Communications Server for z/OS V1R2 TCP/IP Implementation Guide

Volume 5: Availability, Scalability, and Performance

Covers load balancing with DNS/WLM, Sysplex Distributor and MNLB

Describes high availability with dynamic VIPA scenarios

Details z/OS TCP/IP settings to increase performance

Adolfo Rodriguez
Marcia Maria Fascini
Giancarlo Rodolfi
Heather Woods

Redbooks

ibm.com/redbooks

International Technical Support Organization

Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 5: Availability, Scalability, and Performance

October 2002

SG24-6517-00

Take Note! Before using this information and the product it supports, be sure to read the general information in “Notices” on page vii.

First Edition (October 2002)

This edition applies to Volume 1, Release 2 of Communications Server for z/OS IP.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 662
P.O. Box 12195
Research Triangle Park, NC 27709-2195

Contents

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

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

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

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

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

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

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

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

Trademarks

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

Redbooks(logo)™
AIX®
AnyNet®
CICS®
DB2®
DFS™
ESCON®
FFST™
IBM®
IBM eServer™
MVS™
OpenEdition®
OS/390®
PAL®
Parallel Sysplex®
RACF®
Redbooks™
RMF™
RS/6000®
S/390®
SecureWay®
SP™
SP1®
TCS®
Tivoli®
VTAM®
WebSphere®
z/OS™
z/VM™
zSeries™
3890™

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

Domino™
Lotus®

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

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

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

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

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

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

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

Preface

The Internet and enterprise-based networks have led to a rapidly increasing reliance upon TCP/IP implementations. The zSeries platform provides an environment in which critical business applications flourish. The demands placed on these systems are growing and require a solid, scalable, highly available, and highly performing operating system and TCP/IP component. z/OS and Communications Server for z/OS provide for such a requirement with a TCP/IP stack that is robust and rich in functionality. The *Communications Server for z/OS TCP/IP Implementation Guide* series provides a comprehensive, in-depth survey of CS for z/OS.

Volume 5 concentrates on the availability and scalability of z/OS TCP/IP. We cover load-balancing solutions including DNS/WLM, Sysplex Distributor, and the preferred Sysplex Distributor/MNLB joint solution. We further describe mechanisms by which availability is increased on z/OS systems with dynamic VIPA and Automatic VIPA Takeover. Finally, we provide a survey of tuning exercises that can be employed to further enhance the performance of your z/OS TCP/IP system.

Because of the varied scope of CS for z/OS, this volume is not intended to cover all aspects of it. The main goal of this volume is to provide an insight into the different functions available in CS for z/OS to increase availability, scalability, and performance through the use of VIPAs, load-balancing mechanisms, and performance tuning. For more information, including applications available with CS for z/OS IP, please refer to the other volumes in the series These are:

- *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 1: Base and TN3270 Configuration*, SG24-5227
- *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 2: UNIX Applications*, SG24-5228
- *OS/390 eNetwork Communications Server for V2R7 TCP/IP Implementation Guide Volume 3: MVS Applications*, SG24-5229
- *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 4: Connectivity and Routing*, SG24-6516
- *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 6: Policy and Network Management*, SG24-6839
- *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 7: Security*, SG24-6840

The team that wrote this redbook

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

Adolfo Rodriguez is a Senior I/T Specialist at the International Technical Support Organization, Raleigh Center. He writes extensively and teaches IBM classes worldwide on all areas of TCP/IP. Before joining the ITSO, Adolfo worked in the design and development of CS for z/OS, in RTP, NC. He holds a B.A. degree in Mathematics and B.S. and M.S. degrees in Computer Science, from Duke University. He is currently pursuing the Ph.D. degree in Computer Science at Duke University, with a concentration on Networking Systems.

Marcia Maria Fascini is a Systems Specialist in Brazil. She has four years of experience in CICS and six years of experience in the networking field. She holds a degree in Mathematics from Fundação Santo André. Her areas of expertise include VTAM and TCP/IP networks.

Giancarlo Rodolfi is a zSeries FTSS for Latin America. He has 16 years of experience in the zSeries field. His areas of expertise include TCP/IP, z/VM, z/OS, UNIX System Services, CS for z/OS, Security, Linux, WebSphere Application Server, Firewall, TN3270 Server, and Domino. He has written extensively on CS for z/OS TCP/IP services and security.

Heather Woods is a Network Systems Programmer with IBM Strategic Outsourcing in the UK. She has eight years' experience in S/390 systems, and has spent the past four years working mainly with TCP/IP, CS for OS/390 (z/OS), NCP, and SNA.

Thanks to the following people for their contributions to this project:

Bob Haimowitz, Jeanne Tucker, Margaret Ticknor, Tamikia Barrow, Gail Christensen, Linda Robinson
International Technical Support Organization, Raleigh Center

Jeff Haggar, Bebe Isrel, Van Zimmerman, Jerry Stevens, Tom Moore, Robert Perrone, Michael Fitzpatrick, Gus Kassimis, Dinakaran Joseph, Jay Aiken, Jeremy Geddes
Communications Server for z/OS Development, Raleigh, NC

Comments welcome

Your comments are important to us!

We want our IBM 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 Internet note to:

  redbook@us.ibm.com

- Mail your comments to the address on page ii.

1

Introduction

The increasing demands of network servers, and in particular zSeries servers, has led to a focus on performance within Communications Server for z/OS IP. For example, the stack has been improved in terms of performance in every release since its inception. In addition, CS for z/OS IP has seen the creation of different techniques to address performance requirements when a single server is not capable of providing the availability and scalability demands placed on it by its clients. Specifically, network solutions make use of what is referred to as the *clustering* technique, whereby multiple servers are associated together into a *cluster* to provide sufficient processing power and availability characteristics to handle the demands of the clients.

Within the scope of this book, the cluster functionality is provided by the sysplex. That is, the sysplex provides the necessary capability to cluster together a number of zSeries servers that can cooperate with one another to deliver the processing power and availability needed to service the demands required of a particular service environment.

This redbook provides insight into the performance tuning of Communications Server for z/OS IP and solutions implementing various forms of the clustering technique. We compare these approaches, noting the advantages and disadvantages of each. Additionally, we describe the Virtual IP Addressing (VIPA) concept and the high availability problems that it addresses.

A previous redbook regarding this topic was based on a Communications Server for OS/390 V2R8 environment. In this updated redbook, we have not made any changes to some of the examples provided in the earlier edition. This applies to the examples in Chapter 2, “DNS/WLM (connection optimization)” on page 17. The examples are still valid, but you should be aware that the details of the displays may have changed when you implement Communications Server for z/OS V1R2 IP.

1.1 The role of the sysplex

Beyond providing basic tuning guidelines to improve the performance of CS, this book describes solutions utilizing the clustering approach. These functions increase server availability and processing capability and attempt to provide mechanisms by which they ensure the viability of the cluster in an environment containing a large number of clients generating a potentially high number of requests. To do so, the clustering technique can provide for two main objectives: high availability and load balancing. In some cases, clustering techniques address only high availability, as is the case with Dynamic VIPA that provides for availability in spite of potential TCP/IP stack or z/OS image failures. In other cases, the intent is to provide for both high availability and load balancing, as is done by the Domain Name System/Workload Manager solution (DNS/WLM), Network Dispatcher, Sysplex Distributor, and MultiNode Load Balancing (MNLB).

In general, load balancing refers to the ability to utilize different systems within the cluster simultaneously, thereby taking advantage of the additional computational function of each. Further, clustering techniques addressing load balancing lead to other system requirements, such as that of a single system-wide image (one identity by which clients access the system), horizontal growth, and ease of management.

1.1.1 High availability

The traditional view of a single server has been primarily a single machine with perhaps a few network interfaces (IP addresses). This tends to lead to many potential points of failure within the server: the machine itself (hardware), the operating system (including TCP/IP stack) kernel executing on the machine, or a network interface (and the IP address associated with it). Static Virtual IP Addresses (VIPAs) exclude the network interface as a point of failure, while Dynamic VIPAs additionally aid with server (image) or kernel failure. In this way, high availability is seen as the availability of the entire server cluster and the *service* it provides. Further, VIPAs can be used in conjunction with the three load-balancing solutions discussed in this book: DNS/WLM, Network Dispatcher, and Sysplex Distributor.

Clustering techniques that address the load balancing of connection requests also typically provide for some high availability. That is, these techniques *dispatch* connections to target servers and can exclude failed servers from the list of *target* servers that can receive connections. In this way, the dispatching function avoids routing connections and requests to a server incapable of satisfying such requests.

1.1.2 Load balancing

Load balancing is the ability of a cluster to spread workload evenly (or based on some policy) to target servers comprising the cluster. Usually, this load balancing is measured by some notion of perceived load on each of the target servers. This book describes and compares three CS for z/OS IP-based techniques that provide load balancing: DNS/WLM, Sysplex Distributor, and MNLB (in conjunction with Sysplex Distributor). Though not implemented directly in CS for z/OS IP, we also provide some information on Network Dispatcher. Each of these methodologies identifies the target zSeries servers willing to receive client connections based on some specification.

By providing load balancing, clustering techniques must also provide for other system requirements in addition to the dispatching of connections. These include the ability to advertise some single system-wide image or identity so that clients can uniquely and easily identify the service. Additionally, clustering techniques should also provide for horizontal growth of the system and ease of management.

Single system-wide image

Clients connecting to a cluster should not be aware of the internal makeup of a cluster. More specifically, clients should not even be aware that the service they are requesting is actually being serviced by a collection or cluster of servers. Instead, clients must be provided with some single image identifier to be used when connecting to the service. DNS/WLM uses some specific host name to identify a service within the cluster. In this manner, clients making requests of the service use the host name as the single system-wide identity. In Network Dispatcher (ND), Sysplex Distributor, and MNLB, however, the identity is that of some IP address associated with the cluster. In the case of Sysplex Distributor and MNLB (in conjunction with Sysplex Distributor), this address is called a distributed Virtual IP Address (VIPA).

Horizontal growth

As the clients' demands on the service increase, clusters must provide a way to expand the set of cooperating servers to accommodate such growing demand. Put another way, the cluster must provide a mechanism by which to add servers without disrupting the operation of the cluster. To this end, the service is made available to clients at all times and can grow to accommodate the increased demand placed on the cluster by the clients.

Ease of management

The administrative burden associated with the cluster should not increase as we add servers to the cluster. It is desirable to use the same configurations for many systems in the cluster (sysplex). Within a sysplex, servers are homogenous, since a sysplex is comprised solely of zSeries servers. As such, many of the configurations can be shared among the different zSeries servers, thereby reducing the administrative burden associated with the sysplex. Additionally, as the size of the cluster increases, the administrative overhead in adding systems to the cluster should be as low as possible.

1.2 Sysplex overview

Within this redbook, we use the term *sysplex* to refer to a group of loosely coupled z/OS (MVS) images. For example, a sysplex could be comprised of several LPARs within a physical host, or it could be multiple physical hosts connected via ESCON channels. In this way, the sysplex provides the basis for implementations of the clustering technique. For the remainder of this book, we use the terms *cluster* and *sysplex* interchangeably.

Sysplex systems can be distinguished between base sysplex and Parallel Sysplex, the Parallel Sysplex having a coupling facility. The coupling facility is a high-speed shared medium that improves availability and performance by allowing vital data to be stored independently of any attached z/OS system, yet retrieved more quickly than if it were on disk. Although the SNA component of Communications Server for z/OS makes use of the coupling facility to improve service to VTAM users, the IP component does not (although it is enhanced in V1R4 to do so). Therefore, all the functions described in this book apply to a base sysplex as well as to a Parallel Sysplex.

Regardless of their physical connectivity, sysplex z/OS hosts are able to cooperate with each other to such a degree that they are able to share disks, provide backup capabilities for availability, and distribute workload (load balancing).

1.3 Communications Server for z/OS

Communications Server for z/OS (formerly known as IBM Communications Server for OS/390 and SecureWay Communications Server for OS/390) is the communications engine of z/OS. It is comprised of IP (TCP/IP), SNA (VTAM), and AnyNet components. With CS for z/OS, TCP/IP and SNA are very closely integrated in the z/OS environment.

The performance of TCP/IP was greatly improved with the redesign of the stack in CS for OS/390 V2R5, and further improved in subsequent releases. The TCP/IP stack was made multiprocessor-capable in OS/390 Version 1 Release 3; this reduces the advantage of running multiple stacks, although such a configuration is still supported. Although the reasons for running with a multiple stack environment are rapidly fading, we demonstrate multiple stack configurations to aid in the understanding of the concepts involved in doing so. Additionally, in our environment, there are many different tests involved with CS for z/OS IP that require the exclusive use of a particular stack. Running a multiple stack environment allows our LPARs to have more stacks on which to run these tests.

For a complete list of changes in recent releases of the TCP/IP for MVS product, see *z/OS V1R2.0 CS: IP Migration*, GC31-8773.

1.4 Network interfaces to the sysplex

One of the major ways in which the SNA and IP components of CS for z/OS (formerly VTAM and TCP/IP for MVS respectively) are integrated is the use of a common DLC connection manager to handle most of the network connections available to CS for z/OS. The DLC connection manager implements the following DLCs:

- Multipath Channel Plus (MPC+)
- Asynchronous Transfer Mode (ATM)
- XCF (see 1.6, “Cross-system coupling facility” on page 5)
- LAN Channel Station (LCS)
- Channel Data Link Control (CDLC)
- Channel to Channel (CTC)
- HYPERchannel
- Common Link Access to Workstations (CLAW)
- SAMEHOST, providing connectivity between the stack and gateway applications:
  - X.25
  - SNALINK
  - SNALINK LU 6.2
- Queued Direct I/O (QDIO) providing direct memory access to the OSA-Express for connectivity to Gigabit and Fast Ethernet
- Hipersockets

The first three of these (MPC+, ATM and XCF) are shared between VTAM and TCP/IP; the remainder are used exclusively by TCP/IP. VTAM retains its own DLC management for all SNA connections other than the three listed here.

In viewing the collection of zSeries images within a sysplex as a cluster, we can in turn view network interfaces available on each of the images collectively as the set of network interfaces available to the sysplex as a whole. That is, systems in the sysplex may take advantage of network interfaces attached to other systems in the sysplex. Potentially this may include high-performing network interfaces, such as the QDIO interface to the OSA-Express.

1.5 Workload Manager

The Workload Manager (WLM) is a component of z/OS used to control work scheduling, load balancing, and performance (goal) management. Additionally, WLM can provide information regarding the current load on systems within the sysplex. It is because of these features that WLM is the basis for operation within DNS/WLM, Network Dispatcher, and Sysplex Distributor. That is, periodic updates received from WLM tell the server which hosts in the sysplex have the most processing resources available so that these systems can receive more of the workload from the *dispatching agent*, the function that routes connection requests based on feedback from WLM.

To take advantage of WLM-based sysplex load balancing (as opposed to round-robin), TCP/IP requires that WLM be configured in *goal mode*. For more information on how WLM functions, see *z/OS V1R2.0 MVS Planning: Workload Management*, SA22-7602. For more information specific to CS for z/OS IP, see *z/OS V1R2.0 CS: IP Configuration Guide*, SC31-8775.

1.6 Cross-system coupling facility

The z/OS systems in a sysplex have an additional communication path between them that is denied to non-sysplex systems. This is the cross-system coupling facility (XCF), not to be confused with the coupling facility that marks a Parallel Sysplex. Data using XCF connections may travel over ESCON channels or through the coupling facility, depending on the configuration. CS for z/OS can use XCF for communication between sysplex members (for coordination among them) and for the transfer of IP packets between them. The SNA component (VTAM) requires no definitions to use the facility, but the earlier releases of the IP component had to have them predefined.

Starting with OS/390 V2R7 IP, you have the option of defining IP connectivity over XCF to other TCP/IP stacks dynamically. XCF dynamics provides nondisruptive horizontal growth for TCP/IP in a sysplex, allowing you to add new TCP/IP images without requiring the coordination of definitions for existing sysplex members. Because only a single definition for each new TCP/IP image is needed to establish IP connections between every system in the sysplex, it is easier to scale to handle higher workloads without impacting existing systems and their users. XCF dynamics automatically create device and link definitions and you need to define only one new IP address per system.

Aside from transferring IP traffic, CS for z/OS IP can use XCF signalling for communication and coordination between the stacks themselves. The dynamic VIPA function (see Chapter 3, “Dynamic VIPA (for application instance)” on page 61), for example, uses XCF signalling to coordinate the placement of VIPA addresses within a sysplex.

1.7 High availability with Virtual IP Addressing (VIPA)

The original purpose of (static) VIPA was to eliminate a host application's dependence on a particular network attachment. A client connecting to a server would normally select one of several network interfaces (IP addresses) to reach the server. If the chosen interface goes down, the connection also goes down and has to be reestablished over another interface. Additionally, while the interface is down, new connections to the failed interface (and IP address) cannot be established.

With VIPA, you define a virtual IP address that does not correspond to any physical attachment or interface. CS for z/OS IP then makes it appear to the IP network that the VIPA address is on a separate subnetwork, and that CS for z/OS itself is the gateway to that subnetwork. A client selecting the VIPA address to contact its server will have packets routed to the VIPA via any one of the available real host interfaces. If that interface fails, the packets will be rerouted nondisruptively to the VIPA address using another active interface.

CS for OS/390 V2R8 IP extended the availability coverage of the VIPA concept to allow for the recovery of failed system images or entire TCP/IP stacks. In particular, it introduced two enhancements to VIPA:

- The automatic VIPA takeover function allows you to define the same VIPA address on multiple TCP/IP stacks in a sysplex. One stack is defined as the primary or owning stack and the others are defined as secondary or backup stacks for the VIPA. Only the primary one is made known to the IP network. If the owning stack fails, then one of the secondary stacks takes its place and assumes ownership of the VIPA. The network simply sees a change in the routing tables. In this case, applications associated with these DVIPAs are active on the backup systems, thereby providing a *hot standby* for the services.
- Dynamic VIPA (for an application instance) allows an application to register to the TCP/IP stack with its own VIPA address. This lets the application server move around the sysplex images without affecting the clients that know it by name or address; the name and address stay constant although the physical location of the single application instance may move. In this way, the application can dynamically activate the VIPA on the system image it wishes to host the application. Because the application instance is only active on one image in the sysplex at a time, the other images provide a *cold standby* of the service.

These VIPA enhancements were enabled by the use of XCF to communicate between the TCP/IP stacks. That is, XCF has become the basis for communication regarding VIPAs within the sysplex. Because of the ease of configuration provided by XCF dynamics, many of the newer VIPA functions in turn are easily configurable.

Because of the different functions provided by each of the two flavors of Dynamic VIPA, we recommend these general rules in regards to the applicability of each:

- If load balancing is required, you should consider using DNS/WLM, Network Dispatcher, Sysplex Distributor, or MNLB as outlined in 1.8, “Providing load balancing and high availability simultaneously” on page 7.
- If more than one instance of the application can run simultaneously within the sysplex and the flexibility to dynamically activate the DVIPA by starting the application is not necessary, you should look toward automatic VIPA takeover. Note that in this case, the DVIPA will not move unless the stack owning the DVIPA has failed, at which time the backup stack assumes ownership.
- In the event that multiple applications cannot run simultaneously in the sysplex, or the ability to activate the DVIPA when the application starts is desired, we recommend the use of a Dynamic VIPA for an application instance.

1.8 Providing load balancing and high availability simultaneously

The main objective of a sysplex is high availability without compromising perceived client performance. High availability requires a number of servers providing the same service to their clients so that these servers allow for the recovery of the service in the presence of failures. That is, servers perform some sort of backup functionality for each other within the cluster.

In contrast, load balancing ensures that such a group or cluster of servers can maintain optimum performance by serving client requests simultaneously. Additionally, an evenly spread workload minimizes the number of users affected by the failure of a single server. Thus, load balancing and availability are closely linked; in this chapter, and throughout the book, we consider these two functions together as they apply to TCP/IP in a sysplex.

The ultimate goal is, of course, to provide 100% *perceived* service availability and at the same time provide top performance to end users when they request server functions. The latter is achieved by implementing some sort of *connection dispatching* technology such as DNS/WLM, Network Dispatcher, Sysplex Distributor, or MNLB. Because these solutions can exclude failed servers from connection reception, they also inherently allow for increased service availability. Additionally, all of these solutions can take advantage of increased availability associated with Virtual IP Addresses.

In general, there are various ways of addressing load balancing within a cluster:

- One technique is to let the users themselves choose a server at random from a number of host names or addresses. When users try to connect to a server they are not aware if the server is available, nor do the users know how well the server will perform when they connect to it. This approach is quite common but very inefficient. Web-based applications use this technique by creating multiple copies of Web pages with different explicit HTTP links.
- Another technique is round-robin, in which a function independent of the users selects a server to handle requests. This approach is better, but it does not take into consideration the current load on the target server or even whether the target server is available.
- A third approach is to use a simple advisor that checks availability (and, perhaps, the number of connections) on different servers to some extent, before selecting a server. With this approach, failed servers can be excluded from server selection, thereby increasing availability of the service.
- The most sophisticated technique is to have performance agents in all application servers that feed managers with statistics; the absence of any statistics also gives an indication of non-availability. The managers, armed with this information, select servers based on the overall service levels that they expect to be delivered.
- A final approach could be to grow the size of a single server, avoiding the use of a clustering technique altogether. This approach can be extremely costly and not possible in some circumstances (if, for example, the current demand on the service could overload even the most powerful of servers).

Additionally, all of these clustering techniques could be used together to some extent. In this way, simplicity, accuracy, and performance are balanced somewhat.

Availability is also highly dependent on the IP network, not so much on the transport network between servers and clients (because IP can reroute around failures), but on the network adapters through which the servers access the network. By using functions such as Virtual IP Addressing (VIPA), together with sophisticated routing techniques such as OSPF equal-cost multipath, we can improve availability even further.

In this book we have used the three main approaches available to TCP/IP users in a sysplex to perform load balancing and to accomplish high availability. These techniques can (and usually do) make use of the MVS Workload Manager to distribute IP traffic across a number of servers, but they are different in the approach they take. While we summarize Network Dispatcher in this introduction, we do not provide detailed information on its implementations, since it is a function external to CS for z/OS IP. Hence, this book concentrates on the following z/OS-based solutions: DNS/WLM, Sysplex Distributor, and MNLB (in conjunction with SD).

1.8.1 DNS/WLM solution

The DNS solution is based on the DNS name server BIND 4.9.3 and the z/OS Workload Manager. Intelligent sysplex distribution of connections is provided through cooperation between WLM and DNS. For customers who elect to place a name server in a z/OS sysplex, the name server can utilize WLM to determine the best system to service a given client request. Please note that the latest version of DNS, BIND 9, does not support this connection dispatching capability. As a result, you must continue to use BIND 4.9.3 for this function.

In general, DNS/WLM relies on the host name to IP address resolution for the mechanism by which to distribute load among target servers. Hence, the single system image provided by DNS/WLM is that of a specific host name. Note that the system most suitable to receive an incoming client connection is determined only at the time of host name resolution. Once the resolution is completed, the client may cache the results and therefore re-use the address on subsequent connections.

The DNS approach works only in a sysplex environment, because the Workload Manager requires it. If the server applications are not all in the same sysplex, then there can be no single WLM policy and no meaningful coordination between WLM and DNS.

The operation of the DNS/WLM combination is described more fully in Chapter 2, “DNS/WLM (connection optimization)” on page 17, but in essence it functions as follows:

- The TCP/IP stack registers its IP addresses with WLM.
- The servers register with WLM under a particular *server group name* for the purpose of providing some service described by this name (for example, TN3270). WLM then matches each server instance with the IP addresses registered by the TCP/IP stack in that same LPAR.
- At regular intervals, DNS asks WLM for its workload measurements.
- The client requests a server by IP *host name* via a process called *host name resolution*. This host name is the server group name known to WLM.
- The DNS name server in the sysplex checks its WLM information for details of the server instances registered under that host (server group) name.
- DNS responds to the client with the IP address of the most suitable server instance so that the client can connect to that specific server.

All BIND-based name servers use a simple sorting algorithm (by default) when returning one of multiple addresses during host name resolution. This is more or less the basic round-robin technique. However, WLM also has some additional notion of performance or server load and can return addresses for the server with the least current load. As a result, we get a lot more than just simple round-robin selection.

WLM will provide host and application weights that help the name server to load balance the connections to the sysplex servers. If the application is registered to WLM, the name server will resolve the client's name query for the application; and if the application is de-registered, WLM will no longer provide its details to the DNS. This means that dead applications will not be selected by the DNS.

The advantages of the DNS approach are:

- Familiar technology

  Most installations already utilize DNS in their organizations even though it may not have been implemented on z/OS. Additionally, the technique of simply modifying host name to IP address mapping for load balancing is somewhat easier to understand than other clustering techniques.

- No additional software needed

  The prerequisites are there when you have a z/OS system; it is just a matter of configuration. That is, the DNS server is included with the z/OS operating system as part of CS for z/OS IP.

- Performance

  Once the host name is resolved to a server address, there is no more involvement from the DNS/WLM load balancing function. No state needs to be maintained to identify the current connection and its distribution.

- High availability

  Secondary name servers can be implemented in a sysplex to fulfill the functions of the primary one should it fail. Further, the DNS/WLM function will not distribute connections to failed application servers within the sysplex. In this way, perceived client availability is increased.

- Ease of use

  Users (clients) are not aware of changes in the location or the IP address of an application server. Clients simply resolve some host name and use the IP address returned on the host name resolution request.

- Ease of administration

  Because applications register with DNS/WLM, adding servers as candidates for connection requests is administratively simple and dynamic.

Possible drawbacks include:

- Potential end-user configuration

  The users may need to learn new names for the generic applications or services, although the use of aliases can reduce or eliminate this work. DNS/WLM creates these names dynamically as applications register for use with the function.

- Distribution available for sysplex systems only

  Because of its inherent dependence on the sysplex, DNS/WLM can only be used with applications running within the sysplex and cannot be used with applications running on other servers.

- Requires application support

  Because applications must register with DNS/WLM, application support is necessary to take advantage of the function. Currently, not many applications provide this support.

- Users can bypass load balancing

  Users that know the real host names or IP addresses of the servers can bypass the DNS/WLM load balancing algorithm altogether.

- Stale WLM information

  The name server queries WLM periodically for updated information, by default every 60 seconds. This means that the name server's information may become stale between intervals. Additionally, clients and intermediate name servers can cache host name-to-IP address mappings (even if they contain a low TTL) and propagate stale mappings.

- Increased network utilization

  To avoid caching side effects, administrators can reduce the TTL field in DNS resource records (RRs). This generally causes increased network utilization because clients must resolve host names on every connection request.

1.8.2 Network Dispatcher

The Network Dispatcher (ND) is load-balancing software that is part of the WebSphere Edge Server. ND determines the most appropriate server to receive new incoming IP connections. This section gives an overview of Network Dispatcher. Please reference *TCP/IP in a Sysplex*, SG24-5235 for a more detailed description of Network Dispatcher.

With the Network Dispatcher, it is possible to link many servers that provide equivalent applications with common data into what appears to be a single virtual server. Servers may have different hardware architectures and operating systems, as long as the TCP/IP services are the same. The clients just reference one special IP address (known as a *cluster* address), which is shared between the cluster of servers and the Network Dispatcher. All client requests to the shared IP address are sent to the Network Dispatcher. The Network Dispatcher then selects the optimal server at that time and sends the connection request to that server. The server sends a response back to the client directly, without any involvement of the Network Dispatcher.

The Network Dispatcher also provides a high-availability option, utilizing a standby machine that remains ready to take over load balancing in case of failure of the primary Network Dispatcher.

The Network Dispatcher technique does not depend on host names; rather it provides an IP address (the cluster address) as the single system image specification. Clients send connection requests to the cluster address. These packets reach the Network Dispatcher, which then forwards them to the chosen server. ND has knowledge of the available servers through advisors that keep a watch on various protocols (HTTP, Telnet, FTP) and an MVS advisor that communicates with WLM regarding server load. ND uses all the information obtained from the advisors to select a server.

All packets from the client to the server pass through the Network Dispatcher, since the IP network knows only one address for the servers (the cluster address) and that address belongs to the ND. From the server back to the client, packets use normal IP routing because the client's IP address is given to the server as the source of the packet.

Although the ND solution will function with separate hosts (not part of the same sysplex), the load balancing will not work correctly for the same reason as in the DNS case: WLM instances in separate sysplexes do not communicate with each other. In this case ND will rely on its own perception of the workload, which is confined to the inbound TCP/IP traffic.

Advantages of the Network Dispatcher approach include:

- Ease of configuration

  It is very easy to change the server configuration, for example, to add, quiesce, stop and delete servers dynamically.

- Comprehensive advisors

  An MVS advisor that connects to WLM gets load metrics for connection optimization, and also determines availability. Protocol advisors poll different ports and measure response times. This ensures availability, functionality, and high performance.

- High availability

  There is a built-in option to configure a standby ND that remains ready to take over if the primary fails.

- Easy integration

  The end users do not know that the ND exists; they just connect to a new server IP address (or host name).

- Flexibility

  The ND solution can be used with IP hosts other than a sysplex, although the workload-balancing functions will not be able to use WLM input.

- Independence from DNS

  ND does not depend on host name resolution for load balancing. Rather, the workload distribution occurs at TCP connection setup. As a result, ND is not susceptible to host name caching effects or increased network utilization resulting from low TTL settings of DNS resource records.

Possible disadvantages of ND may be:

- Extra hardware costs

  Currently, the Network Dispatcher is part of the WebSphere Edge Server, which is available on AIX, Windows NT, Red Hat Linux, and Sun Solaris operating systems. Hence, additional hardware running on these operating systems is required. ND is also implemented in discontinued IBM hardware, such as the IBM 2216.

- Performance and capacity

  Advisors typically poll servers for information every five seconds. Together with the fact that each packet may (depending on the configuration) require one extra hop on the IP network, this can place additional loading on that network. In addition, the ND machine must maintain knowledge of the TCP connections to the servers in the cluster and thus requires more capacity than a simple router.

- Incompatibility with IPSec and VPN

  IPSec in tunnel mode encrypts the true destination IP address, so Network Dispatcher cannot be an intermediate node on an IPSec connection. It must itself be the endpoint (firewall). In the kinds of environments we are describing, however, that is probably not an issue, since the Network Dispatcher will usually be in a secure environment.

- FTP support inconsistent

  Because FTP connection flow is somewhat different from the traditional client/server application flow, ND must make special considerations for this protocol. In general, FTP connections originating from the *same* client are always directed to the same target server. This may be an issue when using an FTP proxy in which a larger number of clients appear to be a single client.

1.8.3 Sysplex Distributor

Sysplex Distributor is the state of the art in connection dispatching technology among z/OS IP servers. Essentially, Sysplex Distributor extends the notion of Dynamic VIPA and automatic VIPA takeover to allow for load distribution among target servers within the sysplex. It combines technology used with Network Dispatcher for the distribution of incoming connections with that of Dynamic VIPAs to ensure high availability of a particular service within the sysplex.

Technically speaking, the functionality of Sysplex Distributor is similar to that of Network Dispatcher in that one IP entity advertises ownership of some IP address by which a particular service is known. In this fashion, the single system image of Sysplex Distributor is also that of a special IP address. However, in the case of Sysplex Distributor, this IP address (known as the cluster address in Network Dispatcher) is called a *distributed DVIPA*. Further, in Sysplex Distributor, the IP entity advertising the distributed VIPA and dispatching connections destined for it is itself a system image within the sysplex, referred to as the *distributing stack*.

Like Network Dispatcher and DNS/WLM, Sysplex Distributor also makes use of Workload Manager (WLM) and its ability to gauge server load. In this paradigm, WLM informs the distributing stack of server load so that it may make the most intelligent decision regarding where to send incoming connection requests. Additionally, Sysplex Distributor has the ability to specify certain policies within the Policy Agent so that it may use QoS information from target stacks in addition to WLM server load. Further, these policies can specify which target stacks are candidates for clients in particular subnetworks.

As with ND, connection requests are directed to the distributing stack of Sysplex Distributor. The stack selects which target server is the best candidate to receive an individual request and routes the request to it. It maintains state so that it can forward data packets associated with this connection to the correct stack. Additionally, data sent from the servers within the sysplex to clients need not travel through the distributing stack.

Sysplex Distributor also enhances the Dynamic VIPA and automatic VIPA takeover functions introduced in SecureWay Communications Server for OS/390 V2R8 IP. The enhancements allow a VIPA to move *nondisruptively* to another stack. That is, in the past, a VIPA was only allowed to be active on one single stack in the sysplex. This led to potential disruptions in service when connections existed on one stack, yet the intent was to move the VIPA to another stack. With Sysplex Distributor, the movement of VIPAs can occur without disrupting existing connections on the original VIPA owning stack.

In summary, Sysplex Distributor offers the following advantages:

- Ease of configuration

  Sysplex Distributor takes ease of configuration to another level. The initial configuration of a distribution is made extremely easy. Additionally, servers can be added to a distribution without the need for any configuration.

- More accurate measure of server load

  Sysplex Distributor makes use of WLM-provided server load information. But it can also use QoS performance metrics from target servers to guide in the selection of target servers on incoming connections. Additionally, the set of potential target stacks can be different depending on which client is requesting the connection. This allows for the reservation of particular stacks to some subset of clients in a straightforward manner.

- The ultimate in availability

  Sysplex Distributor can function with automatic VIPA takeover to ensure that the distribution of connections associated with a particular service is made available. Because every stack in the sysplex distribution can be a backup distributing stack, the survival of just one system image ensures the availability of the service. In this way, target servers can become backup stacks for the distribution of incoming connections.

- Easy integration

  End users are not aware of the distribution being performed by Sysplex Distributor; they just connect to a server IP address (or host name).

- Independence from DNS

  Sysplex Distributor does not depend on host name resolution for load balancing. Rather, the workload distribution occurs at TCP connection setup, as with ND. As a result, Sysplex Distributor is not susceptible to host name caching effects or increased network utilization resulting from low TTL settings of DNS resource records, as is the case with DNS/WLM.

- No additional hardware required

  Because all of the Sysplex Distributor function is contained within the sysplex, no additional hardware is necessary to take advantage of this function.

- Performance

  Again, because Sysplex Distributor is contained within the sysplex, CS for z/OS IP can take advantage of the homogeneity within the cluster. That is, forwarding connection and data through the distributing stack is done extremely fast. Additionally, extra communication occurs between the stacks in the sysplex, allowing for fast recognition of VIPA failure and enhanced backup functions.

- No need for application-specific pings

  Another advantage is that Sysplex Distributor knows when an application establishes a relevant listening socket on a target stack, through exchange of messages between the target stack and the routing stack, so no application advisors or pings are required to determine server instance availability.

Possible disadvantages of Sysplex Distributor may be:

- The cluster *is* the sysplex

  In Sysplex Distributor, all target servers must be zSeries servers and resident within a single sysplex. In some regards, this limits the flexibility in having heterogeneity among servers within the cluster, as is available with ND.

- Incompatibility with IPSec and VPN

  As with ND, the end of an IPSec tunnel mode must not be a target server. This is hardly much of a limitation, but worth mentioning. Additionally, this restriction is removed in V1R4.

- FTP support limited

  Being exposed to the intricacies of the FTP protocol, as was Network Dispatcher, Sysplex Distributor has limited support. The distribution of non-passive mode FTP is fully supported, including the capability of establishing data connections. Passive mode FTP, however, is currently not supported. This restriction is also removed in V1R4.

- Lack of TN3270 printer session support

  Another limitation is support for TN3270 printer sessions. These are established via a second, parallel connection from the client.

1.8.4 MultiNode Load Balancing and Sysplex Distributor

Cisco's MultiNode Load Balancing (MNLB) architecture is designed to perform similar functionality as Network Dispatcher and Sysplex Distributor. In its essence, MNLB dispatches connections by distributing incoming TCP SYN packets to destination target servers. The one main advantage that this solution has over ND and Sysplex Distributor is that the forwarding decision of where to send data packets is done within the router (or switch) where the packets are processed anyway. Note that the initial selection of target servers for incoming connections need not necessarily be done with the router (forwarding agent). The disadvantage, however, is that information regarding target server load must be pushed out to the router so that it may select the appropriate target server. This introduces issues of staleness as this information rapidly becomes stale and not indicative of current server load and increase the load on the network outside the sysplex.

With CS for z/OS V1R2 IP, the MNLB architecture can cooperate with the Sysplex Distributor function. In this paradigm, the connection dispatching function once performed by Sysplex Distributor is divided into two main portions: the distribution of connections and the forwarding of data. What results is a 'best of both worlds' scenario in which the Sysplex Distributor leverages its strength in using up-to-date information to select the most appropriate target stack to receive a connection and in which the MNLB router can learn of this selection and can directly forward data to target servers without the need to place additional load on the Sysplex Distributor. That is, the Sysplex Distributor is relieved of the job of multiplexing incoming data packets to target servers while the MNLB router is relieved of the task of selecting the appropriate target server with potentially state information.

In the MNLB architecture, the entity that dispatches connections to the appropriate target server is called the Service Manager. The entity that forwards data to target servers is called the Forwarding Agent. Hence, in this hybrid environment, Sysplex Distributor becomes the Service Manager and the Cisco router is the Forwarding Agent.

In summary, this hybrid MNLB/SD solution has the following advantages over a pure Sysplex Distributor or a pure MNLB solution:

- The router performs the forwarding of data packets to target servers, thereby substantially reducing the load placed on the Sysplex Distributor. Routers have been engineered to forward packets very quickly. This solution allows routers do what they do best.
- The Sysplex Distributor makes the appropriate target stack selection based on current information. The distributing stack in the sysplex is the best entity to make the right decision as to which target server should receive an incoming connection request. This solution leverages this as well.
- Separation of distinct functionality. The MNLB architecture allows the separation of distinct functionality so that each may be placed in the appropriate location. For example, this solution encourages routers to be the Forwarding Agents, a function that is clearly in the router domain.

In contrast, the MNLB/SD joint solution suffers from the following drawbacks:

- Increased complexity. Of course, whenever new hardware is brought into the mix, complexity is likely to increase. We must now be familiar with CS for z/OS and with Cisco router configuration to make this work. Though Cisco's penetration into the market implies that most people have Cisco experience, the complexity of the architecture is not trivial.
- Additional hardware. In order to fully leverage the advantages of this functionality, you must have an MLNB-supporting Cisco router. This implies increased cost. However, if such a router already exists in your network, the price of this solution is suddenly more appealing.

1.8.5 Which solution is best?

The connection dispatching technologies described within this chapter all implement some sort of clustering technique. All of these techniques can learn the performance and availability characteristics of the target servers to which traffic is directed. Sysplex Distributor (and MNLB/SD) provides a bit more functionality here by taking into consideration QoS performance metrics in dispatching decisions.

Also, all of these techniques can be configured (by means of redundancy) to provide very high availability. In the case of DNS/WLM and Network Dispatcher, the entity making dispatching solutions can have a single backup. With Sysplex Distributor (and MNLB/SD), the dispatching function can be backed up by all systems within the sysplex, yielding increased availability characteristics.

Because of these and the other benefits of Sysplex Distributor, we generally recommend it as the dispatching solution of choice. To leverage Cisco router functionality, we recommend the use of Sysplex Distributor with Cisco MNLB-enabled routers.

However, DNS/WLM and Network Dispatcher should be considered in some cases. In general terms, the circumstances under which the DNS/WLM solution and/or Network Dispatcher should be considered are:

- It is desired that host name be the single system image rather than IP address
- Passive mode FTP must be supported
- Heterogeneity among target servers is desired

If any of these conditions are the case, then you might consider the use of either DNS/WLM or Network Dispatcher depending on the type of application. In general:

- If the application tends to use a large number of short connections (UDP or Web access), use Network Dispatcher because it does not require name resolution on every connection.
- If connections are long-lived (Telnet and especially FTP), use DNS/WLM. The overhead of name resolution is infrequent, and thus small compared with the performance gained by not sending the traffic through the ND function.

2

DNS/WLM (connection optimization)

Officially known as *connection optimization*, DNS/WLM provides intelligent sysplex distribution of requests through cooperation between the WLM and the DNS server. For customers who elect to place a DNS in a z/OS sysplex, DNS will invoke WLM sysplex routing services to determine the best system to service a given client request.

WLM provides various workload related services: performance administration, performance management and workload distribution. WLM is capable of dynamically assessing resource utilization on all participating hosts within a sysplex.

A DNS server running on a host in the sysplex can take advantage of WLM's knowledge and use it to control how often an address for a particular host in the sysplex is returned on a DNS query. When a sysplex name server queries the WLM for information, it is provided with a weight corresponding to the relative resource availability of each participating host in the sysplex. These weights are used by the name server to control the frequency with which an address will be returned for a given host. If a host in the sysplex is relatively busy, its address will not be returned by the server as often as a less busy host's address. As you might have guessed, this means that you *must* use host names when accessing an application in the sysplex. If you use an IP address directly, no workload balancing can occur with the DNS/WLM solution.

Please note that DNS BIND 9 does not support connection optimization. You must use BIND 4.9.3, which is still currently being shipped with CS for z/OS IP. For more information, please refer to *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 2: UNIX Applications*, SG24-5228.

2.1 Domain Name System (DNS) overview

This section provides a very brief overview of the Domain Name System (DNS). This subject is very complex and numerous books on DNS have been written. One of the most popular books is *DNS and BIND* by Paul Albitz and Cricket Liu. If you are going to implement a DNS, we strongly recommend you refer to such texts on the subject for more complete descriptions.

2.1.1 Why DNS?

The TCP/IP applications refer to host computers by their IP addresses. IPv4 addresses are numeric, in the format `nnn.nnn.nnn.nnn` where `nnn` can range from 0 to 255 (with a few exceptions). The major drawback of this system is that, for most people, numbers are difficult to remember. As a result, today's IP-based networks use a mapping of host names to host numbers or addresses. The obvious advantage of this name-to-IP address mapping is that we can assign easily remembered names to hosts in the network. For example, what if we map the host `Garth` to the number 9.24.104.200? We no longer need to memorize the numeric address; just use the name `Garth` instead. What happens, though, if another `Garth` wants to use this name on the network too? The Domain Name System not only handles the name to address (and vice versa) mapping, it also encompasses a system that is capable of ensuring that names are unique throughout all interconnected networks.

2.1.2 What is the Domain Name System?

The Domain Name System (DNS) is a distributed database providing mappings between host names and IP addresses. Essentially, the increased importance of host names and the size of the Internet hosts file led to the creation of the DNS. It is now the method of choice for resolving host names to IP addresses and vice versa.

The DNS is a client/server model in which programs called *name servers* contain information about host systems and IP addresses. Name servers provide this information to clients called *resolvers*.

Logistically, it is best likened to a hierarchical file system. All levels of directories in a file system begin with a root directory. All levels of a domain in the DNS also begin with a root domain. Instead of separating each level of domain with a slash ("/"), the DNS uses a dot (".").

Controlling the names

The uniqueness of host names within a domain is managed in a similar fashion to the way file names are used within a directory. For example, the files /bin/matt and /sbin/matt are obviously different. The DNS uses the same principle, but the root directory is listed on the far right, and successive subdomains (equivalent to successive subdirectories) are listed from right to left. For example, if we have an address such as:

```
buddha.ral.ibm.com
```

our highest (or closest to the root) domain is `com`. Note that the root domain is represented by a dot, just as on a file system the root directory is a slash. The same address could correctly be written as

```
buddha.ral.ibm.com.
```

Often we leave the final dot out of the address, but you will see situations later in this text where it becomes very important. The next subdomain is `ibm`, and we continue down to the lowest subdomain of `buddha`.

At this point, you might be wondering where the hosts are. Any domain name can represent a host while at the same time it can represent the domain of a group of hosts (or more subdomains even). In other words, we know the domain `ibm.com` represents the domain of the IBM Corporation, but there could also be a host called `ibm.com` out there as well.

So how does DNS get hold of these name-to-address mappings? The DNS is essentially a distributed database system. A network administrator chooses a host on the network as a DNS server. This server will usually have a zone for which it is responsible for resolving names to addresses (and addresses to names, called reverse mapping). A zone can describe an entire domain of mappings, more than one domain, or only a subset of a domain. In each case, the server will refer to a configuration file containing simple lists of address and name records, often referred to as a data file. A host-to-address (and vice versa) mapping is referred to as a *resource record*. For example:

```
mvs03a     IN    A   172.16.250.3
```

is the resource record that maps host mvs03a to the address 172.16.250.3.

The name server's job is to respond to queries by providing either an address for a name, or a name for a supplied address. Initially, each server will know only about the hosts within the zones for which it is configured to respond. Caching allows the server to learn and remember data acquired from other name servers. It should also be noted that the name-to-address mapping can be one-to-many, because a host can have more than one IP address.

Finding an address

Hosts on a network must be configured to look for a DNS server when a host name is used instead of an address. The request for name resolution is handed to a resolver routine. The resolver routine will have an address or list of addresses that points to hosts running a DNS server. In the z/OS TCP/IP environment, this is controlled by the NSINTERADDR parameter. The resolver routine will send the query to the host listed in NSINTERADDR, and the resolver routine waits for a response and passes the answer back to the application that requested the resolution. When a query is sent to a name server and the name server is expected to find the answer, this is referred to as a *recursive query*. Later, we discuss a situation where we simply want a name server to give us the best answer it has (that is, the name and address of a more likely name server). This is referred to as an *iterative (or non-recursive) query*.

It can happen that a name server does not know the mapping that is being requested. When this happens, there are several courses of action the server can take. Usually, there is a name server record pointed to by a data file that maps a domain name to a specific server for resolution. This hard-coded record gives the name server a mapping between a domain for which it does not have data and the address of a name server that should have the mapping. That name server will respond back to the original name server with its answer, and the original name server will then respond back to the resolver routine on the host that originated the request.

So what if we get a request for a domain that is completely separate from any domain for which we have data files? This distributed database Domain Name System contains something called a root name server. A root name server's purpose in the DNS world is to provide other servers with information on where to find the top-level domain server for a given domain. In other words, if a name server gets a request for `where.world.ca`, and the name server knows nothing about the `ca` domain, we can send the request to our root name server. The root name server probably will not resolve the request, but it should return the address of a name server more likely to be able to resolve the request (for example, it could return the address for a name server responsible for a `world.ca` zone). This process of sending the request to another name server and receiving ever better responses is called *iterative resolution*.

Once we have the IP address of the host, the work of the DNS is done.

Reverse mappings

Sometimes, we might already know an IP address, but we want to find the host name associated with the address. When such a request comes to a name server, it is referred to as a *reverse lookup*. Reverse mappings are considered to be in a domain called `in-addr.arpa`. The term `in-addr.arpa` is associated with the actual coding of the resource record for a reverse mapping. For example, the reverse mapping for host mvs03a would look like this:

```
3.250.16.172    IN  PTR     mvs03a.buddha.ral.ibm.com.
```

Primary (master) and secondary (slave) servers

As with any good distributed database system, we do not want to duplicate our database entries, but we also want to have backup data available in the event of a problem. When we implement a name server, we have the ability to load some or all of our data file contents from another name server dynamically. If a name server is running as the primary one within a zone, this indicates that we own our data file resource records (forward and/or reverse mappings). The data physically exists on the system where the name server is running.

Alternatively, a name server can indicate it wants to act as a secondary (also referred to as a slave) server for a particular zone. To do this, we provide the name of the zone and the address of the primary (also referred to as the master) name server from which we want to transfer the data. When the name server starts up, it will request a zone transfer from the primary name server, and will load its data files dynamically with the received data.

There should be only one primary server for any given zone, but there can be many secondaries.

2.1.3 DNS implementation with CS for z/OS IP

Communications Server for z/OS IP provides two Domain Name System implementations based on Berkeley Internet Name Domain (BIND). The ported BIND servers enable the ability to implement a z/OS-based DNS that is similar to de facto standard UNIX implementations of DNS in configuration and processing. Please note that the later implementation, BIND 9, does not support connection optimization. BIND 4.9.3 does support it and is therefore used as the basis for the discussion in this chapter.

The BIND-based DNS was first made available as a kit with OS/390 Version 2 Release 4 and could be run as an OpenEdition (now UNIX System Services) server on an OS/390 TCP/IP OpenEdition stack or an IBM TCP/IP Version 3 Release 2 for MVS stack. The BIND DNS has since been enhanced and is available as part of Communications Server for z/OS.

DNS/WLM provides intelligent sysplex distribution of requests through cooperation between WLM and the DNS server. WLM provides sysplex services to determine the best system to service a given client request.

2.1.4 Files to support a DNS implementation

A DNS server usually contains at least three configuration files:

1. A startup file or boot file, by default /etc/named.boot.
2. A data file or zone file that maps host names to IP addresses for specific domains. We refer to this zone file as the *forward domain* file. It conventionally uses a suffix of *for*, as in named.for. The file name is usually the zone name, but it can be anything.

3. A data file or zone file that resolves IP addresses to host names for IP networks. We refer to this zone file as the *reverse domain* file, or the in-addr.arpa file. It conventionally uses a suffix of *rev*, as in named.rev. Again, the file name is usually the zone name, but it can be anything.

Optionally a DNS server may have additional configuration files:

- Extra forward files for additional zones that the name server may be servicing
- Extra reverse files if the name server manages more than one IP network
- *A loopback file*, which by convention uses the suffix *lbk*, to define the loopback names and addresses
- *A cache file* that references standard domains in the Internet
- Forward and reverse zone files that are used to manage a name server that cooperates with WLM
- Forward and reverse zone files that are used in a network with DHCP and the Dynamic Domain Name System (DDNS)

All of these files may be present at both primary and secondary name servers. A secondary name server obtains most of its data from the primary name server through a process called *zone transfer*. The secondary server uses its forward and reverse files to store the data obtained from the primary name server.

2.2 How load distribution works using DNS/WLM

Before any application can take part in workload balancing (connection optimization), the TCP/IP stack *should* register itself to WLM. It, and only it, can correctly maintain the IP addresses that DNS/WLM will need to resolve a host name to an address. Once the stack has been registered, WLM knows its name and its interface addresses. The SYSPLEXRouting keyword in the IPCONFIG statement in the TCP/IP profile tells the stack to do this. As interfaces are activated and deactivated, the stack keeps WLM informed of the status. Note that if the stack does not register, DNS uses the defined interface addresses but they are never checked for active status.

Note: For each TCP/IP stack within a sysplex, only the first 15 addresses listed in the HOME statement of your profile will be registered to WLM (and passed on to the name server when it requests the information). If an interface is not active, its address will not be forwarded to WLM. When the name server receives these active addresses from WLM, it will accept only the addresses that have a matching address listed in its data file. This requirement for statically defined addresses ensures full administrative control over the workload distribution.

Once the TCP/IP stacks have registered with WLM, the following occurs (see Figure 2-1 on page 22):

1. When each application becomes active in the sysplex, it registers with WLM using the appropriate group, server, and host (stack) name.
2. The name server will query WLM periodically for a list of available applications. WLM returns the names of the applications within each group, the active IP addresses associated with them (obtained from the associated stack name), and a set of workload-related weights.

3. Resource records representing the application's group name are dynamically (and not permanently) added to the name server's data files. These entries will now be treated the same as any hard-coded entries read from the server's data file.
4. When a request to resolve one of these group (server) names comes in, the server will choose an address to return based upon the weighting factor provided from WLM.
5. The next request for the same group name within the sysplex will be given the next address according to the weighting. Depending on the relative resource utilization of the hosts in the sysplex, this could be the same address as retrieved in the previous query, it could be a different address for the same host, or it could be an address for another host in the sysplex.
6. WLM is queried by the name server every 60 seconds (by default) for a new set of addresses and host weights. This can be controlled by the -t parameter at startup of the name server task (daemon).

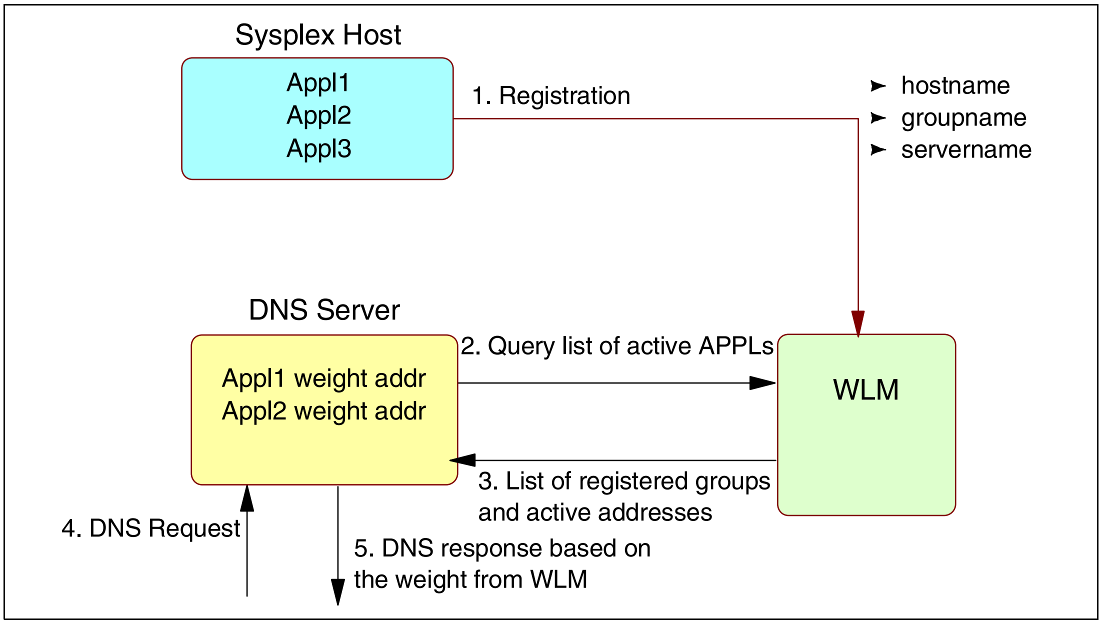


Figure 2-1 WLM and name server working together

If you are familiar with DNS, you might have already noticed that we appear to have crossed a boundary: an application registers with WLM, providing its service name or identification, and then the name server picks up this information and creates a host name entry. The name server dynamically maps an application (actually the group name representing the application) within the sysplex to a host address. While this might be confusing at first, it makes great sense.

If an application goes down, either WLM will be notified by a deregistration command, or else it will detect automatically that the address space has terminated. Then WLM will remove the entry from its tables. When the name server next queries WLM, it will no longer be given that application, group, and host name. Since we can have multiple applications within a single group, another request for the same group will still succeed if another application registered with the same group is active within the sysplex.

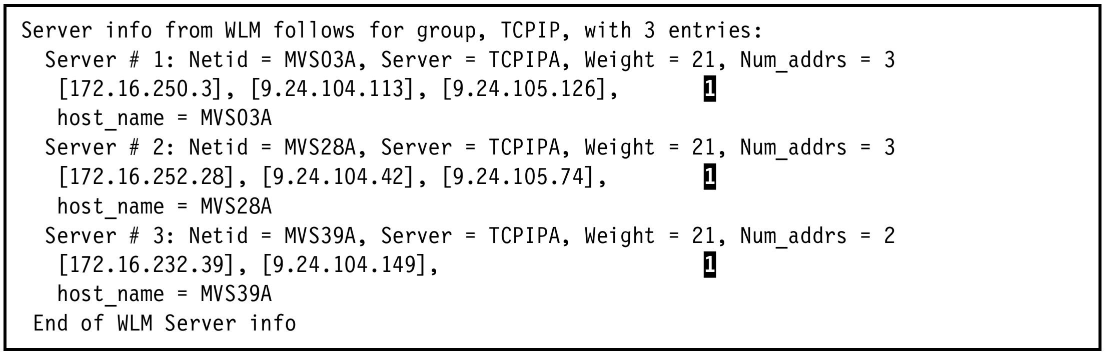
2.2.1 Data returned by the name server

When the sysplex DNS returns information to a DNS requesting data about WLM resources, the sysplex DNS returns a time-to-live (TTL) of 0 so that the local DNS does not cache the results. However, some resolver and name server implementations do not honor a zero TTL, thus reducing the effect of connection optimization during the time they preserve knowledge of cached resources. If you find that there are too many queries on the network for sysplex resources and you wish to choose reduced network traffic over completely optimized connections, you may start the DNS/WLM with a longer TTL value by overriding the default of 0 with the `-l` option.

WLM weights

The DNS/WLM queries the WLM every 60 seconds by default for information regarding resource usage (weights) and available resource addresses. Weights are reflected in the server entries and represent available capacity. The highest weight possible for a server is 64, which indicates the highest capacity available.

The weight 1 of a resource is only visible in a debug trace of the DNS as you can see in Figure 2-2.



```
Server info from WLM follows for group, TCPIP, with 3 entries:
  Server # 1: Netid = MVS03A, Server = TCPIPA, Weight = 21, Num_addrs = 3
   [172.16.250.3], [9.24.104.113], [9.24.105.126],      1
   host_name = MVS03A
  Server # 2: Netid = MVS28A, Server = TCPIPA, Weight = 21, Num_addrs = 3
   [172.16.252.28], [9.24.104.42], [9.24.105.74],       1
   host_name = MVS28A
  Server # 3: Netid = MVS39A, Server = TCPIPA, Weight = 21, Num_addrs = 2
   [172.16.232.39], [9.24.104.149],                     1
   host_name = MVS39A
 End of WLM Server info
```

Figure 2-2 Partial output of debug trace with WLM weight for MVS images

Static addresses versus registered addresses

The static addresses are those that are defined to the DNS in the sysplex (cluster) domain file. The registered addresses are those that have been defined and are active within the TCP/IP protocol stack. In response to queries, DNS/WLM sends a list of available addresses comprised of the intersection of active addresses registered to the WLM and active addresses that have been defined to the DNS zone files. The idea is to present a list of addresses that are reachable by any host that needs to know. If you have VIPAs configured in your TCP/IP stacks, only VIPA addresses should be statically defined in DNS data files. For a TCP/IP stack within the sysplex, only the first 15 addresses listed in the HOME statement of TCPIP.PROFILE will be registered to WLM (and passed on to the name server when it requests the information). If an interface is not active, its address will not be forwarded to WLM. When the name server receives these active addresses from WLM, it will accept only the addresses that have matching address listed in its data file. This requirement for statically defined addresses ensures full administrative control over the workload distribution.

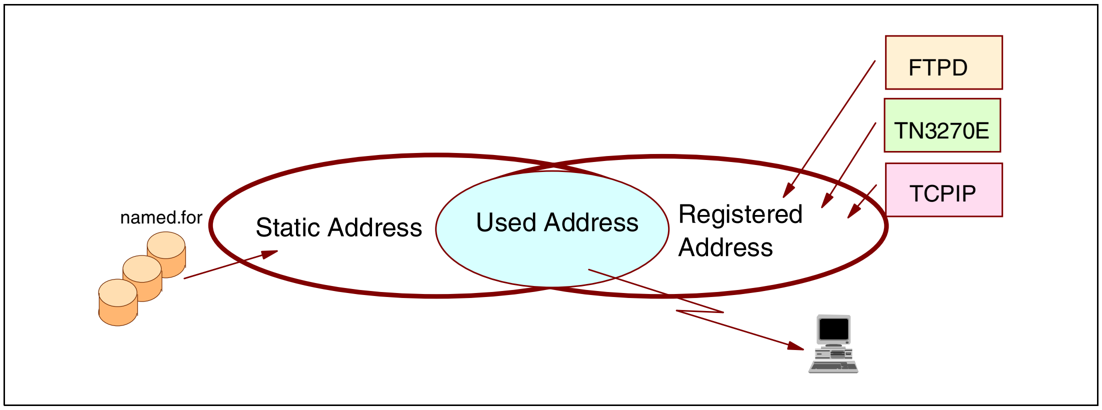


Figure 2-3 DNS addresses used in WLM distribution

The emphasis should be on the words *reachable addresses*. Whether you use static or dynamic routing protocols, you must ensure that any address returned in response to a DNS query can be reached.

If you are running static routing in your network without a comprehensive set of static routing definitions on the appropriate platforms, many such addresses could be unreachable. A DNS extraction might present a list of addresses, some of which would not be reachable by every host in the network.

Recommendation for DNS/WLM address definition

Based on the discussion so far, we have come to the following conclusions:

1. If you have implemented dynamic routing protocols in your network, limit your statically defined addresses in the sysplex subdomain to the VIPA address and use SOURCEVIPA.
2. If you use dynamic routing protocols throughout your network, but you do not use VIPA at the z/OS IP host, you may still successfully use multiple addresses in your name server forward zone files.
3. If you use static routes in your network, limit the statically defined name server addresses to those that are reachable throughout the network.

Round-robin technique and addresses returned

If the intersection of active addresses and DNS-defined addresses yields multiple potential addresses for a query and if all systems have the same weights, you would expect to see a rotation of the addresses being offered a client. Yet you may test with DNS/WLM and never perceive the phenomenon. This is due to the default for the -t option on the DNS startup. -t represents the amount of time between queries to the WLM about sysplex names, addresses, and weights. When the time specified in -t (default of 60 seconds) expires, DNS resets its list of potential addresses to the order specified in the DNS definition sequence. The default of 60 seconds has been deemed optimal for a production system, because weights can change rapidly; DNS should refresh its knowledge of weights frequently. A great number of connections occur in a production environment in those 60 seconds, and these will receive the benefits of the round-robin address offers. If you set -t to a value greater than 60 seconds, you defeat the purpose of connection balancing by overriding refreshed knowledge about sysplex weights.

However, if you want to see the round-robin effect in action *during testing,* you can temporarily set the value higher than 60 seconds. Multiple (o)nslookups in rapid succession should deliver a list of addresses that rotate with each command.

2.2.2 Using VIPA and a dynamic routing protocol with DNS/WLM

From a purely DNS and WLM perspective, the use of VIPA does not alter the host selection criteria for a sysplex application or WLM group. From a routing perspective, there are benefits in using the VIPA address for a TCP/IP stack rather than the physical address.

In general, access to the mainframe will traverse a router somewhere in the network for most users. That router can make routing decisions based upon the topology of the network. When a z/OS TCP/IP stack has multiple physical connections, a router can choose the most direct link and should that link fail, an alternate may be transparently substituted.

There is enough evidence to suggest the most viable configuration is to use the DNS/WLM sysplex functions with a VIPA address. This provides a very high availability solution with a minimal configuration requirement at the workstation. If a route fails, including a channel connection, the router can simply redirect the traffic to an available route. If a stack fails, the user can reconnect immediately to an alternate stack without changing the application destination. This means there is no longer a requirement to define "backup icons" for sysplex-supported services or for the user to have to choose between normal Telnet or backup Telnet. DNS will automatically select what is available and will balance the load when the failing system returns.

2.3 The pros and cons of DNS/WLM

The DNS/WLM solution provides for a nice workload distribution, particularly when using an application with long-lived connections. Because of its dependence on the DNS host name to IP address mapping, this solution also has some potential drawbacks. This section lists these advantages and disadvantages.

2.3.1 Benefits of DNS/WLM workload distribution

Since the target TCP/IP stack is chosen by the DNS server using the workload information provided by WLM, the workload is balanced in a sysplex based on the current load and system capacity.

Clients use the sysplex name as a server’s host name. In case of TCP/IP stack failure, the connection can be re-established to an appropriate surviving TCP/IP stack within the sysplex.

Summary of benefits:

- Distributes connections in a sysplex based on current load and capacity.
- Distributes load across adapters on a single host.
- Dynamically avoids crashed hosts and servers.
- Dynamically avoids crashed TCP/IP stacks when using sysplex name.
- Highly scalable - new servers may be added without DNS administration.
- Inexpensive to deploy - uses existing technology. No special software/hardware is required.
- Provides for high performance since the distribution is done during host name resolution.

2.3.2 DNS/WLM limitations

Most of the DNS/WLM solution's limitations arise from its inherent dependence on the Domain Name System. The following lists some of these drawbacks:

- To take advantage of DNS/WLM connection optimization, the clients must be using DNS to resolve addresses.
- Additionally, the DNS server must be implemented within the sysplex. Further, the dynamic naming structure may require client re-configuration.
- The DNS/WLM solution is not applicable to all applications, since application software support is required.
- The DNS/WLM implementation does not distinguish among multiple servers on the same host but using a different port.
- If caching is enabled at other name servers or at hosts and these name servers or hosts ignore the TTL value, full connection optimization is defeated.
- The DNS/WLM can optimize connections only within a single sysplex.
- DNS/WLM is intended primarily for long-lived connections. Although short-lived connections do exploit the potential of DNS/WLM, the added network traffic they generate may outweigh the benefits.
- DNS/WLM is currently only supported by BIND 4.9.3. DNS BIND 9 does not support this function.

2.4 Application and stack registration to WLM

In order for Workload Manager to become aware of an application, the application must register with WLM. There is an assembler macro and a C function available for doing this (IWMSRSRG and IWMDNREG, respectively). Although they have different names, the C function is just a wrapper to call IWMSRSRG. When an application registers, the following information must be passed to WLM:

- Group (cluster) name

  A generic name used to represent a group of applications running on the sysplex. This can be considered the name of the *service* provided.

- Server name

  The name of the application running on that particular host in the sysplex. Each application in a group must register with a different server name. Essentially, this is the name of the application instance providing the service.

- Host name

  The TCP/IP host name of the stack associated with the application (server). This can be obtained by issuing the gethostname() call.

When the name server requests a list of registered applications from WLM, the above information is returned for each one, along with a list of active addresses associated with the host name; that is, all the interfaces that have been successfully activated and have an address assigned via the HOME statement in the TCP/IP profile.

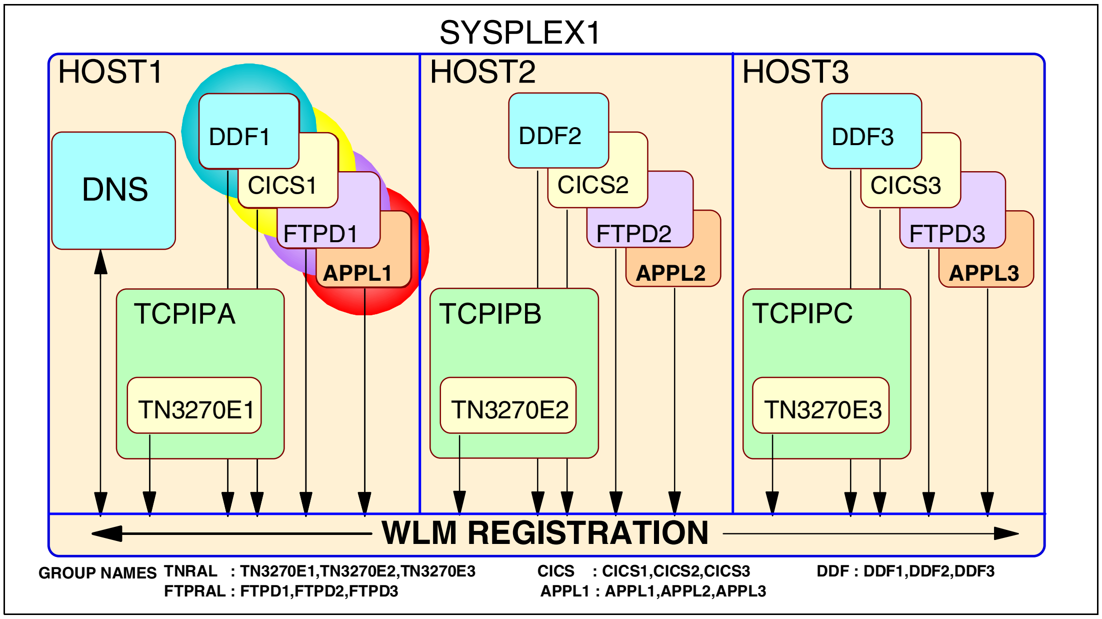


Figure 2-4 Application and stack registration to WLM

2.4.1 Stack registration with DNS/WLM

If you plan to use the connection optimization features of the BIND DNS server that is exploiting WLM, then you need to be aware of several additions to your TCP/IP Profile data set:

```
IPCONFIG
  SYSPLEXRouting  1
```

SYSPLEXRouting 1 indicates that this CS for z/OS IP stack participates in a sysplex and should notify the Workload Manager (WLM) of any changes in interface definitions or statuses. This statement allows the stack to register itself and its interfaces with the WLM for connection optimization purposes. SYSPLEXRouting is part of the IPCONFig statements of the PROFILE.TCPIP.

2.4.2 Communications Server for z/OS V1R2 IP application support

Several applications shipped with CS for z/OS IP are able to register with WLM. For example, the TN3270 server and the FTP server can be configured to register with WLM. Our scenarios used these two applications. Additionally, other IBM applications for the z/OS platform have the ability to register with WLM.

For CICS, you specify the group names in the listener definition. You may specify up to three group names for a CICS listener. If you want to change the registration, you have to stop and restart the listener.

TN3270 configuration

```
TELNETPARMS
   WLMCLUSTERNAME     2
       TNO3           3
       TNRAL          3
       TNTSO          3
   ENDWLMCLUSTERNAME  2
ENDTELNETPARMS
```

If you want Telnet to register with WLM, you need to add WLMCLUSTERNAME and ENDWLMCLUSTERNAME 2 to the TELNETPARMS section of your profile coding. Imbedded between WLMC and ENDWLMC you may specify the names that you would like the TN3270 Server to register as with WLM. TNRAL 3 and the other names in the WLMC list are known as *group names*. They represent a cluster of equivalent server names in a sysplex environment that provide some common service.

The TCP/IP stack and the Telnet server are registered at stack startup if the appropriate definitions have been placed in the PROFILE.TCPIP. Re-registration occurs after every OBEYFILE command. You may deregister by stopping the stack or by issuing an OBEYFILE command against a PROFILE that has coded NOSYSPLEXrouting or by changing the specifications in the PROFILE between WLMC and ENDWLMC.

The Telnet server can also be deregistered with:

```
V TCPIP,procname,TELNET,QUIESCE
```

and

```
V TCPIP,procname,TELNET,STOP
```

A re-registration is accomplished with:

```
V TCPIP,procname,TELNET,RESUME
```

You may specify up to 16 group names for a TN3270 server.

FTP configuration

If you want FTP to register with WLM, you need to add WLMCLUSTERNAME to the FTP.DATA file.

```
WLMCLUSTERNAME FTPRAL
```

You may also specify up to 16 group names for the FTP server. If you want to change the registration, you have to stop and restart the FTP server.

Registering your own applications

To register your own server application, use a C interface or an assembler interface. For C language, the IWMDNREG and IWMDNDRG API is provided. For assembler applications, use IWMSRSRG and IWMSRDRS macros. Appendix C, “Sample applications and programs” on page 273 contains the description of a sample application that is capable of registering with WLM. The source code for this application is included in Appendix D, “Sample C program source code” on page 287.

For information on how to code the assembler macro, see *z/OS V1R2.0 MVS Workload Management Services*, SA22-7619.

2.4.3 DNS/WLM registration results

Unfortunately there is no generic query available to determine from z/OS WLM the names of resources registered with it. Some applications issue messages about a successful registration like the FTP server, as shown in Figure 2-5.

```
   EZYFT57I FTP registering with WLM as group = FTPRAL host = MVS03A
EZY2702I Server-FTP: Initialization completed at 12:39:25 on 09/21/00.
```

Figure 2-5 FTP server registration with WLM

Other applications, such as Telnet, have commands available to display the group names registered with WLM (see Figure 2-6).

```
D TCPIP,TCPIPA,T,WLM
EZZ6067I TELNET WLM DISPLAY 240
WLM CLUSTER NAME          STATUS
------------------  ----------------------
----- PORT:    23  ACTIVE   BASIC
TNTSO               Registered
TNRAL               Registered
TN03                Registered
4 OF 4 RECORDS DISPLAYED
```

Figure 2-6 Display Telnet group names registration with WLM

If there is no command or message available to check the registration with WLM, you can use the nslookup or any other function that resolves host names to verify the results. Another possibility to be certain if an application has registered is to dump the DNS database. Please refer to 2.5.6, “Dumping the DNS server cache” on page 35 for a detailed explanation.

2.5 Working with DNS/WLM

The hosts, servers, and stacks register with Workload Manager (WLM), and DNS/WLM retrieves its knowledge of system loads and available resources from the WLM. DNS/WLM distributes connections in a sysplex based on currently available system capacities. It also distributes the system load across active adapters on a single host and through dynamically updated awareness of crashed stacks, servers, and adapters. It can avoid them in distributing traffic across a sysplex. Since server registration is dynamic, no DNS administration is required to recognize new resources in the network.

2.5.1 WLM configuration

Load balancing in the sysplex requires that all hosts participating in connection optimization be operating in *WLM goal mode*. Without goal mode, all hosts are treated equally and there is no WLM consideration of the different workloads they may have. To set goal mode, you have two choices:

1. Omit the keyword `IPS=` from IEASYSxx in SYS1.PARMLIB
2. Dynamically configure goal mode by issuing the MVS console command:

   ```
   F WLM,MODE=GOAL
   ```

You can discover whether your system is in goal mode or not with the command D WLM,SYSTEM=sysname. You see the output of the command in Figure 2-7 on page 30.

```
   D WLM,SYSTEMS
IWM025I  11.33.23  WLM DISPLAY 295
  ACTIVE WORKLOAD MANAGEMENT SERVICE POLICY NAME: EQUALESY
  ACTIVATED: 2000/06/28  AT: 08:32:08  BY: LUNA      FROM: RA03
  DESCRIPTION: Policy for equal imp case
  RELATED SERVICE DEFINITION NAME: CICSpol
  INSTALLED: 1998/01/30  AT: 15:44:05  BY: WOZA      FROM: SA28
  WLM VERSION LEVEL:       LEVEL011
  WLM FUNCTIONALITY LEVEL: LEVEL004
  WLM CDS FORMAT LEVEL:    FORMAT 3
  STRUCTURE SYSZWLM_WORKUNIT STATUS: DISCONNECTED
  STRUCTURE SYSZWLM_61989672 STATUS: DISCONNECTED
  *SYSNAME*  *MODE*  *POLICY*  *WORKLOAD MANAGEMENT STATUS*
  RA03       GOAL    EQUALESY  ACTIVE
  RA28       GOAL    EQUALESY  ACTIVE
  RA39       GOAL    EQUALESY  ACTIVE
```

Figure 2-7 Display of WLM status

2.5.2 DNS/WLM TCPDATA consideration

The TCPDATA file (or resolver file) is used by clients to determine, among other things, the stack name they should have affinity to and the domain name that will automatically be appended to their name-based queries. The fully qualified name for `mvs03` could end up being `mvs03.itso.ral.ibm.com.` or `mvs03.ralplex1.itso.ral.ibm.com.` even if all you entered at the client was `ping mvs03`. This depends on the resolver file your client is using.

If you are implementing the DNS server for a sysplex domain, you have a decision to make about how you designate the domain name for the TSO or shell client. If you leave the domain name as `itso.ral.ibm.com.`, then every time your client needs to have, for example, `ftpral` converted into a fully qualified name, it will be resolved into `ftpral.itso.ral.ibm.com`. You may not find the group name `ftpral` under these conditions; however, you would find the thousands of other resources in the domain `itso.ral.ibm.com` and served by another name server very easily by allowing the default domain to be appended.

Your network users who need to get to the few resources managed by the sysplex domain simply need to change those requests to something like `ftp ftpral.ralplex1`, ensuring that they do not append a period to the request. (The period or dot would indicate that the fully qualified name has been specified and the current domain name should not be appended.) Their client resolver process will expand the two-part name into the fully qualified `ftpral.ralplex1.itso.ral.ibm.com`. To avoid your network users having to specify the long sysplex names, you can simply code CNAME records for your sysplex resources as shown in Figure 2-28 on page 45.

On the other hand, if you make the domain name something like `ralplex1.itso.ral.ibm.com`, then every time your TSO client needs to find the group name `ftpral`, it will be correctly resolved into `ftpral.ralplex1.itso.ral.ibm.com`. However, to reach the thousands of resources by name that are actually in the `itso.ral.ibm.com` domain, the TSO or shell client would have to specify the fully qualified name to begin with or would have to rely on your CNAME coding in the name server. (The CNAME coding would spell out the fully qualified name of the resource.)

You have probably figured out that the issue of what domain name to put into the TCPDATA file is one of degree: how many host-based clients are there versus workstation clients? If there are few z/OS-based clients trying to reach resources that are based mostly in the sysplex, then you might decide to use the sysplex subdomain as your TCPDATA domain. If the same clients are trying to reach resources in a completely different domain, then you might decide to use the domain name that represents the greater number of resources.

In our network, we left the TCPDATA file at the host with a domain of `itso.ral.ibm.com.` for the clients to use and CNAME records to point to the sysplex resources.

2.5.3 Client/server affinity

Some client/server applications require that the client connects to the same server instance after an interruption. This is achieved in the following way:

1. The server uses the new ioctl() function SIOCGSPLXFQDN to get its fully qualified name from the TCP/IP stack.
2. After the connection has been established using only the group name or the sysplex name the server sends its fully qualified name (*server_instance.groupname.sysplex_subdomain.domain*) to the client.
3. After an interruption, the client uses this fully qualified name to make sure it connects to the same server it was connected to before the interruption.

To enable this function, the following definition has to be added to the sysplex name servers loopback file:

```
127.0.0.128 IN PTR ralplex1.itso.ral.ibm.com; Sysplex Loopback Address (SLA)
```

The loopback address range 127.0.0.128-127.0.0.255 has been reserved by IBM for this purpose.

2.5.4 Starting the DNS server

The BIND DNS name server must be associated with a TCP/IP stack. This process occurs by default if there is only one copy of CS for z/OS IP in the z/OS image. A single copy will probably be the more common implementation, since the reasons for running multiple stacks are rapidly disappearing. The establishing of affinity to a particular stack then becomes an issue only if you are running in a CINET configuration with multiple stacks. This issue is easy to solve with a pointer to the correct TCPDATA file. The TCPDATA file (also called the resolver file) is found according to the following search sequence:

1. RESOLVER_CONFIG environment variable
2. /etc/resolv.conf
3. //SYSTCPD DD
4. *jobname* or *userid*.TCPIP.DATA
5. SYS1.TCPPARMS(TCPDATA)
6. TCPIP.TCPIP.DATA

There can be problems *with some applications* using the TCPDATA reference by way of the //SYSTCPD DD card. (Forked tasks do not resolve correctly if the DD card is used; for such forked tasks you must duplicate the contents of the //SYSTCPD DD card in an HFS data set called /etc/resolv.conf.) Although the BIND name server forks a new task, this task already has the information it needs from the parent task; therefore, at the ITSO, we had no problems running with the //SYSTCPD DD card.

If you have an /etc/resolv.conf in place, you can, of course, omit the //SYSTCPD DD statement from your name server JCL. If you need to override the default /etc/resolv.conf for a particular name server procedure, you may use //SYSTCPD DD or you may reference an HFS resolver configuration file with the environment variable `RESOLVER_CONFIG`. We used the //SYSTCPD DD 1 as you can see in Figure 2-8.

```
//NAMED PROC B='/etc/named.boot',P='53'        2
//**************************************************************
//NAMED    EXEC PGM=EZANSNMD,REGION=OK,TIME=NOLIMIT,
//      PARM='POSIX(ON) ALL31(ON)/ -b &B -p &P -d 11'
//*STEPLIB DD DISP=SHR,DSN=TCPIP.SEZALINK
//SYSPRINT DD SYSOUT=*,DCB=(RECFM=F,LRECL=132,BLKSIZE=132)
//SYSIN    DD DUMMY
//SYSERR   DD SYSOUT=*
//SYSOUT   DD SYSOUT=*,DCB=(RECFM=F,LRECL=132,BLKSIZE=132)
//CEEDUMP  DD SYSOUT=*
//SYSTCPD  DD DISP=SHR,DSN=TCPIP.TCPPARMS.R2611(TCPD&SYSCLONE.A) 1
```

Figure 2-8 DNS server startup procedure

2 The PORT value and the boot file parameter are not required in the example above since it is the default, but are included for illustration purposes.

For the DNS server you reserve the PORT in the PROFILE data set with the name of the parent process. This varies from procedure to procedure, with some procedures requiring that the child process be named. If you autolog a DNS server procedure, then both the AUTOLOG and the PORT statement must reference the parent process. This is in marked contrast to what occurs with FTP, where the child process must be named on the PORT and AUTOLOG statements:

```
AUTOLOG
  NAMED
;
PORT
  53  TCP   NAMED
  53  UDP   NAMED
```

The DNS server can also be started from the UNIX system services shell environment, but starting it requires superuser authority or an authorized TSO user ID. You cannot start the name server with the inetd daemon. The startup must know either via default or via parameters what the boot file name is, so that the correct data can be loaded. Figure 2-9 shows how you can start the DNS server with a UNIX System Services shell environment.

Note: If we had coded a default /etc/resolv.conf, we would not have needed to specify the RESOLVER_CONFIG parameter.

```
export RESOLVER_CONFIG="//’TCPIP.TCPPARMS(TCPD03A)’"
_BPX_JOBNAME=’NAMED’  /usr/sbin/named  -b  /etc/named.boot
```

Figure 2-9 Start DNS server in UNIX System Services shell environment

Table 2-1 lists additional startup parameters with a brief description:

Table 2-1 Additional start options for the DNS server

Option Name	**Description**
-d *n*	Specifies a debugging option and causes the named daemon to write debugging information to the file /tmp/named.run. Valid debug levels are one to 11, where 11 supplies the most information.
-p	Reassigns the port that is used in queries to other name servers. (The default is 53.) The local / remote port number can be specified.
-b *filename*	Specifies an alternate boot file to /etc/named.boot.
-q	Enables the logging of queries received by the name server.
-r	Disables recursive query processing.
-t *nn*	Specifies the time (nn, in seconds) between refreshes of sysplex names and addresses and of the weights associated with those names and addresses. The default is sixty seconds.
-l *nn*	Specifies the time-to-live (nn, in seconds) for sysplex names and addresses after they are sent into the network. The default is zero seconds.

We have started the name server with a procedure from the MVS console. Figure 2-10 shows the console log of the startup of the DNS server process on our MVS03 system.

Note that procedure NAMED 1 has ended, but you will later see that it has created a child process called NAMED1. The message EZZ6475I 2 actually tells you that the domain name server finished loading its resources and is now ready to respond to query requests.

```
S NAMED
$HASP100 NAMED    ON STCINRDR
IEF695I START NAMED    WITH JOBNAME NAMED    IS ASSIGNED TO USER
TCPIP3  , GROUP OMVSGRP
$HASP373 NAMED    STARTED
IEF403I NAMED - STARTED - TIME=16.01.35
EZZ6452I NAMED STARTING. @(#) DDNS/NS/NS_MAIN.C, DNS_NS, DNS_R1.1 1.
239
62  9/23/97 10:57:21
-                                                --TIMINGS (MINS.)--
          ----PAGING COUNTS---
-JOBNAME  STEPNAME PROCSTEP    RC   EXCP   CONN    TCB    SRB  CLOCK
 SERV  PG  PAGE  SWAP   VIO SWAPS
-NAMED             NAMED       00    731     31    .00    .00     .0
11841   0     0     0     0     0
IEF404I NAMED - ENDED - TIME=16.01.36
-NAMED    ENDED.  NAME-                   TOTAL TCB CPU TIME=   .00
 TOTAL ELAPSED TIME=    .0
$HASP395 NAMED    ENDED                   1
+EZZ6475I NAMED:  READY TO ANSWER QUERIES. 2
```

Figure 2-10 Console log of DNS start

Depending on whether you have started the syslog daemon, you will see additional messages about the files that DNS is loading either on the MVS console or in the syslogd.log in HFS. Debugging is usually easier if you allow the messages to be sent to syslogd. The messages you see at MVS03 from the initialization with our boot file are shown in Figure 2-11 on page 34.

```
EZZ6452I named starting. @(#) ddns/ns/ns_main.c, dns_ns, dns_r1.1 1.62  9/23/97
EZZ6701I named established affinity with 'TCPIPA'  1
EZZ6540I Static primary zone '104.24.9.in-addr.arpa' loaded (serial 1999040101) 2
EZZ6540I Static primary zone '16.172.in-addr.arpa' loaded (serial 1999040101)
EZZ6540I Static primary zone '0.0.127.in-addr.arpa' loaded (serial 1999040101)
EZZ6540I Static cache zone '' loaded (serial 0)
EZZ6475I named:  ready to answer queries.
```

Figure 2-11   DNS messages in syslogd.log: file loading

You can see in Figure 2-11 (1) how our server has been associated via the resolver process with the TCPIPA stack. You can also see which level of your customization has been loaded (2) as shown by the displayed serial number. You may want to maintain the serial number designation in the files as you customize them in order to understand which version of a file has been loaded. The serial number is also used at secondary name servers to determine whether data needs to be reloaded after a zone transfer. If you discover a mismatch between the data at a secondary name server and that at a primary, the discrepancy could be due to one of the following:

- A view of the secondary name server prior to its having pulled new data from the primary
- The failure of the secondary to pull new data from the primary because a matching serial number at the primary signalled the secondary not to update its data

We get back to this in more detail when we review the configuration files of the primary and secondary name server.

2.5.5 Displaying the DNS active sockets

Once the DNS server process has started, you can display the active sockets with an **onetstat -a** display. Also, you can use the -c and the -s options of **onetstat** to display the active sockets. Figure 2-12 shows the result of an onetstat display and you can observe 1 how the port we reserved in the PROFILE.TCPIP and specified (or defaulted) in the DNS startup is port 53.

```
MVS TCP/IP onetstat CS V2R10      TCPIP Name: TCPIPA          13:49:51
User Id  Conn     Local Socket          Foreign Socket        State
-------  ----     ------------          --------------        -----
BPXOINIT 0000000A 0.0.0.0..10007        0.0.0.0..0            Listen
FTPDA1   000000ED 0.0.0.0..21           0.0.0.0..0            Listen
NAMED    00000107 0.0.0.0..53  1        0.0.0.0..0            Listen
OMPROUTA 0000001B 127.0.0.1..1027       127.0.0.1..1028       Establsh
TCPIPA   0000000C 0.0.0.0..1025         0.0.0.0..0            Listen
TCPIPA   00000016 0.0.0.0..23           0.0.0.0..0            Listen
TCPIPA   000000F7 9.24.104.113..23      9.24.106.31..4906     Establsh
TCPIPA   00000011 127.0.0.1..1025       127.0.0.1..1026       Establsh
TCPIPA   00000010 127.0.0.1..1026       127.0.0.1..1025       Establsh
TCPIPA   000000F2 172.16.250.3..23      9.24.106.102..2889    Establsh
TCPIPA   00000014 127.0.0.1..1028       127.0.0.1..1027       Establsh
NAMED    00000108 0.0.0.0..53  1        *..*                  UDP
```

Figure 2-12   Display of active sockets: onestat -a

2.5.6 Dumping the DNS server cache

When you start the DNS server process, it will read all the zone files and place the information in memory. This memory database will get updated with entries that it learns from other DNS servers because of the recursive searching that may go on between DNS servers.

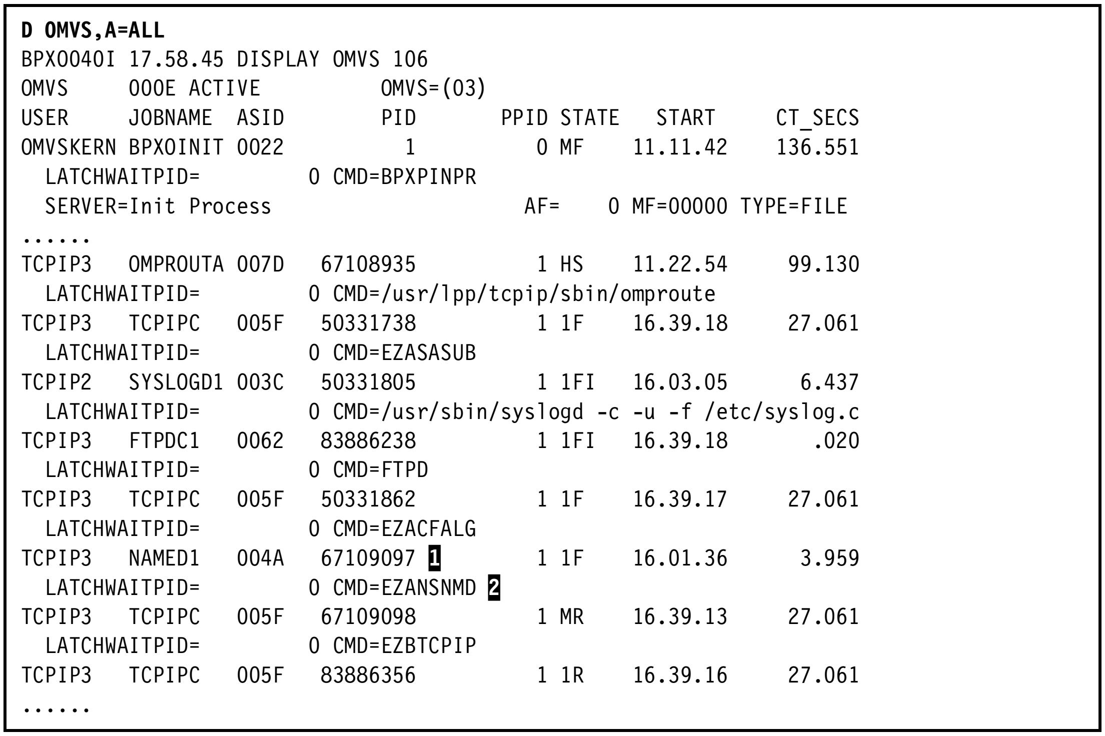
You have the ability to dump this memory table by sending a signal to the DNS server. The `SIGINT` signal dumps the name server memory database in the HFS file /tmp/named_dump.db. You can issue the signal through the ISPF command ISHELL, or you can issue the command in the UNIX System Services shell by entering:

```
kill -INT $(cat /etc/named.pid)
```

The process ID of the named daemon is stored in the /etc/named.pid file at the named startup. Alternatively you might enter the PID directly:

```
kill -INT 402653187
```

You can obtain the PID 1 with a UNIX System Services shell command **`ps -e`** or with the D OMVS,A=ALL console command, as you see in Figure 2-13. Note the name of the executed program: `EZANSMD` 2.



```
D OMVS,A=ALL
BPXO040I 17.58.45 DISPLAY OMVS 106
OMVS     000E ACTIVE         OMVS=(03)
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
OMVSKERN BPXOINIT 0022          1          0 MF    11.11.42    136.551
  LATCHWAITPID=        0 CMD=BPXPINPR
  SERVER=Init Process                    AF=    0 MF=00000 TYPE=FILE
......
TCPIP3   OMPROUTA 007D   67108935          1 HS    11.22.54     99.130
  LATCHWAITPID=        0 CMD=/usr/lpp/tcpip/sbin/omproute
TCPIP3   TCPIPC   005F   50331738          1 1F    16.39.18     27.061
  LATCHWAITPID=        0 CMD=EZASASUB
TCPIP2   SYSLOGD1 003C   50331805          1 1FI   16.03.05      6.437
  LATCHWAITPID=        0 CMD=/usr/sbin/syslogd -c -u -f /etc/syslog.c
TCPIP3   FTPDC1   0062   83886238          1 1FI   16.39.18       .020
  LATCHWAITPID=        0 CMD=FTPD
TCPIP3   TCPIPC   005F   50331862          1 1F    16.39.17     27.061
  LATCHWAITPID=        0 CMD=EZACFALG
TCPIP3   NAMED1   004A   67109097 1        1 1F    16.01.36      3.959
  LATCHWAITPID=        0 CMD=EZANSNMD 2
TCPIP3   TCPIPC   005F   67109098          1 MR    16.39.13     27.061
  LATCHWAITPID=        0 CMD=EZBTCPIP
TCPIP3   TCPIPC   005F   83886356          1 1R    16.39.16     27.061
......
```

Figure 2-13 Displaying process ID and DNS program

See Figure 2-14 on page 36 for a partial copy of the /tmp/named_dump.db file that was created when you issued signal #2 (-INT) against the DNS program, EZANSNMD. The output shows you only the zone table and a few lines in the beginning of the dump, but it should give an idea how it looks. You will see a more detailed dump later in 2.6.6, “DNS DUMP of primary DNS server in the sysplex” on page 49.

```
; Dumped at Tue Sep 19 14:36:12 2000
;; ++zone table++
; ralplex1.itso.ral.ibm.com (type 1, class 1, source ralplex1.for)
;       time=969374208, lastupdate=937949821, serial=1999040102,
;       refresh=7200, retry=3600, expire=604800, minimum=3600
;       ftime=937949821, xaddr=[0.0.0.0], state=20041, pid=0
; 104.24.9.in-addr.arpa (type 1, class 1, source ralplex1.rev9)
;       time=969378400, lastupdate=937951070, serial=1999040101,
;       refresh=7200, retry=3600, expire=604800, minimum=3600
;       ftime=937951070, xaddr=[0.0.0.0], state=0041, pid=0
; 16.172.in-addr.arpa (type 1, class 1, source ralplex1.rev)
;       time=969379773, lastupdate=937949840, serial=1999040101,
;       refresh=7200, retry=3600, expire=604800, minimum=3600
;       ftime=937949840, xaddr=[0.0.0.0], state=0041, pid=0
; 0.0.127.in-addr.arpa (type 1, class 1, source mvs03a.lbk)
;       time=969377837, lastupdate=937950997, serial=1999040101,
;       refresh=7200, retry=3600, expire=604800, minimum=3600
;       ftime=937950997, xaddr=[0.0.0.0], state=0041, pid=0
;; --zone table--
; Note: Cr=(auth,answer,addtnl,cache) tag only shown for non-auth RR's
; Note: NT=milliseconds for any A RR which we've used as a nameserver
; --- Cache & Data ---
 $ORIGIN itso.ral.ibm.com.
 ralplex1        IN      SOA     mvs03a.ralplex1.itso.ral.ibm.com.
                 1999040102 7200 3600 604800 3600 )      ;Cl=5
                 IN      NS      mvs03a.ralplex1.itso.ral.ibm.com.
                 IN      A       172.16.250.3    ;Cl=5
                 IN      A       172.16.252.28   ;Cl=5
                 IN      A       172.16.232.39   ;Cl=5
 $ORIGIN ralplex1.itso.ral.ibm.com.
```

Figure 2-14 Partial copy of /tmp/named_dump.db (from a SIGINT to DNS process)

2.5.7 DNS statistics

You can obtain DNS statistics by using the signal #3 (ABRT), available either from the ISHELL selection menus or by issuing **`kill -3 $(cat /etc/named.pid)`** from the shell. The data is stored in /tmp/named.stats. See Figure 2-15 on page 37 for a sample output.

```
+++ Statistics Dump +++ (969401611) Tue Sep 19 22:13:31 2000
7915.time since boot (secs)
7915.time since reset (secs)
0.Unknown query types
2.PTR queries
++ Name Server Statistics ++
(Legend)
.RQ.RR.RIQ.RNXD.RFwdQ
.RFwdR.RDupQ.RDupR.RFail.RFErr
.RErr.RTCP.RAXFR.RLame.ROpts
.SSysQ.SAns.SFwdQ.SFwdR.SDupQ
.SFail.SFErr.SErr.RNotNsQ.SNaAns
.SNXD
(Global)
.2 3 0 0 2  0 0 0 3 0  0 0 0 0 0  1 0 2 0 0  2 0 0 2 0  0
›9.24.104.125®
.0 3 0 0 0  0 0 0 3 0  0 0 0 0 0  1 0 2 0 0  0 0 0 0 0  0
›172.16.250.3®
.2 0 0 0 2  0 0 0 0 0  0 0 0 0 0  0 0 0 0 0  2 0 0 2 0  0
-- Name Server Statistics --
--- Statistics Dump --- (969401611) Tue Sep 19 22:13:31 2000
```

Figure 2-15 Name server statistics

2.5.8 Discovering signals available for process

Most of the signal numbers are documented in *z/OS V1R2.0 UNIX System Services Command Reference*, SA22-7802. If you want a list of the available signals in an environment, you may issue a LIST 1 command, as in Figure 2-16.

```
IBM
Licensed Material - Property of IBM
5647-A01 (C) Copyright IBM Corp. 1993, 2000
(C) Copyright Mortice Kern Systems, Inc., 1985, 1996.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

 - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - Improve performance by preventing the propagation -
 - of TSO/E or ISPF STEPLIBs                         -
 - - - - - - - - - - - - - - - - - - - - - - - - - - -
@ RA03:/u/pabst>kill -l  1
 NULL HUP INT ABRT ILL POLL URG STOP FPE KILL BUS SEGV SYS PIPE ALRM TERM USR1 U
SR2 ABND CONT CHLD TTIN TTOU IO QUIT TSTP TRAP IOERR WINCH XCPU XFSZ VTALRM PROF
 DCE DUMP
```

Figure 2-16 kill -l: requesting a list of available process signals

Table 2-2 lists all the valid signals for the domain name server.

Table 2-2 Valid signals for the domain name server

Signal Name	Description
SIGHUP	Reloads the boot file from the disk
SIGINT	Dumps the name server's database and hints file into the /tmp/named_dump.db file
SIGABRT	Dumps the current statistics of the name server in the /tmp/named.stats file
SIGUSR1	Starts debug tracing for the name server, or increment the debug level by one if the tracing has been activated already
SIGUSR2	Stops debug tracing
SIGWINCH	Toggles query logging on and off

A sample started procedure in TCPIP.SEZAINST named NSSIG allows you to issue signals to the name server from the MVS console:

```
S NSSIG,SIG=SIGINT
```

By sending the signal `SIGINT`, you can dump the memory table into the HFS file /tmp/named_dump.db. This is just another way to produce a dump, as was mentioned in 2.5.6, “Dumping the DNS server cache” on page 35.

The documentation for issuing signals with the name server is in *z/OS V1R2.0 CS: IP Configuration Guide*, SC31-8775. In Figure 2-17, you see the a copy of the NSSIG JCL we used in ITSO Raleigh.

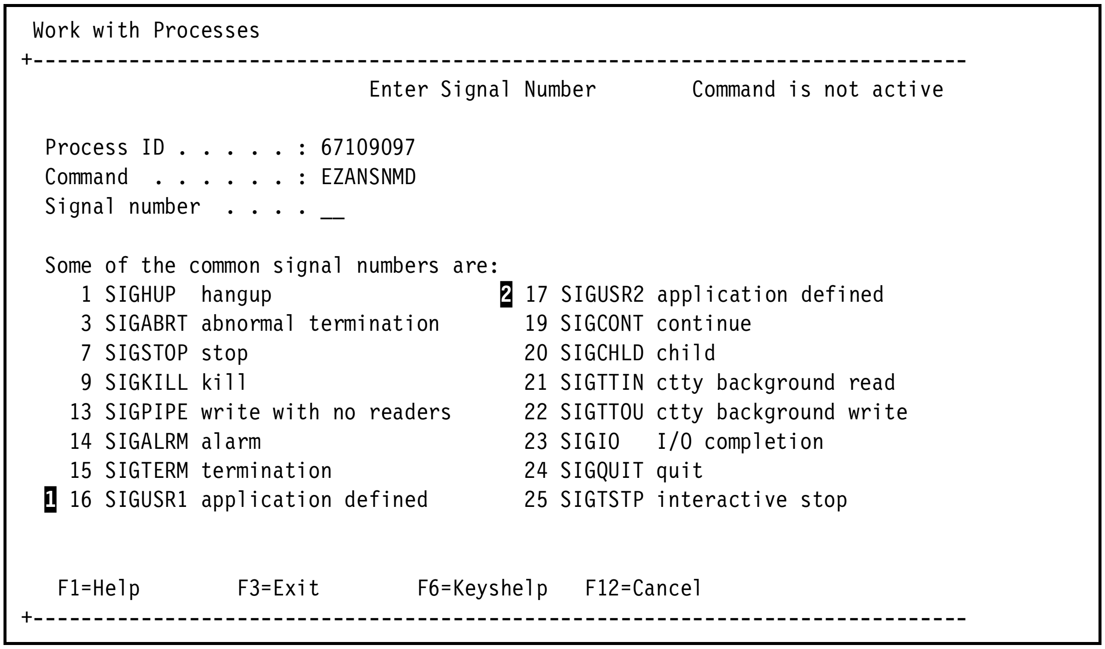
```
//NSSIG PROC SIG=''
//NSSIG EXEC PGM=BPXBATCH,REGION=30M,TIME=NOLIMIT,
//             PARM='SH kill -s &SIG $(cat /etc/named.pid)'
//* STDIN and STDOUT are both defaulted to /dev/null
//STDERR      DD PATH='/etc/log',PATHOPTS=(OWRONLY,OCREAT,OAPPEND),
//            PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
```

Figure 2-17 Running NSSIG procedure from ITSO Raleigh

2.5.9 Tracing the name server

As you can see in the output in Figure 2-18 on page 39, there are other ways to send the same signals. In this case we use the ISPF command ISHELL. We can select the **Work with Processes** option by doing the following:

1. Select **TOOLS**
2. Select **Work with Processes**
3. Find **process ID** or use command EZANSNMD
4. Select process with **S**=SIGNAL



```
 Work with Processes
+------------------------------------------------------------------------------
                             Enter Signal Number       Command is not active

  Process ID . . . . . : 67109097
  Command  . . . . . . : EZANSNMD
  Signal number  . . . . __

  Some of the common signal numbers are:
     1 SIGHUP  hangup                 2 17 SIGUSR2 application defined
     3 SIGABRT abnormal termination      19 SIGCONT continue
     7 SIGSTOP stop                      20 SIGCHLD child
     9 SIGKILL kill                      21 SIGTTIN ctty background read
    13 SIGPIPE write with no readers     22 SIGTTOU ctty background write
    14 SIGALRM alarm                     23 SIGIO   I/O completion
    15 SIGTERM termination               24 SIGQUIT quit
  1 16 SIGUSR1 application defined       25 SIGTSTP interactive stop


   F1=Help        F3=Exit        F6=Keyshelp   F12=Cancel
+------------------------------------------------------------------------------
```

Figure 2-18   Another way to issue signals to the DNS server

Signal #16 (USR 1) 1 begins a trace of the DNS server process. Signal #17 (USR2) 2 terminates the trace. The data is written to /tmp/named.run. See Figure 2-19 for a partial trace output. A more detailed trace is included in 2.6.7, “DNS trace of WLM data for the primary DNS in the sysplex” on page 50.

```
Debug turned ON, Level 1
Debug turned ON, Level 2
Debug turned ON, Level 3
Debug turned ON, Level 4
Debug turned ON, Level 5
Debug turned ON, Level 6
Debug turned ON, Level 7
Debug turned ON, Level 8
Debug turned ON, Level 9
datagram from [172.16.250.3].1027, fd 5, len 43; now Tue Sep 19 15:37:15 2000
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 5657
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; 102.106.24.9.in-addr.arpa, type = PTR, class = IN
;; ...truncated
ns_req(from=[172.16.250.3].1027)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 5657
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; 102.106.24.9.in-addr.arpa, type = PTR, class = IN
```

Figure 2-19   /tmp/named.run partial trace output

The debug level was raised to 9 in the trace because we issued the USR1 signal (#16) multiple times. We really wanted a debug level of 11, the recommended (and highest allowable) setting. Attempting to reach debug level 11 via the ISHELL TOOLS menu can be tedious, as you see. We recommend instead bypassing the ISHELL process and setting the debug trace level to 11 by one of two methods:

- Use `-d 11` on the named start command or in your JCL.
- Issue the **`kill -USR1 ($cat /etc/named.pid)`** multiple times from the shell with the help of the retrieve key.

Resolver tracing is available with the name server and is enabled by default in the TCPDATA file. The output goes to the MVS console or to the syslogd output if you have started the syslog daemon.

2.5.10 Reloading DNS data

To reload data that you may have changed during a particular lifetime of the name server, you can send a signal to the UNIX System Services shell asking the DNS to reread its configuration files. Issue **`kill -HUP ($cat /etc/named.pid)`** or use signal #1 from the ISHELL after you select **Tools** -> **Work with Processes**.

This command works only at a primary name server. A secondary server periodically returns to query the primary for new data, theoretically eliminating the need to reload a secondary with a signal. (The refresh interval is one of the settings in the SOA record. See 2.6.1, “Primary DNS configuration on MVS03” on page 42 for further details on SOA records and their configuration.)

The reload process is not designed for dynamic domains, since these are updated via the **`nsupdate`** command.

2.5.11 Stopping the DNS server

You can stop the DNS server with the MVS STOP command P T03DNS1. The advantage of the STOP command is the graceful termination of the name server and the issuing of messages in syslogd. Other alternatives are to use the MVS CANCEL command or the OMVS KILL command. The OMVS KILL command can be issued from OMVS or from the NSSIG procedure.

2.6 Implementation scenario

When implementing DNS/WLM, you can leave in place your primary name server, which may reside on a platform other than z/OS, and *still take advantage of the DNS/WLM*. The only changes to your standard DNS definitions are minimal additions to allow for the specification of the zSeries DNS as the authoritative name server for some subdomain. That is, the sysplex subdomain that is created within your network’s domain will be serviced by the zSeries, but that does not necessarily force you to use the zSeries DNS as the authoritative server for your entire network domain.

In our scenario, our existing domain, itso.ral.ibm.com, has been using a name server built on an AIX platform (RSSERVER, an RS/6000). We leave that name server in place, adding to it a subdomain definition for the new sysplex subdomain. Within this new sysplex subdomain, we will place our authoritative DNS server that will balance load among the different servers providing the same service.

Figure 2-20 depicts the servers and groups running in our environment. You may be wondering why we chose different group names for the TN3270 function. Functionally each group name is not equivalent to every other group name. TNRAL represents a group name that allows users to reach the same application on all three MVS images. TN03 is a group name that allows you only to access the MVS image MVS03. TN28 is a group name that allows you only to access the MVS image MVS28, and so on for TN39.

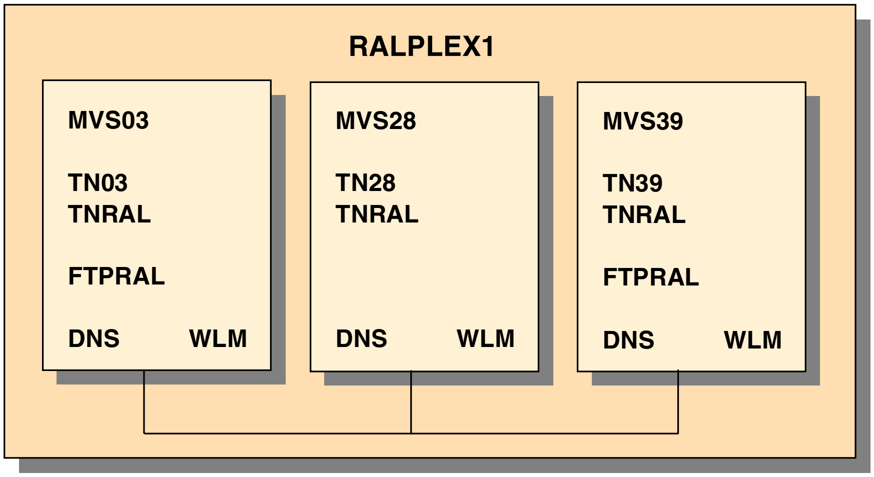


Figure 2-20 Telnet and FTP distribution in a sysplex

Figure 2-21 on page 42 shows the environment under which we have implemented our scenario. We use a sysplex system that consists of three OS/390 images running with IBM Communications Server for OS/390 V2R10 IP on each stack with OMPROUTE routing daemon executing the OSPF routing protocol. Though this scenario used OS/390 V2R10, it is still valid for z/OS V1R2. On system RA03, the primary DNS server for the sysplex domain `ralplex1.itso.ral.ibm.com` has been configured. As a secondary DNS server for this sysplex subdomain, the system RA28 is used. The parent DNS server, which is authoritative for the parent domain `itso.ral.ibm.com`, is located on the local LAN.

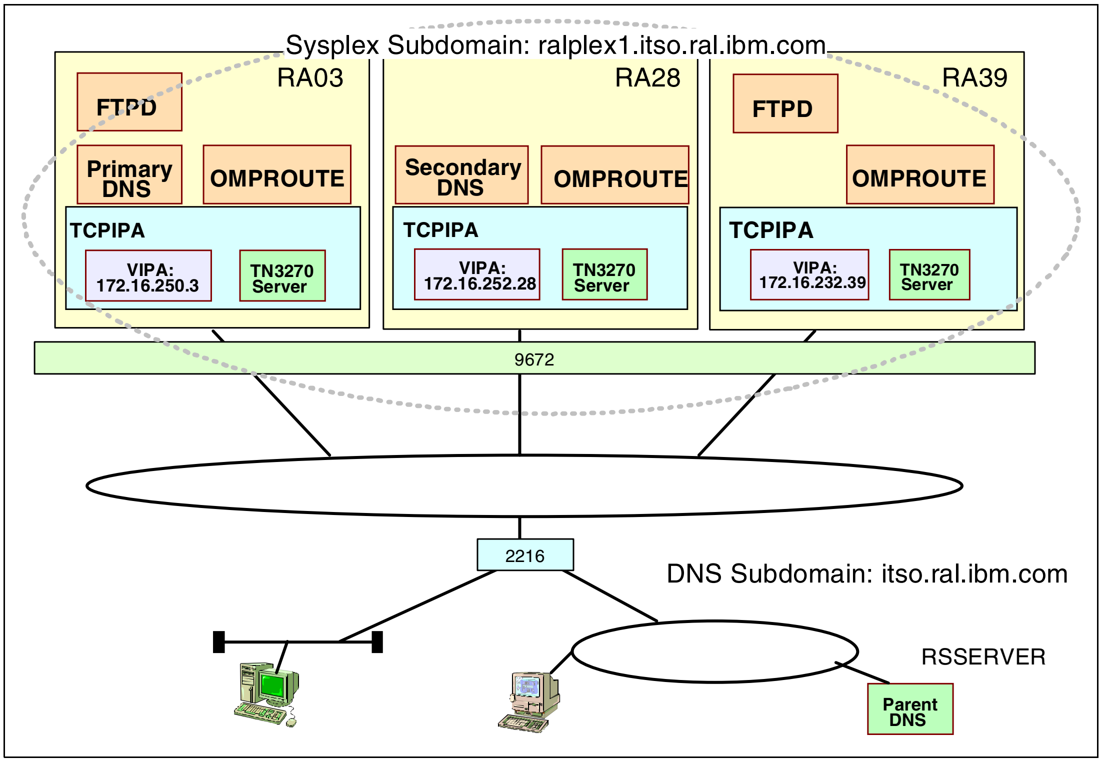


Figure 2-21 Environment at ITSO Raleigh

This type of DNS solution may also be desirable if you have implemented a firewall in your network. You may want only the primary name server (in our case, RSSERVER) to be accessible by workstations in the network. The workstations would not reference the sysplex name server in their configuration files but would rather point to the parent name server RSSERVER. RSSERVER, on the other hand, would be allowed to penetrate the firewall to reach the sysplex name server.

2.6.1 Primary DNS configuration on MVS03

The boot file initializes the name server environment and points to the individual name server definition files and to the options that the name server will provide for each zone it supports. Figure 2-22 on page 43 shows our /etc/named.boot file.

The following definitions can be specified in the BOOT file:

`directory` Defines the location of the files that are listed within the boot file.

`primary` Defines the domain name for the zone followed by the file to read for the name-to-IP/IP-to-name address mapping called the forward file. The mapping file to map the loopback address also has to be specified.

`cache` Corresponds to the root level domain, identified by a dot(.), and indicates the file in which the IP address of the root DNS server can be found. The cache file is also known as the *hint* file.

```
;
;    /etc/named.boot for TCPIPA on RA03
;
;  TYPE      DOMAIN                   HOST        FILE
;
directory    /etc/dnsdata
;
primary    ralplex1.itso.ral.ibm.com  1          ralplex1.for  cluster 2
primary    104.24.9.in-addr.arpa                 ralplex1.rev9
primary    16.172.in-addr.arpa                   ralplex1.rev
primary    0.0.127.in-addr.arpa                  mvs03a.lbk
cache      .                                     ralplex1.ca
```

Figure 2-22 /etc/named.boot file for DNS/WLM

This file looks similar to many other boot files, but there are two significant differences:

- 1 shows that our name server is primary for the sysplex subdomain called `ralpex1.itso.ral.ibm.com`
- 2 the cluster keyword indicates that this name server will communicate with the Workload Manager to achieve connection optimization. The keyword is used only once in the boot file for a DNS/WLM configuration.

Figure 2-23 shows you the forward file that we have pointed to in our boot file. Notice in 1 how MVS03 is the authority for the sysplex domain called `ralplex1.itso.ral.ibm.com.` If you have VIPAs 2 configured on your TCP/IP stacks, only VIPAs should be specified in the forward file, so that only VIPAs are returned to the DNS queries from clients. We have defined only VIPA addresses, as was shown in Figure 2-21 on page 42.

```
;  /etc/dnsdata/ralplex1.for for TCPIPA on RA03
;
$ORIGIN ralplex1.itso.ral.ibm.com.    1
@ IN      SOA     mvs03a.ralplex1.itso.ral.ibm.com. garthm@mvs03a    (
            1999040102  ; Serial
            7200        ; Refresh time after 2 hours
            3600        ; Retry after 1 hour
            604800      ; Expire after 1 week
            3600    )   ; Minimum TTL of 1 hour
                IN      NS      mvs03a
                IN      NS      mvs28a
mvs03a          IN      A       172.16.250.3      2
mvs03c          IN      A       172.16.251.5      2
mvs28a          IN      A       172.16.252.28     2
mvs39a          IN      A       172.16.232.39     2
```

Figure 2-23 DNS/WLM forward file

The reverse file, which is also referred to as an in-addr.arpa file, is referenced in the named.boot file as the primary DNS information for the in-addr.arpa domain (for example, 16.172.in-addr.arpa domain as shown in Figure 2-24 on page 44). Note the 1 inverse syntax used for referencing the in-addr.arpa file. The @ sign 2 on the start of authority (SOA) record is a special character that indicates the SOA is for the zone named in the named.boot file. It is used as a shorthand method, but it is equally as valid to specify the ORIGIN statement in the configuration. See Figure 2-24 on page 44 for details.

```
@ 2   IN    SOA     mvs03a.ralplex1.itso.ral.ibm.com. garthm@mvs03a (
            1999040101  ; Serial
            7200        ; Refresh time after 2 hours
            3600        ; Retry after 1 hour
            604800      ; Expire after 1 week
            3600    )   ; Minimum TTL of 1 hour
             IN  NS      mvs03a.ralplex1.itso.ral.ibm.com.
             IN  NS      mvs28a.ralplex1.itso.ral.ibm.com.
3.250   1    IN  PTR     mvs03a.ralplex1.itso.ral.ibm.com.
3.251   1    IN  PTR     mvs03c.ralplex1.itso.ral.ibm.com.
28.252  1    IN  PTR     mvs28a.ralplex1.itso.ral.ibm.com.
39.232  1    IN  PTR     mvs39a.ralplex1.itso.ral.ibm.com.
```

Figure 2-24 DNS/WLM reverse file

The loopback file is also referenced in the named.boot file as the primary name server information for the domain 0.0.127.in-addr.arpa. Please refer to Figure 2-25.

```
;
;        /etc/dnsdata/mvs03a.lbk for TCPIPA on RA03
;
@   IN      SOA     mvs03a.ralplex1.itso.ral.ibm.com. garthm@mvs03a    (
            1999040101  ; Serial
            7200        ; Refresh time after 2 hours
            3600        ; Retry after 1 hour
            604800      ; Expire after 1 week
            3600    )   ; Minimum TTL of 1 hour
                IN      NS      mvs03a.ralplex1.itso.ral.ibm.com.
1               IN      PTR     loopback.
127             IN      PTR     ralplex1.itso.ral.ibm.com.
```

Figure 2-25 DNS/WLM loopback file

2.6.2 Secondary DNS configuration MVS28

A secondary name server could be primary for some zones and secondary for others. Its boot file indicates for which zones it is primary and for which it is secondary. The cluster 1 keyword appears in the secondary name server's boot file, as it did in the primary name server. Again we identify the sysplex domain in the domain record. Figure 2-26 shows the boot file for the secondary name server.

```
;
;    /etc/named.boot for TCPIPA on RA28
;
;  TYPE      DOMAIN                      FILE OR HOST
directory      /etc/dnsdata
;
secondary   ralplex1.itso.ral.ibm.com     172.16.250.3  fback  cluster 1
secondary   104.24.9.in-addr.arpa         172.16.250.3  rback9
secondary   16.172.in-addr.arpa           172.16.250.3  rback
primary     0.0.127.in-addr.arpa          mvs28a.lbk
cache       .                             mvs28a.ca
```

Figure 2-26 /etc/named.boot file for MVS28 (secondary DNS server)

Using the zone transfer process, the secondary server retrieves the files for specified zones from the primary name server that it points to. The secondary stores this information in its own files if the retrieved serial number is higher than the current serial number stored in the secondary. The secondary obtains the files from the primary name server based upon the refresh interval coded on the SOA record or the resource record (RR) itself.

Figure 2-27 shows you the loopback file for the secondary DNS server that has to be configured.

```
;
;  /etc/dnsdata/mvs28a.lbk for T28ATCP
;
@   IN      SOA     mvs28a.ralplex1.itso.ral.ibm.com. garthm@mvs03a  (
              1999040101  ; Serial
              7200        ; Refresh time after 2 hours
              3600        ; Retry after 1 hour
              604800      ; Expire after 1 week
              3600    )   ; Minimum TTL of 1 hour
                IN      NS      mvs28a.ralplex1.itso.ral.ibm.com.
1               IN      PTR     loopback.
127             IN      PTR     ralplex1.itso.ral.ibm.com.
```

Figure 2-27 DNS/WLM loopback file of secondary DNS server

2.6.3 Parent DNS configuration

In this sample configuration, the parent name server is authoritative for the domain itso.ral.ibm.com. Additional configurations for the parent name server would be required if you want to use the resources in a sysplex without using fully qualified domain names (FQDN). 1 NS records identify the name servers for the sysplex subdomain, and two A 2 records identify the fully qualified domain names and addresses of the DNS servers. CNAME records identify the fully qualified names of resources that can be found by registering their short names, as in ping tnral. CNAME 3 allows the tnral resource name to work as if it were in the itso.ral.ibm.com domain. Clients that submit a query for tnral use their resolver code to fully qualify the name so that the name server sees it as tnral.itso.ral.ibm.com. The CNAME record for the tnral entry at the parent name server resolves tnral.itso.ral.ibm.com into an alias called tnral.ralplex1.itso.ral.ibm.com. The parent DNS server knows that the authoritative name server for ralplex1.itso.ral.ibm.com is either mvs03.ralplex1.itso.ral.ibm.com or mvs28.ralplex1.itso.ral.ibm.com. The parent DNS server sends the query to the sysplex name server, which then sends back a list of usable addresses.

```
$ORIGIN ral.ibm.com.
itso IN SOA rsserver.itso.ral.ibm.com. karina@itso.ral.ibm.com. (
1 10800  3600  60 800  86 00 )
$ORIGIN itso.ral.ibm.com.
ralplex1  IN  NS  mvs03.ralplex1.itso.ral.ibm.com. 1
ralplex1  IN  NS  mvs28.ralplex1.itso.ral.ibm.com. 1
mvs03.ralplex1.itso.ral.ibm.com.  IN  A 172.16.250.3  2
mvs28.ralplex1.itso.ral.ibm.com.  IN  A 172.16.252.28 2
tnral    IN CNAME  tnral.ralplex1.itso.ral.ibm.com.   3
tn03     IN CNAME  mvs03a.tnral.ralplex1.itso.ral.ibm.com. 3
tn28     IN CNAME  mvs28a.tnral.ralplex1.itso.ral.ibm.com. 3
:
```

Figure 2-28 Additions to parent domain server to reflect sysplex subdomain resources

You would have to add the NS records for the sysplex name server in the parent name server's reverse file configuration, because some applications, such as nslookup, rely on address-to-name resolution and will fail if your definitions are not comprehensive. Additional applications that also rely on address-to-name resolution are NFS and the DDNS client.

2.6.4 BIND DNS resource records

The entries configured in DNS data files are defined using a special format defined by RFC. Each resource record (RR) has the format shown below:

```
name ttl address_class record_type record_data
```

The `record_type` and `record_data` fields are the only required fields. The `name`, `ttl`, and `address_class` have defaults that may be set if they are not specified. The `record_type` indicates the type of resource record that defines the record data format. Valid resource record types include, but are not limited to, those shown in Table 2-3.

Table 2-3 DNS resource record types

Record Type	Description
SOA	Start of ZONE authority for the stated domain
A	Name to IP address translation
PTR	IP address to name translation
NS	Name of the authoritative DNS server for the stated domain
CNAME	Alias name

Resource record details

The SOA record specifies the fully qualified name of the host that has the domain name server authority for the zone. Note the period at the end of the resource name, which means that this name is fully qualified and should be appended with the ORIGIN. The SOA record includes a mailbox address of the user who is responsible for the zone, for example karina@itso.ral.ibm.com.

The SOA record is continued into the following master data set records. A left parenthesis signals that everything between here and a succeeding right parenthesis should be considered as belonging to the same resource record, despite record boundaries of the master data set.

You use the SERIAL field to keep track of your changes to the master data set. A good practice is to enter the date when you last made changes to the data set. A suggestion is to use `YYMMDDx`, where `x` is the number of updates per day. We were making changes several times a day, so we used a simple sequence number instead of the date. You must update the serial value every time you make changes to the zone. The secondary name servers determine when to do a zone transfer based on an increment in this value.

The REFRESH field is expressed in seconds. A secondary name server that has transferred this zone from the primary name server should not wait more than this number of seconds before it requests a refresh (a full zone transfer) from the primary name server. Before requesting a zone transfer, the secondary name server checks if the value of the serial field for the zone in question has changed or not. If not, a zone transfer is not necessary.

The RETRY field is expressed in seconds. If a secondary name server fails to refresh its copy of resource records, it should wait this number of seconds before it retries the refresh from the primary name server.

The EXPIRE field is expressed in seconds. This is the maximum time a secondary name server should consider its copy of resource data valid. If the secondary name server does not succeed with a zone transfer from the primary name server within this amount of time, it should consider its copy of the resource data obsolete, and stop answering queries for this zone.

The MINIMUM TTL field is expressed in seconds. Every time a response from this name server is sent out, it contains a time-to-live (TTL) field, which signifies how long the receiver should be able to consider the response valid. In BIND name servers, this MINIMUM TTL field really represents the DEFAULT if no TTL value has been specified on an individual resource. With the BIND name server, the TTL value on a resource takes precedence over the DEFAULT coded on the SOA record.

Note: For dynamic WLM resources, the TTL value defaults to 0 and can be specified with the *-l* start option.

2.6.5 Observing the effects of WLM and DNS

Where are all these definitions leading? We are using the configuration as documented earlier in this chapter. The configuration diagram is shown in Figure 2-21 on page 42. As you can see, we used VIPAs together with OMPROUTE.

The theory of sysplex operation indicates that we should be able to balance the load between applications registered with the same group name on more than one host in the sysplex. We have registered an application called TNRAL to each WLM in the ralplex1 sysplex. TNRAL represents a TN3270E server application. All the stacks are configured for TN3270E, so each stack has registered TNRAL with its respective Workload Manager using the VIPA address. We have also registered an application called ftpral to WLM running on MVS03 and MVS39.

Earlier in this chapter, we described a configuration with two name servers. The primary name server for the sysplex runs on MVS03 and a secondary sysplex name server runs on MVS28. In many organizations the TCP/IP network was established long before the sysplex. As a result, workstations have been configured to use an existing name server for name resolution. In most cases this happened before TCP/IP was installed on the mainframe. In our implementation, this "existing" name server resides on a third system.

We execute our test using a small REXX EXEC that sends as many pings as needed to a specified application and keeps a tally of the resource for each address returned by the name server, and writes it to a file. The source is provided in Appendix B, “REXX EXECs to gather connection statistics” on page 261 for Windows 95/98 and Windows NT workstations. By the way, REXX is not provided as part of Windows 95/98/NT, so a separate REXX interpreter is required.

The REXX EXEC is executed by the following line commands:

```
REXX SYSPLEXW WLM_registered_name number_of_pings extra_delay
```

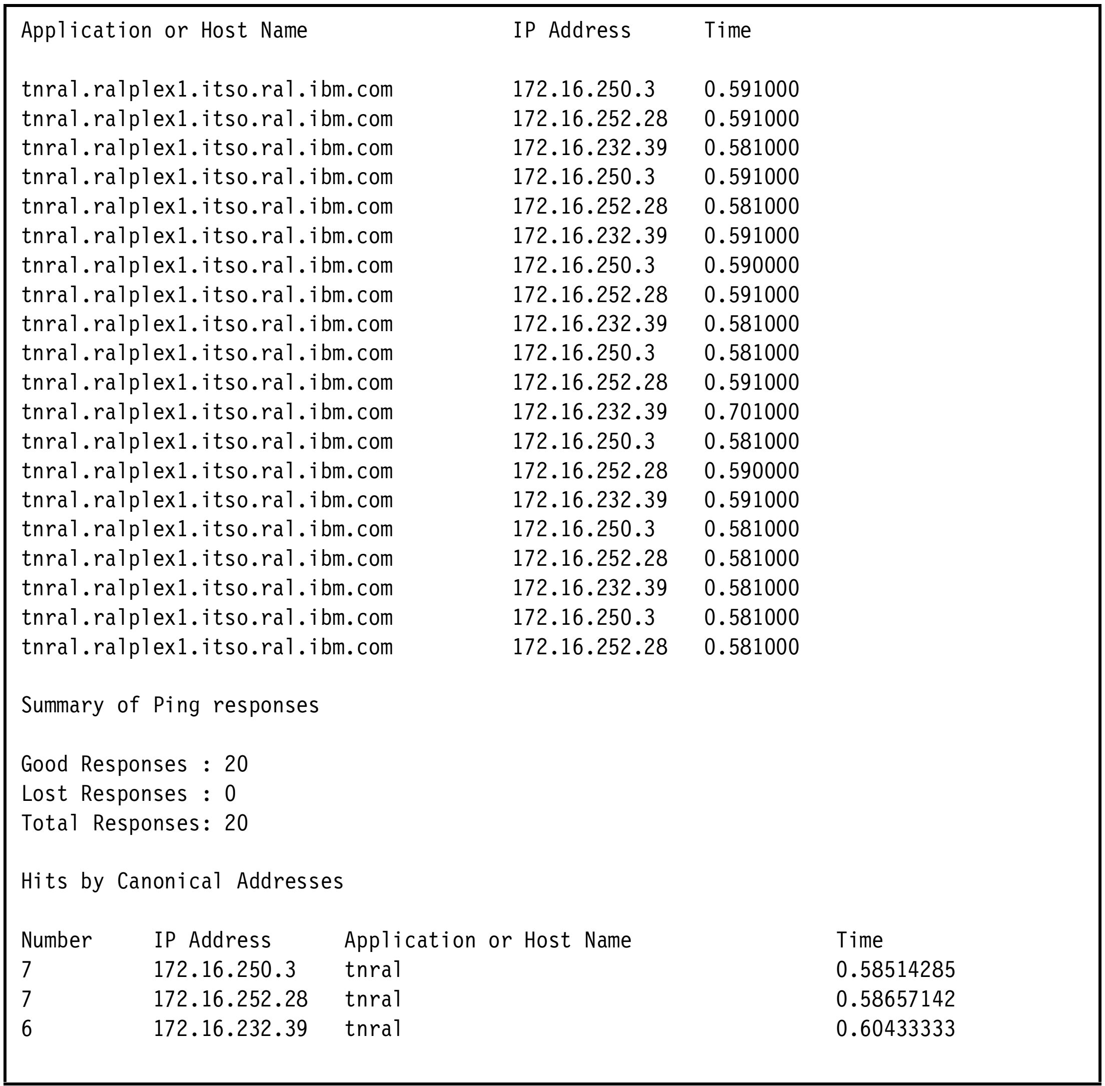
Where the parameters are:

- WLM_registered_name is the name used to define the application group to WLM (for example tnral)
- number_of_pings is how many pings you want to send (default 10)
- extra_delay is an optional value that inserts a delay in seconds in the ping loop (default 0)

This test of the balancing effect of DNS/WLM was run when the systems were lightly loaded. In this environment we expected an even balance of system selection. The workstation was connected directly to the sysplex. The following command was issued:

```
REXX SYSPLEXW TNRAL 20
```

The results of running the REXX EXEC are shown in Figure 2-29.



```
Application or Host Name               IP Address      Time

tnral.ralplex1.itso.ral.ibm.com        172.16.250.3    0.591000
tnral.ralplex1.itso.ral.ibm.com        172.16.252.28   0.591000
tnral.ralplex1.itso.ral.ibm.com        172.16.232.39   0.581000
tnral.ralplex1.itso.ral.ibm.com        172.16.250.3    0.591000
tnral.ralplex1.itso.ral.ibm.com        172.16.252.28   0.581000
tnral.ralplex1.itso.ral.ibm.com        172.16.232.39   0.591000
tnral.ralplex1.itso.ral.ibm.com        172.16.250.3    0.590000
tnral.ralplex1.itso.ral.ibm.com        172.16.252.28   0.591000
tnral.ralplex1.itso.ral.ibm.com        172.16.232.39   0.581000
tnral.ralplex1.itso.ral.ibm.com        172.16.250.3    0.581000
tnral.ralplex1.itso.ral.ibm.com        172.16.252.28   0.591000
tnral.ralplex1.itso.ral.ibm.com        172.16.232.39   0.701000
tnral.ralplex1.itso.ral.ibm.com        172.16.250.3    0.581000
tnral.ralplex1.itso.ral.ibm.com        172.16.252.28   0.590000
tnral.ralplex1.itso.ral.ibm.com        172.16.232.39   0.591000
tnral.ralplex1.itso.ral.ibm.com        172.16.250.3    0.581000
tnral.ralplex1.itso.ral.ibm.com        172.16.252.28   0.581000
tnral.ralplex1.itso.ral.ibm.com        172.16.232.39   0.581000
tnral.ralplex1.itso.ral.ibm.com        172.16.250.3    0.581000
tnral.ralplex1.itso.ral.ibm.com        172.16.252.28   0.581000

Summary of Ping responses

Good Responses : 20
Lost Responses : 0
Total Responses: 20

Hits by Canonical Addresses

Number     IP Address      Application or Host Name                Time
7          172.16.250.3    tnral                                   0.58514285
7          172.16.252.28   tnral                                   0.58657142
6          172.16.232.39   tnral                                   0.60433333
```

Figure 2-29 Distribution test of telnet in the sysplex DNS/WLM environment

Examining the output of the EXEC, we can see the name `tnral` has been fully qualified as `tnral.ralplex1.itso.ral.ibm.com`. This is the name reported by the ping application. The address assigned is in the next column and we see that the address provided by DNS started with 172.16.250.3, 172.16.252.28 and then 172.16.232.39 and then it repeats continuously. The time entry is the elapsed time between ping iterations, not the response time reported by ping. The summary of ping responses is used to check whether any pings were lost. Hits by canonical addresses summarize how DNS distributed the workload requests from this workstation. If there were other requests for the same application, then the results may well look different. Again, time shows the average elapsed time between pings, including the DNS lookup time, the ping response time, and any delays introduced.

The results from running the EXEC showed an evenly balanced workload between applications running on all three MVS systems.

2.6.6 DNS DUMP of primary DNS server in the sysplex

To confirm the number of instances of an application in a WLM group, we can dump the information in the DNS to see what has been registered to WLM, since WLM data is periodically retrieved by DNS. From the WLM information, DNS determines and then stores the current state of application availability. The dump of DNS does not show the relative weighting of the host or application instances, but only their registration. There are two ways to dump the DNS server. Please refer to 2.5.6, “Dumping the DNS server cache” on page 35 for a detailed description. We divided the dump in different parts to make it easier to go through the dump and explain it. You will find the whole dump in Appendix A, “Dump of T28ATCP name server - single-path network” on page 259.

```
; Dumped at Tue Sep 19 14:36:12 2000
;; ++zone table++ 1
; ralplex1.itso.ral.ibm.com (type 1, class 1, source ralplex1.for)
;  time=969374208, lastupdate=937949821, serial=1999040102,
;  refresh=7200, retry=3600, expire=604800, minimum=3600
;  ftime=937949821, xaddr=[0.0.0.0], state=20041, pid=0
; 104.24.9.in-addr.arpa (type 1, class 1, source ralplex1.rev9)
;  time=969378400, lastupdate=937951070, serial=1999040101,
;  refresh=7200, retry=3600, expire=604800, minimum=3600
;  ftime=937951070, xaddr=[0.0.0.0], state=0041, pid=0
; 16.172.in-addr.arpa (type 1, class 1, source ralplex1.rev)
;  time=969379773, lastupdate=937949840, serial=1999040101,
;  refresh=7200, retry=3600, expire=604800, minimum=3600
;  ftime=937949840, xaddr=[0.0.0.0], state=0041, pid=0
; 0.0.127.in-addr.arpa (type 1, class 1, source mvs03a.lbk)
;  time=969377837, lastupdate=937950997, serial=1999040101,
;  refresh=7200, retry=3600, expire=604800, minimum=3600
;  ftime=937950997, xaddr=[0.0.0.0], state=0041, pid=0
;; --zone table--
```

Figure 2-30 Zone table of the primary DNS server in the sysplex

The first thing we see in the dump is the zone table (1) from the primary DNS on MVS03. It defines the domain name, `ralplex1.itso.ral.ibm.com`, the time and date of the dump, and the zone information. The zone information has three sections:

- The sysplex domain
- The in-addr.arpa
- The loopback

These three sections refer to the three records defined in named.boot on MVS03. Figure 2-22 on page 43 shows the corresponding boot records.

In Figure 2-31 on page 50, we can see the information obtained from the primary DNS concerning `ralplex1`. This information comes from the primary DNS forward cluster file pointed to in the named.boot. The forward cluster file is shown in Figure 2-23 on page 43.

The $ORIGIN and SOA records identify the source of the information, `mvs03a.ralplex1.itso.ral.ibm.com` 4. The `IN NS` record 5 defines the primary DNS while the `IN A` records 6 define the IP home addresses available to the sysplex. Each home address also identifies a separate IP stack.

```
$ORIGIN itso.ral.ibm.com.
ralplex1   4    IN      SOA     mvs03a.ralplex1.itso.ral.ibm.com. (
                1999040102 7200 3600 604800 3600 )      ;Cl=5
           5    IN      NS      mvs03a.ralplex1.itso.ral.ibm.com.
           6    IN      A       172.16.250.3    ;Cl=5
           6    IN      A       172.16.252.28   ;Cl=5
           6    IN      A       172.16.232.39   ;Cl=5
```

Figure 2-31 More data from the primary DNS server

Figure 2-32 shows the domain statements for `ralplex1.itso.ral.ibm.com`. There we can see entries for the WLM-managed applications: `TNRAL`, `TN03`,`TN28`,`TN39`, and `FTPRAL`. `TNRAL` 7 is available on all three stacks 03,28 and 39, where `FTPRAL` 8 is only available on stacks 03 and 39. `TN03`,`TN28`, and `TN39` 9 are only available on their corresponding stacks as you can see in the dump. Even though we are looking at the DNS entries for the sysplex domain, not all of the DNS entries necessarily reflect applications that are sysplex capable. Host entries come either as part of a zone transfer from the primary name server or dynamically from WLM. Only those defined to WLM will have multiple entries.

The information in this part of the dump will be updated dynamically to reflect the application availability. Should one instance (for example `FTPRAL`) of a WLM-managed application be brought down, then this DNS record would be updated accordingly. Similarly, should an additional instance of the application be started, we would then see the new application instance in this record. For information on how applications can register to WLM, see 2.4, “Application and stack registration to WLM” on page 26.

```
$ORIGIN ralplex1.itso.ral.ibm.com.
FTPRAL    8     IN      A       172.16.232.39   ;Cl=5
          8     IN      A       172.16.250.3    ;Cl=5
TN28      9     IN      A       172.16.252.28   ;Cl=5
mvs03a          IN      A       172.16.250.3    ;Cl=5
TN03      9     IN      A       172.16.250.3    ;Cl=5
mvs03c          IN      A       172.16.251.5    ;Cl=5
mvs28a          IN      A       172.16.252.28   ;Cl=5
TNTSO           IN      A       172.16.250.3    ;Cl=5
TNRAL     7     IN      A       172.16.250.3    ;Cl=5
          7     IN      A       172.16.252.28   ;Cl=5
          7     IN      A       172.16.232.39   ;Cl=5
ralplex1        IN      CNAME   ralplex1.itso.ral.ibm.com.
TN39      9     IN      A       172.16.232.39   ;Cl=5
mvs39a          IN      A       172.16.232.39   ;Cl=5
```

Figure 2-32 Servers defined in the sysplex domain

2.6.7 DNS trace of WLM data for the primary DNS in the sysplex

The DNS dump showed us which applications have multiple instances. How can we determine the balance weights that DNS will use when building the application selection table, based on CPU utilization of each system? There is no way to directly generate queries to WLM, so the next best approach is to trace the activity in DNS itself. When DNS is started, the parameter `-d 11` can be passed from the EXEC PARM keyword.

The DNS trace is much longer than the dump we showed earlier, so we will split the trace into different parts for illustrative purposes.

The first area of interest comprises the applications managed by WLM. Note that there are no static records for them. We have not defined them anywhere in DNS. We have told DNS to query WLM to see what additional applications are available.

We can see the WLM query 1 for `ralplex1.itso.ral.ibm.com`. We have listed here only three entries returned by WLM: `TCPIP` 2, `TNRAL` 3 and `FTPRAL` 4. TCP/IP is automatically generated by each stack when it starts up, but DNS *will not* respond to this application name, and it will not be shown in a dump of DNS. Please review Figure 2-33 for this trace output. Note that some of the traces included in this section are from an earlier release and have not been generated again due to the small amount of change in the output.

```
wlm_load ralplex1.itso.ral.ibm.com        1
 group list retcode = -1 rsncode = 1034
 group list retcode = 0 rsncode = 0
 group list entry_count = 7
 Group list from WLM follows:
  Group number 1 = TCPIP
  Group number 2 = TN03
  Group number 3 = TNRAL
  Group number 4 = TNTSO
  Group number 5 = FTPRAL
  Group number 6 = TN28
  Group number 7 = TN39
 End of Group list
.......
querying group =  TCPIP                   2
server list retcode = -1 rsncode = 1034
server list retcode = 0 rsncode = 0
querying group =  TNRAL                   3
server list retcode = -1 rsncode = 1034
server list retcode = 0 rsncode = 0
querying group =  FTPRAL                  4
server list retcode = -1 rsncode = 1034
server list retcode = 0 rsncode = 0
```

Figure 2-33 WLM data from DNS trace part 1 (querying group names)

In Figure 2-34 on page 52, we can see the entries for each of the stacks and the relative weights associated with each system. The entry for the 6 processing group shows the minimum weight available for a server to be 21. Once the processing group parameters have been established, each of the servers is assigned a weight. The assigned weights 7 of the servers are 21 for all three servers. As you can see, the hosts are fairly evenly balanced.

```
processing group = ralplex1.itso.ral.ibm.com count = 3
minimum weight = 21  6
processing server = TCPIPA weight = 21 host = MVS03A   7
adding MVS03A to list
db_update(ralplex1.itso.ral.ibm.com, 0x16154630, 0x16154630, 01, 0x16153970)
match(0x161539e0, 1, 6) 1, 6
match(0x161539e0, 1, 1) 1, 6
match(0x16153bf0, 1, 1) 1, 2
db_update: adding 16154630
processing server = TCPIPA weight = 21 host = MVS28A   7
adding MVS28A to list
db_update(ralplex1.itso.ral.ibm.com, 0x16154668, 0x16154668, 01, 0x16153970)
match(0x161539e0, 1, 6) 1, 6
match(0x161539e0, 1, 1) 1, 6
match(0x16153bf0, 1, 1) 1, 2
match(0x16154630, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for ralplex1.itso.ral.ibm.com is 4(5) from [0.0.0.0].0, is 4(5) in
db_update: adding 16154668
 processing server = TCPIPA weight = 21 host = MVS39A    7
 adding MVS39A to list
```

Figure 2-34   WLM data from DNS trace part2 (assigned WLM weight to OS/390 images)

In Figure 2-35 on page 53 we can see that the minimum weight for the TNRAL application is 21 8 and that there are three instances of the application. All three applications have the same minimum weight of 21 9 on each of the systems, as you can see in the trace. This would provide a fairly evenly balanced load.

```
processing group = TNRAL.ralplex1.itso.ral.ibm.com count = 3
minimum weight = 21                                           8
processing server = MVS03A weight = 21 host = MVS03A  9
db_update(TNRAL.ralplex1.itso.ral.ibm.com, 0x161547f8, 0x161547f8, 01, 0x16153
match(0x161539e0, 1, 6) 1, 6
db_update: adding 161547f8
db_update(MVS03A.TNRAL.ralplex1.itso.ral.ibm.com, 0x16154868, 0x16154868, 01,
savehash GROWING to 2
match(0x161547f8, 1, 6) 1, 1
match(0x161547f8, 1, 2) 1, 1
match(0x161539e0, 1, 6) 1, 6
db_update: adding 16154868
processing server = MVS28A weight = 21 host = MVS28A  9
db_update(TNRAL.ralplex1.itso.ral.ibm.com, 0x161548f0, 0x161548f0, 01, 0x16153
match(0x161547f8, 1, 6) 1, 1
match(0x161547f8, 1, 2) 1, 1
match(0x161539e0, 1, 6) 1, 6
match(0x161547f8, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for TNRAL.ralplex1.itso.ral.ibm.com is 4(5) from [0.0.0.0].0, is 4
db_update: adding 161548f0
db_update(MVS28A.TNRAL.ralplex1.itso.ral.ibm.com, 0x16154928, 0x16154928, 01,
match(0x161547f8, 1, 6) 1, 1
match(0x161548f0, 1, 6) 1, 1
match(0x161547f8, 1, 2) 1, 1
match(0x161548f0, 1, 2) 1, 1
match(0x161539e0, 1, 6) 1, 6
db_update: adding 16154928
processing server = MVS39A weight = 21 host = MVS39A  9
db_update(TNRAL.ralplex1.itso.ral.ibm.com, 0x16154998, 0x16154998, 01, 0x16153
match(0x161547f8, 1, 6) 1, 1
match(0x161548f0, 1, 6) 1, 1
match(0x161547f8, 1, 2) 1, 1
match(0x161548f0, 1, 2) 1, 1
match(0x161539e0, 1, 6) 1, 6
match(0x161547f8, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp 1)
credibility for TNRAL.ralplex1.itso.ral.ibm.com is 4(5) from [0.0.0.0].0, is 4
match(0x161548f0, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp 1)
credibility for TNRAL.ralplex1.itso.ral.ibm.com is 4(5) from [0.0.0.0].0, is 4
db_update: adding 16154998
db_update(MVS39A.TNRAL.ralplex1.itso.ral.ibm.com, 0x161549d0, 0x161549d0, 01,
match(0x161547f8, 1, 6) 1, 1
match(0x161548f0, 1, 6) 1, 1
match(0x16154998, 1, 6) 1, 1
match(0x161547f8, 1, 2) 1, 1
match(0x161548f0, 1, 2) 1, 1
match(0x16154998, 1, 2) 1, 1
match(0x161539e0, 1, 6) 1, 6
db_update: adding 161549d0
```

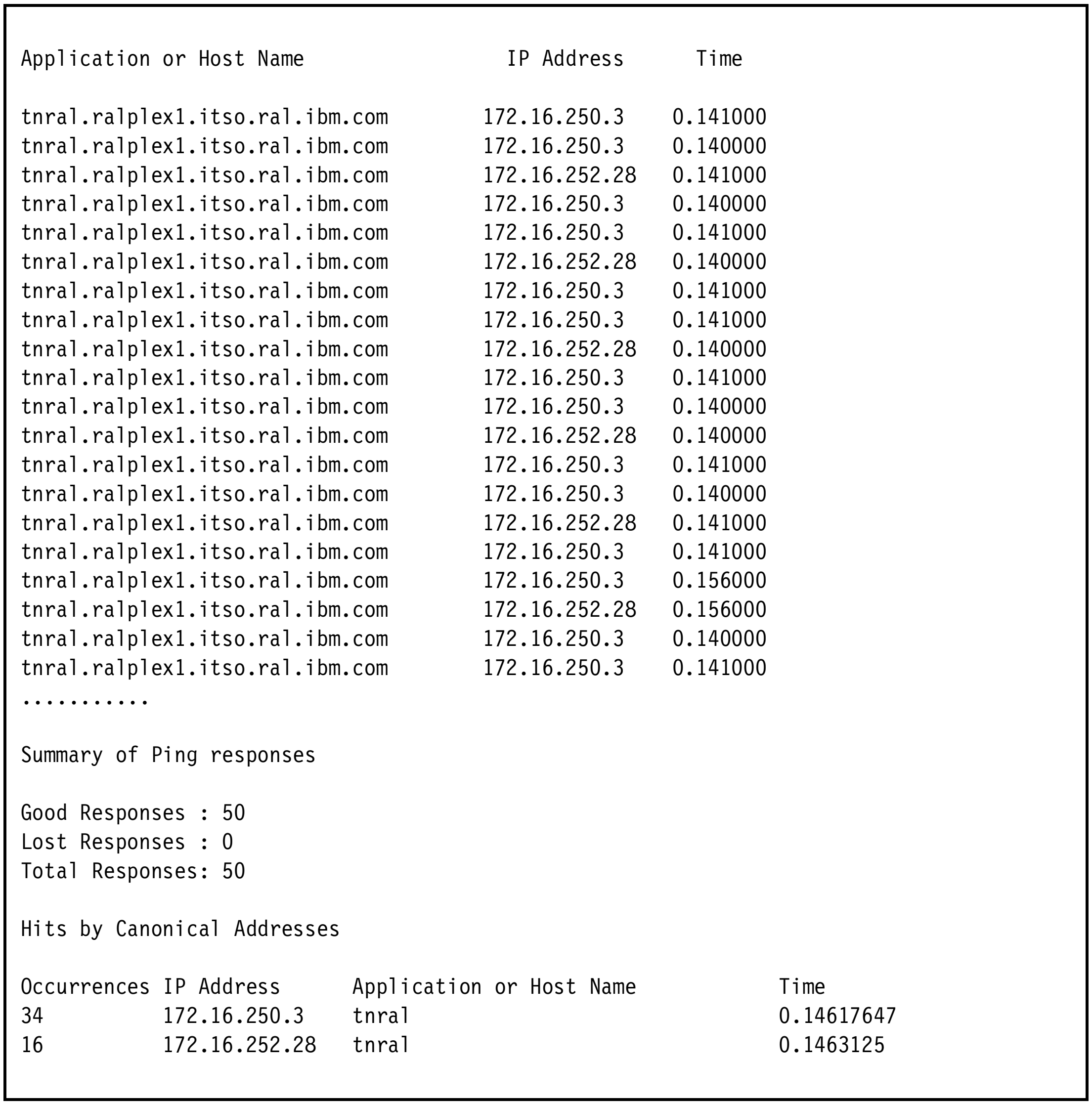
Figure 2-35 Application weights

2.6.8 Testing workload distribution with different CPU utilizations

We now looked at DNS/WLM distribution to target servers that had varying CPU utilizations. During this test, Telnet and FTP servers were running on the systems MVS03 and MVS28 (we exclude MVS39 from this test). The application instances have registered to WLM with the group names `TNRAL` and `FTPRAL`.

Since all of our previous examples showed lightly loaded systems and even balancing, we now show an unbalanced environment where one of the hosts has a noticeably heavier workload than the other. When this test was running, we introduced a job on the MVS28 system to increase CPU utilization to over 50% while the MVS03 system was running around 20% CPU utilization. The load observations are very rough in that they were taken from SDSF samples and made no allowance for other resource utilization such as paging, I/O rate, or memory, but it gives you a general idea of the overall load on each of the systems.

The output from the SYSPLEXW EXEC for this run is shown in Figure 2-36. The run was for 50 samples at 0 delay, but only the first 20 entries are shown.



```
Application or Host Name                 IP Address      Time

tnral.ralplex1.itso.ral.ibm.com       172.16.250.3    0.141000
tnral.ralplex1.itso.ral.ibm.com       172.16.250.3    0.140000
tnral.ralplex1.itso.ral.ibm.com       172.16.252.28   0.141000
tnral.ralplex1.itso.ral.ibm.com       172.16.250.3    0.140000
tnral.ralplex1.itso.ral.ibm.com       172.16.250.3    0.141000
tnral.ralplex1.itso.ral.ibm.com       172.16.252.28   0.140000
tnral.ralplex1.itso.ral.ibm.com       172.16.250.3    0.141000
tnral.ralplex1.itso.ral.ibm.com       172.16.250.3    0.141000
tnral.ralplex1.itso.ral.ibm.com       172.16.252.28   0.140000
tnral.ralplex1.itso.ral.ibm.com       172.16.250.3    0.141000
tnral.ralplex1.itso.ral.ibm.com       172.16.250.3    0.140000
tnral.ralplex1.itso.ral.ibm.com       172.16.252.28   0.140000
tnral.ralplex1.itso.ral.ibm.com       172.16.250.3    0.141000
tnral.ralplex1.itso.ral.ibm.com       172.16.250.3    0.140000
tnral.ralplex1.itso.ral.ibm.com       172.16.252.28   0.141000
tnral.ralplex1.itso.ral.ibm.com       172.16.250.3    0.141000
tnral.ralplex1.itso.ral.ibm.com       172.16.250.3    0.156000
tnral.ralplex1.itso.ral.ibm.com       172.16.252.28   0.156000
tnral.ralplex1.itso.ral.ibm.com       172.16.250.3    0.140000
tnral.ralplex1.itso.ral.ibm.com       172.16.250.3    0.141000
...........

Summary of Ping responses

Good Responses : 50
Lost Responses : 0
Total Responses: 50

Hits by Canonical Addresses

Occurrences IP Address      Application or Host Name           Time
34          172.16.250.3    tnral                              0.14617647
16          172.16.252.28   tnral                              0.1463125
```

Figure 2-36   Workload distribution of DSN/WLM with different CPU utilizations

As can be seen from the results, the load was balanced 2 to 1 in favor of the MVS03 system. How did DNS determine the appropriate load balance? To see this, it is again necessary to look at the DNS trace. Figure 2-37 on page 55 shows us only a part of the trace where the WLM weight is shown for the processing group `TNRAL`.

```
wlm_load ralplex1.buddha.ral.ibm.com
 querying group =  FTPRAL
 querying group =  TNRAL
 querying group =  TCPIP
 processing group = FTPRAL.ralplex1.buddha.ral.ibm.com count = 2
 minimum weight = 21
 processing server = MVS03A weight = 42 host = MVS03A
savehash GROWING to 2
 processing server = MVS28A weight = 21 host = MVS28A
db_update(FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14dce9f8, 0x14dce9f8, 01, 0x14dcd8b8)
credibility for FTPRAL.ralplex1.buddha.ral.ibm.com is 4(5) from  0.0.0.0 .0, is 4(5) in
cache
db_update(MVS28A.FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14dcea30, 0x14dcea30, 01,
0x14dcd8b8)
 processing group = TNRAL.ralplex1.buddha.ral.ibm.com count = 2
 minimum weight = 21                                    1
 processing server = MVS03A weight = 42 host = MVS03A    2
db_update(TNRAL.ralplex1.buddha.ral.ibm.com, 0x14dceaa0, 0x14dceaa0, 01, 0x14dcd8b8)
db_update(MVS03A.TNRAL.ralplex1.buddha.ral.ibm.com, 0x14dceb10, 0x14dceb10, 01,
0x14dcd8b8)
savehash GROWING to 2
 processing server = MVS28A weight = 21 host = MVS28A    3
```

Figure 2-37 Application weights for test with different CPU utilizations

At 1 we see the minimum weight 21 that can be assigned to TNRAL. WLM assigns a weight of 42 for the processing server MVS03A, while MVS28A was assigned a weight of 21. As you may have noticed, the weight is identical for TNRAL and for FTPRAL, because they are running on the same MVS images.

2.6.9 More on resource record TTL

We were running the SYSPLEXW EXEC from two different clients; one of them was configured to use the external name server and the other one to use the sysplex name server.

We started our external name server with a time-to-live of three seconds to illustrate the difference in operation between our sysplex DNS (which always returns TLL=0 to the client) and a typical name server (which is unlikely to return TTL=0 even if the sysplex DNS is configured to do so).

First, we pointed our client to the external name server. We again used the SYSPLEXW REXX EXEC. We issued the following command:

```
SYSPLEXW TNRAL 300
```

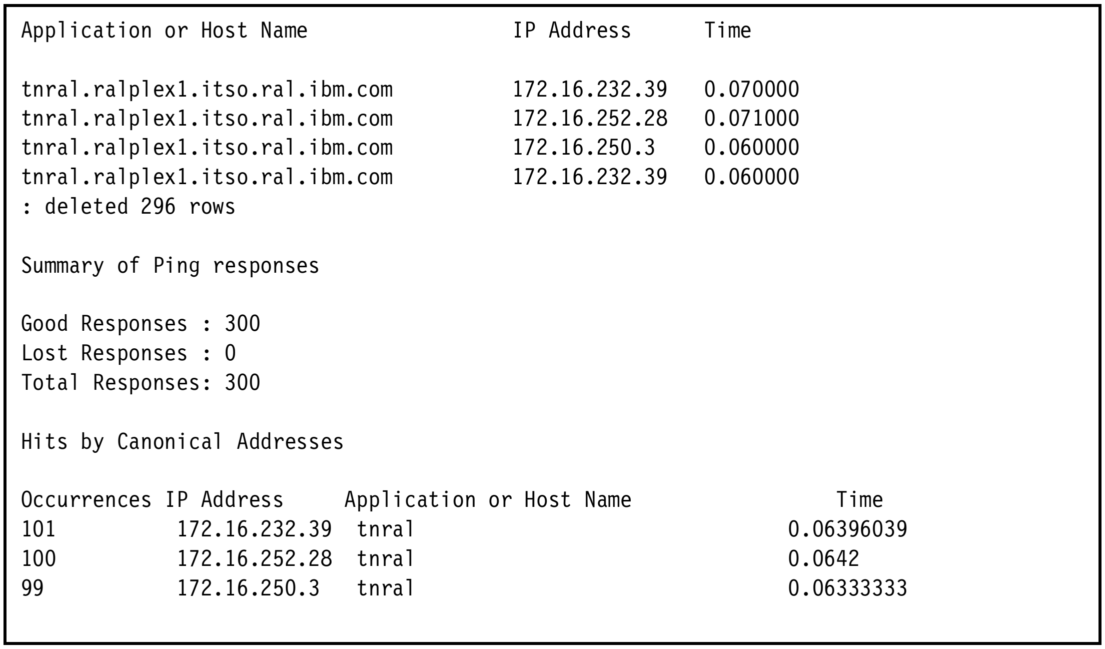
Figure 2-38 on page 56 shows the resulting output.

```
Application or Host Name                 IP Address      Time

tnral.ralplex1.itso.ral.ibm.com          172.16.250.3    0.094000
tnral.ralplex1.itso.ral.ibm.com          172.16.252.28   0.125000
: deleted 35 rows to 172.16.252.28
tnral.ralplex1.itso.ral.ibm.com          172.16.232.39   0.125000
: deleted 35 rows to 172.16.232.39
tnral.ralplex1.itso.ral.ibm.com          172.16.250.3    0.125000
: deleted 35 rows to 172.16.250.3
tnral.ralplex1.itso.ral.ibm.com          172.16.252.28   0.172000
: deleted * rows

Summary of Ping responses

Good Responses : 330
Lost Responses : 0
Total Responses: 330

Hits by Canonical Addresses

Occurrences IP Address     Application or Host Name                 Time
109         172.16.250.3   tnral                                    0.09606422
113         172.16.252.28  tnral                                    0.11131858
108         172.16.232.39  tnral                                    0.09563888
```

Figure 2-38   Distributing workload result - external name server

As you can see the client using the external name server received the same name resolution from the name server for about three seconds. The external name server responded with its cached entry during the TTL period. After the TTL period the name server returned a new address and in the long run the returned addresses seemed to be equally balanced among the three stacks in the sysplex.

Figure 2-39 on page 57 shows the corresponding output on the client using the sysplex name server. In this case the lines deleted looked like the first three repeated.



```
Application or Host Name                IP Address      Time

tnral.ralplex1.itso.ral.ibm.com         172.16.232.39   0.070000
tnral.ralplex1.itso.ral.ibm.com         172.16.252.28   0.071000
tnral.ralplex1.itso.ral.ibm.com         172.16.250.3    0.060000
tnral.ralplex1.itso.ral.ibm.com         172.16.232.39   0.060000
: deleted 296 rows

Summary of Ping responses

Good Responses : 300
Lost Responses : 0
Total Responses: 300

Hits by Canonical Addresses

Occurrences IP Address     Application or Host Name                Time
101          172.16.232.39  tnral                              0.06396039
100          172.16.252.28  tnral                              0.0642
99           172.16.250.3   tnral                              0.06333333
```

Figure 2-39 Distributing workload result - sysplex name server

The client configured to use the sysplex name server received a different resolution on each query. The sysplex name server did not cache a single answer for three seconds, as did the external name server; it returned exactly what WLM indicated it should.

2.6.10 Test application

Next, we ran a job that registered our application to DNS with the name `TESTRAL`, then started the socket server application. The application was set to listen on port 1234. We ran this job on all of our sysplex hosts. Please refer to 2.4, “Application and stack registration to WLM” on page 26 for how to register your own application.

On the client side we used the client test program described in “EXEC to connect to server using TCP” on page 266 to perform our tests. The EXEC will connect to a server on a given host name/port a specified number of times. On each connection it will read 4 bytes from the server.

We executed **`sysplex2 testral 1234 -c 350 -b 0.1`** from both of our clients, one at a time. Figure 2-40 on page 58 shows the results for the client with the external name server, and Figure 2-41 on page 58 shows those for the other client.

```
+---------------------+-----------------+---------------+--------+--------+
| Hostname            | Resolved Addr   | Connected Addr | Resolv | Connec |
+---------------------+-----------------+---------------+--------+--------+
| testral             | 172.16.250.3    | 172.16.250.3   | 0.0630 | 0.0160 |
| : deleted 22 rows to 172.16.250.3
| testral             | 172.16.252.28   | 172.16.252.28  | 0.0320 | 0.0160 |
| : deleted 27 rows to 172.16.252.28
| testral             | 172.16.232.39   | 172.16.232.39  | 0.0310 | 0.0160 |
| : deleted 27 rows to 172.16.232.39
| testral             | 172.16.250.3    | 172.16.250.3   | 0.0310 | 0.0160 |
| : deleted 28 rows to 172.16.250.3
  :

+---------------+---------------+
| Resolved Addr  | Resolved Count |
+---------------+---------------+
| 172.16.250.3   | 115            |
| 172.16.252.28  | 112            |
| 172.16.232.39  | 123            |

+---------------+---------------+
| Connected To   | Connected Count|
+---------------+---------------+
| 172.16.250.3   | 115            |
```

Figure 2-40   Socket application client result - external name server

```
+---------------------+-----------------+---------------+--------+--------+
| Hostname            | Resolved Addr   | Connected Addr | Resolv | Connec |
+---------------------+-----------------+---------------+--------+--------+
| testral             | 172.16.232.39   | 172.16.232.39  | 0.0200 | 0.0100 |
| testral             | 172.16.250.3    | 172.16.250.3   | 0.0100 | 0.0100 |
| testral             | 172.16.252.28   | 172.16.252.28  | 0.0100 | 0.0100 |
   :

+---------------+---------------+
| Resolved Addr  | Resolved Count |
+---------------+---------------+
| 172.16.250.3   | 100            |
| 172.16.252.28  | 100            |
| 172.16.232.39  | 100            |

+---------------+---------------+
| Connected To   | Connected Count|
+---------------+---------------+
| 172.16.250.3   | 100            |
| 172.16.252.28  | 100            |
| 172.16.232.39  | 100            |
```

Figure 2-41   Socket application client result - sysplex name server

2.6.11 Test application - server failure case

Next, we simulated the failure of one of the socket application servers while the socket application client was running. Figure 2-42 shows how the DNS/WLM dealt with the failure in the external name server case. Figure 2-43 on page 60 shows the sysplex name server case.

```
 sysplex2 testral 1234 -c 100 -b 1.5

+----------------------+------------------+----------------+--------+--------+
| Hostname             | Resolved Addr    | Connected Addr | Resolv | Connec
|+----------------------+------------------+----------------+--------+--------+
| testral              | 172.16.232.39    | 172.16.232.39  | 0.0630 | 0.0160 |
| : deleted 3 rows to 172.16.232.39
| testral              | 172.16.252.28    | 172.16.252.28  | 0.0310 | 0.0160 |
| : deleted 2 rows to 172.16.252.28
| testral              | 172.16.250.3     | 172.16.250.3   | 0.0320 | 0.0150 |
| : deleted 2 rows to 172.16.250.3
| testral              | 172.16.252.28    | 172.16.252.28  | 0.0310 | 0.0310 |
| : deleted 2 rows to 172.16.252.28
| testral              | 172.16.232.39    | 172.16.232.39  | 0.0310 | 0.0160 |
| : deleted 3 rows to 172.16.232.39
Error on connecting socket to '172.16.250.3': ECONNREFUSED
Error on connecting socket to '172.16.250.3': ECONNREFUSED
Error on connecting socket to '172.16.250.3': ECONNREFUSED
| testral              | 172.16.232.39    | 172.16.232.39  | 0.0320 | 0.0150 |
| : deleted 3 rows to 172.16.232.39
| testral              | 172.16.252.28    | 172.16.252.28  | 0.0310 | 0.0160 |
| : deleted 2 rows to 172.16.252.28
Error on connecting socket to '172.16.250.3': ECONNREFUSED
Error on connecting socket to '172.16.250.3': ECONNREFUSED
Error on connecting socket to '172.16.250.3': ECONNREFUSED
Error on connecting socket to '172.16.250.3': ECONNREFUSED
| testral              | 172.16.232.39    | 172.16.232.39  | 0.0310 | 0.0160 |
| : deleted 5 rows to 172.16.232.39
| testral              | 172.16.252.28    | 172.16.252.28  | 0.0310 | 0.0150 |
| : deleted 5 rows to 172.16.252.28
| testral              | 172.16.232.39    | 172.16.232.39  | 0.0310 | 0.0160 |
| : deleted 5 rows to 172.16.232.39
| testral              | 172.16.252.28    | 172.16.252.28  | 0.0310 | 0.0150 |
  : deleted *

+----------------+----------------+
| Resolved Addr  | Resolved Count |
+----------------+----------------+
| 172.16.250.3   | 10             |
| 172.16.252.28  | 41             |
| 172.16.232.39  | 49             |

+----------------+----------------+
| Connected To   | Connected Count|
+----------------+----------------+
| 172.16.250.3   | 3              |
| 172.16.252.28  | 41             |
| 172.16.232.39  | 49             |
| ECONNREFUSED   | 7              |
```

Figure 2-42 Socket application server failure case - external name server

After we stopped the application server on MVS03, the client tried to connect to the failed server a few times. DNS on the sysplex continued to return the address of the failed server according to its last communication with WLM, so the external name server passed it on to the client for the defined three seconds. For those three seconds, the client tried and failed to connect. Eventually the sysplex DNS received updated information from WLM, returned new information to the external name server, and the client never saw the address 172.16.250.3 again.

The sysplex name server client is shown in Figure 2-43.

```
 sysplex2 testral 1234 -c 30 -b 3

+----------------------+------------------+----------------+--------+--------+
| Hostname             | Resolved Addr    | Connected Addr | Resolv | Connec |
+----------------------+------------------+----------------+--------+--------+
| testral              | 172.16.252.28    | 172.16.252.28  | 0.0300 | 0.0100 |
| testral              | 172.16.232.39    | 172.16.232.39  | 0.0100 | 0.0200 |
| testral              | 172.16.250.3     | 172.16.250.3   | 0.0100 | 0.0200 |
| testral              | 172.16.252.28    | 172.16.252.28  | 0.0100 | 0.0100 |
| testral              | 172.16.232.39    | 172.16.232.39  | 0.0100 | 0.0200 |
Error on connecting socket to '172.16.250.3': ECONNREFUSED
| testral              | 172.16.252.28    | 172.16.252.28  | 0.0100 | 0.0200 |
| testral              | 172.16.232.39    | 172.16.232.39  | 0.0100 | 0.0200 |
Error on connecting socket to '172.16.250.3': ECONNREFUSED
| testral              | 172.16.252.28    | 172.16.252.28  | 0.0200 | 0.0100 |
| testral              | 172.16.232.39    | 172.16.232.39  | 0      | 0.0100 |
Error on connecting socket to '172.16.250.3': ECONNREFUSED
| testral              | 172.16.252.28    | 172.16.252.28  | 0.0200 | 0.0200 |
| testral              | 172.16.232.39    | 172.16.232.39  | 0.0100 | 0.0200 |
| testral              | 172.16.252.28    | 172.16.252.28  | 0.0100 | 0.0200 |
| testral              | 172.16.232.39    | 172.16.232.39  | 0.0100 | 0.0200 |
| testral              | 172.16.252.28    | 172.16.252.28  | 0.0100 | 0.0100 |
| testral              | 172.16.232.39    | 172.16.232.39  | 0.0100 | 0.0200 |
| testral              | 172.16.252.28    | 172.16.252.28  | 0.0100 | 0.0200 |
  :

+----------------+----------------+
| Resolved Addr  | Resolved Count |
+----------------+----------------+
| 172.16.250.3   | 4              |
| 172.16.252.28  | 13             |
| 172.16.232.39  | 13             |

+----------------+----------------+
| Connected To   | Connected Count|
+----------------+----------------+
| 172.16.250.3   | 1              |
| 172.16.252.28  | 13             |
| 172.16.232.39  | 13             |
| ECONNREFUSED   | 3              |
```

Figure 2-43 Socket application server failure case - sysplex name server

Here a similar thing occurs: DNS returns each server address in turn (they seem to be equally weighted) until the first WLM call after the failure sets the record straight.

3

Dynamic VIPA (for application instance)

By providing IP addresses for the z/OS applications that are not associated with a specific physical network attachment or gateway, a Virtual IP Address (VIPA) enables fault tolerance against outages in the IP interfaces on the z/OS host. However, in earlier releases, if the stack itself failed, you had to move the application workload manually and activate a VIPA on another stack via the VARY OBEY command. Since V2R8, VIPAs can be dynamically activated. Furthermore, a VIPA can now be regarded as the private address of an application server in the sysplex and can follow that server across sysplex images.

CS for z/OS IP has two flavors of Dynamic VIPA: the application-specific Dynamic VIPA (designed for single instance applications) and the automatic VIPA takeover/takeback flavor (designed for sysplex-wide VIPA takeover). The former is the subject of this chapter, while the latter is covered in Chapter 4, “Automatic VIPA takeover and takeback” on page 87.

3.1 Benefits of Dynamic VIPA

In general, the application-specific Dynamic VIPA allows an application to activate a VIPA dynamically. This allows the application instances to have control of when the VIPA is active and in which stack in the sysplex the VIPA is active. The Dynamic VIPA is usually only active in at most one stack in a sysplex. That is, one stack *owns* the VIPA and advertises reachability to that stack (usually via some dynamic routing protocol). We will see some cases in which a VIPA appears active in more than one stack, but it is only advertised by one.

The application-specific Dynamic VIPA allows the VIPA address to be associated with a particular application instance. VIPA activation is performed, without any DEVICE, LINK or HOME definitions, either by the application issuing bind() to that particular IP address, or by an APF-authorized program issuing an IOCTL to the stack or by invoking the MODDVIPA utility. The stack must be configured appropriately to permit activation of such a Dynamic VIPA, with a VIPA subnet range defined to ensure that unwanted IP addresses are not created.

In this method, the failure of the application instance (or stack, or z/OS) means that the application instance could be restarted elsewhere in a sysplex environment. How this restart is accomplished is not determined by the stack function. It could be by operator intervention, Automatic Restart Manager (ARM), or other sysplex-wide mechanism.

The application-specific Dynamic VIPA function allows VIPA IP addresses to be defined and activated by individual applications (with or without modifying the applications), so that the VIPA IP address moves when the application is moved to another z/OS host image in the sysplex environment. In this regard, a VIPA is typically associated with some application. The movement of this Dynamic VIPA is done by the activation of the application instance or of the Dynamic VIPA itself on some other system in the sysplex.

When some application fails in a sysplex environment, this application can be restarted on another stack. If correctly defined, the application can bind() to the same IP address without any intervention. The VIPA will be activated dynamically at the second stack.

Application-specific Dynamic VIPAs are intended for applications for which only one instance of the application can be running simultaneously. If a second stack activates the VIPA at the same time that a first stack has the VIPA active, the resulting behavior may be confusing. Because of this, it is important to have a good understanding of the behavior associated with multiple Dynamic VIPA activations. With CS for OS/390 V2R8 IP, this behavior was considered disruptive. CS for OS/390 V2R10 IP alleviated this disruptiveness by allowing a smooth transition in VIPA ownership.

3.2 Implementing Dynamic VIPA

It is important to note that the application-specific Dynamic VIPA is defined exclusively with the VIPARange statement within the VIPADynamic block in the TCP/IP profile. The VIPADynamic block has other statements as shown in Figure 3-1 on page 63.

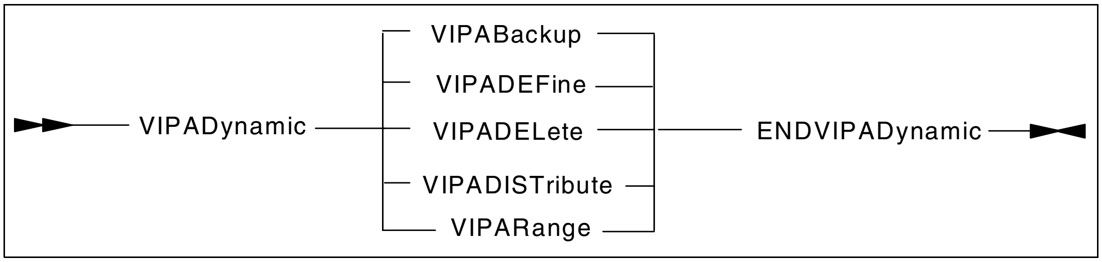


Figure 3-1 Definition format for Dynamic VIPA statement

VIPADEFine, VIPABackup, VIPADELete, and VIPADISTribute statements are used in conjunction with the automatic takeover flavor of VIPA as discussed in Chapter 4, “Automatic VIPA takeover and takeback” on page 87. The VIPADEFine statement designates one or more VIPA IP addresses that a stack should initially own. VIPADELete is used to delete one of these defined VIPA addresses. The VIPADISTribute statement is used to configure the Sysplex Distributor and is the subject of Chapter 5, “Sysplex Distributor” on page 109.

The VIPARange statement is used to define and delete IP subnets from which an application can activate a Dynamic VIPA by issuing a bind() or IOCTL. That is, if an application expects to activate a VIPA dynamically, it must be contained within some subnet specification of an active VIPARange statement.

3.2.1 Dynamic VIPA configuration (for application instance)

Activation of an application-specific Dynamic VIPA IP address associated with a specific application instance occurs only via an application program's API call, in either of the following ways:

- An application issues a bind() to that particular (specific) IP address.
- An application binds to INADDR_ANY instead of a specific IP address, but the Server Bind Control function changes the generic bind() to a specific one. This situation is discussed in 3.2.2, “Solutions for applications that bind() to INADDR_ANY” on page 64.
- An authorized application issues the SIOCSVIPA IOCTL() command. An example of such an application is the MODDVIPA utility.

Since the VIPA IP address is specified by the application, it need not be defined in the TCP/IP profile. However, we must ensure that the addresses being used by the application correspond to our IP addressing scheme. We use the VIPARange statement in the TCP/IP profile to indicate the range of VIPAs that we are willing to dynamically activate as shown in Figure 3-2.

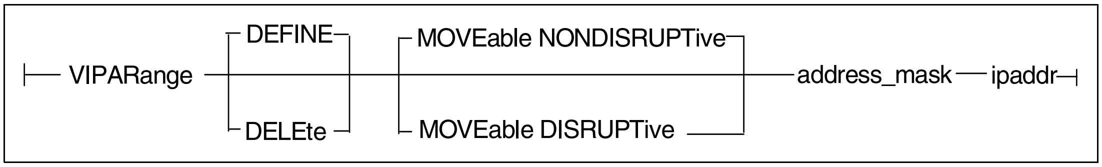


Figure 3-2 Definition format for VIPARange

The VIPARange statement defines an IP subnetwork using the network address (prefix) and the subnet mask. Since the same VIPA address may not be activated by IOCTL/bind() while also participating in automatic takeover as defined by VIPADEFine/VIPABackup, it is recommended that subnets for VIPADEFine be different from subnets for VIPARange.

VIPARANGE in itself does not create or activate or reserve any Dynamic VIPAs. It merely defines an allocated range within which program action (or the BIND parameter or a PORT statement) may cause a Dynamic VIPA to be created and activated, for a specific IP address. Also, the same range may have DVIPAs defined via VIPADEFINE or VIPABACKUP, as long as no attempt is made to use the same IP address for both VIPADEFINE and activation via program BIND.

For our tests in 3.2.3, “Examples of Dynamic VIPA” on page 65, we use the configuration in Figure 3-3.

```
VIPADYNAMIC
   VIPARANGE  DEFINE MOVEABLE NONDISRUPT 255.255.255.0 172.16.240.193
ENDVIPADYNAMIC
```

Figure 3-3 VIPARange statement

Once activated on a stack via bind() or IOCTL, a Dynamic VIPA IP address remains active unless the VIPA IP address is moved to another stack or it is deleted. The system operator may delete an active application-specific Dynamic VIPA IP address by using the MODDVIPA utility or by stopping the application that issued the bind() to activate the VIPA. To remove an active VIPARange statement, the VIPARange DELETE statement may be used.

Deleting a VIPARANGE does not affect existing DVIPAs that are already activated within the range, but will simply ensure that new ones may not be activated within the deleted range until the VIPARANGE is reinstated.

3.2.2 Solutions for applications that bind() to INADDR_ANY

An application may issue a bind() to INADDR_ANY to accept connection requests from any IP address associated with the stack. In that case, if the application is to be associated with a specific VIPA address, it is not possible to determine which VIPA IP address it should be associated with. There are three ways to get around this situation:

- Define the application to bind() to a specific address instead of INADDR_ANY using the new function Server Bind Control, implemented by the BIND keyword on the PORT statement. Note that the port must be known ahead of time, since this will be coded in the TCPIP profile.
- Modify the application to bind() to a specific address (however, sometimes this is not possible without source code modifications).
- Use the utility MODDVIPA or change the application to send the appropriate IOCTL.

The most attractive solution is to use the new Server Bind Control function because it does not require changing the application or the use of a manual utility. Using this function, a generic server (such as the TN3270 Server) will bind to a specific address instead of INADDR_ANY. When the application binds to INADDR_ANY, the bind() is intercepted and converted to the specified IP address. The process then continues as if the server had issued a bind() to that specific address. If the application does not support the ability to specify a specific local address to which to bind, the Server Bind Control function provides an attractive alternative to changing application source code. In order to use this function, however, the port used by the application must be known in advance so that it can be added to the PORT statement in the TCPIP profile.

If the Server Bind Control function cannot be used and the application can be modified, change the target address for the bind() from INADDR_ANY to the specific Dynamic VIPA IP address. In “SOCSRVR, a simple socket server program” on page 279, we show how we did this for the sample sockets application used in many of our tests.

To address the case in which the application cannot take advantage of the Server Bind Control function and it cannot be modified, CS for z/OS IP provides a utility MODDVIPA to create a Dynamic VIPA IP address using the IOCTL call. The utility can be initiated via JCL, from the OMVS command line, or from a shell script. MODDVIPA is the name of utility EZBXFDVP, which was available in CS for OS/390 IP V2R8. The name EZBXFDVP can still be used to run the utility, but this name is not mentioned in CS for OS/390 IP V2R10 (or later release) documentation anymore. See “Using MODDVIPA utility” on page 65 for an example.

3.2.3 Examples of Dynamic VIPA

In this section, we give examples of Dynamic VIPAs activated in three different ways:

- Using the MODDVIPA utility
- Server issues a bind() to INADDR_ANY which is converted to a specific address via the Server Bind Control function
- Application issues a bind() to a specific address directly

Using MODDVIPA utility

Figure 3-4 shows our sample procedure to invoke the utility.

```
//TCPDVP   PROC
//TCPDVP   EXEC PGM=MODDVIPA,REGION=0K,TIME=1440,
//   PARM='-p TCPIPC -c 172.16.240.193'
//SYSPRINT DD SYSOUT=A,DCB=(RECFM=FB,LRECL=132,BLKSIZE=132)
//SYSERR   DD SYSOUT=A
//SYSERROR DD SYSOUT=A
//SYSDEBUG DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSABEND DD SYSOUT=A
```

Figure 3-4 Sample JCL to run MODDVIPA

The utility expects a parameter specifying the VIPA IP address to be activated. It may also be used to delete a VIPA IP address as an alternative to the VARY OBEY command by a system operator. The parameter option field can be `-c` for create or `-d` for delete. The example above will create a Dynamic VIPA with IP address 172.16.240.193. Activation of the Dynamic VIPA IP address will succeed as long as the desired IP address is not claimed by any other stack, is not an IP address of a physical interface or a static VIPA, and is not defined via VIPADEFine or VIPABackup in a VIPADynamic block.

The following completion codes are expected when creating (`-c`) a DVIPA IP address:

0 Success: The DVIPA was activated.

4 Warning: The required DVIPA was not activated because the specified IP is already active on this stack.

8 Error: The IP address is not defined as a DVIPA on this TCP/IP

The following completion codes are expected when deleting (`-d`) a DVIPA IP address:

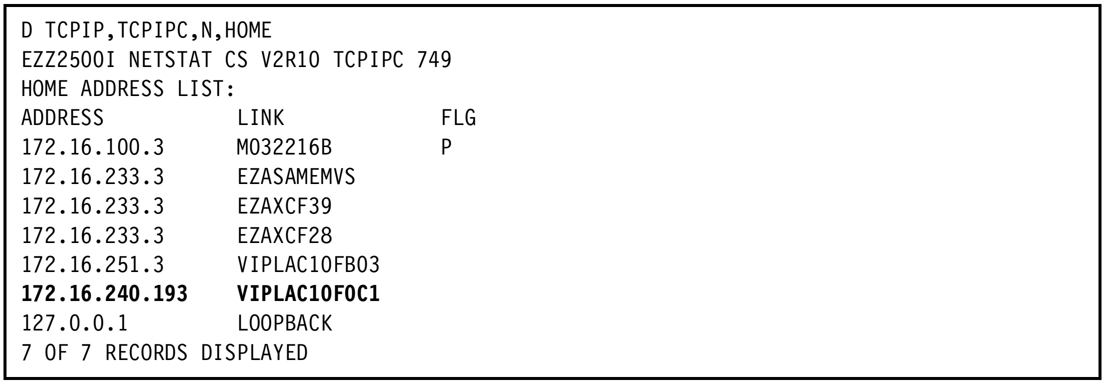
0 Success: The DVIPA was deleted.

8 The requested DVIPA was not deleted.

Note that the issuer of this utility must be APF authorized and have root authority. If the user is not APF authorized, the following message is issued:

```
SIOCSVIPA IOCTL failed: EDC5111I Permission denied.  errno2=74057209
```

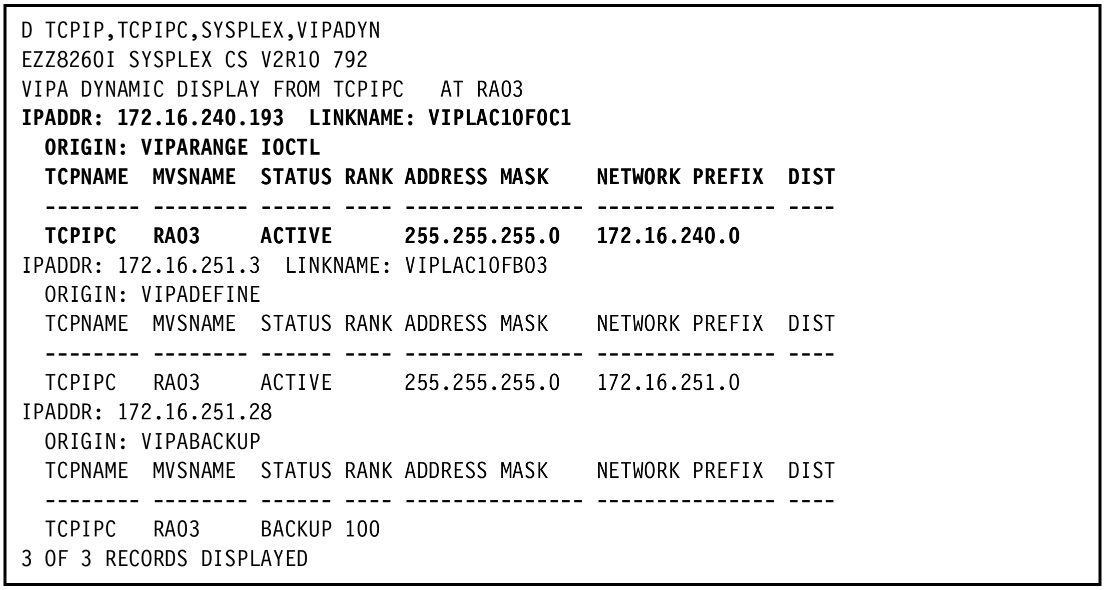
After authorizing the user, the job was executed again. Figure 3-5 shows that the DVIPA IP address 172.16.240.193 was added to this stack.



```
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 749
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.100.3     M032216B         P
172.16.233.3     EZASAMEMVS
172.16.233.3     EZAXCF39
172.16.233.3     EZAXCF28
172.16.251.3     VIPLAC10FB03
172.16.240.193   VIPLAC10F0C1
127.0.0.1        LOOPBACK
7 OF 7 RECORDS DISPLAYED
```

Figure 3-5 Display NETSTAT,HOME

Figure 3-6 shows the result of SYSPLEX,VIPADYN and how this address was activated (IOCTL).



```
D TCPIP,TCPIPC,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V2R10 792
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.240.193  LINKNAME: VIPLAC10F0C1
  ORIGIN: VIPARANGE IOCTL
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.240.0
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.251.0
IPADDR: 172.16.251.28
  ORIGIN: VIPABACKUP
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     BACKUP 100
3 OF 3 RECORDS DISPLAYED
```

Figure 3-6 Display SYSPLEX,VIPADYN

Using Server Bind Control

Some servers can only bind to INADDR_ANY (0.0.0.0). For these servers, we can use the Server Bind Control function to convert the bind to INADDR_ANY to a bind to a specific IP address as defined in the PORT statement. Figure 3-7 on page 67 shows the TCP/IP profile including the port reservation statement for port 23, the Telnet well-known port.

```
PORT
  7   UDP MISCSERV
  7   TCP MISCSERV
  9   UDP MISCSERV
  9   TCP MISCSERV
  19  UDP MISCSERV
  19  TCP MISCSERV
  20  TCP OMVS      NOAUTOLOG ; FTP SERVER
  21  TCP FTPDC1              ; FTP SERVER
  23 TCP INTCLIEN BIND 172.16.240.193  ;
  23 TCP INETD1   BIND 9.24.105.74  ;
```

Figure 3-7 Port definition

Refer to *z/OS V1R2.0 CS: IP Configuration Reference,* SC31-8776 for a complete description of the PORT statement.

We established a TN3270 connection to the DVIPA IP address and issued some display commands. Figure 3-8 shows that the DVIPA IP address 172.16.240.193 was added to stack TCPIPC.

```
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 580
HOME ADDRESS LIST:
ADDRESS           LINK             FLG
172.16.100.3      M032216B         P
172.16.233.3      EZASAMEMVS
172.16.233.3      EZAXCF28
172.16.233.3      EZAXCF39
172.16.251.3      VIPLAC10FB03
172.16.240.193    VIPLAC10F0C1
127.0.0.1         LOOPBACK
7 OF 7 RECORDS DISPLAYED
```

Figure 3-8 Display NETSTAT,HOME

Figure 3-9 on page 68 shows the result of SYSPLEX,VIPADYN and how this IP address was activated (bind).

```
D TCPIP,TCPIPC,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V2R10 662
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.240.193  LINKNAME: VIPLAC10F0C1
  ORIGIN: VIPARANGE BIND
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.240.0
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.251.0
IPADDR: 172.16.251.28
  ORIGIN: VIPABACKUP
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     BACKUP 100
3 OF 3 RECORDS DISPLAYED
```

Figure 3-9 Display SYSPLEX,NETSTAT

Our example shows how this function can allow multiple servers to bind to the same port on different interfaces. Specifically, Telnet 3270 server and OE Telnet server can both listen on the same port 23 simultaneously.

Application issues a bind() to a specific address

In this test, we used an application that binds a specific address and give some displays. The application receives the port number and the IP address as parameters to use on the bind() call. Figure 3-10 shows the invocation of the tool.

```
RA03:/u/claudia>jcs -s 1/0/0/0 -b 4343/172.16.240.193/0/0/0 -l -d 200
```

Figure 3-10 Application jcs

With this invocation, the application binds to port 4343 and the specific IP address 172.16.240.193. Figure 3-11 shows that the Dynamic VIPA IP address was activated on TCPIPC stack.

```
RO RA03,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 248
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.100.3     M032216B         P
172.16.233.3     EZASAMEMVS
172.16.233.3     EZAXCF28
172.16.233.3     EZAXCF39
172.16.251.3     VIPLAC10FB03
172.16.240.193   VIPLAC10F0C1
127.0.0.1        LOOPBACK
7 OF 7 RECORDS DISPLAYED
```

Figure 3-11 Display NETSTAT,HOME

Figure 3-12 shows how the Dynamic VIPA IP address was activated (bind).

```
RO RA03,D TCPIP,TCPIPC,SYSPLEX,VIPAD
D TCPIP,TCPIPC,SYSPLEX,VIPAD
EZZ8260I SYSPLEX CS V2R10 252
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.240.193  LINKNAME: VIPLAC10F0C1
  ORIGIN: VIPARANGE BIND
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.240.0
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.251.0
IPADDR: 172.16.251.28
  ORIGIN: VIPABACKUP
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     BACKUP 100
3 OF 3 RECORDS DISPLAYED
```

Figure 3-12 Display SYSPLEX,VIPAD

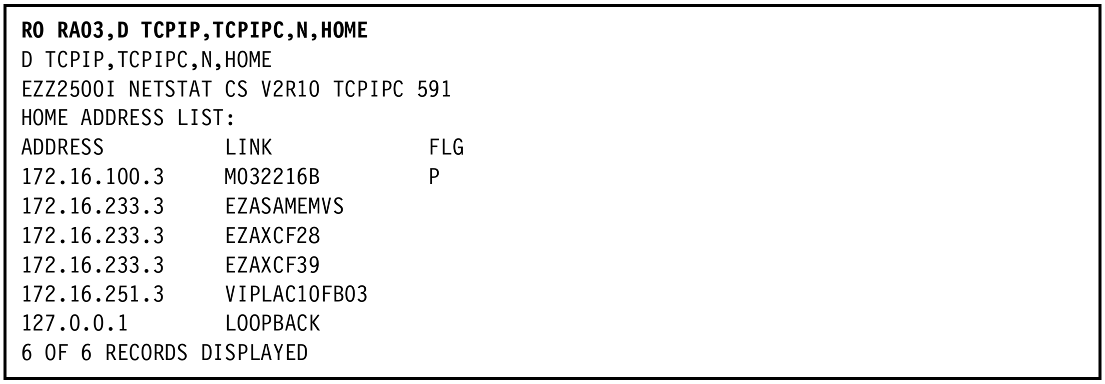
Figure 3-13 shows the application using port 4343 and the Dynamic VIPA IP address added.

```
RO RA03,D TCPIP,TCPIPC,N,CONN
D TCPIP,TCPIPC,N,CONN
EZZ2500I NETSTAT CS V2R10 TCPIPC 258
USER ID  CONN     LOCAL SOCKET         FOREIGN SOCKET    STATE
BPXOINIT 0000000B 0.0.0.0..10007       0.0.0.0..0        LISTEN
CLAUDIA4 0000056A 172.16.240.193..4343 0.0.0.0..0        LISTEN
FTPDC1   00000011 0.0.0.0..21          0.0.0.0..0        LISTEN
OMPROUTC 0000001A 127.0.0.1..1026      127.0.0.1..1027   ESTBLSH
TCPIPC   00000018 172.16.240.193..23   0.0.0.0..0        LISTEN
```

Figure 3-13 Display NETSTAT,CONN

In this case, the Dynamic VIPA IP address is deleted when the application finishes.

Figure 3-14 on page 70 shows that the Dynamic VIPA IP address is not active anymore.



```
RO RA03,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 591
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.100.3     M032216B         P
172.16.233.3     EZASAMEMVS
172.16.233.3     EZAXCF28
172.16.233.3     EZAXCF39
172.16.251.3     VIPLAC10FB03
127.0.0.1        LOOPBACK
6 OF 6 RECORDS DISPLAYED
```

Figure 3-14 Display NETSTAT,HOME after the application ends

3.3 Dynamic VIPA conflicts

TCP/IP stacks have a mechanism for preventing conflicts when the same Dynamic VIPA IP address is activated in more than one stack. Sometimes a conflict can occur as a result of changes in the sysplex environment. For example, a DVIPA could be activated due to a stack or application failure. To help with these conflicts, only one TCP/IP stack advertises a Dynamic VIPA IP address to routers. The stack that receives packets destined for the VIPA is considered its *owning stack*.

The next section explains the behavior of a Dynamic VIPA IP address that is activated and another application tries to activate the same Dynamic VIPA IP address in another IP stack. As of CS for OS/390 V2R10 IP, the resulting behavior depends on the VIPARange definition in the first stack.

3.3.1 bind()

Every time an application issues a bind() to a specific IP address, this IP address is checked against IP addresses in the HOME list. If the IP address is already active, the bind() is successful. If the IP address is not active on this stack, the VIPARange statement is checked. If there is no VIPARange statement corresponding to the address requested in an application call, the call is rejected.

Otherwise, if this IP address is already active in another stack, the behavior will be established by the options MOVEable DISRUPTive and MOVEable NONDISRUPTive in the VIPARange statement corresponding to this VIPA.

The DVIPA IP address is immediately moved to the second stack if the VIPARange is defined as MOVEable NONDISRUPTive. The DVIPA IP address is added to the HOME list of a new stack and this stack notifies neighboring routers that it is now the owner of this IP address. New connections will be directed to the new owning stack and outstanding connections to the previously owning stack will remain active and functional, despite its having lost the ownership of the DVIPA. The new owning stack routes the data for these existing connections to the old stack as shown in Figure 3-15 on page 71. In this regard, the movement of the DVIPA is *nondisruptive*. The DVIPA is eventually deleted when the application closes its DVIPA owning socket.

Note: Bringing up a third application before the first has closed its socket will lead to errors because there cannot be two stacks with the same DVIPA in MOVING status.

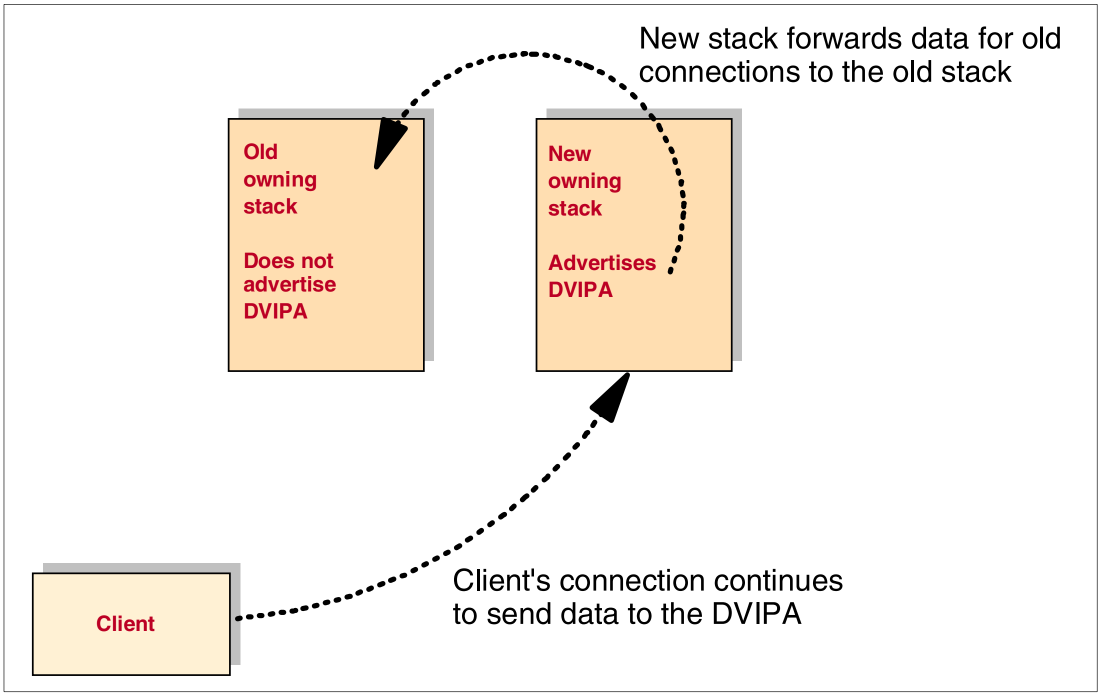

New stack forwards data for old connections to the old stack
Old owning stack
Does not advertise DVIPA
New owning stack
Advertises DVIPA
Client
Client's connection continues to send data to the DVIPA

Figure 3-15 With nondisruptive behavior, the new owning stack forwards data for old connections

If the VIPARange is defined as MOVEable DISRUPTive, the VIPA is not moved and the bind() request for the application on the second stack fails. In this case, the second application issuing the bind() is said to have been *disrupted*. This is the only behavior allowed with CS for OS/390 V2R8 IP. NONDISRUPTive is the default behavior for CS for z/OS V1R2 IP. Both stacks should be at a V2R10 or later code level or the behavior will be DISRUPTive.

3.3.2 IOCTL

The TCP/IP configuration for the IOCTL() call is the same as for the bind(specific) call, namely a VIPARange defining a subnet containing the desired VIPA IP addresses. The same VIPARange may be used for both if desired. However, the behavior when the IP address is already active in another stack is different.

The DVIPA IP address is immediately transferred to the second stack in both cases (MOVEable DISRUPTive and NONDISRUPTive).

When VIPARange is defined as NONDISRUPTive, the routers are notified about the ownership change and the old connections are preserved on the old stack. The new stack routes the old connections data to the old stack. The status for the DVIPA IP address on the old stack will be *moving* until the existing connections with the old stack terminate.

When VIPARange is defined as DISRUPTive, the routers are notified about the new ownership and the DVIPA IP address is deleted from the HOME list on the first stack. All existing connections on the first stack will be broken.

3.3.3 Scenarios

In this section we show four different scenarios:

- Dynamic VIPA IP address activated via IOCTL and VIPARange defined as MOVEable NONDISRUPTive
- Dynamic VIPA IP address activated via IOCTL and VIPARange defined as MOVEable DISRUPTive
- Dynamic VIPA IP address activated via bind() and VIPARange defined as MOVEable NONDISRUPTive
- Dynamic VIPA IP address activated via bind() and VIPARange defined as MOVEable DISRUPTive

In our tests, the environment shown in Figure 3-16 was used.

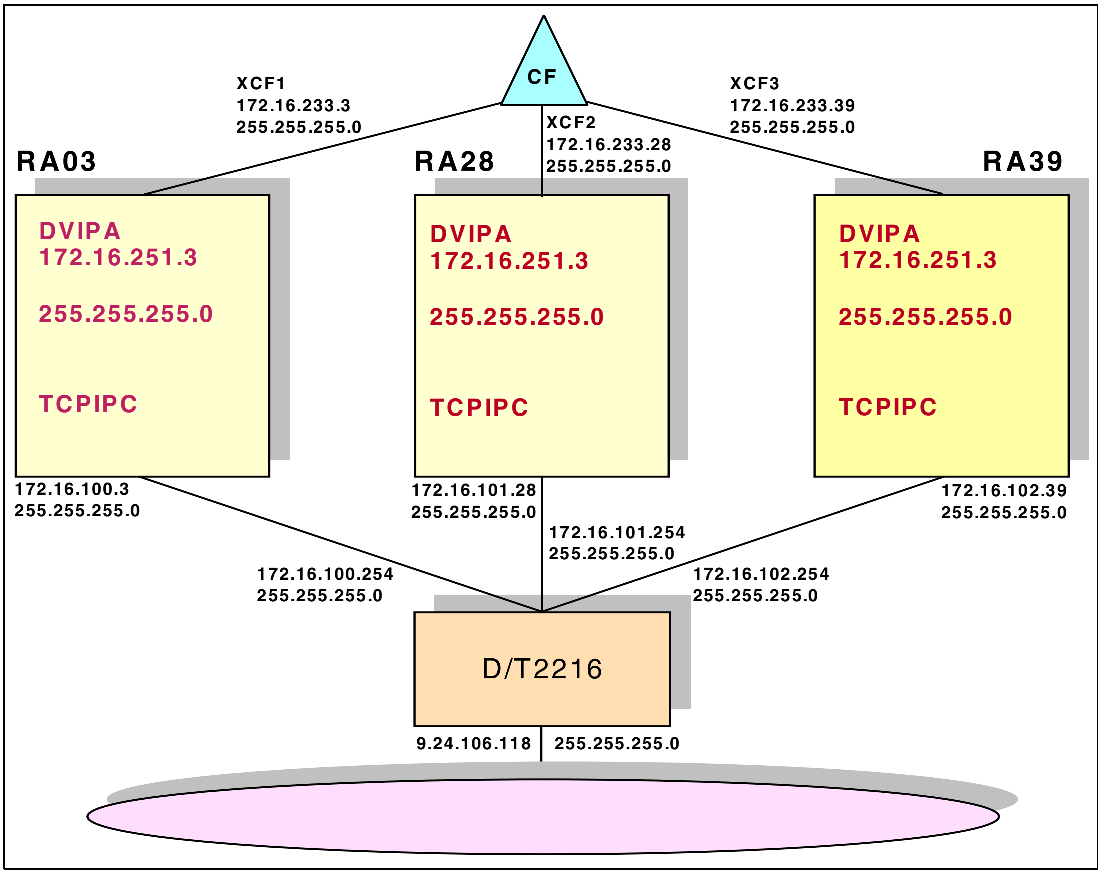


Figure 3-16 Test environment

For convenience, stack TCPIPC on RA39 was not used.

IOCTL and VIPARange defined as MOVEable NONDISRUPTive

In this test, we followed this sequence of actions:

1. Define Dynamic VIPA IP address in VIPARange as shown in Figure 3-3 on page 64 on TCPIPC on RA03.
2. Define Dynamic VIPA IP address in VIPARange as shown in Figure 3-3 on page 64 on TCPIPC on RA28.

3. Run MODDVIPA to activate Dynamic VIPA IP address 172.16.240.193 on TCPIPC on RA03.
4. Verify the Dynamic VIPA IP address 172.16.240.193 is active on TCPIPC on RA03. See Figure 3-17 and Figure 3-18.

```
RO RA03,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 768
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.100.3     M032216B         P
172.16.233.3     EZASAMEMVS
172.16.233.3     EZAXCF39
172.16.233.3     EZAXCF28
172.16.251.3     VIPLAC10FB03
172.16.240.193   VIPLAC10F0C1
127.0.0.1        LOOPBACK
7 OF 7 RECORDS DISPLAYED
```

Figure 3-17 Display NETSTAT,HOME on TCPIPC on RA03

```
D TCPIP,TCPIPC,SYSPLEX,VIPAD
EZZ8260I SYSPLEX CS V2R10 770
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.240.193  LINKNAME: VIPLAC10F0C1
  ORIGIN: VIPARANGE IOCTL
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.240.0
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.251.0
  TCPIPC   RA28     BACKUP 200
IPADDR: 172.16.251.28
  ORIGIN: VIPABACKUP
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA28     ACTIVE      255.255.255.0   172.16.251.0
  TCPIPC   RA03     BACKUP 100
5 OF 5 RECORDS DISPLAYED
```

Figure 3-18 Display SYSPLEX,VIPAD

5. Start an FTP server in both stacks.
6. Establish a connection to an FTP server using Dynamic VIPA 172.16.240.193 (activated on TCPIPC on RA03). The resulting connections are shown in Figure 3-19 on page 74.
7. Start transfer of a big file from a client to the server (TCPIPC on RA03).

```
RO RA03,D TCPIP,TCPIPC,N,CONN
D TCPIP,TCPIPC,N,CONN
EZZ2500I NETSTAT CS V2R10 TCPIPC 841
USER ID  CONN     LOCAL SOCKET           FOREIGN SOCKET         STATE
BPXOINIT 00002F15 0.0.0.0..10007         0.0.0.0..0             LISTEN
CLAUDIA  000044B6 172.16.240.193..20     9.24.104.75..3356      ESTBLSH
FTPDC1   000044B1 172.16.240.193..21     9.24.104.75..3355      ESTBLSH
FTPDC1   00000011 0.0.0.0..21            0.0.0.0..0             LISTEN
```

Figure 3-19 Display NETSTAT,CONN

8. Run MODDVIPA to activate Dynamic VIPA 172.16.240.193 on TCPIPC on RA28. Figure 3-20 shows the messages on the console log (the first one is issued on RA28 and the second one on RA03) when the Dynamic VIPA IP address is moved.

```
EZZ8302I VIPA 172.16.240.193 TAKEN FROM TCPIPC ON RA03
EZZ8303I VIPA 172.16.240.193 GIVEN TO TCPIPC ON RA28
```

Figure 3-20 Console log

9. Verify the Dynamic VIPA 172.16.240.193 is active on TCPIPC on RA28. This is shown in the displays in Figure 3-21 and Figure 3-22 on page 75. Notice the status of `MOVING` associated with the VIPA during this process.

```
RO RA28,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 820
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.101.28    M282216B         P
172.16.233.28    EZASAMEMVS
172.16.233.28    EZAXCF39
172.16.233.28    EZAXCF03
172.16.251.28    VIPLAC10FB1C
172.16.240.193   VIPLAC10F0C1
127.0.0.1        LOOPBACK
7 OF 7 RECORDS DISPLAYED
```

Figure 3-21 Display NETSTAT,HOME

```
RO RA03,D TCPIP,TCPIPC,SYSPLEX,VIPAD
D TCPIP,TCPIPC,SYSPLEX,VIPAD
EZZ8260I SYSPLEX CS V2R10 873
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.240.193
  ORIGIN: VIPARANGE IOCTL
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA28     ACTIVE      255.255.255.0   172.16.240.0
  TCPIPC   RA03     MOVING      255.255.255.0   0.0.0.0
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.251.0
  TCPIPC   RA28     BACKUP 200
IPADDR: 172.16.251.28
  ORIGIN: VIPABACKUP
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA28     ACTIVE      255.255.255.0   172.16.251.0
  TCPIPC   RA03     BACKUP 100
6 OF 6 RECORDS DISPLAYED
```

Figure 3-22 Display SYSPLEX,VIPAD

10. Check if the FTP connections are still active on TCPIPC on RA03. Figure 3-23 shows that they are.

```
RO RA03,D TCPIP,TCPIPC,N,CONN
D TCPIP,TCPIPC,N,CONN
EZZ2500I NETSTAT CS V2R10 TCPIPC 844
USER ID  CONN     LOCAL SOCKET           FOREIGN SOCKET         STATE
BPXOINIT 00002F15 0.0.0.0..10007         0.0.0.0..0             LISTEN
CLAUDIA  000044B6 172.16.240.193..20     9.24.104.75..3356      ESTBLSH
FTPDC1   000044B1 172.16.240.193..21     9.24.104.75..3355      ESTBLSH
FTPDC1   00000011 0.0.0.0..21            0.0.0.0..0             LISTEN
```

Figure 3-23 Display NETSTAT,CONN

The FTP protocol makes use of two TCP ports, 21 for the control connection and 20 for the data connections. When the session is established, only port 21 is allocated. As data needs to be transferred, connections from port 20 are created by the server. If the Dynamic VIPA IP address is moved before the port 20 connection is created, the information regarding that particular connection cannot be moved (since it does not yet exist). As a result, when any additional data is to be transferred, the data connection will not be established because the second stack does not know to where the information should be routed. In this case, the connection is hung. Figure 3-24 on page 76 shows an example of this case, with a status of `SYNSENT` for a data connection. Because of this type of problem, it is very important to keep in mind the movement of a DVIPA, particularly when using FTP.

```
RO RA03,D TCPIP,TCPIPC,N,CONN
D TCPIP,TCPIPC,N,CONN
EZZ2500I NETSTAT CS V2R10 TCPIPC 088
USER ID  CONN     LOCAL SOCKET           FOREIGN SOCKET         STATE
BPXOINIT 00002F15 0.0.0.0..10007         0.0.0.0..0             LISTEN
CLAUDIA  00004B5C 172.16.240.193..20     9.24.106.64..1037      SYNSENT
FTPDC1   00000011 0.0.0.0..21            0.0.0.0..0             LISTEN
FTPDC1   00004B55 172.16.240.193..21     9.24.106.64..1035      ESTBLSH
```

Figure 3-24 Display NETSTAT,CONN

IOCTL and VIPARange defined as MOVEable DISRUPTive

In this test, we followed this sequence of actions:

1. Define Dynamic VIPA in VIPARange as shown in Figure 3-25 on TCPIPC on RA03.
2. Define Dynamic VIPA in VIPARange as shown in Figure 3-25 on TCPIPC on RA28.

```
VIPADYNAMIC
 VIPARANGE DEFINE MOVEABLE DISRUPT 255.255.255.0 172.16.240.193
ENDVIPADYNAMIC
```

Figure 3-25 Definition VIPARange MOVEable DISRUPTive

3. Run MODDVIPA to activate Dynamic VIPA 172.16.240.193 on TCPIPC on RA03.
4. Verify the Dynamic VIPA 172.16.240.193 is active on TCPIPC on RA03. This is illustrated in the displays in Figure 3-26 and Figure 3-27 on page 77.

```
RO RA03,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 718
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.100.3     M032216B         P
172.16.233.3     EZASAMEMVS
172.16.233.3     EZAXCF39
172.16.233.3     EZAXCF28
172.16.251.3     VIPLAC10FB03
172.16.240.193   VIPLAC10F0C1
127.0.0.1        LOOPBACK
7 OF 7 RECORDS DISPLAYED
```

Figure 3-26 Display NETSTAT,HOME on TCPIPC on RA03

```
D TCPIP,TCPIPC,SYSPLEX,VIPAD
EZZ8260I SYSPLEX CS V2R10 720
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.240.193  LINKNAME: VIPLAC10F0C1
  ORIGIN: VIPARANGE IOCTL
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.240.0
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.251.0
  TCPIPC   RA28     BACKUP 200
IPADDR: 172.16.251.28
  ORIGIN: VIPABACKUP
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA28     ACTIVE      255.255.255.0   172.16.251.0
  TCPIPC   RA03     BACKUP 100
5 OF 5 RECORDS DISPLAYED
```

Figure 3-27 Display SYSPLEX,VIPAD

5. Start the FTP server in both stacks.
6. Establish a connection to the FTP server using Dynamic VIPA 172.16.240.193 (activated on TCPIP on RA03). The resulting connection displays are shown in Figure 3-28.
7. Start transfer of a file from the client to the server (TCPIPC on RA03).

```
RO RA03,D TCPIP,TCPIPC,N,CONN
D TCPIP,TCPIPC,N,CONN
EZZ2500I NETSTAT CS V2R10 TCPIPC 758
USER ID  CONN     LOCAL SOCKET           FOREIGN SOCKET         STATE
BPXOINIT 0000000B 0.0.0.0..10007         0.0.0.0..0             LISTEN
CLAUDIA  00000067 172.16.240.193..20     9.24.104.75..3358      ESTBLSH
FTPDC1   00000011 0.0.0.0..21            0.0.0.0..0             LISTEN
FTPDC1   00000065 172.16.240.193..21     9.24.104.75..3357      ESTBLSH
OMPROUTC 0000001C 127.0.0.1..1027        127.0.0.1..1028        ESTBLSH
TCPIPC   00000014 127.0.0.1..1025        127.0.0.1..1026        ESTBLSH
```

Figure 3-28 Display NETSTAT,CONN

8. Run MODDVIPA to activate Dynamic VIPA 172.16.240.193 on TCPIPC on RA28. Figure 3-29 shows the messages on the console log (the first one is issued on RA28 and the second one on RA03) when the Dynamic VIPA IP address is moved.

```
EZZ8302I VIPA 172.16.240.193 TAKEN FROM TCPIPC ON RA03
EZZ8304I VIPA 172.16.240.193 SURRENDERED TO TCPIPC ON RA28
```

Figure 3-29 Console log

9. Verify the Dynamic VIPA 172.16.240.193 was deleted from stack TCPIPC on RA03 (see Figure 3-30 on page 78) and activated in stack TCPIPC on RA28 (see Figure 3-31 on page 78).

```
RO RA03,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 831
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.100.3     M032216B         P
172.16.233.3     EZASAMEMVS
172.16.233.3     EZAXCF39
172.16.233.3     EZAXCF28
172.16.251.3     VIPLAC10FB03
127.0.0.1        LOOPBACK
6 OF 6 RECORDS DISPLAYED
```

Figure 3-30 Display NETSTAT,HOME on RA03

```
RO RA28,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 496
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.101.28    M282216B         P
9.24.104.34      LOOPBACK
172.16.233.28    EZASAMEMVS
172.16.233.28    EZAXCF39
172.16.233.28    EZAXCF03
172.16.251.28    VIPLAC10FB1C
172.16.240.193   VIPLAC10F0C1
127.0.0.1        LOOPBACK
8 OF 8 RECORDS DISPLAYED
```

Figure 3-31 Display NETSTAT,HOME on RA28

10. Verify the status of Dynamic VIPA IP address on the sysplex as shown in Figure 3-32 on page 79.

```
RO RA03,D TCPIP,TCPIPC,SYSPLEX,VIPAD
D TCPIP,TCPIPC,SYSPLEX,VIPAD
EZZ8260I SYSPLEX CS V2R10 838
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.240.193
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA28     ACTIVE      255.255.255.0   172.16.240.0
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.251.0
  TCPIPC   RA28     BACKUP 200
IPADDR: 172.16.251.28
  ORIGIN: VIPABACKUP
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA28     ACTIVE      255.255.255.0   172.16.251.0
  TCPIPC   RA03     BACKUP 100
5 OF 5 RECORDS DISPLAYED
```

Figure 3-32 Display SYSPLEX,VIPAD

11. Check that the FTP connection is not active on TCPIPC on RA03 anymore. The connection was broken as can be seen in Figure 3-33.

```
RO RA03,D TCPIP,TCPIPC,N,CONN
D TCPIP,TCPIPC,N,CONN
EZZ2500I NETSTAT CS V2R10 TCPIPC 146
USER ID  CONN     LOCAL SOCKET           FOREIGN SOCKET         STATE
BPXOINIT 0000000B 0.0.0.0..10007         0.0.0.0..0             LISTEN
FTPDC1   00000011 0.0.0.0..21            0.0.0.0..0             LISTEN
OMPROUTC 0000001C 127.0.0.1..1027        127.0.0.1..1028        ESTBLSH
TCPIPC   00000015 0.0.0.0..23            0.0.0.0..0             LISTEN
```

Figure 3-33 Display NETSTAT,CONN

bind() and VIPARange defined as MOVEable NONDISRUPTive

In this test, we performed the following sequence of actions:

1. Define Dynamic VIPA in VIPARange as shown in Figure 3-3 on page 64 on TCPIPC on RA03.
2. Define Dynamic VIPA in VIPARange as shown in Figure 3-3 on page 64 on TCPIPC on RA28.
3. Start application to bind Dynamic VIPA 172.16.240.193 on RA03.
4. Verify the Dynamic VIPA 172.16.240.193 is active on TCPIPC on RA03. Figure 3-34 on page 80 and Figure 3-35 on page 80 show the Dynamic VIPA's active state.

```
RO RA03,D TCPIP,TCPIPC,SYSPLEX,VIPAD
D TCPIP,TCPIPC,SYSPLEX,VIPAD
EZZ8260I SYSPLEX CS V2R10 662
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.240.193  LINKNAME: VIPLAC10F0C1
  ORIGIN: VIPARANGE BIND
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.240.0
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.251.0
  TCPIPC   RA28     BACKUP 200
IPADDR: 172.16.251.28
  ORIGIN: VIPABACKUP
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA28     ACTIVE      255.255.255.0   172.16.251.0
  TCPIPC   RA03     BACKUP 100
```

Figure 3-34 Display SYSPLEX,VIPAD

```
RO RA03,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 664
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.100.3     M032216B         P
9.24.105.76      EN103
172.16.233.3     EZASAMEMVS
172.16.233.3     EZAXCF39
172.16.233.3     EZAXCF28
172.16.251.3     VIPLAC10FB03
172.16.240.193   VIPLAC10F0C1
127.0.0.1        LOOPBACK
8 OF 8 RECORDS DISPLAYED
```

Figure 3-35 Display NETSTAT,HOME

5. Establish a connection to Dynamic VIPA 172.16.240.193 (TN3270). This connection can be seen in Figure 3-36.

```
RO RA03,D TCPIP,TCPIPC,N,CONN
D TCPIP,TCPIPC,N,CONN
EZZ2500I NETSTAT CS V2R10 TCPIPC 674
USER ID  CONN     LOCAL SOCKET           FOREIGN SOCKET         STATE
BPXOINIT 0000000B 0.0.0.0..10007         0.0.0.0..0             LISTEN
CLAUDIA5 00000053 172.16.240.193..1500   0.0.0.0..0             LISTEN
FTPDC1   00000011 0.0.0.0..21            0.0.0.0..0             LISTEN
OMPROUTC 00000019 127.0.0.1..1027        127.0.0.1..1028        ESTBLSH
TCPIPC   00000016 0.0.0.0..23            0.0.0.0..0             LISTEN
TCPIPC   0000005A 172.16.240.193..23     9.24.106.64..1126      ESTBLSH
```

Figure 3-36 Display NETSTAT,CONN

6. Start the same application to bind to Dynamic VIPA 172.16.240.193 on RA28. Figure 3-37 shows the messages on the console logs (the first one is issued on RA28 and the second one on RA03) when the Dynamic VIPA IP address is moved.

```
EZZ8302I VIPA 172.16.240.193 TAKEN FROM TCPIPC ON RA03
EZZ8303I VIPA 172.16.240.193 GIVEN TO TCPIPC ON RA28
```

Figure 3-37 Console log

7. Verify the Dynamic VIPA 172.16.240.193 was activated on TCPIPC on RA28 as shown in Figure 3-38.

```
RO RA28,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 832
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.101.28    M282216B         P
9.24.104.34      LOOPBACK
172.16.233.28    EZASAMEMVS
172.16.233.28    EZAXCF39
172.16.233.28    EZAXCF03
172.16.251.28    VIPLAC10FB1C
172.16.240.193   VIPLAC10F0C1
127.0.0.1        LOOPBACK
```

Figure 3-38 Display NETSTAT,HOME on TCPIPC on RA28

8. Verify the Dynamic VIPA 172.16.240.193 still exists on TCPIPC on RA03 because there are active connections on this stack. Figure 3-39 and Figure 3-40 on page 82 show the `MOVING` state.

```
RO RA03,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 830
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.100.3     M032216B         P
9.24.105.76      EN103
172.16.233.3     EZASAMEMVS
172.16.233.3     EZAXCF39
172.16.233.3     EZAXCF28
172.16.251.3     VIPLAC10FB03
172.16.240.193   VIPLAC10F0C1     I
127.0.0.1        LOOPBACK
8 OF 8 RECORDS DISPLAYED
```

Figure 3-39 Display NETSTAT,HOME on TCPIPC on RA03

```
RO RA03,D TCPIP,TCPIPC,SYSPLEX,VIPAD
D TCPIP,TCPIPC,SYSPLEX,VIPAD
EZZ8260I SYSPLEX CS V2R10 832
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.240.193
  ORIGIN: VIPARANGE BIND
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA28     ACTIVE      255.255.255.0   172.16.240.0
  TCPIPC   RA03     MOVING      255.255.255.0   0.0.0.0
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.251.0
  TCPIPC   RA28     BACKUP 200
IPADDR: 172.16.251.28
  ORIGIN: VIPABACKUP
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA28     ACTIVE      255.255.255.0   172.16.251.0
  TCPIPC   RA03     BACKUP 100
```

Figure 3-40 Display SYSPLEX,VIPAD

9. Verify the connection status of TCPIPC on RA03 as illustrated in Figure 3-41.

```
RO RA03,D TCPIP,TCPIPC,N,CONN
D TCPIP,TCPIPC,N,CONN
EZZ2500I NETSTAT CS V2R10 TCPIPC 834
USER ID  CONN     LOCAL SOCKET           FOREIGN SOCKET         STATE
BPXOINIT 0000000B 0.0.0.0..10007         0.0.0.0..0             LISTEN
CLAUDIA5 00000053 172.16.240.193..1500   0.0.0.0..0             LISTEN
FTPDC1   00000011 0.0.0.0..21            0.0.0.0..0             LISTEN
OMPROUTC 00000019 127.0.0.1..1027       127.0.0.1..1028         ESTBLSH
TCPIPC   00000016 0.0.0.0..23            0.0.0.0..0             LISTEN
TCPIPC   0000005A 172.16.240.193..23     9.24.106.64..1126      ESTBLSH
```

Figure 3-41 Display NETSTAT,CONN

10. Start a new connection to Dynamic VIPA 172.16.240.193 and check the connection is established to TCPIPC on RA28 as shown in Figure 3-42.

```
RO RA28,D TCPIP,TCPIPC,N,CONN
D TCPIP,TCPIPC,N,CONN
EZZ2500I NETSTAT CS V2R10 TCPIPC 838
USER ID  CONN     LOCAL SOCKET           FOREIGN SOCKET         STATE
BPXOINIT 0000000B 0.0.0.0..10007         0.0.0.0..0             LISTEN
FTPDC1   00000011 0.0.0.0..21            0.0.0.0..0             LISTEN
OMPROUTC 00000019 127.0.0.1..1026        127.0.0.1..1027        ESTBLSH
TCPIPC   00000018 0.0.0.0..23            0.0.0.0..0             LISTEN
TCPIPC   000000B9 172.16.240.193..23     9.24.106.64..1127      ESTBLSH
```

Figure 3-42 Display NETSTAT,CONN

bind() and VIPARange defined as MOVEable DISRUPTive

In this test, we performed the following sequence of actions:

1. Define Dynamic VIPA in VIPARange as shown in Figure 3-25 on page 76 on TCPIPC on RA03.
2. Define Dynamic VIPA in VIPARange as shown in Figure 3-25 on page 76 on TCPIPC on RA03.
3. Start application to bind Dynamic VIPA 172.16.240.193 on stack TCPIPC on RA03. The resulting bind and listen can be displayed as in Figure 3-43.

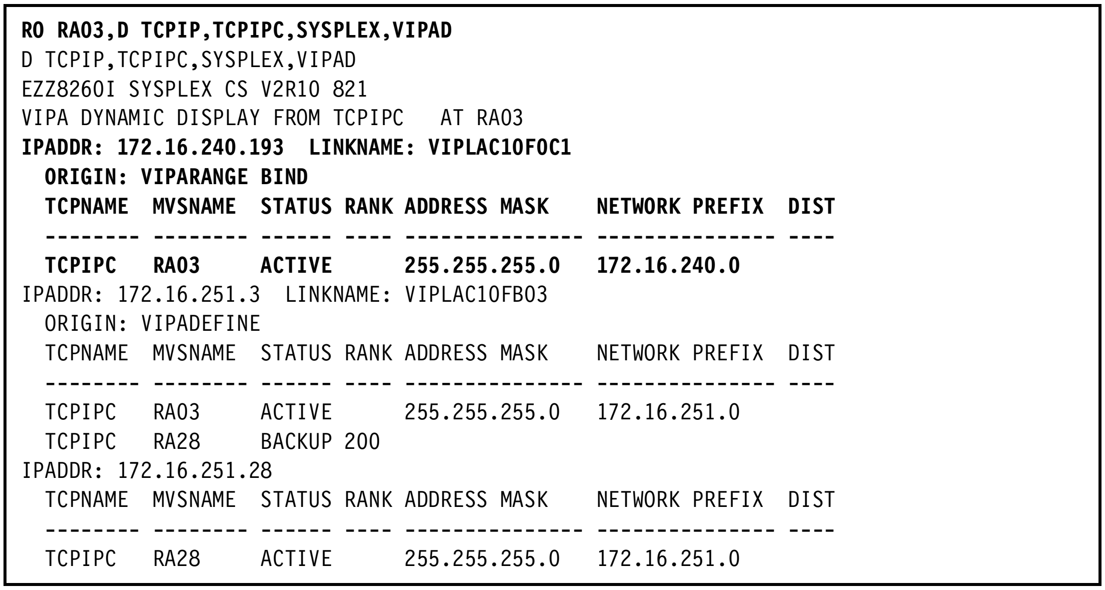
```
RO RA03,D TCPIP,TCPIPC,N,CONN
D TCPIP,TCPIPC,N,CONN
EZZ2500I NETSTAT CS V2R10 TCPIPC 261
USER ID  CONN     LOCAL SOCKET           FOREIGN SOCKET         STATE
BPXOINIT 0000000B 0.0.0.0..10007         0.0.0.0..0             LISTEN
CLAUDIA2 00000028 172.16.240.193..1500   0.0.0.0..0             LISTEN
FTPDC1   00000012 0.0.0.0..21            0.0.0.0..0             LISTEN
OMPROUTC 00000019 127.0.0.1..1027        127.0.0.1..1028        ESTBLSH
```

Figure 3-43 Display NETSTAT,CONN

4. Verify the Dynamic VIPA 172.16.240.193 is activated on TCPIPC on RA03 as illustrated in Figure 3-44 and Figure 3-45 on page 84.

```
RO RA03,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 819
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.100.3     M032216B         P
9.24.105.76      EN103
172.16.233.3     EZASAMEMVS
172.16.233.3     EZAXCF39
172.16.233.3     EZAXCF28
172.16.251.3     VIPLAC10FB03
172.16.240.193   VIPLAC10F0C1
127.0.0.1        LOOPBACK
8 OF 8 RECORDS DISPLAYED
```

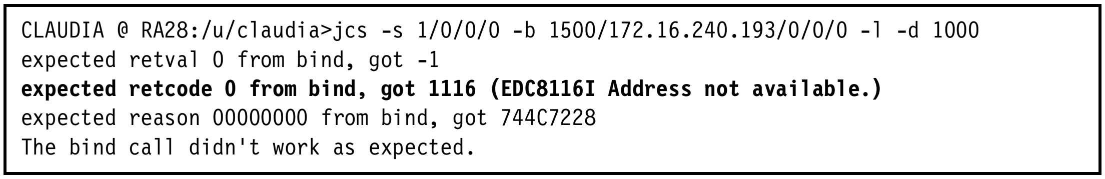
Figure 3-44 Display NETSTAT,HOME



```
RO RA03,D TCPIP,TCPIPC,SYSPLEX,VIPAD
D TCPIP,TCPIPC,SYSPLEX,VIPAD
EZZ8260I SYSPLEX CS V2R10 821
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.240.193  LINKNAME: VIPLAC10F0C1
  ORIGIN: VIPARANGE BIND
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.240.0
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.251.0
  TCPIPC   RA28     BACKUP 200
IPADDR: 172.16.251.28
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA28     ACTIVE      255.255.255.0   172.16.251.0
```

Figure 3-45   Display SYSPLEX,VIPAD

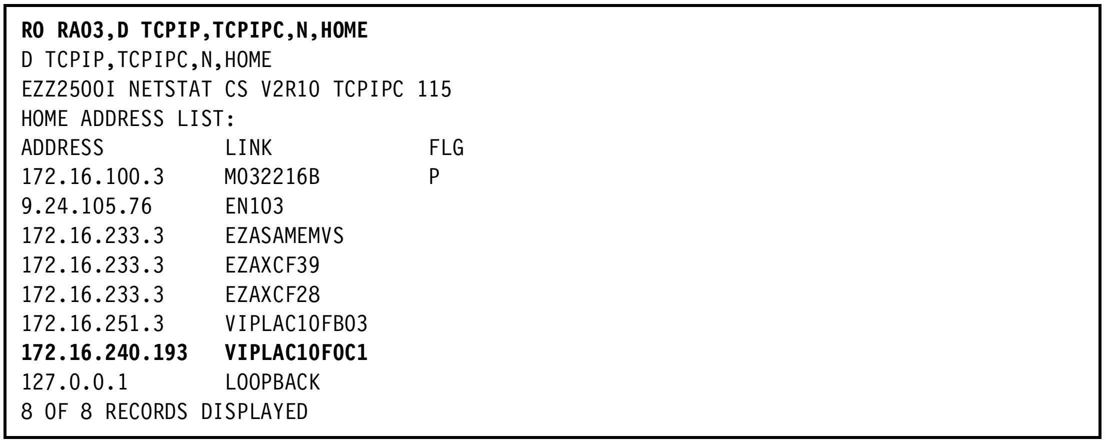
5. Start the same application on TCPIPC on RA28 to bind to the same Dynamic VIPA 172.16.240.193. The bind fails as shown in Figure 3-46.



```
CLAUDIA @ RA28:/u/claudia>jcs -s 1/0/0/0 -b 1500/172.16.240.193/0/0/0 -l -d 1000
expected retval 0 from bind, got -1
expected retcode 0 from bind, got 1116 (EDC8116I Address not available.)
expected reason 00000000 from bind, got 744C7228
The bind call didn't work as expected.
```

Figure 3-46   Messages for application jcs

6. Verify the Dynamic VIPA 172.16.240.193 is still active on TCPIPC on RA03. Figure 3-47 shows the home list display and Figure 3-48 on page 85 shows the VIPAD display.



```
RO RA03,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 115
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.100.3     M032216B         P
9.24.105.76      EN103
172.16.233.3     EZASAMEMVS
172.16.233.3     EZAXCF39
172.16.233.3     EZAXCF28
172.16.251.3     VIPLAC10FB03
172.16.240.193   VIPLAC10F0C1
127.0.0.1        LOOPBACK
8 OF 8 RECORDS DISPLAYED
```

Figure 3-47   Display NETSTAT,HOME

```
RO RA03,D TCPIP,TCPIPC,SYSPLEX,VIPAD
D TCPIP,TCPIPC,SYSPLEX,VIPAD
EZZ8260I SYSPLEX CS V2R10 121
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.240.193  LINKNAME: VIPLAC10F0C1
  ORIGIN: VIPARANGE BIND
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.240.0
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.251.0
  TCPIPC   RA28     BACKUP 200
IPADDR: 172.16.251.28
  ORIGIN: VIPABACKUP
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA28     ACTIVE      255.255.255.0   172.16.251.0
  TCPIPC   RA03     BACKUP 200
```

Figure 3-48 Display SYSPLEX,VIPAD

7. The active connection to Dynamic VIPA 172.16.240.193 is not broken, as is shown in the connection display in Figure 3-48.

```
RO RA03,D TCPIP,TCPIPC,N,CONN
D TCPIP,TCPIPC,N,CONN
EZZ2500I NETSTAT CS V2R10 TCPIPC 263
USER ID  CONN     LOCAL SOCKET           FOREIGN SOCKET         STATE
BPXOINIT 0000000B 0.0.0.0..10007         0.0.0.0..0             LISTEN
CLAUDIA2 00000028 172.16.240.193..1500   0.0.0.0..0             LISTEN
FTPDC1   00000012 0.0.0.0..21            0.0.0.0..0             LISTEN
OMPROUTC 00000019 127.0.0.1..1027        127.0.0.1..1028        ESTBLSH
```

Figure 3-49 Display NETSTAT,CONN

More information about the results of attempting to create a Dynamic VIPA IP address when it already exists in the sysplex or when there is the same IP address configured in a HOME statement can be found in *z/OS V1R2.0 CS: IP Configuration Guide*, SC31-8775.

4

Automatic VIPA takeover and takeback

By providing IP addresses for the z/OS applications that are not associated with a specific physical network attachment or gateway, Virtual IP Addresses (VIPAs) facilitate fault tolerance against outages of the IP interfaces on the zSeries. With static VIPAs, however, if the stack itself failed, the VIPA also failed. To overcome such a failure, the VIPA IP address had to move to another stack via the manual VARY OBEY command. Since CS for OS/390 IP V2R8, the following improvements were provided to allow more systematic VIPA takeover operation:

- VIPA can be taken over automatically by another stack.
- VIPA can be taken back automatically by the original stack.

However, in CS for OS/390 IP V2R8, the VIPA IP address was not taken back by the original stack while there were any active connections on the VIPA backup stack. So, the takeback process could be delayed, potentially for a long period of time.

CS for OS/390 V2R10 IP solved this problem and provided the following improvements:

- VIPA takeback can be immediate and nondisruptive.
- A DVIPA can also be *distributed* for load balancing using the Sysplex Distributor function. Sysplex Distributor is covered in detail in Chapter 5, “Sysplex Distributor” on page 109.

4.1 Overview of VIPA takeover/takeback

In this section, we explain the theory behind the VIPA concept. We discuss the goals of Dynamic VIPA and VIPA takeover/takeback and how these goals are met by CS for z/OS IP.

4.1.1 VIPA concept

An IP network provides nondisruptive rerouting of traffic in the event of a failure, but only within the routing network itself, not at the endpoint hosts. For most client hosts (PCs or workstations), failure of the host, the network adapter, or the connected link will just isolate the client application from the network, if it does not take down the client application altogether. For servers, on the other hand, particularly large-capacity and highly scalable servers such as z/OS, it is extremely common to have more than one link into a z/OS image and its associated IP stack. While connections may be distributed among the various links and adapters, failure of one such will mean loss of all TCP connections associated with the failing device or link, because the TCP connection is in part defined by the IP address of the failed adapter. In addition, no new data destined for this address, regardless of whether it is TCP or UDP, may be received.

CS for z/OS addresses the requirement of nondisruptive rerouting around a failing network adapter by allowing the customer to define a virtual adapter with an associated Virtual IP Address (VIPA). A virtual adapter (interface) has no real existence, and a VIPA is really associated with the stack as a whole. To the routers attached to the stack via physical adapters, a VIPA appears to be on a subnet on the other side of the z/OS IP stack, and the TCP stack looks like another router that happens to have reachability to that IP address. On the z/OS IP stack, on the other hand, the VIPA acts somewhat like a loopback address; incoming packets addressed to the VIPA are routed up the stack for handling by TCP or UDP as with any other home IP interface. Dynamic routing protocols can provide transparent rerouting around the failure of an adapter on the endpoint stack, in that the VIPA still appears reachable to the routing network via one of the other adapters on the z/OS.

4.1.2 Dynamic VIPA enhancements

While VIPA removes a single hardware interface and the associated transmission medium as a single point of failure for a large number of connections, the connectivity of the server can still be lost through a failure of a single stack or an MVS image. Of course, we can move a VIPA manually to the other stack, but customers require automatic recovery wherever possible, especially in a sysplex environment.

Therefore, CS for OS/390 IP V2R8 and later provides improvements by adding the VIPA takeover function. VIPA takeover builds on the VIPA concept, but automates the movement of the VIPA to an appropriate surviving stack. *Automatic VIPA takeover* allows a VIPA address to move automatically to a stack where an existing suitable application instance already resides, allowing that instance to serve clients formerly connecting to the failed stack. *Automatic VIPA takeback* allows a VIPA address to move back automatically to the failed stack once it is restored.

The VIPA takeover function is supported since CS for OS/390 IP V2R8, but VIPA takeback was either disruptive or occurred only when all connections on the stack that originally took over the VIPA terminated. In CS for OS/390 IP V2R10 and later, VIPA takeback can be immediate and nondisruptive. That is, the VIPA can be taken back by its rightful owner immediately without disrupting existing connections to the current owner.

The Sysplex Distributor function allows connections to be distributed among TCP/IP stacks in a sysplex environment. The distributed DVIPA is defined with the VIPADISTribute statement and is discussed in Chapter 5, “Sysplex Distributor” on page 109.

Application-specific Dynamic VIPAs allow VIPAs to be defined and activated by individual applications (with or without modifying the applications), so that the VIPA moves when the application is moved. Dynamic VIPA is defined by the VIPARANGE statement that is discussed in Chapter 3, “Dynamic VIPA (for application instance)” on page 61.

4.1.3 VIPA takeover and VIPA takeback

Automatic VIPA takeover requires that dynamic VIPA IP addresses (as opposed to traditional static VIPA IP addresses) be defined as having a normal "home" stack, and optionally one or more backup stacks. All the stacks share information regarding dynamic VIPAs using z/OS XCF messaging (the same mechanism as dynamic XCF and sysplex sockets), so that, for each dynamic VIPA, all stacks know:

- Which stack has the VIPA active
- Which stack(s), and in what order, will participate in backup if the active one fails

When a failure of a stack owning an active dynamic VIPA is detected, the first stack in the backup list automatically defines DEVICE, LINK, and HOME statements for the same dynamic VIPA, and notifies its attached routing daemon of the activation. This information is passed onto the routing network via dynamic routing protocols and ultimately ensures that the DVIPA is still reachable.

Figure 4-1 shows a sysplex with three TCPIP images. In this example, stack RA03 fails and its DVIPA is subsequently taken over by RA28, which is providing the backup capability for this address.

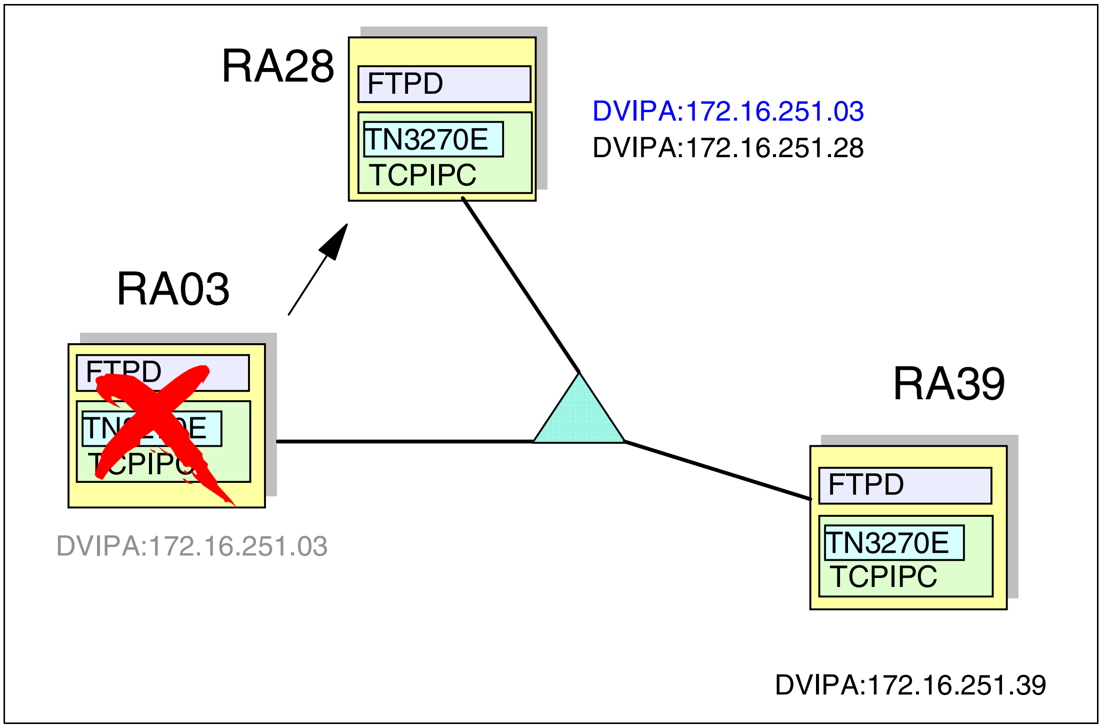


Figure 4-1 Automatic VIPA takeover of 172.16.251.03 by stack RA28

When the original "normal home" stack is reactivated, the dynamic VIPA may be taken back from the backup stack to the original stack automatically as shown in Figure 4-2. In this case, new connections are sent to the reactivated stack and the connections with the backup stack are not necessarily broken. The data of the old connections is forwarded to the backup stack. This is the default behavior for CS for z/OS V1R2 IP. In CS for z/OS V1R2, the older behavior can be enabled by setting different parameters on the VIPADEFINE statement.

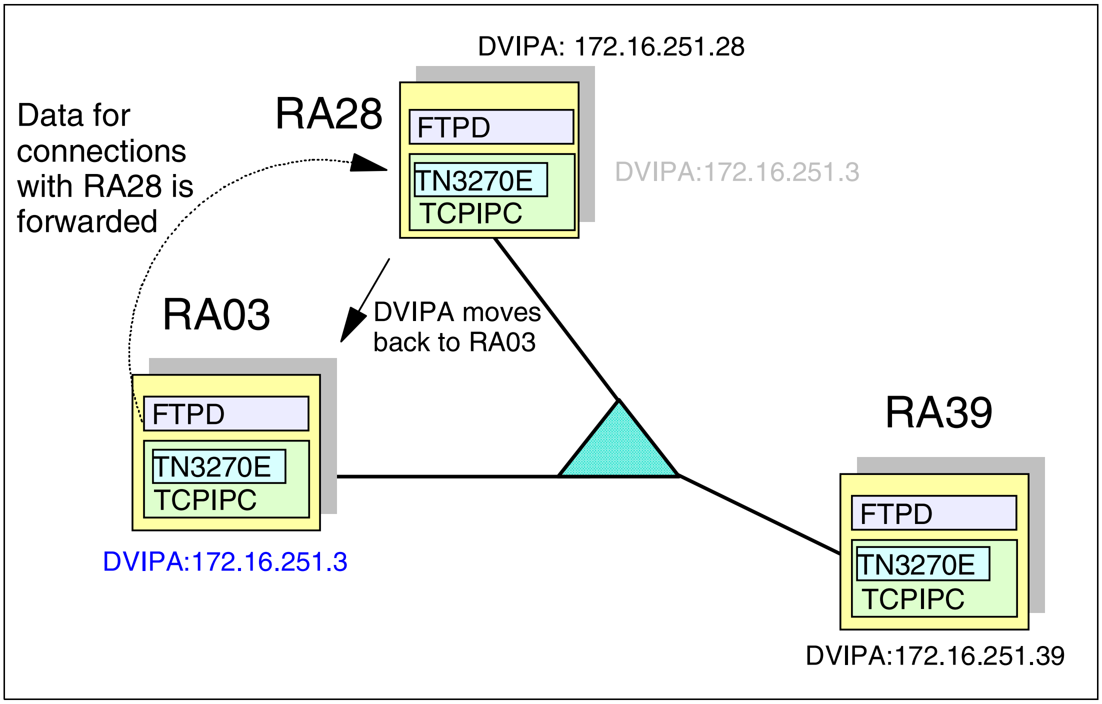


Figure 4-2   172.16.251.03 is taken back by the restarted RA03 stack

4.1.4 Benefits of sysplex-wide VIPA takeover

When a stack or its underlying z/OS fails, it is not necessary to restart the stack with the same configuration profile on a different z/OS image as is needed with static VIPA.

After the stack failure, the VIPA address is automatically moved to another stack without the need for human intervention. The new stack will have received information regarding the connections from the original stack and will accept new connections for the DVIPA. The routers are automatically informed about the change. This increases availability because multiple server instances can be started in different stacks.

VIPA takeover allows complete flexibility in the placement of servers within sysplex nodes, not limited to traditional LAN interconnection with MAC addresses. Building on the VIPA concept means that spare adapters do not have to be provided, as long as the remaining adapters have enough capacity to handle the increased load. Spare processing capacity may be distributed across the sysplex rather than on a single node.

4.1.5 Benefits of sysplex-wide VIPA takeback

When a TCP/IP stack fails, its load may be assumed by another stack. As soon as the failed stack is activated, this stack may take back the control of all connections. No connections are lost, because only new connections will be established with the original stack. The connections that were already established in the backup stack are kept. However, the information about these connections is sent by the backup stack to the original owning stack. Using this information, the original stack routes data packets for connections terminating on the backup stack to it.

In CS for OS/390 IP V2R8, the original stack could take back the VIPA only when there were no more active connections on the backup stack. The takeback could be delayed for a long time or the outstanding connections on the backup could be broken. This behavior was viewed as being too restrictive and was improved in CS for OS/390 V2R10 IP.

If the original stack has defined a VIPA address supporting the Sysplex Distributor function (see Chapter 5, "Sysplex Distributor" on page 109), this function automatically will be taken back by the original distributing VIPA owner immediately. That is, a distributed VIPA can be backed up and moved around immediately and nondisruptively.

4.2 Implementing VIPA takeover and takeback

In this section we give an overview of how VIPA takeover and takeback can be defined. Practical examples are shown in 4.4, "Examples of VIPA takeover and takeback" on page 96.

4.2.1 Automatic VIPA takeover/takeback configuration

Each VIPA has a preferred home stack and set of backup stacks. Additionally, the backup stacks have a preferred order. CS for z/OS provides configuration options that allow the administrator to define which stacks should start off owning a VIPA, and which stacks should provide backup for that VIPA in the event of failure of the primary stack.

Every Dynamic VIPA parameter is defined in the VIPADynamic block in the TCP/IP profile, as shown in Figure 4-3.

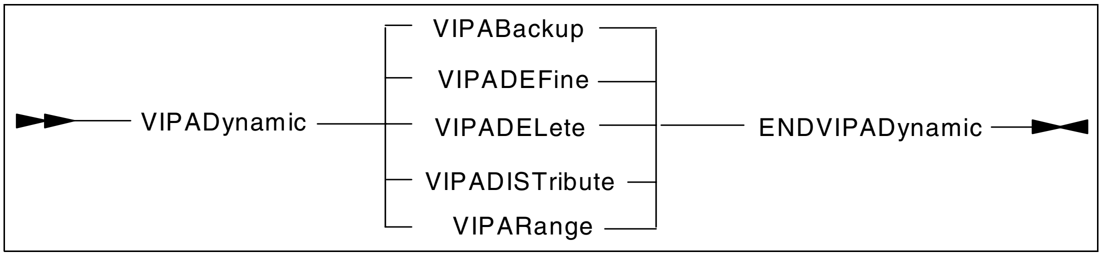


Figure 4-3 Definition format for dynamic VIPA block

Automatic VIPA takeover/takeback requires you to define the primary and backup VIPA addresses to each stack that will participate. Everything else is automatic. The configuration options are illustrated in Figure 4-4 on page 92.

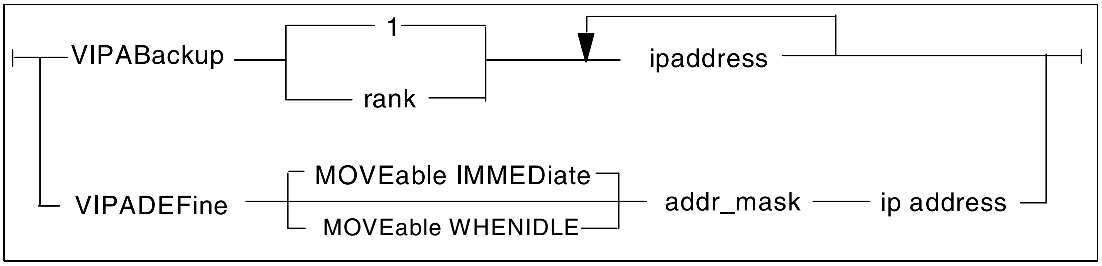


Figure 4-4 Definition format for VIPA backup

The VIPADEFine statement designates one or more VIPAs that this stack should initially own. Each is known throughout the IP network, so it requires an address and a subnet mask to determine how many of the bits of the IP address specify the network.

More than one IP address can be defined within one VIPADEFine statement, but every IP address defined should belong to the same network. It means that each IP address in a subnet should be in the same range. To check if this rule is being used, convert the subnet mask value to binary and verify the following points:

- The most significant bit should be 1.
- After the first 0 encountered (to the right of the most significant bit), all bits should be 0.
- If every subnet mask is logically ANDed with all IP addresses in the list, the result should be the same.

CS for OS/390 IP V2R10 introduced the parameters MOVE IMMEDiate and MOVE WHENIDLE. MOVE IMMEDiate means that when an original stack comes back up after it has failed, the VIPA addresses are taken back immediately, independent of the existing connections on the backup stack. If MOVE WHENIDLE is defined, the VIPA address will be owned by the backup stack while there is at least one connection active managed by the backup stack. This is the only possible situation in CS for OS/390 IP V2R8. So, if one of the stacks is running CS for OS/390 IP V2R8, the MOVE IMMED definition will be ignored and MOVE WHENIDLE will be used instead.

To preserve connections, the configuration option IPCONFIG DATAGRAMFWD must be specified in the TCP/IP profile.

The VIPABackup statement designates one or more VIPAs for which this stack will provide automatic backup when the owning stack fails. It is not necessary to define a subnet mask because it is the same as that on the primary stack for the address in question. Valid values for the rank parameter are from 0 to 255. Larger rank values move the respective stacks closer to the top of the backup chain, which means a higher priority when the need for activating a backup stack arises. That is, the higher the rank, the higher its backup priority.

There is also a statement to delete a VIPA that has been defined with VIPADEFine or VIPABackup called VIPADELete. It is coded in the VIPADynamic block and simply specifies the IP addresses to be deleted.

VIPADELete may also be used to delete a VIPA address that was defined in a VIPARange subsequently created via BIND to a specific address or via IOCTL. The VIPADELete command is executed immediately. If there are any connections to the VIPA address, they will be lost. The connections will no longer be able to communicate. However, the connections will not be removed until either they time out or are closed by the application. If the application doest not attempt to send data and the application never closes the connections, they could actually be active forever, although they will never be able to send data again.

4.3 Monitoring VIPA status

Several operator commands are provided to monitor dynamic VIPA and VIPA backup configuration and status. We used some of them in our tests, but we summarize them in this section for convenience. Figure 4-5 shows the base network configuration used.

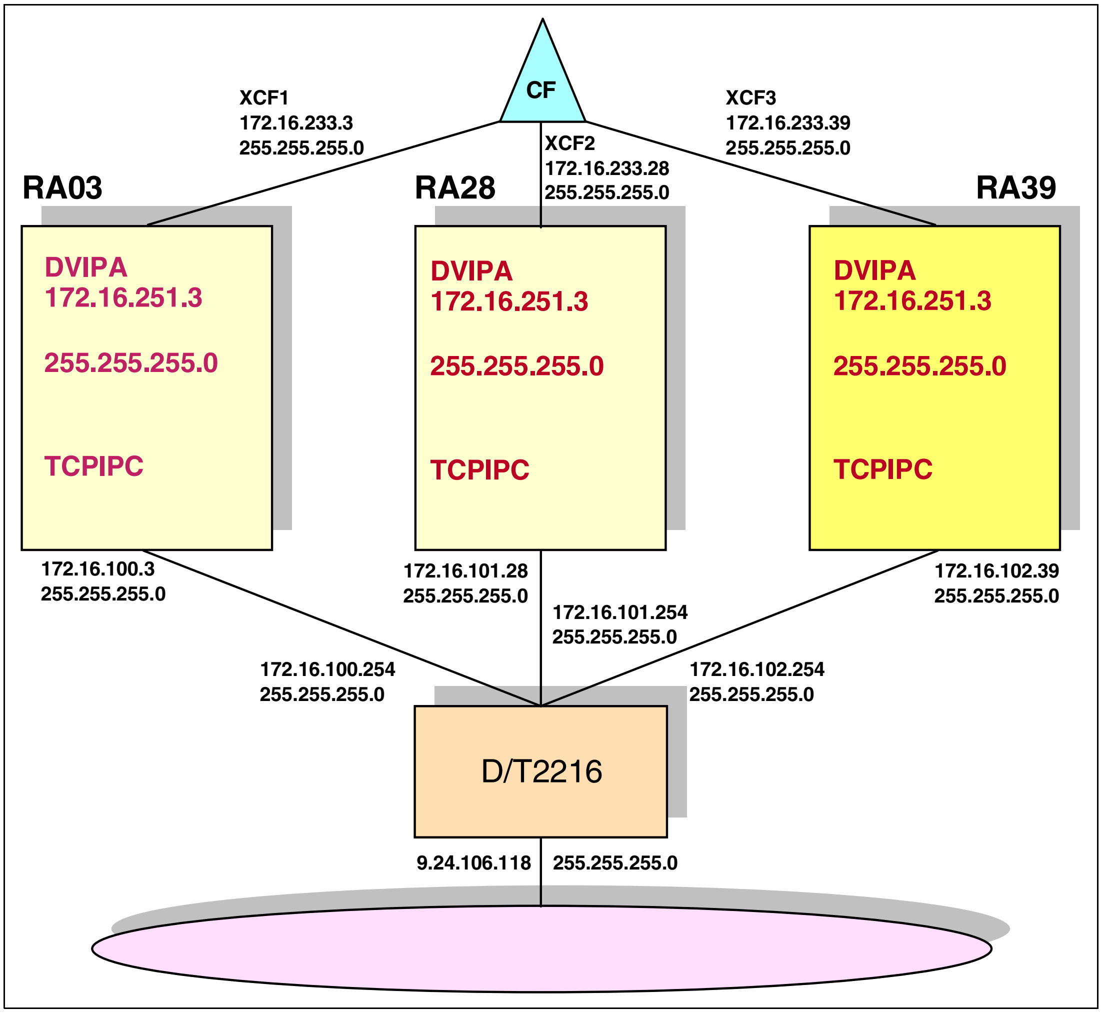


Figure 4-5 Base network configuration

4.3.1 Display Sysplex command

The command D TCPIP,<tcpipjobname>,SYSplex,VIPADyn is available to show you the status of dynamic VIPAs (both application related as defined by VIPARange and takeover flavor as defined by VIPADEFine) in the sysplex. Figure 4-6 on page 94 shows an example with VIPAs configured via VIPADEFine and VIPABackup. The origin (configuration statement responsible for the VIPA) and status of each dynamic VIPA on the stack (TCPNAME) and system (MVSNAME) in the sysplex are shown. The RANK value indicates the order in which the backup stacks will be chosen if the stack on which the dynamic VIPA is active is stopped. The active system with the highest rank is the one that will take over the dynamic VIPA.

```
D TCPIP,TCPIPC,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V2R10 779
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.251.0    BOTH     1
  TCPIPC   RA28     BACKUP 200                                  DEST     2
  TCPIPC   RA39     BACKUP 100                                  DEST     3
3 OF 3 RECORDS DISPLAYED
```

Figure 4-6 Display VIPA backup configuration in the sysplex

The Distribution Status (DIST) field show how this stack is configured. It can be a distributor stack and/or a destination stack. In our example, 1 is a distributing and destination stack (BOTH status), while 2 and 3 are destination stacks.

4.3.2 NETSTAT commands

In addition to the Display SYSplex command, there are several new parameters in the NETSTAT commands related to dynamic VIPA. The NETSTAT commands include the following:

- NETSTAT for TSO
- **onetstat** for UNIX System Services
- D TCPIP,tcpname,NETSTAT for the MVS console

The output for all of the commands have a new section in the CONFIG (-f) report. Additionally, each is now able to generate a new VIPADYN (-v) report. These reports show the dynamic VIPAs on a system basis, not a sysplex-wide basis. The global configuration information section (1) of the CONFIG report is displayed whether there are any dynamic VIPAs known to this system (see 1 in Figure 4-7 on page 95).

```
D TCPIP,TCPIPC,N,CONFIG
EZZ2500I NETSTAT CS V2R10 TCPIPC 124
TCP CONFIGURATION TABLE:
DEFAULTRCVBUFSIZE:  00016384  DEFAULTSNDBUFSIZE: 00016384
DEFLTMAXRCVBUFSIZE: 00262144
MAXRETRANSMITTIME:  120.000   MINRETRANSMITTIME: 0.500
ROUNDTRIPGAIN:      0.125     VARIANCEGAIN:      0.250
VARIANCEMULTIPLIER: 2.000     MAXSEGLIFETIME:    60.000
DEFAULTKEEPALIVE:   0.120     LOGPROTOERR:       00
TCPFLAGS:           90
UDP CONFIGURATION TABLE:
DEFAULTRCVBUFSIZE: 00016384  DEFAULTSNDBUFSIZE: 00016384
CHECKSUM:          00000001  LOGPROTOERR:       01
UDPFLAGS:          2C
IP CONFIGURATION TABLE:
FORWARDING: YES    TIMETOLIVE: 00060  RSMTIMEOUT:  00060
FIREWALL:   00000  ARPTIMEOUT: 01200  MAXRSMSIZE:  65535
IGREDIRECT: 00001  SYSPLXROUT: 00001  DOUBLENOP:   00000
STOPCLAWER: 00001  SOURCEVIPA: 00001  VARSUBNET:   00001
MULTIPATH:  NO     PATHMTUDSC: 00000  DEVRTRYDUR: 0000000090
DYNAMICXCF: 00001
  IPADDR: 172.16.233.3     SUBNET: 255.255.255.0    METRIC: 01
SMF PARAMETERS:
INITTYPE: 00  TERMTYPE: 00  CLIENTTYPE: 00  TCPIPSTATS: 00
GLOBAL CONFIGURATION INFORMATION:                       1
TCPIPSTATS: 00
```

Figure 4-7 Display NETSTAT, CONFIG command

The NETSTAT,CONFIG command was changed in CS for OS/390 IP V2R10 and a new command, NETSTAT,VIPADCFG, is available to show the information about VIPA related to the stack where the command is issued. An example is shown in Figure 4-8.

```
RO RA03,D TCPIP,TCPIPC,N,VIPADCFG
D TCPIP,TCPIPC,N,VIPADCFG
EZZ2500I NETSTAT CS V2R10 TCPIPC 190
DYNAMIC VIPA INFORMATION:
  VIPA BACKUP:
    IP ADDRESS       RANK
    ----------       ----
    172.16.251.28    000100
    172.16.251.39    000200
  VIPA DEFINE:
    IP ADDRESS       ADDRESSMASK      MOVEABLE
     ----------       -----------      --------
     172.16.251.3     255.255.255.0    IMMEDIATE
   VIPA RANGE:
     ADDRESSMASK      IP ADDRESS       MOVEABLE
     -----------      ----------       --------
     255.255.255.0    172.16.251.193   NONDISR
   VIPA DISTRIBUTE:
     IP ADDRESS       PORT   XCF ADDRESS
     ----------       ----   -----------
     172.16.251.3     00020  ALL
     172.16.251.3     00021  ALL
```

Figure 4-8 Display NETSTAT,VIPADFCG command

Figure 4-9 shows the results of a DISPLAY N,VIPADYN command. The same display can be achieved using **onetstat -v** from a UNIX System Services prompt.

```
RO RA03,D TCPIP,TCPIPC,N,VIPADYN
D TCPIP,TCPIPC,N,VIPADYN
EZZ2500I NETSTAT CS V2R10 TCPIPC 382
IP ADDRESS     ADDRESSMASK    STATUS   ORIGINATION  DISTSTAT
172.16.251.3   255.255.255.0  ACTIVE   VIPADEFINE   DIST/DEST
172.16.251.28  255.255.255.0  BACKUP   VIPABACKUP
172.16.251.39  255.255.255.0  BACKUP   VIPABACKUP
172.16.251.193  255.255.255.0 ACTIVE   VIPARANGE IOCTL
4 OF 4 RECORDS DISPLAYED
```

Figure 4-9 Display NETSTAT,VIPADYN(-v) command

With the advent of Sysplex Distributor, the stack may need to keep track of a VIPA Connection Routing Table (VCRT). Figure 4-10 shows the results of a DISPLAY N,VCRT command.

```
RO RA03,D TCPIP,TCPIPC,N,VCRT
D TCPIP,TCPIPC,N,VCRT
EZZ2500I NETSTAT CS V2R10 TCPIPC 882
DYNAMIC VIPA CONNECTION ROUTING TABLE:
DEST IPADDR      DPORT  SRC IPADDR       SPORT  DESTXCF ADDR
172.16.251.3     00021  9.24.106.64      01105  172.16.233.39
172.16.251.3     00023  9.24.106.64      01106  172.16.233.39
2 OF 2 RECORDS DISPLAYED
```

Figure 4-10 Display NETSTAT,VCRT command

4.4 Examples of VIPA takeover and takeback

In this section we demonstrate two examples of VIPA takeover and takeback using different configurations. These are:

- VIPA takeover/takeback when the VIPADEFine statement is defined with the MOVE IMMED parameter. VIPA definitions are coded on multiple stacks, so that the failure of one stack results in the takeover of its VIPA address by another stack. When the original stack is recovered, the VIPA address is taken back immediately.
- VIPA takeover/takeback when the VIPADEFine statement is defined with the MOVE WHENIDLE parameter. After a failure of one stack, the backup stack takes over the VIPA address. However, after recovering from the failure on the original stack, the takeback is delayed until all connections are finished on backup stack.

For this test, we used only two TCP/IP stacks: TCPIPC on RA28 and TCPIPC on RA03. For convenience, TCPIPC on RA39 is not running.

4.4.1 Automatic VIPA takeover/takeback - MOVE IMMED

We configured on each stack a primary VIPA address and a backup address for the other stack's primary VIPA. We performed the following sequence of actions:

1. Define the primary VIPA IP addresses: 172.16.251.3 on TCPIPC on RA03 and 172.16.251.28 on TCPIPC on RA28.

2. Define the backup VIPA IP addresses: 172.16.251.3 on TCPIPC on RA28 and 172.16.251.28 on TCPIPC on RA03.
3. Start our server applications on both of the stacks, and establish connections from a client to the server on TCPIPC on RA03 using the VIPA 172.16.251.3.
4. Log on to the TN3270 server on TCPIPC on RA03 using the VIPA.
5. Stop the TCPIPC on RA03 stack.
6. Make sure the backup stack TCPIPC on RA28 takes over the VIPA 172.16.251.3.
7. Via dynamic routing (OSPF), the network is informed that the VIPA has moved and all connection requests are routed to the stack owning the VIPA now.
8. Restart the TCPIPC on RA03 stack and the server application.
9. Check that the VIPA moves back to the original stack.

   Figure 4-11 shows our VIPA definitions on TCPIPC on RA03, and Figure 4-12 shows the corresponding definitions on TCPIPC on RA28.

```
VIPADYNAMIC
 VIPADEFINE MOVE IMMED 255.255.255.0 172.16.251.3
 VIPABACKUP 100 172.16.251.28
ENDVIPADYNAMIC
```

Figure 4-11 Dynamic VIPA definition for TCPIPC on RA03

```
VIPADYNAMIC
  VIPADEFINE MOVE IMMED 255.255.255.0 172.16.251.28
  VIPABACKUP 100 172.16.251.3
ENDVIPADYNAMIC
```

Figure 4-12 Dynamic VIPA definition for TCPIPC on RA28

10. Before stopping the TCPIPC stack on RA03, we used the DISPLAY SYSplex command on TCPIPC on RA28 to see which dynamic VIPA was known to each stack. Figure 4-13 shows the results that match our definitions.

```
D TCPIP,TCPIPC,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V2R10 194
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.251.0
  TCPIPC   RA28     BACKUP 200
IPADDR: 172.16.251.28
  ORIGIN: VIPABACKUP
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA28     ACTIVE      255.255.255.0   172.16.251.0
  TCPIPC   RA03     BACKUP 100
4 OF 4 RECORDS DISPLAYED
```

Figure 4-13 Display VIPA definition for TCPIPC on RA28 before TCPIPC on RA03 stack failure

11. We also displayed the OSPF routing table on TCPIPC on RA28, as shown in Figure 4-14. Note the presence of the local dynamic VIPA 1 with its stack-generated link name, and the remote VIPA 2 belonging to TCPIPC on RA03.

```
RO RA03,D TCPIP,TCPIPC,OMPROUTE,RTTABLE
D TCPIP,TCPIPC,OMPROUTE,RTTABLE
EZZ7847I ROUTING TABLE 370
TYPE   DEST NET      MASK      COST    AGE     NEXT HOP(S)

SPE2   0.0.0.0              0  1       536     172.16.100.254
SPE2   9.0.0.0       FF000000  1       536     172.16.100.254
 SPF   9.1.150.0     FFFFFE00  1814    1359    172.16.100.254
 SPF   9.3.1.0       FFFFFF00  1814    1359    172.16.100.254
 SPF   9.3.240.0     FFFFFF00  1814    1359    172.16.100.254
SPE2   9.12.0.0      FFFFFF00  1       536     172.16.100.254
 SPF   9.12.2.0      FFFFFF00  14      1359    172.16.100.254
 SPF   9.12.3.0      FFFFFFF0  1808    1359    172.16.100.254
 SPF   9.12.3.16     FFFFFFF0  1808    1359    172.16.100.254
 SPF   9.12.3.32     FFFFFFF0  1808    1359    172.16.100.254
 SPF   9.12.3.48     FFFFFFF0  1808    1359    172.16.100.254
SPE2   9.12.6.0      FFFFFF00  1       536     172.16.100.254
SPE2   9.12.9.0      FFFFFF00  1       536     172.16.100.254
SPE2   9.12.13.0     FFFFFF00  1       536     172.16.100.254
 SPF   9.12.14.0     FFFFFF00  14      1359    172.16.100.254
SPE2   9.12.15.0     FFFFFF00  1       536     172.16.100.254
 SPF   9.24.104.0    FFFFFF00  7       1359    172.16.100.254
 SPF   9.24.104.1    FFFFFFFF  7       1359    172.16.100.254
 SPF   9.24.104.18   FFFFFFFF  1       1359    172.16.100.254
 SPF   9.24.105.0    FFFFFF00  17      1359    172.16.100.254
 SPF   9.24.106.0    FFFFFF00  7       1359    172.16.100.254
SPE2   9.32.41.40    FFFFFFFC  1       536     172.16.100.254
 DIR*  172.16.233.0  FFFFFF00  1       568     172.16.233.3(2)
 SPF   172.16.233.3  FFFFFFFF  0       1430    EZASAMEMVS
STAT*  172.16.233.28 FFFFFFFF  0       1431    172.16.233.3
 SPF*  172.16.251.0  FFFFFF00  14      607     172.16.100.254
 DIR*  172.16.251.3  FFFFFFFF  1       1430    VIPLAC10FB03   2
 SPF   172.16.251.28 FFFFFFFF  14      1359    172.16.100.254 1
.
```

Figure 4-14 Display OMPROUTE table before TCPIPC on RA03 stack failure

12. We then stopped the TCP/IP stack TCPIPC on RA03, which caused the servers (TN3270 and FTP) to go down. All the other sysplex stacks were informed of the failure through XCF and took steps to recover the failed VIPA. The stack with the highest rank (indeed, the only stack) on the backup list for 172.16.251.3 was TCPIPC on RA28. Therefore, TCPIPC on RA28 defined and activated this VIPA address itself. The console log on RA28 confirmed this, as shown in Figure 4-15.

```
EZZ8301I VIPA 172.16.251.3 TAKEN OVER FROM TCPIPC ON RA03
EZZ4323I CONNECTION TO 172.16.233.3 CLEARED FOR DEVICE RA03M
```

Figure 4-15 Console message on RA28 at VIPA takeover

13. We then issued some TCP/IP display commands on RA28 to check the new status, as in Figure 4-16 on page 99.

```
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 121
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.101.28    M282216B         P
172.16.233.28    EZASAMEMVS
172.16.233.28    EZAXCF39
172.16.233.28    EZAXCF03
172.16.251.28    VIPLAC10FB1C
172.16.251.3   VIPLAC10FB03
127.0.0.1        LOOPBACK
7 OF 7 RECORDS DISPLAYED                   1

D TCPIP,TCPIPC,SYSPLEX,VIPAD
EZZ8260I SYSPLEX CS V2R10 124
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA28
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPABACKUP
  TCPNAME MVSNAME STATUS RANK ADDRESS MASK NETWORK PREFIX DIST
  ------- ------- ------ ---- ------- ---- ------------- ----
  TCPIPC   RA28    ACTIVE     255.255.255.0 172.16.251.0 2
IPADDR: 172.16.251.28  LINKNAME: VIPLAC10FB1C
  ORIGIN: VIPADEFINE
  TCPNAME MVSNAME STATUS RANK ADDRESS MASK NETWORK PREFIX DIST
  ------- ------- ------ ---- ----------- ------------- ----
  TCPIPC  RA28    ACTIVE     255.255.255.0 172.16.251.0
2 OF 2 RECORDS DISPLAYED

D TCPIP,TCPIPC,N,CON
EZZ2500I NETSTAT CS V2R10 TCPIPC 209
USER ID  CONN      LOCAL SOCKET      FOREIGN SOCKET    STATE
BPXOINIT 00000F2C  0.0.0.0..10007    0.0.0.0..0        LISTEN
FTPDC1   00000012  0.0.0.0..21       0.0.0.0..0        LISTEN
OMPROUTC 0000001A  127.0.0.1..1026   127.0.0.1..1027   ESTBLSH
TCPIPC   000040AB  172.16.251.3..23  9.24.106.64..1092 ESTBLSH 3
TCPIPC   00000019  0.0.0.0..23       0.0.0.0..0        LISTEN
TCPIPC   000040A1  172.16.251.28..23 9.24.106.64..1091 ESTBLSH
.
.
D TCPIP,TCPIPC,N,SOCKETS
EZZ2500I NETSTAT CS V2R10 TCPIPC 219
SOCKETS INTERFACE STATUS:
TYPE   BOUND TO          CONNECTED TO        STATE    CONN
NAME: BPXOINIT  SUBTASK: 006ECAE8
NAME: TCPIPC    SUBTASK: 00000000
STREAM 172.16.251.3..23  9.24.106.64..1092   ESTBLSH  000040AB
STREAM 172.16.251.28..23 9.24.106.64..1091   ESTBLSH  000040A1
.
```

Figure 4-16 Displays after VIPA takeover on TCPIPC on RA28

In these displays:

- 1 shows the VIPA address 172.16.251.3 in TCPIPC on RA28's HOME list. This stack now owns the address.
- 2 is a DISPLAY SYSplex command showing the status of the dynamic VIPA addresses known to TCPIPC on RA28. The recovered address 172.16.251.3 was defined as VIPABackup but is now active.
- 3 shows that new connections to our server application are connected to the instance on TCPIPC on RA28.

Figure 4-17 shows an FTP client connection to TCPIPC on RA03 (IP address 172.16.251.3). Even though stack TCPIPC on RA03 is not active, the connection is established with TCPIPC on RA28C (MVS28C).

```
C:\>ftp 172.16.251.3
Connected to 172.16.251.3.
220-FTPDC1 IBM FTP CS V2R10 at MVS28C, 14:59:37 on 2000-09-12.
220 Connection will close if idle for more than 5 minutes.
User (172.16.251.3:(none)): claudia
331 Send password please.
Password:
230 CLAUDIA is logged on.  Working directory is "CLAUDIA.".
```

Figure 4-17 FTP connection using the VIPA 172.16.251.3 after stopping TCPIPC on RA03

Figure 4-18 shows that the connections with destination address to RA03 are being connected through XCF address 172.16.233.28.

```
RO RA28,D TCPIP,TCPIPC,N,VCRT
D TCPIP,TCPIPC,N,VCRT
EZZ2500I NETSTAT CS V2R10 TCPIPC 403
DYNAMIC VIPA CONNECTION ROUTING TABLE:
DEST IPADDR      DPORT  SRC IPADDR       SPORT  DESTXCF ADDR
172.16.251.3     00021  9.24.106.64      01138  172.16.233.28
172.16.251.3     00023  9.24.106.64      01139  172.16.233.28
172.16.251.28    00023  9.24.106.64      01140  172.16.233.28
3 OF 3 RECORDS DISPLAYED
```

Figure 4-18 VIPA connection routing table after stopping TCPIPC on RA03

Some time later, we restarted the failed TCP/IP stack TCPIPC on RA03 and the server application on TCPIPC on RA03. Figure 4-19 shows the console message at RA03 system after restarting of stack TCPIPC on RA03.

```
EZZ8302I VIPA 172.16.251.3 TAKEN FROM TCPIPC ON RA28
EZZ8303I VIPA 172.16.251.3 GIVEN TO TCPIPC ON RA03
```

Figure 4-19 Console message on RA03 after restarting TCPIPC on RA03

Although TCPIPC on RA28 had the VIPA 172.16.251.3 active and some active connections through the dynamic VIPA, this address was taken back by RA03. The old connections are not broken, however. TCPIPC on RA03 is now receiving all new connections. Figure 4-20 on page 101 shows the creation of a new FTP connection.

```
C:\>ftp 172.16.251.3
Connected to 172.16.251.3.
220-FTPDC1 IBM FTP CS V2R10 at MVS03C, 15:45:38 on 2000-09-12.
220 Connection will close if idle for more than 5 minutes.
User (172.16.251.3:(none)): claudia
331 Send password please.
Password:
230 CLAUDIA is logged on.  Working directory is "/u/claudia".
```

Figure 4-20 FTP connection using the VIPA 172.16.251.3 after restarting TCPIPC on RA03

Figure 4-21 shows the VIPA Connection Routing Table (VCRT) in RA03 and RA28, which include the old and new connections to 172.16.251.3.1, 2 and 3 are old connections that are still active on RA28. 4 is the new FTP connection to RA03.

```
RO RA28,D TCPIP,TCPIPC,N,VCRT
D TCPIP,TCPIPC,N,VCRT
EZZ2500I NETSTAT CS V2R10 TCPIPC 717
DYNAMIC VIPA CONNECTION ROUTING TABLE:
DEST IPADDR      DPORT  SRC IPADDR       SPORT  DESTXCF ADDR
172.16.251.3     00020  9.24.106.64      01181  172.16.233.28 1
172.16.251.3     00021  9.24.106.64      01180  172.16.233.28 2
172.16.251.3     00023  9.24.106.64      01179  172.16.233.28 3
172.16.251.28    00023  9.24.106.64      01178  172.16.233.28
4 OF 4 RECORDS DISPLAYED

RO RA03,D TCPIP,TCPIPC,N,VCRT
D TCPIP,TCPIPC,N,VCRT
EZZ2500I NETSTAT CS V2R10 TCPIPC 932
DYNAMIC VIPA CONNECTION ROUTING TABLE:
DEST IPADDR      DPORT  SRC IPADDR       SPORT  DESTXCF ADDR
172.16.251.3     00021  9.24.106.64      01182  172.16.233.3  4
172.16.251.3     00021  9.24.106.64      01180  172.16.233.28 2
172.16.251.3     00023  9.24.106.64      01179  172.16.233.28 3
3 OF 3 RECORDS DISPLAYED
```

Figure 4-21 Display VCRT on RA03 and RA28 after takeback

Figure 4-22 on page 102 shows the status of stacks TCPIPC on RA03 and TCPIPC on RA28 in each system. The status `MOVING` for IP address 172.16.251.3 in the RA28 display means that another stack has activated the VIPA and has advertised reachability to it.

```
RO RA03,D TCPIP,TCPIPC,N,VIPAD
D TCPIP,TCPIPC,N,VIPAD
EZZ2500I NETSTAT CS V2R10 TCPIPC 562
IP ADDRESS      ADDRESSMASK     STATUS  ORIGINATION   DISTSTAT
172.16.251.3    255.255.255.0   ACTIVE  VIPADEFINE
172.16.251.28   255.255.255.0   BACKUP  VIPABACKUP
2 OF 2 RECORDS DISPLAYED
RO RA28,D TCPIP,TCPIPC,N,VIPAD
D TCPIP,TCPIPC,N,VIPAD
EZZ2500I NETSTAT CS V2R10 TCPIPC 721
IP ADDRESS      ADDRESSMASK     STATUS  ORIGINATION   DISTSTAT
172.16.251.3    255.255.255.0   MOVING  VIPABACKUP
172.16.251.28   255.255.255.0   ACTIVE  VIPADEFINE
2 OF 2 RECORDS DISPLAYED
```

Figure 4-22 Display VIPAD after takeback

4.4.2 Automatic VIPA takeover/takeback - MOVE WHENIDLE

On each stack we configured a primary VIPA address and a backup address for the other stack's primary VIPA. We performed the following sequence of actions:

1. Define the primary VIPA IP addresses: 172.16.251.3 on TCPIPC on RA03 and 172.16.251.28 on TCPIPC on RA28. This time, the primary VIPA 172.16.251.3 on TCPIPC on RA03 was defined with parameter MOVE WHENIDLE.
2. Define the backup VIPA IP addresses: 172.16.251.3 on TCPIPC on RA28 and 172.16.251.28 on TCPIPC on RA03.
3. Start our server applications on both of the stacks, and establish connections from a client to the server on TCPIPC on RA03 using the VIPA 172.16.251.3.
4. Log on to the TN3270 server on TCPIPC on RA03 using the VIPA.
5. Stop the TCPIPC on the RA03 stack.
6. Make sure the backup stack TCPIPC on RA28 takes over the VIPA 172.16.251.3.
7. Via dynamic routing (OSPF) the network is informed that the VIPA has moved and all connection requests are routed to the stack owning the VIPA now (RA28).
8. Restart the TCPIPC and our server application at RA03.
9. Check that the VIPA is not moved back to the original stack.
10. Close all existing connections to VIPA address 172.16.251.3.
11. Make sure the original stack TCPIPC on RA03 takes back the VIPA 172.16.251.3.
12. Establish a new connection on TCPIPC on RA03 using VIPA 172.16.251.3.

Figure 4-23 shows our VIPA definitions on TCPIPC on RA28 and Figure 4-24 on page 103 shows the corresponding definitions on TCPIPC on RA03.

```
VIPADYNAMIC
 VIPADEFINE MOVE WHENIDLE 255.255.255.0 172.16.251.3
 VIPABACKUP 100 172.16.251.28
ENDVIPADYNAMIC
```

Figure 4-23 Dynamic VIPA definition for TCPIPC on RA03

```
VIPADYNAMIC
  VIPADEFINE MOVE IMMED 255.255.255.0 172.16.251.28
  VIPABACKUP 100 172.16.251.3
ENDVIPADYNAMIC
```

Figure 4-24 Dynamic VIPA definition for TCPIPC on RA28

Before stopping the TCPIPC on RA03, we used the DISPLAY SYSplex command to see our definitions. Figure 4-25 shows the resulting output.

```
RO RA03,D TCPIP,TCPIPC,SYSPLEX,VIPADYN
D TCPIP,TCPIPC,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V2R10 173
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME MVSNAME STATUS RANK ADDRESS MASK NETWORK PREFIX DIST
  ------- ------- ------ ---- ------------ ------------- ----
  TCPIPC   RA03   ACTIVE     255.255.255.0 172.16.251.0
  TCPIPC   RA28   BACKUP 200
IPADDR: 172.16.251.28
  ORIGIN: VIPABACKUP
  TCPNAME MVSNAME STATUS RANK ADDRESS MASK NETWORK PREFIX DIST
  -------- ------ ------ ---- ------------ ------------- ----
  TCPIPC  RA28    ACTIVE     255.255.255.0 172.16.251.0
  TCPIPC  RA03    BACKUP 100
4 OF 4 RECORDS DISPLAYED
```

Figure 4-25 Display VIPA definition for TCPIPC on RA03 before stack failure

We also displayed the OSPF routing table on TCPIPC on RA03 as shown in Figure 4-26.

```
RO RA03,D TCPIP,TCPIPC,OMPROUTE,RTTABLE
D TCPIP,TCPIPC,OMPROUTE,RTTABLE
EZZ7847I ROUTING TABLE 268
TYPE   DEST NET       MASK      COST   AGE     NEXT HOP(S)
 SPF   172.16.232.39  FFFFFFFF  8      1309    172.16.100.254
 DIR*  172.16.233.0   FFFFFF00  1      313     172.16.233.3
 SPF   172.16.233.3   FFFFFFFF  0      312     EZASAMEMVS
STAT*  172.16.233.28  FFFFFFFF  0      313     172.16.233.3
 SPF*  172.16.251.0   FFFFFF00  14     1309    172.16.100.254
 DIR*  172.16.251.3   FFFFFFFF  1      1304    VIPLAC10FB03
 SPF   172.16.251.28  FFFFFFFF  14     1309    172.16.100.254
 SPF   172.16.252.0   FFFFFF00  8      1309    172.16.100.254
 SPF   172.16.252.28  FFFFFFFF  8      1309    172.16.100.254
.
.
```

Figure 4-26 OMPROUTE table before TCPIPC on RA03 stack failure

We then stopped the TCP/IP stack TCPIPC on RA03, which caused the servers (TN3270 and FTP) to terminate. The behavior in this case is exactly the same as in the previous scenario. RA28 was informed of the failure through XCF and took steps to recover the failed VIPA. TCPIPC on RA28 dynamically defined and activated this VIPA address itself because it is the only backup. If more than one backup is defined, the stack defined with the highest rank will receive the VIPA address. The console log on RA28 confirmed this, as shown in Figure 4-27.

```
EZZ8301I VIPA 172.16.251.3 TAKEN OVER FROM TCPIPC ON RA03
EZZ4323I CONNECTION TO 172.16.233.3 CLEARED FOR DEVICE RA03M
```

Figure 4-27 Console message on RA28 at VIPA takeover

Figure 4-28 shows an FTP client connection to TCPIPC on RA03 (address 172.16.251.3). Although stack TCPIPC on RA03 is not active, the connection is established with TCPIPC on RA28 (MVS28C), the stack that took over ownership of the VIPA.

```
C:\>ftp 172.16.251.3
Connected to 172.16.251.3.
220-FTPDC1 IBM FTP CS V2R10 at MVS28C, 19:47:45 on 2000-09-15.
220 Connection will close if idle for more than 5 minutes.
User (172.16.251.3:(none)): claudia
331 Send password please.
Password:
230 CLAUDIA is logged on.  Working directory is "CLAUDIA.".
ftp>
```

Figure 4-28 FTP connection established after stopping TCPIPC stack on RA03

We then issued some TCP/IP display commands at RA28 to check the new status. Figure 4-29 shows the VIPA address 172.16.251.3 is active on RA28.

```
RO RA28,D TCPIP,TCPIPC,SYSPLEX,VIPADYN
D TCPIP,TCPIPC,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V2R10 153
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA28
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPABACKUP
  TCPNAME MVSNAME STATUS RANK ADDRESS MASK NETWORK PREFIX DIST
  ------- ------- ------ ---- ------------ ------------- ----
  TCPIPC  RA28    ACTIVE     255.255.255.0 172.16.251.0
IPADDR: 172.16.251.28  LINKNAME: VIPLAC10FB1C
  ORIGIN: VIPADEFINE
  TCPNAME MVSNAME STATUS RANK ADDRESS MASK NETWORK PREFIX DIST
  ------- ------- ------ ---- ------------ ------------- ----
  TCPIPC  RA28    ACTIVE     255.255.255.0 172.16.251.0
2 OF 2 RECORDS DISPLAYED
```

Figure 4-29 Display NETSTAT,VIPADYN after stopping TCPIPC on RA03

Figure 4-30 on page 105 shows the VIPA address 172.16.251.3 was added to stack TCPIPC on RA28.

```
RO RA28,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 161
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.101.28    M282216B         P
172.16.233.28    EZASAMEMVS
172.16.233.28    EZAXCF39
172.16.233.28    EZAXCF03
172.16.251.28    VIPLAC10FB1C
172.16.251.3     VIPLAC10FB03
127.0.0.1        LOOPBACK
7 OF 7 RECORDS DISPLAYED
```

Figure 4-30 Display NETSTAT,HOME after stopping TCPIPC on RA28

After some time, we restarted TCPIPC on RA03 and started a new connection to VIPA address 172.16.251.3. Since we defined this VIPA as MOVE WHENIDLE, this VIPA was not taken back for TCPIPC on RA03 because there were some active connections to 172.16.251.3 on stack TCPIPC on RA28. Figure 4-31 shows a new FTP connection to VIPA address 172.16.251.3 and shows that this connection was established with RA28, since the VIPA has not yet been taken back.

```
C:\>ftp 172.16.251.3
Connected to 172.16.251.3.
220-FTPDC1 IBM FTP CS V2R10 at MVS28C, 20:06:39 on 2000-09-15.
220 Connection will close if idle for more than 5 minutes.
User (172.16.251.3:(none)): claudia
331 Send password please.
Password:
230 CLAUDIA is logged on.  Working directory is "CLAUDIA.".
ftp>
```

Figure 4-31 FTP connection after restarting TCPIPC on RA03

Figure 4-32 shows active connections to VIPA address 172.16.251.3 on TCPIPC on RA28.

```
O RA28,D TCPIP,TCPIPC,N,SOCKETS
D TCPIP,TCPIPC,N,SOCKETS
EZZ2500I NETSTAT CS V2R10 TCPIPC 279
SOCKETS INTERFACE STATUS:
TYPE   BOUND TO          CONNECTED TO       STATE    CONN
NAME: BPXOINIT  SUBTASK: 006ECB58
STREAM 172.16.251.3..21  9.24.106.64..1234  ESTBLSH  00000678
STREAM 172.16.251.3..23  9.24.106.64..1233  ESTBLSH  00000671
STREAM 172.16.251.28..23 9.24.106.64..1220  ESTBLSH  00000420
```

Figure 4-32 Display NETSTAT,SOCKETS after restarting TCPIPC on RA03

Figure 4-33 on page 106 shows the VIPA address 172.16.151.3 is still active on stack TCPIPC on RA28.

```
   RO RA03,D TCPIP,TCPIPC,SYSPLEX,VIPADYN
   D TCPIP,TCPIPC,SYSPLEX,VIPADYN
   EZZ8260I SYSPLEX CS V2R10 927
   VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
   IPADDR: 172.16.251.3
     ORIGIN: VIPADEFINE CONTENTION
     TCPNAME MVSNAME STATUS RANK ADDRESS MASK NETWORK PREFIX DIST
     ------- ------- ------ ---- ------------ ------------- ----
     TCPIPC  RA28    ACTIVE     255.255.255.0 172.16.251.0
     TCPIPC  RA03    BACKUP 255
   IPADDR: 172.16.251.28
     ORIGIN: VIPABACKUP
     TCPNAME MVSNAME STATUS RANK ADDRESS MASK NETWORK PREFIX DIST
     ------- ------- ------ ---- ------------ ------------- ----
     TCPIPC  RA28    ACTIVE     255.255.255.0 172.16.251.0
     TCPIPC  RA03    BACKUP 100
4 OF 4 RECORDS DISPLAYED
```

Figure 4-33 Display SYSPLEX,VIPAD after restarting TCPIPC on RA03

Figure 4-34 shows that VIPA address 172.16.251.3 is not deleted from TCPIPC on RA28 and is not added to TCPIPC on RA03.

```
RO RA28,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 161
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.101.28    M282216B         P
172.16.233.28    EZASAMEMVS
172.16.233.28    EZAXCF39
172.16.233.28    EZAXCF03
172.16.251.28    VIPLAC10FB1C
172.16.251.3     VIPLAC10FB03
127.0.0.1        LOOPBACK
7 OF 7 RECORDS DISPLAYED

RO RA03,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 634
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.100.3     M032216B         P
172.16.233.3     EZASAMEMVS
172.16.233.3     EZAXCF28
172.16.233.3     EZAXCF39
127.0.0.1        LOOPBACK
5 OF 5 RECORDS DISPLAYED
```

Figure 4-34 Display NETSTAT,HOME after restarting TCPIPC on RA03

After these displays, we closed all connections to VIPA IP address 172.16.251.3, which is active on TCPIPC on RA28. At this moment, the VIPA IP address 172.16.251.3 is taken back to the original stack TCPIPC on RA03 image. The console log on RA28 and RA03 (the first message is issued on RA28 and the second one is issued on RA03) confirmed this as shown in Figure 4-35 on page 107.

```
EZZ8303I VIPA 172.16.251.3 GIVEN TO TCPIPC ON RA03
EZZ8301I VIPA 172.16.251.3 TAKEN OVER FROM TCPIPC ON RA28
```

Figure 4-35 Console message on RA28 and RA03 at VIPA takeback

Then, we issued the NETSTAT,HOME command and verified that the VIPA IP address 172.16.251.3 was added to TCPIPC on RA03 again and deleted from TCPIPC on RA28 as shown in Figure 4-36.

```
RO RA03,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 120
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.100.3     M032216B         P
172.16.233.3     EZASAMEMVS
172.16.233.3     EZAXCF28
172.16.233.3     EZAXCF39
172.16.251.3     VIPLAC10FB03
127.0.0.1        LOOPBACK
6 OF 6 RECORDS DISPLAYED

RO RA28,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 260
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.101.28    M282216B         P
172.16.233.28    EZASAMEMVS
172.16.233.28    EZAXCF39
172.16.233.28    EZAXCF03
172.16.251.28    VIPLAC10FB1C
127.0.0.1        LOOPBACK
6 OF 6 RECORDS DISPLAYED
```

Figure 4-36 Display NETSTAT,HOME after takeback

Figure 4-37 on page 108 shows the VIPA address 172.16.251.03 is active on TCPIPC on RA03 again.

```
RO RA03,D TCPIP,TCPIPC,SYSPLEX,VIPAD
D TCPIP,TCPIPC,SYSPLEX,VIPAD
EZZ8260I SYSPLEX CS V2R10 207
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME MVSNAME STATUS RANK ADDRESS MASK  NETWORK PREFIX DIST
  ------- ------- ------ ---- ------------  ------------- ----
  TCPIPC  RA03    ACTIVE      255.255.255.0 172.16.251.0
  TCPIPC  RA28    BACKUP 200
IPADDR: 172.16.251.28
  ORIGIN: VIPABACKUP
  TCPNAME MVSNAME STATUS RANK ADDRESS MASK  NETWORK PREFIX DIST
  ------- ------- ------ ---- ------------  ------------- ----
  TCPIPC  RA28    ACTIVE      255.255.255.0 172.16.251.0
  TCPIPC  RA03    BACKUP 100
4 OF 4 RECORDS DISPLAYED
```

Figure 4-37 Display SYSPLEX,VIPAD after takeback

Figure 4-38 shows a new connection to VIPA address 172.16.251.3, which is established on TCPIPC on RA03.

```
C:\>ftp 172.16.251.3
Connected to 172.16.251.3.
220-FTPDC1 IBM FTP CS V2R10 at MVS03C, 20:26:53 on 2000-09-15.
220 Connection will close if idle for more than 5 minutes.
User (172.16.251.3:(none)): claudia
331 Send password please.
Password:
230 CLAUDIA is logged on.  Working directory is "/u/claudia".
ftp>
```

Figure 4-38 FTP connection after takeback

5

Sysplex Distributor

Sysplex Distributor is a function introduced in IBM Communications Server for OS/390 V2R10 IP that takes the XCF dynamics and Dynamic VIPA support to a whole new level in terms of availability and workload balancing in a sysplex. Workload can be distributed to multiple server instances within the sysplex without requiring changes to clients or networking hardware and without delays in connection setup. CS for z/OS IP provides the way to implement a dynamic VIPA as a single network-visible IP address for a set of hosts that belong to the same sysplex cluster. Any client located anywhere in the IP network is able to see the sysplex cluster as one IP address, regardless of the number of hosts that it includes.

With Sysplex Distributor, clients receive the benefits of workload distribution provided by both Workload Manager (WLM) and Quality of Service (QoS) Policy Agent. In addition, Sysplex Distributor ensures high availability of the IP applications running on the sysplex cluster, no matter if one physical network interface fails or an entire IP stack or z/OS is lost.

5.1 Static VIPA and Dynamic VIPA overview

The concept of virtual IP address (VIPA) was introduced by IBM to remove the dependencies of other hosts on particular network attachments to CS for z/OS IP. Prior to VIPA, other hosts were bound to one of the home IP addresses and, therefore, to a particular network interface. If the *physical* network interface failed, the home IP address became unreachable and all the connections already established with this IP address also failed. VIPA provides a *virtual* network interface with a *virtual* IP address that other TCP/IP hosts can use to select a z/OS IP stack without choosing a specific network interface on that stack. If a specific physical network interface fails, the VIPA address remains reachable by other physical network interfaces. Hosts that connect to z/OS IP applications can send data to a VIPA address via whatever path is selected by the dynamic routing protocol (such as RIP or OSPF).

A VIPA is configured the same as a normal IP address for a physical adapter, except that it is not associated with any particular interface. VIPA uses a virtual device and a virtual IP address. The virtual IP address is added to the home address list. The virtual device defined for the VIPA using DEVICE, LINK and HOME statements is always active and never fails. Moreover, the z/OS IP stack advertises routes to the VIPA address as if it were one hop away and has reachability to it.

To an attached router, the IP stack in z/OS simply looks like another router. When the IP stack receives a packet destined for the VIPA, the inbound IP function of the stack notes that the IP address of the packet is in the stack's home list and forwards the packet up the stack. Assuming that the IP stack has more than one network interface, if a particular network interface fails, the downstream router will simply route VIPA-targeted packets to the stack via an alternate route. In other words, the destination IP stack on z/OS is still reachable and it looks like another intermediate node. The VIPA may thus be thought of as an address of the stack and not of any particular network interface associated with the stack.

While VIPA certainly removes the dependency on any particular network interface as a single point of failure, the connectivity of a server can still be lost when a single stack or a z/OS image fails. When this occurs, we could manually move a VIPA to another stack using the OBEY command (or semi-automatically using any system management automation of the manual process). This type of VIPA is not viewed as attractive, since the process is inherently manual. As a result, an automatic VIPA movement and activation mechanism were added.

Dynamic VIPA was introduced by SecureWay Communications Server for OS/390 V2R8 IP to enable the dynamic activation of a VIPA as well as the automatic movement of a VIPA to another surviving z/OS image after a z/OS stack failure. There are two forms of Dynamic VIPA, both of which can be used for takeover functionality:

- *A*utomatic VIPA takeover allows a VIPA address to move automatically to a stack (called a backup stack) where an existing suitable application instance is already active and allows the application to serve the client formerly going to the failed stack.
- Dynamic VIPA activation for an application server allows an application to create and activate VIPA so that the VIPA moves when the application moves.

Nondisruptive, immediate, automatic VIPA takeback was introduced by IBM Communications Server for OS/390 V2R10 to move the VIPA back to where it originally belongs once the failed stack has been restored. This takeback is nondisruptive to existing connections with the backup stack and the takeback is not delayed until all connections with the backup stack have terminated (as was the case with CS for OS/390 V2R8 IP). New connections will be handled by the new (original) primary owner, thereby allowing the workload to move back to the original stack.

5.2 What is Sysplex Distributor?

Sysplex Distributor was designed to address the requirement of one single network-visible IP address for the sysplex cluster and let the clients in the network receive the benefits of workload distribution and high availability within the sysplex cluster. With Sysplex Distributor, client connections seem to be connected to a single IP host even if the connections are established with different servers in the same sysplex cluster.

Because the Sysplex Distributor function resides on a system in the sysplex itself, it has the ability to factor "real-time" information concerning the multiple server instances including server status as well as QoS and Policy information provided by CS for z/OS IP’s Policy Agent. By combining these "real-time" factors with the information from WLM, the Sysplex Distributor has the unique ability to ensure that the best destination server instance is chosen for a particular client connection. The Sysplex Distributor has more benefits than other load-balancing implementations, such as the Network Dispatcher or DNS/WLM. Their limitations are removed with Sysplex Distributor.

In summary, the benefits of Sysplex Distributor include:

1. Removes configuration limitations of Network Dispatcher

   Target servers can use XCF (or Hipersockets) links between the distributing stack and target servers as opposed to LAN connections such as an OSA.

2. Removes dependency of specific hardware in WAN

   Provides a total CS for z/OS IP solution for workload distribution.

3. Provides real-time workload balancing for TCP/IP applications

   Even if clients cache the IP address of the server (a common problem for DNS/WLM).

4. Enhances VIPA takeover and takeback support

   – Allows for nondisruptive takeback of VIPA original owner to get workload where it belongs.

   – Distributing function can be backed up and taken over.

5. Enhances Dynamic VIPA support

   Nondisruptive application server instance movement.

6. Has immediate visibility of new server instances or loss of a server instance.

   Target stacks are aware of when an application instance has a listening socket that could accept connections via a distributed DVIPA, or when such a socket is closed (or the application instance terminates for any reason). The target stacks notify the routing stack proactively, and there is no need for application advisors on the routing stack.

In summary, Sysplex Distributor provides a single network-visible IP address of a sysplex cluster service. One IP address can be assigned to the entire sysplex cluster (usually for each service provided, such as Telnet):

- Sysplex Distributor will query the Policy Agent to find if there exists any policy defined for routing the incoming connection requests.
- WLM and QoS policy can be specified for workload balancing in real time on every new connection request.

The specific profile statements that should be used to configure Sysplex Distributor will be detailed later in 5.4, “Sysplex Distributor implementation” on page 116.

5.2.1 Sysplex Distributor functionality

Let us consider the scenario depicted in Figure 5-1 on page 113. This includes four CS for OS/390 V2R10 IP stacks running in the same sysplex cluster in GOAL mode (WLM GOAL mode). All of them have SYSPLEXROUTING, DATAGRAMFWD, and DYNAMICXCF configured. Let us assume that:

- H1 is configured as the distributing IP stack with V1 as the Dynamic VIPA (DVIPA) assigned to the sysplex cluster.
- H2 is configured as backup for V1.
- H3 and H4 are configured as secondary backups for V1.
- Let us suppose that APPL1 is running in all the hosts that are members of the same sysplex cluster. Note that the application could also be running in two or three of the hosts or in all of them at the same time.

With this in mind, we describe how Sysplex Distributor works:

1. When IP stack H1 is activated, the definitions for the local XCF1 link are created dynamically due to DYNAMICXCF being coded in the H1 profile. Through this new link, H1 recognizes the other IP stacks that belong to the same sysplex cluster and their XCF associated links: XCF2, XCF3, and XCF4.
2. The DVIPA assigned to the sysplex cluster and the application ports that this DVIPA serves are read from the VIPADISTRIBUTE statement in the profile data set. An entry in the home list is added with the distributed IP address in all the IP stacks. The home list entry on the target stacks is actually done with a message that H1 sends to all the stacks read from the VIPADISTRIBUTE statement. Only one stack advertises the DVIPA through the RIP or OSPF routing protocol. In this case it is the one that resides in H1, the host in charge of load distribution.
3. H1 monitors whether there is at least one application (APPL1 in Figure 5-1 on page 113) with a listening socket for the designated port and DVIPA. Actually H2, H3, and H4 will send a message to H1 when a server (in our case APPL1) is bound to either INADDR_ANY or specifically to the DVIPA (and, of course, the designated port). With that information H1 builds a table with the name of the application and the IP stacks that could serve any connection request for it. The table matches the application server listening port with the target XCF IP address.
4. When a client in the network requests a service from APPL1, the DNS resolves the IP address for the application with the DVIPA address. This DNS could be any DNS in the IP network and does not need to register with WLM.
5. As soon as H1 receives the connection request (TCP segment with the SYN flag), it queries WLM and/or QoS to select the best target stack for APPL1 and forwards the SYN segment to the chosen target stack. In our example, it is APPL1 in H4 that best fits the request.
6. One entry is created in the connection routing table (CRT) in H1 for this new connection with XCF4 as the target IP address. H4 also adds the connection to its connection routing table.

   Note: If a program binds to DVIPA on H4 and initiates a connection, H4 needs to send a message to H1, so H1 can update its connection routing table accordingly. As an example, this is used when the FTP server on H4 would initiate a data connection (port 20) to a client.

7. The H1 IP stack will forward subsequent incoming data for this connection to the correct target stack.

8. When the H4 IP stack decides that the connection no longer exists, it informs the H1 IP stack with a message so H1 can remove the connection from its connection routing table.

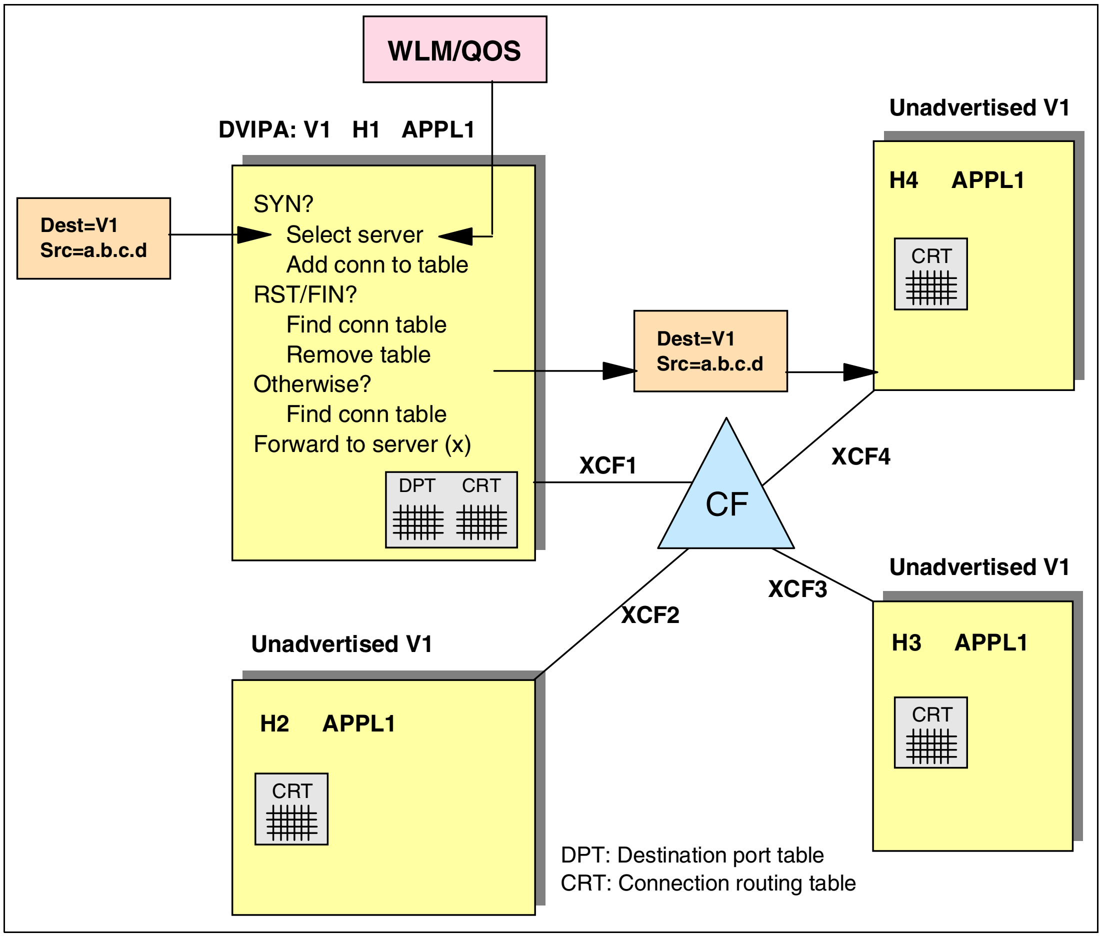


Figure 5-1 Sysplex Distributor functionality

5.2.2 Backup capability

Let us say that the scenario depicted has been running for some time without problems. The new APPL1 connections have been distributed according to WLM and/or QoS to H1, H2, H3, and H4. Suppose that a considerable amount of connections are currently established between several APPL1 server images and clients in the IP network. What would happen if we had a major failure in our distributing IP stack, H1?

Automatic Dynamic VIPA takeover was introduced in SecureWay Communications Server for OS/390 V2R8 IP. This function allows a VIPA address to automatically move from one IP stack where it was defined to another one in the event of the failure of the first. The VIPA address remains active in the IP network, allowing clients to access the services associated with it.

In IBM Communication Server for OS/390 V2R10, this VIPA takeover functionality was enhanced to support Sysplex Distributor. Consider the scenario described in Figure 5-2 on page 114.

H1 is the distributing IP stack and H2 is the primary VIPABACKUP IP stack. When H1 fails (Figure 5-2):

1. All the IP connections terminating at H1 are lost.
2. The Sysplex Distributor connection routing table (CRT) is also lost.

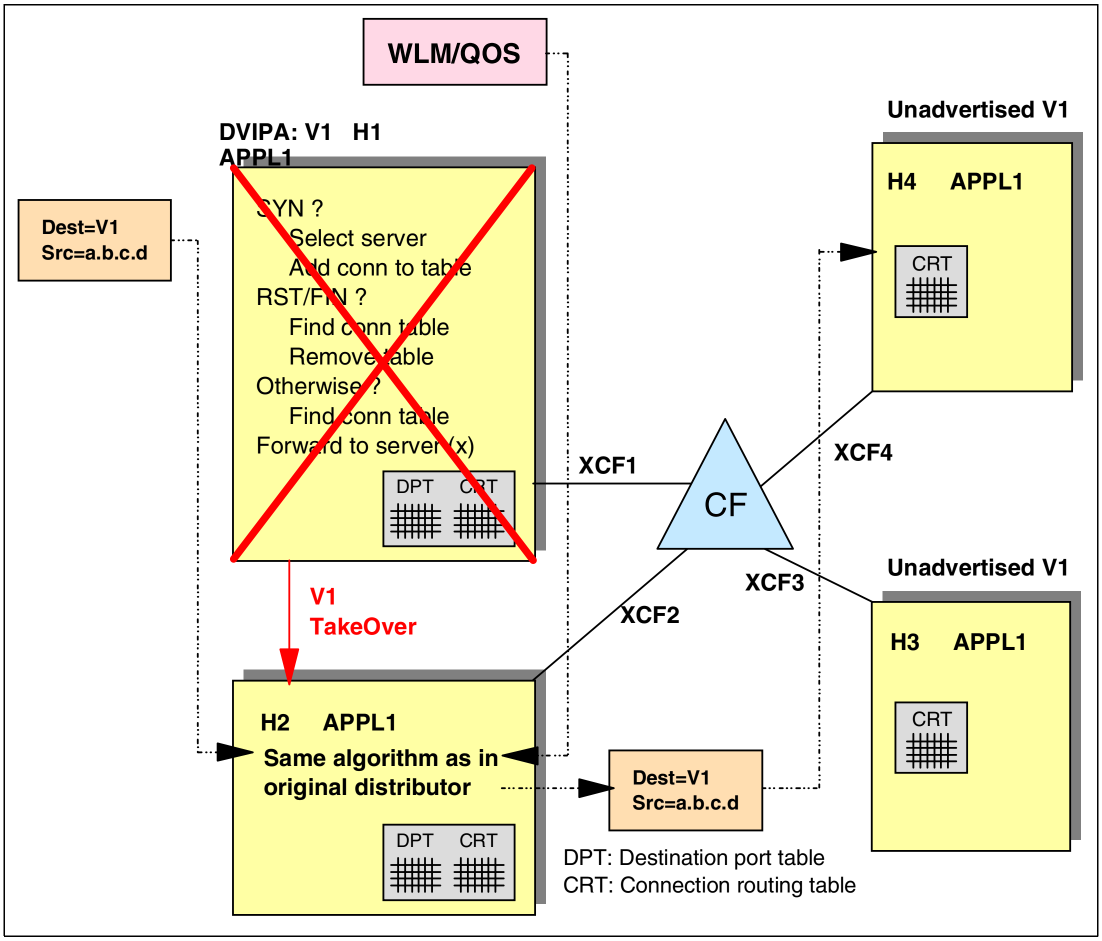


Figure 5-2 Sysplex Distributor and VIPA takeover

3. H2 detects that H1 is down and defines itself as the distributing IP stack for the VIPA.
4. Because H2 saved information about H1, it informs the other target stacks that it knows V1 is distributable.
5. H3 and H4 find out that H2 is the chosen backup for V1 and immediately send connection information regarding V1 to IP stack H2.
6. H2 advertises V1(DVIPA) through the dynamic routing protocol (RIP or OSPF). Retransmitted TCP segments for already existing connections or SYN segments for new connections are hereafter processed by IP stack H2 and routed by H2 to the appropriate target stacks.

Note: Only the IP connections with the failing IP stack were lost. All other connections remain allocated and function properly

5.2.3 Recovery

Once the H1 IP stack is activated again the process of taking back V1 to H1 is started. This process is nondisruptive for the IP connections already established with V1 regardless of which host is the owner at that time (in our example H2). In our example, connection information is maintained by H2. When H1 is re-activated, H2 sends its connection information to H1. This gives H1 the information it needs to once again distribute packets for existing connections to the correct stacks in the sysplex.

Connections with the backup host are not broken when the V1 address is taken back to H1, and takeback is not delayed until all connections with the backup host have terminated (Figure 5-3).

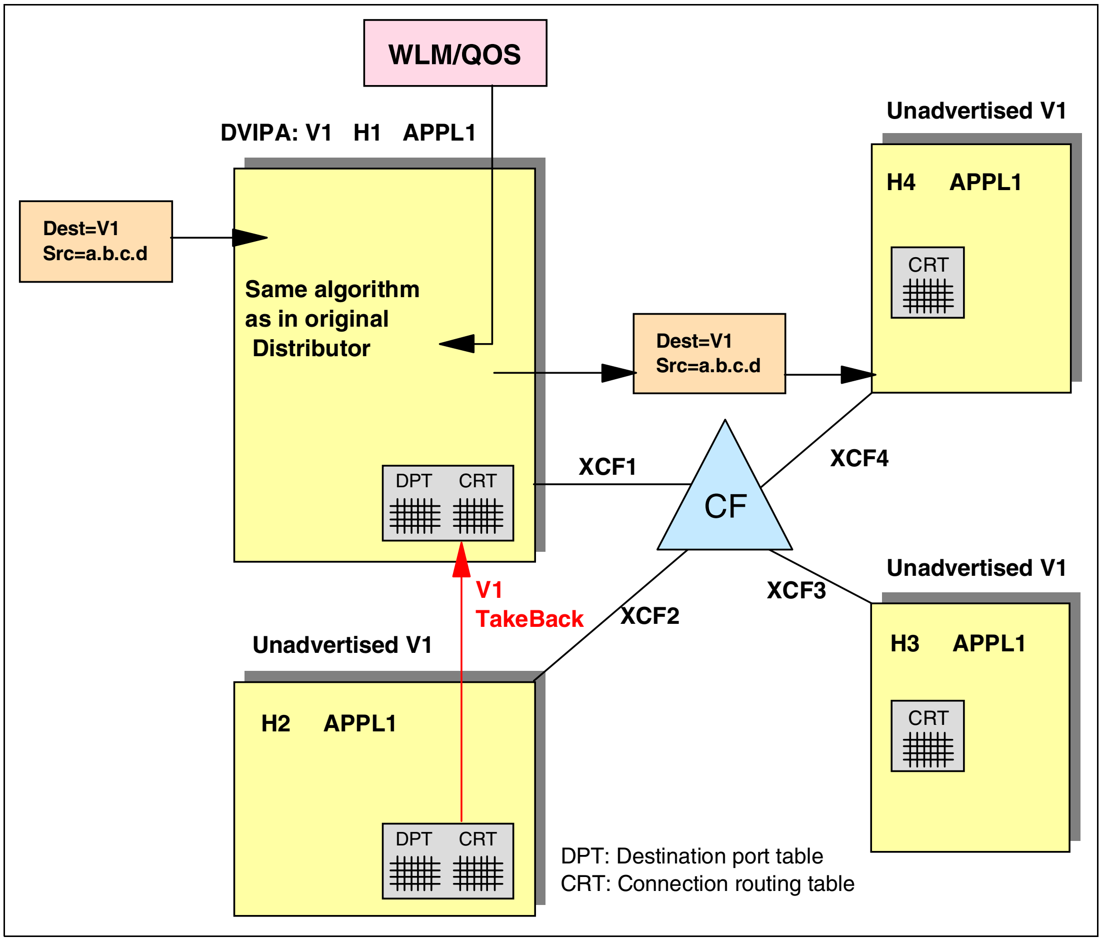


Figure 5-3 Sysplex Distributor and VIPA takeback

5.3 The role of dynamic routing with Sysplex Distributor

Routing IP packets for Sysplex Distributor can be divided into two cases: routing inside the sysplex cluster and routing through the IP network. Routing inside the sysplex cluster is accomplished by the distributing host. All incoming traffic (new connection requests and connection data) arrives first to the distributing stack. It forwards the traffic to the target applications, wherever they are in the sysplex cluster, through the XCF links. Here the routing process is done without considering any IP routing table. The WLM and QoS weights are the factors considered in target selection for new requests and the CRT is the consulted data structure for connection data. On the other hand, the outgoing traffic generated by the applications is routed considering the destination IP address and the routing table in each stack.

Routing outside the sysplex through the IP network is done by the downstream routers. Those routers learn about the DVIPA assigned to the sysplex dynamically using OSPF or RIP routing protocols. As a result, it is usually necessary to implement either one of these routing protocols in all the IP stacks of the sysplex cluster.

The distributing VIPA address is dynamically added to the home list of each IP stack participating in the Sysplex Distributor, but only one IP stack advertises the sysplex VIPA address to the routers: the one defined as the distributing IP stack. The other stacks do not advertise it and only the backup IP stack will do so if the distributing IP stack fails.

If ORouteD is being used, then the Dynamic VIPA support generates the appropriate BSDROUTINGPARMS statement.

If you are using OMPROUTE, you should consider the following as referenced in Figure 5-4:

- The names 2 of Dynamic VIPA interfaces are assigned dynamically by the stack when they are created. Therefore, the name coded for the OSPF_Interface statement in the Name 2 field will be ignored by OMPROUTE.
- It is recommended that each OMPROUTE server have an OSPF_Interface defined for each Dynamic VIPA address that the IP stack might own or, if the number of DVIPAs addresses is large, a wildcard should be used.

It is also possible to define ranges of dynamic VIPA interfaces using the subnet mask and the IP address on the OSPF_Interface statement. The range defined will be all the IP addresses that fall within the subnet defined by the mask and the IP address. The following example 1 defines a range of Dynamic VIPA addresses from 10.138.165.80 to 10.138.165.95:

```
OSPF_Interface
   IP_address = 10.138.165.80   1
   Name = dummy_name            2
   Subnet_mask = 255.255.255.240
```

Figure 5-4 Dynamic VIPA OSPF definition

For consistency with the VIPARANGE statement in the TCPIP.PROFILE, any value that may fall within this range can be used with the mask to define a range of dynamic VIPAs.

For additional information on routing, please see *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 4: Connectivity and Routing*, SG24-6516.

5.4 Sysplex Distributor implementation

The implementation of Sysplex Distributor is very straightforward. The TCP/IP configuration that needs to take place is minimal compared to other connection dispatching technologies. For the most part, the sysplex environment enables the dynamic establishment of links and the dynamic coordination between stacks in the cluster.

5.4.1 Requirements

The IPCONFIG DATAGRAMFWD and DYNAMICXCF statements must be coded in the TCPIP.PROFILE data set in all the IP stacks of the sysplex cluster.

If you want to implement a WLM-based distribution, you have to register all IP stacks participating in the sysplex with WLM coding SYSPLEXROUTING in each IP stack. Also verify that all the participating CS for z/OS IP images are configured for WLM GOAL mode.

To enable the distributing IP stack to forward connections based upon a combination of workload information and network performance information, configure all the participating stacks for WLM GOAL mode. Specify SYSPLEXROUTING in the IPCONFIG statement in all the participating stacks and also define a Sysplex Distributor Performance Policy on the target stack with the Policy Agent. Otherwise, If SYSPLEXROUTING is not coded in any IP stack, the distribution for incoming connections to the target applications will be random.

For those z/OS images that are running more than one IP stack, the recommended way to define XCF and IUTSAMEH links is to use the IPCONFIG DYNAMICXCF. In fact, IUTSAMEH links should not be specified if the IP stack is participating in Sysplex Distributor.

For any IP application that uses both control and data ports, both port numbers must be distributed by the same Dynamic VIPA address (for example FTP).

5.4.2 Limitations

FTP passive mode should not be used with Sysplex Distributor, as documented in *z/OS V1R2.0 CS: IP User's Guide and Commands,* SC31-8780. If the host for the secondary FTP server has the Sysplex Distributor function distributing FTP server workload, then the client should not use the proxy subcommand. The PASV command that is used by the proxy subcommand to allow the secondary FTP server to be the passive side of a data connection cannot be handled properly at the secondary FTP server host. Note that Sysplex Distributor cannot work with FTP passive mode, because the FTP server uses ephemeral ports for the passive connection and ephemeral ports cannot be distributed by Sysplex Distributor function. Note that this restriction is removed in V1R4.

Figure 5-5 on page 118 and the steps below show a short example:

1. The client that requests the passive mode connection to allow the FTP server to be the passive side of the data connection is using the DVIPA V1 of the Sysplex Distributor.
2. Sysplex Distributor in conjunction with WLM selects the target stack.
3. Sysplex Distributor forwards this control connection request to port 21 on the target stack. The FTP server recognizes the request for passive mode and selects an ephemeral port 1428 for the data connection.
4. This information of the ephemeral port is sent to the client over the control connection.
5. The client will use this ephemeral port (1428) to establish the data connection using the DVIPA V1 of the Sysplex Distributor.
6. But the Sysplex Distributor will reject that request, since it is not aware of such a port number. Remember that we have to define on the VIPADISTRibute statement the ports for which we are to distribute workload.

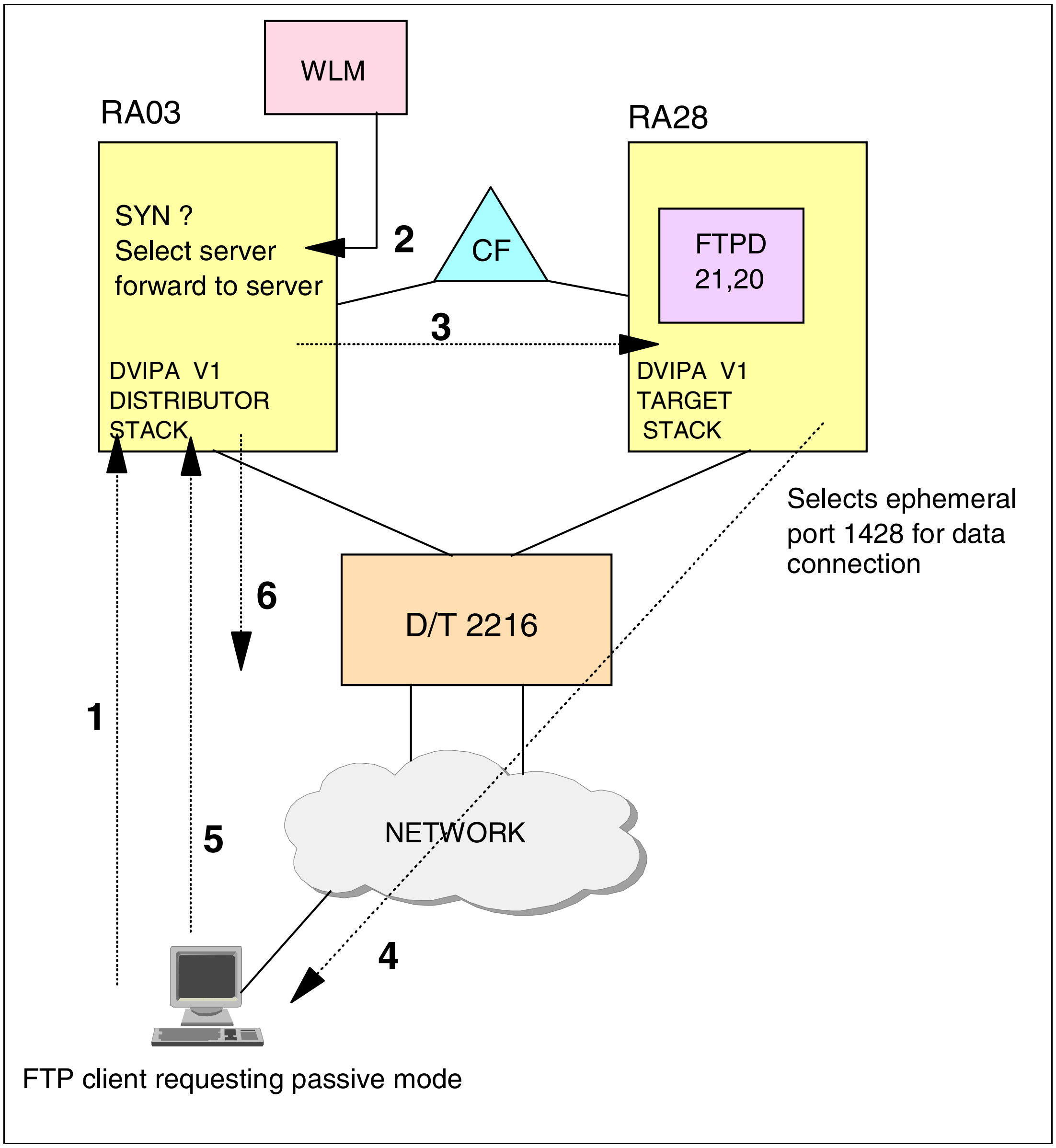


Figure 5-5 FTP passive mode limitations for Sysplex Distributor

During our tests we found another limitation that should be mentioned. If the Sysplex Distributor distributing stack is distributing workload for one port and not for some other (no VIPADISTribute statement for it is defined), the Sysplex Distributor stack will not allow the connection to that port even if a local application is listening on it.

Consider the example shown in Figure 5-6 on page 119:

- Assume that we want to distribute only TN3270E services to the participating stacks RA03,RA28, and RA39. For that we code the following statements:

```
VIPADYNAMIC
 VIPADEFINE MOVE IMMED 255.255.255.0 172.16.251.3
 VIPADISTRIBUTE 172.16.251.3 PORT 23 DESTIP ALL
ENDVIPADYNAMIC
```

- We have only defined one DVIPA to distribute the workload.

- If we now try to connect to the FTP or the Web server on the distributing stack RA03 using the same DVIPA, the Sysplex Distributor stack disallows access to these ports even though the distributing stack has these applications currently active. The connection requests for these applications time out.

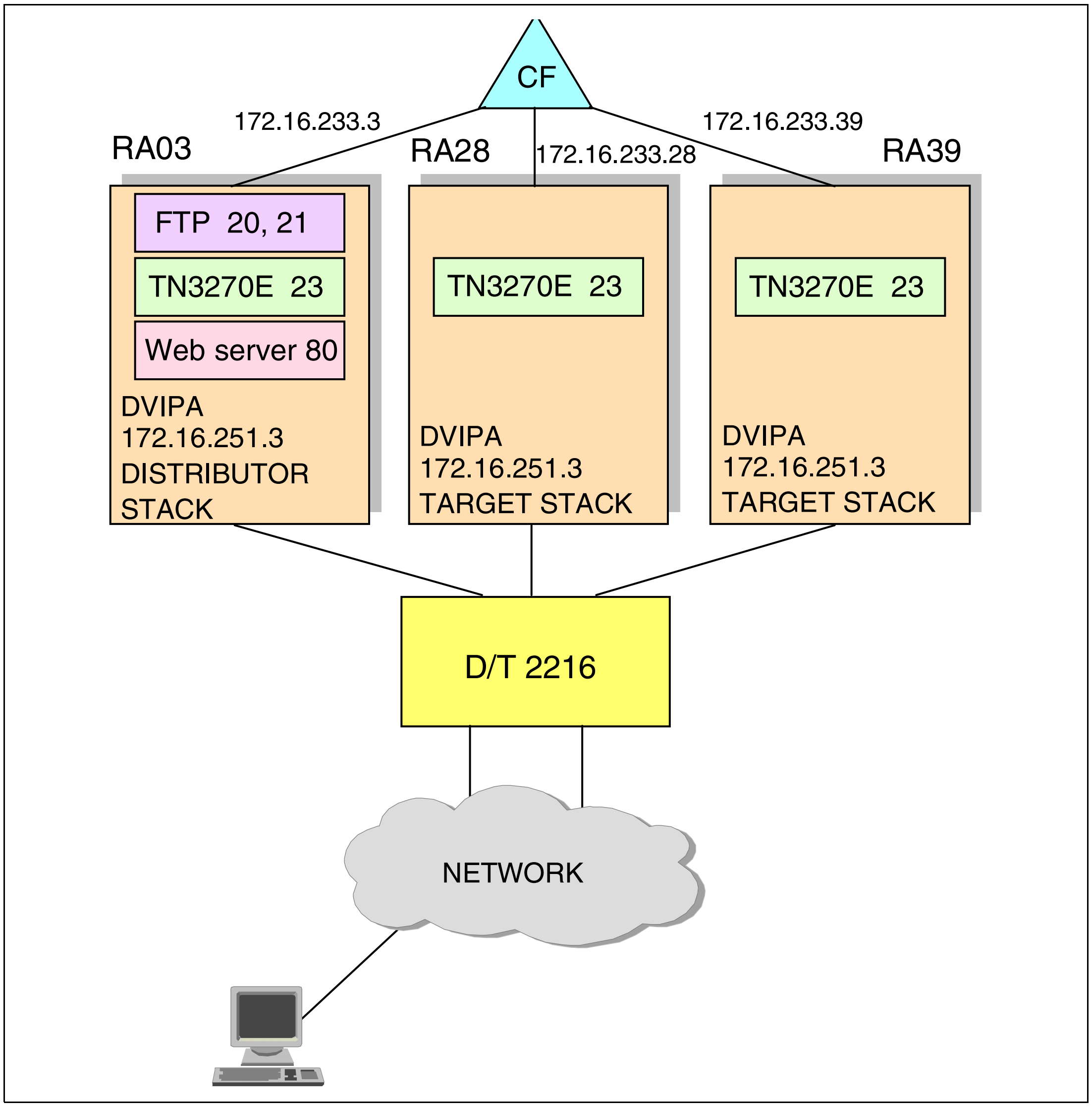


Figure 5-6 Sysplex Distributor limitation

There are actually two bypasses to solve this limitation:

1. We could code another VIPADISTribute statement for the IP service (port) in question (FTP in our case) and explicitly define on the DESTIP keyword the <dynxcfip> address to which target stack it should be distributed (the local, distributing stack). However, note that you can only define four distributed ports per DVIPA. In our case, we would define this for the Web server using the following VIPADISTribute statement:

```
VIPADYNAMIC
 VIPADEFINE MOVE IMMED 255.255.255.0 172.16.251.3
 VIPADISTRIBUTE 172.16.251.3 PORT 23 DESTIP ALL
 VIPADISTRIBUTE 172.16.251.3 PORT 80 DESTIP 172.16.233.3
ENDVIPADYNAMIC
```

   The configuration for the FTP server would be similar.

2. An alternative is to code one DVIPA for every different IP service (port). The limit of DVIPAs per stack has been raised to 256, thereby effectively eliminating this concern. However, you would have to use different host names/IP addresses to connect to the applications, which may pose a problem in already implemented environments in which the same DVIPA is already being used for multiple services.

5.4.3 Implementation

The following list shows what is needed to implement Sysplex Distributor:

1. Choose which IP stack is going to execute the Sysplex Distributor distributing function.
2. Select which IP stacks are going to be the backup stack for the Sysplex Distributor stack and in which order.
3. Ensure that WLM GOAL mode is enabled in all the LPARs participating in the Sysplex Distributor.
4. Enable sysplex routing in all the IP stack participating in the Sysplex Distributor with the SYSPLEXROUTING statement.
5. For those IP stacks that are active under a multi-stack environment, the SAMEHOST links have to be created dynamically. In general, code DYNAMICXCF in all the IP stacks participating in the Sysplex Distributor.

   Note: For Sysplex Distributor you cannot specify the XCF address using the IUTSAMEH DEVICE, LINK, and HOME statements. XCF addresses must be allowed to be dynamically created by coding the IPCONFIG DYNAMICXCF statement.

6. Code DATAGRAMFWD in all IP stacks participating in the Sysplex Distributor.
7. Select, by port numbers, the applications that are going to be distributed using the Sysplex Distributor function. Note that if the application chosen requires data and control ports, both ports have to be considered.
8. Code the VIPADYNAMIC/ENDVIPADYNAMIC block for the distributing IP stack:
   – Define the dynamic VIPA associated to the distributing IP stack with the VIPADEFINE statement.
   – Associate the sysplex Dynamic VIPA to the application's port number with the VIPADISTRIBUTE statement.
9. Code VIPADYNAMIC/ENDVIPADYNAMIC block for the distributor's backup IP stack:
   – Define the IP stack as backup for the sysplex DVIPA address with the VIPABACKUP statement.

Figure 5-7 on page 121 shows the VIPA configuration statements. We show the syntax of the VIPADEFINE, VIPABACKUP, and VIPADISTRIBUTE statements. VIPADYNAMIC and ENDVIPADYNAMIC define the block of statements related to Dynamic VIPAs and Sysplex Distributor. For a complete description of these statements, please refer to *z/OS V1R2.0 CS: IP Configuration Reference,* SC31-8776.

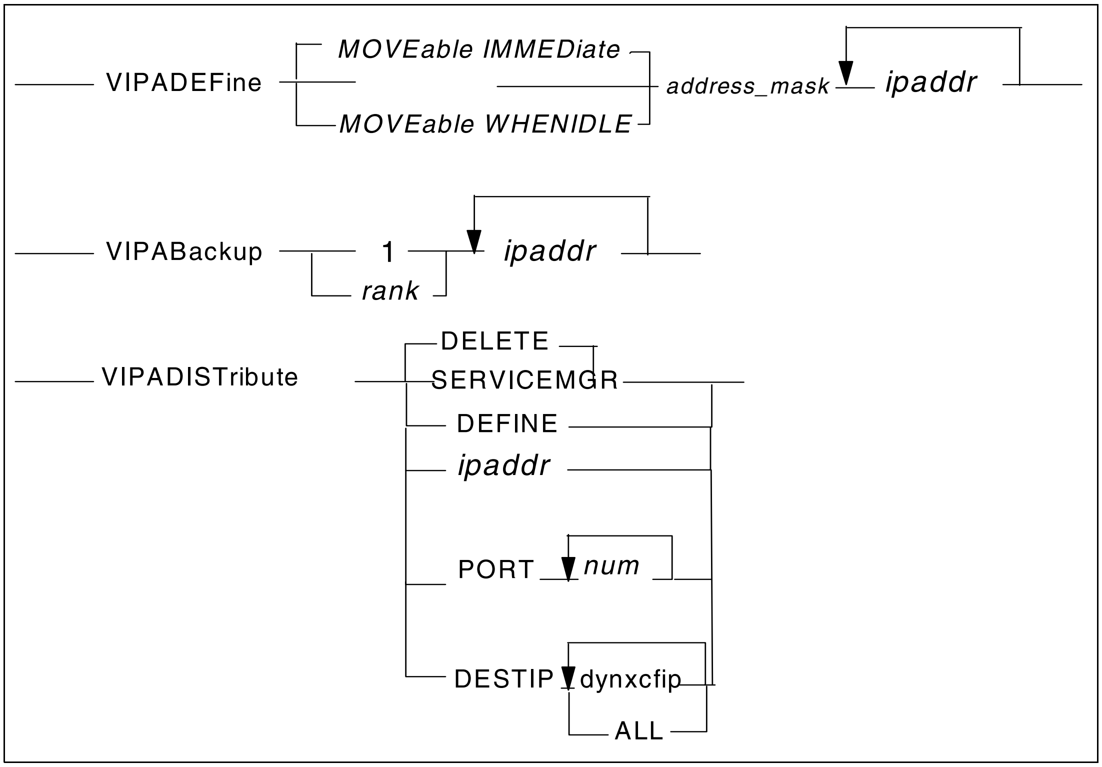


Figure 5-7 VIPADEFINE, VIPABACKUP, and VIPADISTRIBUTE syntax

VIPADEFINE designates one or more Dynamic VIPAs that this IP stack should initially own and support. The parameter MOVEable IMMEDiate indicates that this Dynamic VIPA can be moved to the original owning stack as soon as it re-initializes after a failure. On the other hand, MOVEable WHENIDLE indicates that this Dynamic VIPA can be moved back to the original owning stack only after the backup has no outstanding connections. In this case, new connection requests will continue to be directed to the current owning stack (the backup).

VIPABACKUP designates one or more Dynamic VIPAs for which this IP stack will provide automatic backup if the owning stack fails. The option *rank* specifies the intended backup order; the stack with the highest rank will be used first.

VIPADISTRIBUTE enables (VIPADISTRIBUTE DEFINE) or disables (VIPADISTRIBUTE DELETE) the Sysplex Distributor function on a Dynamic VIPA. PORT limits the scope of this VIPADISTRIBUTE to the specified port number (up to four ports can be coded for the same Dynamic VIPA).

DESTIP specifies the Dynamic XCF addresses that are candidates to receive new incoming connection requests. DESTIP ALL means that all IP stacks in the sysplex that have defined a Dynamic XCF address are candidates for incoming connection requests for this DVIPA, including future stacks that are currently not active and running. Stacks are eligible to receive connections if they have at least one application instance listening on the specified port.

The way that Sysplex Distributor distributes the workload can be modified using the Policy Agent. Please refer to 5.5, “Sysplex Distributor and policy” on page 122. For a detailed description about policy-based network management, refer to *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 6: Policy and Network Management*, SG24-6839.

5.5 Sysplex Distributor and policy

Policies are an administrative means to define controls for a network, in order to achieve the QoS levels promised by a given SLA or to implement security or resource balancing decisions. Quality of Service policy allows classification of IP traffic by application, user group, time of day, and assignment of relative priority. The Policy Agent reads policy entries from a flat file called pagent.conf that can be located in any MVS or HFS file or from an LDAP server or both. The Sysplex Distributor uses these policies to limit the target stack to route its work in conjunction with the WLM weights. Note that the WLM has to run in GOAL mode at the target stacks, or the QoS weights will have no effect and the distribution of the work is random.

The following types of policies are supported in CS for z/OS IP:

Integrated Services Type of service that provides end-to-end QoS using resource reservations along a network path from sender to receiver. This service is provided by the RSVP Agent.

Differentiated Services Type of services that provides aggregate QoS to broad classes of traffic (for example, all FTP traffic).

Sysplex Distribution Policies specify to which target stack the Sysplex Distributor may route incoming connection requests.

Traffic Regulation Mgmt Policies define the maximum number of connections to a TCP port and control the number from a single host to this port.

Intrusion Detection Policies detect possible intrusion attacks.

The Policy Agent performs two distinct functions to assist the Sysplex Distributor:

1. Policies can be defined to control which stack the sysplex distributor routes traffic to. The definition of the outbound interface on the PolicyAction statement can limit the stacks to which work is distributed to a subset of those defined on the VIPADISTRIBUTE statement in the TCPIP.PROFILE. Using a policy, the stack to which work is distributed can vary, for example, based on time periods. Another possibility is to limit the number of SD target stacks for inbound traffic from a given subnet (Figure 5-8 on page 123).
2. The PolicyPerfMonitorForSDR statement in the pagent.conf file will activate the Policy Agent QoS performance monitor function. When activated, the Policy Agent will use data about packet loss and timeouts that exceed defined thresholds and derive a QoS weight fraction for that target stack. This weight fraction is then used to reduce the WLM weight assigned to the target stacks, so that the Sysplex Distributor stack can use this information to better direct workload to where network traffic is best being handled. This policy is activated on SD target stacks (Figure 5-8 on page 123).

To exclude stale data from target stacks where the Policy Agent has terminated, the Policy Agent sends a “heartbeat” to the SD distributing stack at certain intervals. The SD distributing stack deletes QoS weight fraction data from a target stack when the “heartbeat” has not been received within a certain amount of time.

At ITSO Raleigh, we configured SD policies in two ways. In one scenario, the Policy Agent extracts the policy information from a static configured HFS file (PAGENT file), and in another case, an LDAP server running in z/OS provides all policy information in the network.

The sysplex consists of three z/OS systems: RA03, RA28, and RA39. On each system, there is a TCP/IP stack named TCPIPC, which participates in Sysplex Distributor. The TCPIPC stack on RA03 takes the role of the distributing stack, and the stacks on RA39 and RA28 act as the primary and secondary backup respectively. An FTP server has been configured on each SD stack as an application served by SD.

We have defined an SD policy for FTP connections to install into the SD distributing stack, which is TCPIPC on RA03, and have defined an SD performance monitoring policy for the SD target stacks, which are TCPIPC on RA28 and RA39. Figure 5-8 illustrates the system environment at ITSO Raleigh.

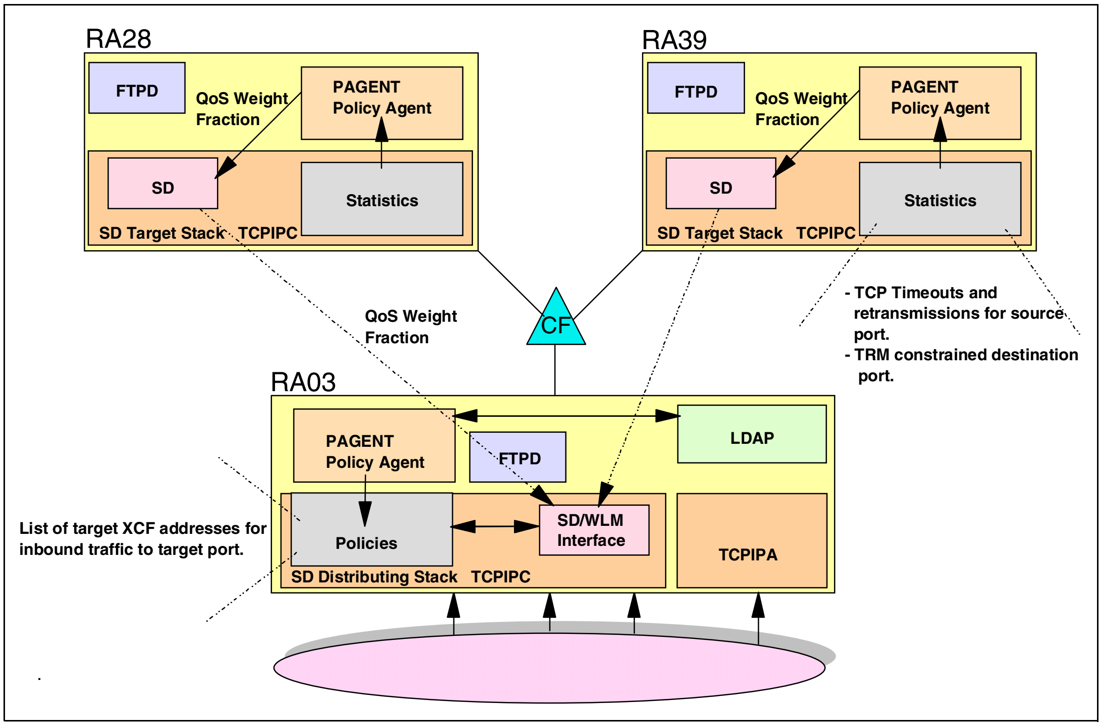


Figure 5-8 Sysplex Distributor policy implementation at ITSO Raleigh

5.5.1 Sysplex Distributor QoS policy in the PAGENT file

Please review the definition of the TCPIP.PROFILE for the stacks RA03, RA28 and RA39 in 5.4, "Sysplex Distributor implementation" on page 116. We did not change anything in the TCPIP.PROFILE to run the QoS policy for Sysplex Distributor.

We defined the SD policies that limit the number of SD target stacks for inbound traffic on the SD target stack, and the SD performance monitoring policies on all the participating stacks. For SD performance monitoring, the traffic to be monitored has to be represented by at least one Differentiated Services policy defined for the target application.

All policies have been configured in the image configuration file, which is the second-level PAGENT configuration file, namely /etc/pagent.r2615c.conf.

You will find further information about SD policy and SD performance monitor policy in *z/OS V1R2.0 CS: IP Configuration Guide*, SC31-8775 and *z/OS V1R2.0 CS: IP Configuration Reference*, SC31-8776. More detailed information is also found in the redbook *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 6: Policy and Network Management*, SG24-6839.

We have the following SD policies configured in the SD distributing stack (TCPIPC on RA03):

```
#
#  IBM Communications Server for OS/390
#  SMP/E distribution path: /usr/lpp/tcpip/samples/IBM/EZAPAGCO
#
```

```
# LogLevel Statement
Loglevel 511

# PolicyPerfMonitorForSDR Statement
PolicyPerfMonitorForSDR  enable
{
  samplinginterval 60
  LossRatioAndWeightFr 20 25
  TimeoutRatioAndWeightFr 50 50
  LossMaxWeightFr 95
  TimeoutMaxWeightFr 100
}

# Policy Action statement
policyAction        ftpaction
{
    policyScope    DataTraffic
    outboundinterface  172.16.233.28                1
    outboundinterface  172.16.233.39                1
#    outboundinterface  0.0.0.0                     2
}

# Policy Rule statement
policyRule          ftprule
{
    ProtocolNumberRange 6
    DestinationPortRange         20 21              3
    SourceAddressRange  9.24.106.0 9.24.106.255     4
    policyactionreference   ftpaction               5
}
```

The policies are identified as SD policies by the presence of the Outboundinterface 1 attribute in the PolicyAction statement. You have to define to which SD target stacks incoming connection requests that map to this rule should be distributed. The target stacks are identified by the IP address of the dynamic XCF link. Up to 32 instances of this attribute can be specified. See 5.4, "Sysplex Distributor implementation" on page 116 for the DESTIP 1 address of the dynamic XCF link participating in Sysplex Distributor.

2 A value of zero can be specified for the interface, which indicates to the SD distributing stack that if it cannot distribute the request to a target stack on one of the other specified interfaces, then the request can be distributed to any of the other eligible target stacks.

Because we commented out the Outboundinterface 0.0.0.0 definition, SD will reject the request when neither of the target stacks is available.

In our implementation, the incoming FTP connection requests will be forwarded to either RA28 or RA39 for inbound traffic from a given 4 subnet, even though there is an FTP server running on RA03. For FTP and other applications that use a 3 control port and a data port, you always have to define both. Note that without this SD policy activated, an incoming FTP connection will be forwarded to either of three systems.

5 You always match the policyRule to the policyAction by the policyActionReference statement.

An additional possibility would be to activate this policy only at certain times, let’s say during normal working hours from Monday to Friday between 08:00 and 17:00, which we didn’t do in our implementation. But you would have to add only two statements to the policyRule definitions. Check the sample file /usr/lpp/tcpip/samples/pagent.conf for the correct syntax and explanation.

```
DayOfWeekMask      0111110
TimeOfDayRange     08:00-17:00
```

On the SD target stacks, the following policies have been activated:

```
#
#  IBM Communications Server for OS/390
#  SMP/E distribution path: /usr/lpp/tcpip/samples/IBM/EZAPAGCO
#
# LogLevel Statement
Loglevel 511

# PolicyPerfMonitorForSDR Statement
PolicyPerfMonitorForSDR  enable                       1
{
  samplinginterval 60                                 2
  LossRatioAndWeightFr 20 25                          3
  TimeoutRatioAndWeightFr 50 50                       4
  LossMaxWeightFr 95                                  5
  TimeoutMaxWeightFr 100                              6
}

# Policy Action statement
policyAction        ftpaction                         7
{
    policyScope    DataTraffic                        8
    MaxConnections 50         # Limit FTP concurrent connections to 50.
    MaxRate        400        # Limit FTP connection throughput to 400
    OutgoingTOS    01000000   # the TOS value of outgoing FTP packets.
}

# Policy Rule statement
policyRule           ftprule                          9
{
    ProtocolNumberRange 6
    DestinationPortRange         20 21
    SourceAddressRange  9.24.106.0 9.24.106.255
    policyactionreference   ftpaction
}
```

1 Enables the policy performance monitor function, which assigns a weight fraction to the monitored policy performance data and sends them to the SD distributing stack as the monitored data crosses defined thresholds. The SD distributing stack uses this weight fraction for its routing decisions for incoming connection requests.

2 With the samplingInterval we specify how often we want to sample the policy information for changes.

3 The LossRatioAndWeightFr has two values: the ratio of retransmitted bytes over transmitted bytes in tenths of a percent, and the weight fraction to be assigned in percentage. In our implementation, if a loss ratio rate of 2% occurs, the loss fraction will be 25%. If a loss ratio rate above 4% occurs, the loss weight fraction will be 50%. Let's assume we have a loss ratio rate of 3%, which means a loss weight fraction of 25%.

4 The TimeoutRatioAndWeightFr has two values: the timeout ratio in tenths of a percent, and the weight fraction to be assigned in percentage. In our implementation, if a timeout ratio rate of 5% occurs, the timeout fraction weight would be 50%. If the timeout ratio rate is above 10%, the timeout weight fraction would be 100%. Let's assume we have a timeout ratio of 6%, which means we have a timeout weight fraction of 50%.

The two weight fractions LossRatioAndWeightFr and TimeoutRatioWeightFr will then be added. In our case, we would have a QoS weight fraction value of 75%, which this target stack sends to the Sysplex Distributor. This value will be used by the Sysplex Distributor stack to reduce the WLM weight given to this stack. So if the WLM weight assigned to this target stack is 50, the QoS weight fraction of 75% will give an effective WLM weight fraction of 37.5.

5 LossMaxWeightFr defines the maximum loss weight fraction.

6 TimeoutMaxWeightFr defines the maximum timeout weight fraction.

The Policy Agent monitors the traffic for which one or more service policy statements have been defined 7, 9. The policyScope attribute for the monitored policy has to be either DataTraffic 8 or Both. We monitored the FTP traffic originated from the 9.24.106 IP subnetwork.

5.5.2 Starting and stopping PAGENT

The Policy Agent can be started from the UNIX System Services shell or as a started task. At ITSO Raleigh, we used a started task procedure to start the Policy Agent.

Although the etc/pagent.conf file is the default configuration file, a specific search order is used when starting the Policy Agent. The following order is used:

1. File or data set specified with the -c startup option
2. File or data set specified with the PAGENT_CONF_FILE environment variable
3. /etc/pagent.conf
4. hlq.PAGENT.CONF

Security product (for example, RACF) authority is required to start the Policy Agent. The following commands can be used to create the necessary profile and permit users to use it:

```
RDEFINE OPERCMDS (MVS.SERVMGR.PAGENT) UACC(NONE)
PERMIT MVS.SERVMGR.PAGENT ACCESS(CONTROL) CLASS(OPERCMDS) ID(userid)SETROPTS
RACLIST(OPERCMDS) REFRESH i
```

For example, the following command could be used to start PAGENT in the shell:

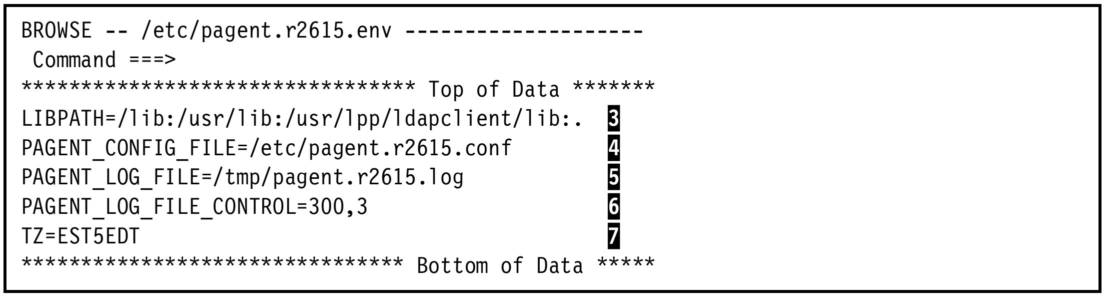
```
pagent -c /etc.pagent.r2615.conf
```

The following is a Policy Agent started task procedure that we used in our system:

```
//PAGENT   PROC
//*
//* SecureWay Communications Server IP
//* SMP/E distribution name: EZAPAGSP
```

```
//*
//* 5647-A01 (C) Copyright IBM Corp. 1999.
//* Licensed Materials - Property of IBM
//*
//PAGENT   EXEC PGM=PAGENT,REGION=OK,TIME=NOLIMIT,
//      PARM='POSIX(ON) ALL31(ON) ENVAR("_CEE_ENVFILE=DD:STDENV")/'
//*
//*       PARM='POSIX(ON) ALL31(ON) ENVAR("_CEE_ENVFILE=DD:STDENV")/-c /
//*             etc/pagent.r2615.conf -d'             1
//*      PARM='POSIX(ON) ALL31(ON) ENVAR("_CEE_ENVFILE=DD:STDENV")/-d'
//*
//* Example of passing parameters to the program (parameters must
//* extend to column 71 and be continued in column 16):
//*
//*STDENV   DD DSN=TCPIP.TCPPARMS.R2615(PAG&SYSCLONE.ENV),DISP=SHR
//STDENV   DD PATH='/etc/pagent.r2615.env',PATHOPTS=(ORDONLY)   2
//*
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//CEEDUMP  DD SYSOUT=*,DCB=(RECFM=FB,LRECL=132,BLKSIZE=132)
```

You can use environment variables, either configured in an MVS data set or HFS file, specified by the STDENV DD (1) to run with the desired configuration. We have configured environment variables in an HFS file(2), /etc/pagent.r2615.env, shown in Figure 5-9.



```
BROWSE -- /etc/pagent.r2615.env --------------------
 Command ===>
********************************* Top of Data *******
LIBPATH=/lib:/usr/lib:/usr/lpp/ldapclient/lib:.  3
PAGENT_CONFIG_FILE=/etc/pagent.r2615.conf        4
PAGENT_LOG_FILE=/tmp/pagent.r2615.log            5
PAGENT_LOG_FILE_CONTROL=300,3                    6
TZ=EST5EDT                                       7
******************************** Bottom of Data *****
```

Figure 5-9 Environment variables for Policy Agent

We have configured four environment variables for the Policy Agent to run successfully. The first variable, LIBPATH, enables PAGENT to search the dynamic link libraries needed to act as an LDAP client (3). The PAGENT_CONFIG_FILE specifies the PAGENT configuration file to use (4). The PAGENT_LOG_FILE specifies the log file name used by PAGENT (5), and the PAGENT_LOG_FILE_CONTROL (6) defines how many PAGENT log files are used in round robin and the size of the file; we use the default value.

If you define the PAGENT to use a syslogd to log messages, which means you define PAGENT_LOG_FILE=SYSLOGD, then the PAGENT_LOG_FILE_CONTROL has no meaning.

For the Policy Agent to run in your local time zone, you might have to specify the time zone in your working location using the TZ environment variable (7) even if you have the TZ environment variable configured in /etc/profile. Note that most OS/390 UNIX applications that start as MVS started tasks cannot use environment variables that have been configured in /etc/profile.

Note that while we do not have the RESOLVER_CONFIG variable configured, PAGENT can establish an affinity to a proper TCP/IP stack. The Policy Agent will use the TCP/IP image name configured in the TcpImage statement in the Policy Agent configuration file to determine to which TCP/IP it will install the policies.

We used the following statement in /etc/pagent.r2615.conf to do that:

```
TcpImage TCPIPC /pagent.r2615c.conf FLUSH 600
```

The Pagent Server can be stopped by:

- Using the **cancel** command; for example C PAGENT
- Using the **kill** command in the OS/390 shell
- Using the operator command STOP

The following command with the TERM signal will enable PAGENT to clean up resources properly before terminating:

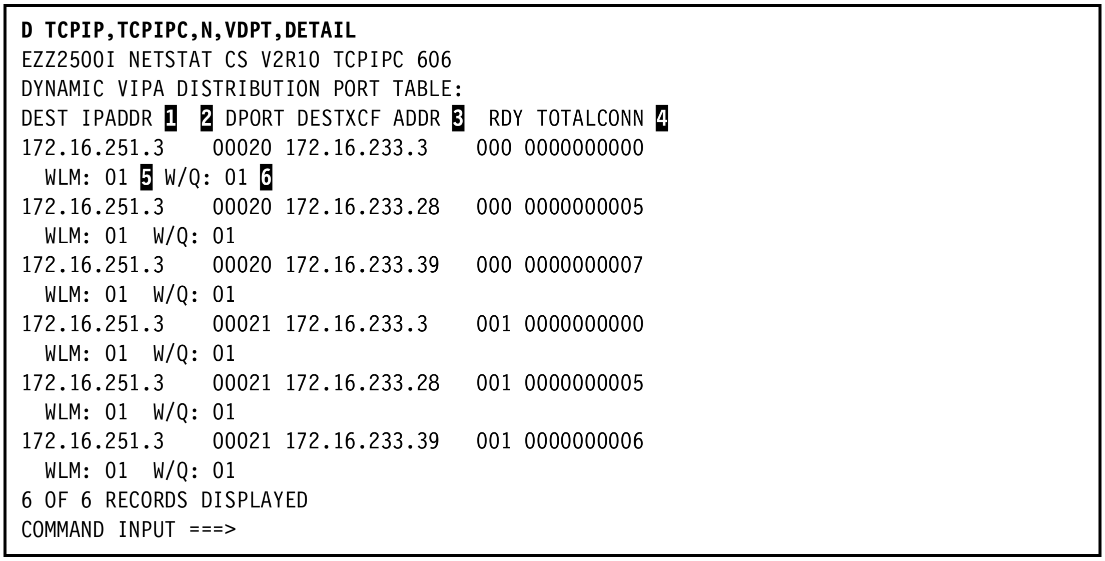
```
kill -s TERM pid
```

The PAGENT process ID can be obtained using the following OS/390 UNIX command:

```
ps -A
```

5.5.3 Monitoring the Sysplex Distributor QoS

You can use the NETSTAT command to display Sysplex Distributor information as shown in Figure 5-10.



```
D TCPIP,TCPIPC,N,VDPT,DETAIL
EZZ2500I NETSTAT CS V2R10 TCPIPC 606
DYNAMIC VIPA DISTRIBUTION PORT TABLE:
DEST IPADDR 1  2 DPORT DESTXCF ADDR 3  RDY TOTALCONN 4
172.16.251.3    00020 172.16.233.3    000 0000000000
  WLM: 01 5 W/Q: 01 6
172.16.251.3    00020 172.16.233.28   000 0000000005
  WLM: 01  W/Q: 01
172.16.251.3    00020 172.16.233.39   000 0000000007
  WLM: 01  W/Q: 01
172.16.251.3    00021 172.16.233.3    001 0000000000
  WLM: 01  W/Q: 01
172.16.251.3    00021 172.16.233.28   001 0000000005
  WLM: 01  W/Q: 01
172.16.251.3    00021 172.16.233.39   001 0000000006
  WLM: 01  W/Q: 01
6 OF 6 RECORDS DISPLAYED
COMMAND INPUT ===>
```

Figure 5-10 Sysplex Distributor VPDT detail display

1 DEST IPADDR: the distributing VIPA address of our sysplex complex.

2 DPORT: the port number to distribute workload in the sysplex.

3 DESTXCF ADDR: IP address of the SD target stacks. This address is configured for the dynamic XCF link in the IPCONFIG statement.

4 TOTALCONN: counter of connections distributed.

5 WLM: Workload Manager weight.

6 W/Q: QoS weight.

As you can see, all FTP connection requests were forwarded to two of three SD target stacks in our sysplex.

To verify that Sysplex Distributor policy has been successfully enabled, we can check the active policies using the **pasearch** command; the command requires a superuser authority. Figure 5-11 on page 130 shows the **pasearch** policies.

```
MVS TCP/IP pasearch CS V2R10            TCP/IP Image:      TCPIPC
  Date:                 08/10/2000       Time:  14:42:26

policyRule:             ftprule
  Version:              2       1        Status:            Active       2
  Priority:             0                Sequence Actions:  Don't Care
  ConditionListType:    DNF              No. Policy Action: 1
  policyAction:         ftpaction
   ActionType:          QOS              Action Sequence:   0
  Time Periods:
   Day of Month Mask:   1111111111111111111111111111111
   Month of Year Mask:  111111111111
   Day of Week Mask:    1111111  (Sunday - Saturday)
   Start Date Time:     None
   End Date Time:       None
   From TimeOfDay:      00:00            To TimeOfDay:      24:00
   From TimeOfDay UTC:  04:00            To TimeOfDay UTC:  04:00
   TimeZone:            Local
  Condition Summary:                     Negative Indicator:  OFF
   RouteCondition:
    InInterface:        0.0.0.0          OutInterface:      0.0.0.0
   HostCondition:
    SourceIpFrom:       9.24.106.0       SourceIpTo:        9.24.106.255
    DestIpFrom:         0.0.0.0          DestIpTo:          0.0.0.0
   ApplicationCondition:
    ProtocolNumFrom:    6                ProtocolNumTo:     6
    SourcePortFrom:     0                SourcePortTo:      0
    DestPortFrom:       20               DestPortTo:        21
    ApplicationName:
    ApplicationData:

  Qos Action:           ftpaction
    Version:            2        1       Status:            Active     2
    Scope:              DataTraffic    5 OutgoingTOS:       00000000
    Permission:         Allowed
    MaxRate:            0                MinRate:           0
    MaxDelay:           0                MaxConn:           0
    Routing Interfaces: 2              3
      Interface Number: 1                Interface:         172.16.233.28   4
      Interface Number: 2                Interface:         172.16.233.39   4
    RSVP Attributes
     ServiceType:       0                MaxRatePerFlow:    0
     MaxTokBuckPerFlw:  0                MaxFlows:          0
    DiffServ Attributes
     InProfRate:        0                InProfPeakRate:    0
     InProfTokBuck:     0                InProfMaxPackSz:   0
     OutProfXmtTOSByte: 00000000         ExcessTrafficTr:   BestEffort
    TR Attributes
     TotalConnections:  0                LoggingLevel:      0
     Percentage:        0                TimeInterval:      0
     TypeActions:       0
```

Figure 5-11 Display policies with pasearch command

The pasearch report shows all attributes of the policy installed, such as version 1 of this policy and the policy status 2.

The Routing Interfaces attribute 3 indicate whether this policy is the SD policy or not. Two interface attributes have been defined for this policy. The value has to be an IP address of the dynamic XCF link that has been defined for the SD target stack 4. Those dynamic XCF links are used to route the incoming connection requests. The PolicyScope attribute 5 must specify either DataTraffic or Both to define interfaces using this attribute.

5.5.4 Sysplex Distributor policies in the LDAP server

In this scenario, the SD policies have been stored in the OS/390 LDAP server, and is retrieved by the Policy Agent running on the SD distributing stack.

We configured the following policies as LDAP objects:

```
dn: pg=SDpolicy, g=policy, o=IBM_US, c= US    1
objectclass:ibm-policyGroup
ibm-policyGroupName:SDpolicy
ibm-policyRulesAuxContainedSet:pr=SDftprule, pg=SDpolicy, g=policy, o=IBM_US, c=US
description:SD policy for the SD distributing stack

dn:pr=SDftprule, pg=SDpolicy, g=policy, o=IBM_US, c= US
objectclass:ibm-policyRule
ibm-policyRuleName:SDftprule
cn:ftp application - rule
ibm-policyRuleEnabled:1
ibm-policyRuleConditionList:1:+:pc=ftpcond, pg=SDpolicy, g=policy, o=IBM_US, c=US
ibm-policyRuleActionList:1:pa=ftpaction, pg=SDpolicy, g=policy, o=IBM_US, c=US
ibm-policyRuleValidityPeriodList:pc=period1, pg=SDpolicy, g=policy, o=IBM_US, c=US
ibm-policyRuleKeywords:SDPolicyRules
ibm-policyRulePriority:2
ibm-policyRuleMandatory:TRUE
ibm-policyRuleSequencedActions:1

dn: pa=ftpaction, pg=SDpolicy, g=policy, o=IBM_US, c=US
objectclass:ibm-policyAction
objectclass:ibm-serviceCategories
ibm-policyActionName:ftpaction
cn:ftp-cos-1
ibm-PolicyScope:DataTraffic
ibm-MaxRate:400
ibm-MaxConnections:50
ibm-OutgoingTOS:01000000
ibm-interface:1--172.16.233.28   2
ibm-interface:1--172.16.233.39   2
ibm-interface:1--0.0.0.0         2

dn:pc=period1, pg=SDpolicy, g=policy, o=IBM_US, c= US
objectclass:ibm-policyTimePeriodCondition
ibm-policyConditionName:timeperiod1
cn:active time period 1
ibm-ptpConditionTime:19990713000000:20021030200000
ibm-ptpConditionMonthOfYearMask:111100111100
ibm-ptpConditionDayOfMonthMask:1111111111111111111111111111111
ibm-ptpConditionDayOfWeekMask:1111111
ibm-ptpConditionTimeOfDayMask:020000:230000
ibm-ptpConditionTimeZone:Z
description:time period 1

dn:pc=ftpcond, pg=SDpolicy, g=policy, o=IBM_US, c= US
objectclass:ibm-networkingPolicyCondition
objectclass:ibm-hostConditionAuxClass
```

```
objectclass:ibm-applicationConditionAuxClass
ibm-policyConditionName:hostftpapplcondition1
cn:ftp host and application condition 1
ibm-ProtocolNumberRange:6
ibm-SourceIPAddressRange:2-9.24.106.0-24
ibm-DestinationPortRange:20:21
```

We defined a new PolicyGroup 1 for the SD policy objects.The policies are identified as SD policies by the presence of the ibm-Interface attribute 2 in the ibm_policyAction object class. Here, we have defined ibm-interface 0.0.0.0, so that when neither RA28 nor RA39 can handle an FTP connection request, it will be forwarded to RA03 if an FTP server is running there.

The second-level PAGENT configuration file also has been updated to communicate with the z/OS LDAP server as shown below:

```
   ReadFromDirectory
   {
     LDAP_Server 172.16.250.3
     LDAP_ProtocolVersion 3
     LDAP_SchemaVersion 2
     SearchPolicyBaseDN  pg=SDpolicy, g=policy, o=IBM_US, c=US  1
   }

   # PolicyPerfMonitorForSDR Statement
   PolicyPerfMonitorForSDR  enable          2
   {
     samplinginterval 60
     LossRatioAndWeightFr 20 25
     TimeoutRatioAndWeightFr 50 50
     LossMaxWeightFr 95
     TimeoutMaxWeightFr 100
   }
```

1 PAGENT will download all objects under this tree. Note that all SD policy-related objects defined belong to this group.

2 Because the SD performance monitor policy is not supported by the LDAP server, it has to be defined in the PAGENT configuration file if necessary.

Using the **pasearch** z/OS UNIX command, you will see the SD policies installed into the SD distributing stack:

```
MVS TCP/IP pasearch CS V2R10            TCP/IP Image:      TCPIPC
  Date:                 08/10/2000       Time:  16:04:37

policyRule:             SDftprule
  Version:              2                Status:            Active
  Distinguish Name:     pr=SDftprule,pg=SDpolicy,g=policy,o=IBM_US,c=US    1
  Group Distinguish Nm: pg=SDpolicy,g=policy,o=IBM_US,c=US
  Priority:             2                Sequence Actions:  Mandatory
  ConditionListType:    DNF              No. Policy Action: 1
  policyAction:         ftpaction
   ActionType:          QOS              Action Sequence:   1
  Time Periods:
   Day of Month Mask:   1111111111111111111111111111111
   Month of Year Mask:  111100111100
   Day of Week Mask:    1111111  (Sunday - Saturday)
   Start Date Time UTC: Tue Jul 13 00:00:00 1999
   End Date Time UTC:   Wed Oct 30 20:00:00 2002
   From TimeOfDay UTC:  02:00            To TimeOfDay UTC:  23:00
```

```
 TimeZone:            UTC
Condition Summary:                      Negative Indicator:  OFF
 RouteCondition:
  InInterface:        0.0.0.0           OutInterface:      0.0.0.0
 HostCondition:
  SourceIpFrom:       9.24.106.0        SourceIpTo:        9.24.106.255
  DestIpFrom:         0.0.0.0           DestIpTo:          0.0.0.0
 ApplicationCondition:
  ProtocolNumFrom:    6                 ProtocolNumTo:     6
  SourcePortFrom:     0                 SourcePortTo:      0
  DestPortFrom:       20                DestPortTo:        21
  ApplicationName:
  ApplicationData:

Qos Action:           ftpaction
  Version:            2                 Status:            Active
  Distinguish Name:   pa=ftpaction,pg=SDpolicy,g=policy,o=IBM_US,c=US    1
  Scope:              DataTraffic       OutgoingTOS:       01000000
  Permission:         Allowed
  MaxRate:            400               MinRate:           0
  MaxDelay:           0                 MaxConn:           50
  Routing Interfaces: 3
    Interface Number: 1                 Interface:         172.16.233.28  2
    Interface Number: 2                 Interface:         172.16.233.39  2
    Interface Number: 3                 Interface:         0.0.0.0        2
  RSVP Attributes
   ServiceType:       0                 MaxRatePerFlow:    0
   MaxTokBuckPerFlw:  0                 MaxFlows:          0
  DiffServ Attributes
   InProfRate:        0                 InProfPeakRate:    0
   InProfTokBuck:     0                 InProfMaxPackSz:   0
   OutProfXmtTOSByte: 00000000          ExcessTrafficTr:   BestEffort
  TR Attributes
   TotalConnections:  0                 LoggingLevel:      0
   Percentage:        0                 TimeInterval:      0

   TypeActions:       0
```

1 The Distinguish Names defined for the LDAP objects.

2 The Routing Interfaces attribute, which indicates that this policy is an SD policy, is defined in the same way when the SD policies are defined in the PAGENT configuration file.

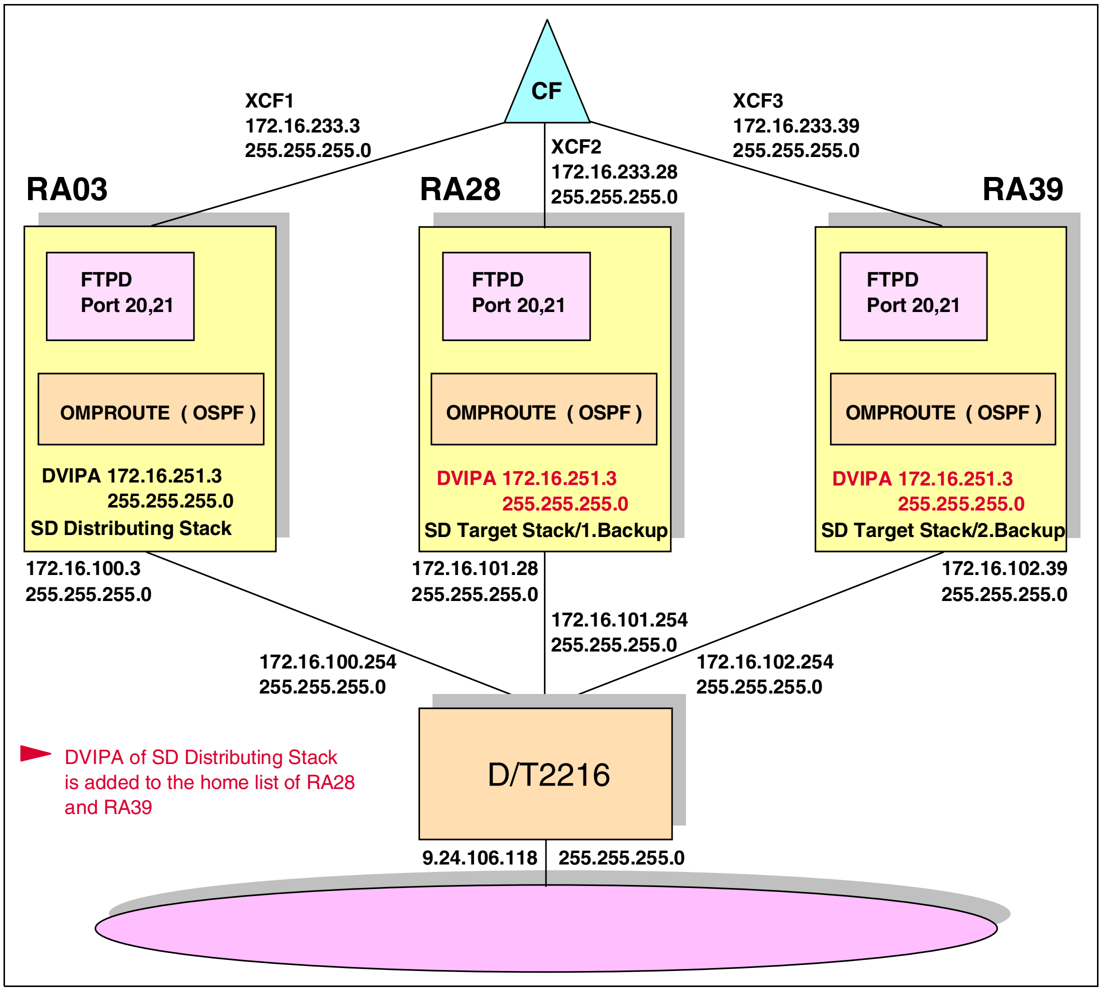
5.6 Implementation examples

In this section, we cover a number of scenarios to illustrate the way in which Sysplex Distributor is implemented.

5.6.1 Scenario 1: Three IP stacks distributing FTP services

Let us consider the scenario depicted in Figure 5-12 on page 134. We have three LPARs, each running CS for OS/390 V2R10 IP. Each LPAR has only one IP stack configured. There are three physical connections to a 2216 router, one per IP stack. The 2216 router connections have been configured as MPC+ and OSPF is the routing protocol selected to work in this scenario. There is one FTP server running in each IP stack using the port numbers 20 and 21.

We will go through the steps listed in 5.4.3, “Implementation” on page 120 to set up our Sysplex Distributor configuration properly.



DVIPA of SD Distributing Stack is added to the home list of RA28 and RA39

Figure 5-12 Scenario 1: Three IP stack in the Sysplex Distributor for FTP services

First, we have to decide which IP stack will be the distributor and which will be the backup. In our case, we have chosen RA03 as the distributing IP stack (1) and RA28 as primary backup (2) and RA39 as secondary backup (2). Please refer to Figure 5-13 on page 135, Figure 5-14 on page 136, and Figure 5-15 on page 137.

We can use the command D WLM,SYSTEMS to ensure that all the LPARs participating in the Sysplex Distributor are running in GOAL mode.

We have coded SYSPLEXROUTING (4), DYNAMICXCF (5), and DATAGRAMFWD (6) in each participating IP stack. Remember that the XCF device/link has to be defined dynamically. Sysplex Distributor will not work for an IP stack that has an IUTSAMEH device explicitly coded.

In this scenario, we describe FTP services being distributed by the RA03 IP stack. We will see later how we can distribute other services (such as TN3270, etc.) as long as we code the proper port number in the VIPADISTRIBUTE statement. For our example we select the ports 20 and 21, which are the default values for FTP server (8). Note that if the application requires more than one port, these ports have to be coded in the same VIPADISTRIBUTE statement.

In the distributing IP stack (RA03 in our example), we have to define the dynamic VIPA that will be associated with the sysplex cluster to distribute the FTP services. In order to do that, we use two statements: VIPADEFINE and VIPADISTRIBUTE. With the VIPADEFINE statement we are designating a set of Dynamic VIPAs that the stack initially owns and supports. We have chosen 172.16.251.3 as the Dynamic VIPA address assigned to the sysplex cluster for distributing FTP services (9). Once we have defined which Dynamic VIPA is going to be used for distributing, we have to assign the port numbers for this Dynamic VIPA using the VIPADISTRIBUTE statement. Finally, for those IP stacks that will serve as backup, it is necessary to code a VIPABACKUP statement with the proper *rank* number and *ip-address* to back up (10).

```
;  SYSPLEX DISTRIBUTOR: RA03  (DISTRIBUTOR)    1
IPCONFIG
   DATAGRAMFWD    6
   DYNAMICXCF 172.16.233.3 255.255.255.0 1   5
   SYSPLEXROUTING  4
   IGNOREREDIRECT
   VARSUBNETTING
...
PORT
   ...
   20  TCP OMVS      NOAUTOLOG ; FTP SERVER
   21  TCP FTPDC1              ; FTP SERVER
   ...
AUTOLOG 5
   FTPDC   JOBNAME FTPDC1
   OMPROUTC
ENDAUTOLOG

DEVICE M032216B MPCPTP AUTORESTART
LINK   M032216B MPCPTP M032216B

HOME
   172.16.100.3      M032216B

VIPADYNAMIC                                 9
   VIPADEFINE MOVE IMMED 255.255.255.0 172.16.251.3
   VIPADISTRIBUTE 172.16.251.3 PORT 20 21 DESTIP ALL
ENDVIPADYNAMIC                      8

START M032216B
```

Figure 5-13   Scenario 1: PROFILE data set used for RA03

```
 ; SYSPLEX DISTRIBUTOR: RA28 (BACKUP)    2
IPCONFIG
   DATAGRAMFWD     6
   DYNAMICXCF 172.16.233.28 255.255.255.0 1   5
   SYSPLEXROUTING   4
   IGNOREREDIRECT
   VARSUBNETTING
...
PORT
   20  TCP OMVS      NOAUTOLOG ; FTP SERVER
   21  TCP FTPDC1              ; FTP SERVER
...
AUTOLOG 5
   FTPDC   JOBNAME FTPDC1
   OMPROUTC
ENDAUTOLOG

DEVICE M282216B MPCPTP AUTORESTART
LINK   M282216B MPCPTP M282216B

HOME
   172.16.101.28    M282216B

VIPADYNAMIC             10
   VIPABACKUP 200 172.16.251.3
ENDVIPADYNAMIC

START M282216B
```

Figure 5-14   Scenario 1: PROFILE data set for RA28

```
 ; SYSPLEX DISTRIBUTOR: RA39 (BACKUP)     2

IPCONFIG
   DATAGRAMFWD    6
   DYNAMICXCF 172.16.233.39 255.255.255.0 1   5
   SYSPLEXROUTING   4
   IGNOREREDIRECT
   VARSUBNETTING
...
PORT
   20  TCP OMVS      NOAUTOLOG ; FTP SERVER
   21  TCP FTPDC1              ; FTP SERVER
...
AUTOLOG 5
   FTPDC   JOBNAME FTPDC1
   OMPROUTC
ENDAUTOLOG

DEVICE M282216B MPCPTP AUTORESTART
LINK   M282216B MPCPTP M282216B

HOME
   172.16.102.39    M282216B

VIPADYNAMIC           10
   VIPABACKUP 100 172.16.251.3
ENDVIPADYNAMIC

START M282216B
```

Figure 5-15 Scenario 1: PROFILE data set for RA39

In our example, we decided to use all of the XCF links (DESTIP ALL). In addition, we wanted the Dynamic VIPA to be taken back as soon the original IP stack is restarted (MOVE IMMED) in the event of a failure.

Because any one of the three IP stacks in our example can eventually be the Dynamic VIPA owner, we had to code the following entry in the OMPROUTE configuration file:

```
OSPF_Interface IP_Address=172.16.251.*
              Subnet_mask=255.255.255.0
              Cost0=8
              Non_Broadcast=Yes
              MTU=32768;
```

Figure 5-16 on page 138, Figure 5-17 on page 138, and Figure 5-18 on page 139 display the OMPROUTE configuration file used for each stack. Note that we have not coded the *name* parameter for these OSPF_Interface statements that define the generic interfaces because this is assigned dynamically. OMPROUTE would ignore any name for these OSPF_Interface statements.

11 represents the Dynamic VIPA definitions while **12** represents the XCF definitions.

```
;  RA03 omproute
;
Area    Area_Number=0.0.0.0
              Stub_Area=NO
              Authentication_type=None;
OSPF_Interface IP_Address=172.16.100.3
              Name=M032216B
              Cost0=1
              Subnet_mask=255.255.255.0    11
              MTU=32768;
OSPF_Interface IP_Address=172.16.251.*
              Subnet_mask=255.255.255.0    12
              Cost0=8
              Non_Broadcast=Yes
              MTU=32768;
OSPF_Interface IP_Address=172.16.233.*
              Subnet_mask=255.255.255.0
              Cost0=8
              Non_Broadcast=Yes
              MTU=32768;
```

Figure 5-16   OMPROUTE configuration file for RA03

```
;  RA28 omproute
;
Area    Area_Number=0.0.0.0
              Stub_Area=NO
              Authentication_type=None;
OSPF_Interface IP_Address=172.16.101.28
              Name=M032216B
              Cost0=1
              Subnet_mask=255.255.255.0     11
              MTU=32768;
OSPF_Interface IP_Address=172.16.251.*
              Subnet_mask=255.255.255.0     12
              Cost0=8
              Non_Broadcast=Yes
              MTU=32768;
OSPF_Interface IP_Address=172.16.233.*
              Subnet_mask=255.255.255.0
              Cost0=8
              Non_Broadcast=Yes
              MTU=32768;
```

Figure 5-17   OMPROUTE configuration file for RA28

```
;  RA39 omproute
;
Area    Area_Number=0.0.0.0
              Stub_Area=NO
              Authentication_type=None;
OSPF_Interface IP_Address=172.16.102.39
              Name=M032216B
              Cost0=1
              Subnet_mask=255.255.255.0    11
              MTU=32768;
OSPF_Interface IP_Address=172.16.251.*
              Subnet_mask=255.255.255.0
              Cost0=8                       12
              Non_Broadcast=Yes
              MTU=32768;
OSPF_Interface IP_Address=172.16.233.*
              Subnet_mask=255.255.255.0
              Cost0=8
              Non_Broadcast=Yes
              MTU=32768;
```

Figure 5-18 OMPROUTE configuration file for RA39

We now describe how this scenario works once all the IP stacks are active. Figure 5-19 on page 140 shows the output from the NETSTAT VIPADCFG command in all the IP stacks participating in the sysplex cluster. As you can see, only the distributing IP stack (TCPIPC on RA03) shows the VIPA DEFINE (1) and VIPA DISTRIBUTE (2) configuration with the port numbers (3) and the XCF IP addresses (4) assigned.

The output from the IP stack in RA28 shows that this stack is the primary backup (5) with rank 200, and the output from the IP stack in RA39 is the secondary backup (6) with rank 100.

```
RO RA03,D TCPIP,TCPIPC,N,VIPADCFG
D TCPIP,TCPIPC,N,VIPADCFG
EZZ2500I NETSTAT CS V2R10 TCPIPC 912
DYNAMIC VIPA INFORMATION:
  VIPA DEFINE:     1
    IP ADDRESS       ADDRESSMASK      MOVEABLE
    ----------       -----------      --------
    172.16.251.3     255.255.255.0    IMMEDIATE
  VIPA DISTRIBUTE:   2
    IP ADDRESS       PORT   XCF ADDRESS
    ----------       ----   -----------
    172.16.251.3     00020  ALL
    172.16.251.3  3  00021  ALL    4



RO RA28,D TCPIP,TCPIPC,N,VIPADCFG
D TCPIP,TCPIPC,N,VIPADCFG
EZZ2500I NETSTAT CS V2R10 TCPIPC 831
DYNAMIC VIPA INFORMATION:
  VIPA BACKUP:
    IP ADDRESS       RANK
    ----------       ----
    172.16.251.3     000200    5



RO RA39,D TCPIP,TCPIPC,N,VIPADCFG
D TCPIP,TCPIPC,N,VIPADCFG
RO RA39,D TCPIP,TCPIPC,N,VIPADCFG
EZZ2500I NETSTAT CS V2R10 TCPIPC 719
DYNAMIC VIPA INFORMATION:
  VIPA BACKUP:
    IP ADDRESS       RANK
    ----------       ----
    172.16.251.3     000100   6
```

Figure 5-19 NETSTAT VIPADCFG for RA03, RA28, and RA39 IP stacks

Figure 5-20 on page 141 shows the output from the NETSTAT HOME command for all the IP stacks participating in the sysplex cluster. Note that only one address is coded below the actual home statement in the PROFILE data set (**1**). The following three IP addresses are one and the same DYNAMICXCF (**2**). The next one is the DVIPA assigned to the Sysplex Distributor function (**3**). Note the flag I in the non-distributing IP stacks. Remember that this DVIPA is learned dynamically for all stacks participating in the Sysplex Distributor but only one stack advertises this IP address (with OSPF in our example).

```
RO RA03,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 982
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.100.3     M032216B         P    1
172.16.233.3     EZASAMEMVS
172.16.233.3     EZAXCF28              2
172.16.233.3     EZAXCF39
172.16.251.3     VIPLAC10FB03          3
127.0.0.1        LOOPBACK
6 OF 6 RECORDS DISPLAYED


RO RA28,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 840
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.101.28    M282216B         P    1
172.16.233.28    EZASAMEMVS
172.16.233.28    EZAXCF39              2
172.16.233.28    EZAXCF03
172.16.251.3     VIPLAC10FB03     I    3
127.0.0.1        LOOPBACK
6 OF 6 RECORDS DISPLAYED


RO RA39,D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 723
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.102.39    M392216B         P   1
172.16.233.39    EZASAMEMVS
172.16.233.39    EZAXCF28             2
172.16.233.39    EZAXCF03
172.16.251.3     VIPLAC10FB03     I   3
127.0.0.1        LOOPBACK
6 OF 6 RECORDS DISPLAYED
```

Figure 5-20 NETSTAT HOME command for RA03, RA28, and RA39 IP stacks

Figure 5-21 on page 142 shows the output from the NETSTAT VDPT command for all the IP stacks participating in the sysplex cluster. The output for this command shows the Dynamic VIPA destination port table. You can see the destination IP address (1) which is the Sysplex Distributor IP address, the port numbers to which connections are being distributed (2), the destination XCF address (3), the number of applications listening on the port number selected (4), and the total number of connections that have been forwarded by the Sysplex Distributor (5).

Note that the output for the stacks in RA28 and RA39 does not show any record because these stacks are not distributing workload. They are considered the target stacks by the distributing IP stack.

```
RO RA03,D TCPIP,TCPIPC,N,VDPT
D TCPIP,TCPIPC,N,VDPT
EZZ2500I NETSTAT CS V2R10 TCPIPC 989
DYNAMIC VIPA DISTRIBUTION PORT TABLE:
DEST IPADDR     DPORT DESTXCF ADDR    RDY TOTALCONN
      1            2       3           4       5
172.16.251.3    00020 172.16.233.3    000 0000000000
172.16.251.3    00020 172.16.233.28   000 0000000000
172.16.251.3    00020 172.16.233.39   000 0000000000
172.16.251.3    00021 172.16.233.3    001 0000000000
172.16.251.3    00021 172.16.233.28   001 0000000000
172.16.251.3    00021 172.16.233.39   001 0000000000
6 OF 6 RECORDS DISPLAYED


RO RA28,D TCPIP,TCPIPC,N,VDPT
D TCPIP,TCPIPC,N,VDPT
EZZ2500I NETSTAT CS V2R10 TCPIPC 846
DYNAMIC VIPA DISTRIBUTION PORT TABLE:
DEST IPADDR     DPORT DESTXCF ADDR    RDY TOTALCONN
0 OF 0 RECORDS DISPLAYED


RO RA39,D TCPIP,TCPIPC,N,VDPT
EZZ2500I NETSTAT CS V2R10 TCPIPC 727
DYNAMIC VIPA DISTRIBUTION PORT TABLE:
DEST IPADDR     DPORT DESTXCF ADDR    RDY TOTALCONN
0 OF 0 RECORDS DISPLAYED
```

Figure 5-21 NETSTAT VDPT command for RA03, RA28, and RA39 IP stacks

Figure 5-22 on page 143 shows the output from the NETSTAT VDPT DET command for the distributing IP stack only. This command provides additional information regarding the Workload Manager weight values (1) for the target IP stacks and the value assigned after modification using QoS information provided by the Policy Agent (2). These values are used by the distributing IP stack to determine the quantity of connections that should be forwarded to each target IP stack. Note that in this scenario no Policy Agent was used.

If all target stacks for a particular destination address and port have zero W/Q values, the connection forwarding will be done randomly rather than based upon WLM/QoS information.

```
RO RA03,D TCPIP,TCPIPC,N,VDPT,DET
D TCPIP,TCPIPC,N,VDPT,DET
EZZ2500I NETSTAT CS V2R10 TCPIPC 515
DYNAMIC VIPA DISTRIBUTION PORT TABLE:
DEST IPADDR     DPORT DESTXCF ADDR    RDY TOTALCONN
172.16.251.3    00020 172.16.233.3    000 0000000000
  WLM: 00 1  W/Q: 00  2
172.16.251.3    00020 172.16.233.28   000 0000000000
  WLM: 00  W/Q: 00
172.16.251.3    00020 172.16.233.39   000 0000000000
  WLM: 00  W/Q: 00
172.16.251.3    00021 172.16.233.3    001 0000000000
  WLM: 00  W/Q: 00
172.16.251.3    00021 172.16.233.28   001 0000000000
  WLM: 00  W/Q: 00
172.16.251.3    00021 172.16.233.39   001 0000000000
  WLM: 00  W/Q: 00
6 OF 6 RECORDS DISPLAYED
```

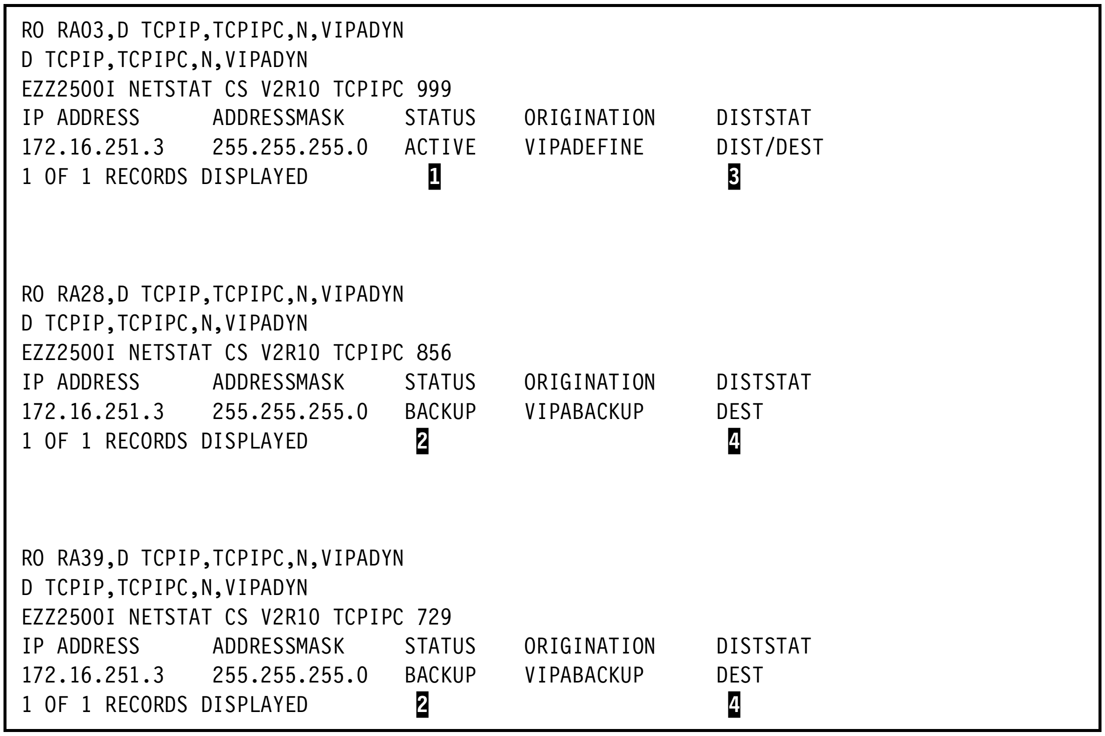
Figure 5-22   NETSTAT VDPT,DET for RA03, RA28 and RA39 IP stacks

Figure 5-23 and Figure 5-24 on page 144 actually show the same information with a slight difference. The SYSPLEX VIPADYN command displays the information for all the stacks participating in the sysplex at once. The command shows the MVS (system) name 1 and the actual status 2 of each stack. If the stack is defined as a backup you also can see the rank 3 value defined for it. The display also indicates whether each stack is defined as the distributor or a destination 4 or both.

```
RO RA03,D TCPIP,TCPIPC,SYSPLEX,VIPADYN
D TCPIP,TCPIPC,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V2R10 027
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- --1----- --2--- -3-- --------------- --------------- -4--
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.251.0    BOTH
  TCPIPC   RA28     BACKUP 200                                  DEST
  TCPIPC   RA39     BACKUP 100                                  DEST
3 OF 3 RECORDS DISPLAYED
```

Figure 5-23   SYSPLEX VIPADYN for all the stacks participating in the sysplex

Figure 5-24 on page 144 shows the output from the NETSTAT VIPADYN command for the IP stacks participating in the sysplex cluster. The output for this command shows the actual dynamic VIPA information for the local host. With this command, you can see if the DVIPA is active (1) or a backup (2) for the local stack. In addition it is shown if the DVIPA is being used as the distributing DVIPA (3) or destination DVIPA (4).



```
RO RA03,D TCPIP,TCPIPC,N,VIPADYN
D TCPIP,TCPIPC,N,VIPADYN
EZZ2500I NETSTAT CS V2R10 TCPIPC 999
IP ADDRESS      ADDRESSMASK     STATUS    ORIGINATION     DISTSTAT
172.16.251.3    255.255.255.0   ACTIVE    VIPADEFINE      DIST/DEST
1 OF 1 RECORDS DISPLAYED          1                        3



RO RA28,D TCPIP,TCPIPC,N,VIPADYN
D TCPIP,TCPIPC,N,VIPADYN
EZZ2500I NETSTAT CS V2R10 TCPIPC 856
IP ADDRESS      ADDRESSMASK     STATUS    ORIGINATION     DISTSTAT
172.16.251.3    255.255.255.0   BACKUP    VIPABACKUP      DEST
1 OF 1 RECORDS DISPLAYED         2                         4



RO RA39,D TCPIP,TCPIPC,N,VIPADYN
D TCPIP,TCPIPC,N,VIPADYN
EZZ2500I NETSTAT CS V2R10 TCPIPC 729
IP ADDRESS      ADDRESSMASK     STATUS    ORIGINATION     DISTSTAT
172.16.251.3    255.255.255.0   BACKUP    VIPABACKUP      DEST
1 OF 1 RECORDS DISPLAYED         2                         4
```

Figure 5-24 NETSTAT VIPADYN for RA03, RA28 and RA39 IP stacks

Figure 5-25 on page 145 shows the output from the NETSTAT VCRT command for all the IP stacks participating in the sysplex cluster.The output of this command displays the dynamic VIPA connection routing table (CRT). During Sysplex Distributor normal operation, this table could be quite large. It contains one entry for each connection being distributed. Figure 5-25 on page 145 shows the initial status just after the IP stacks have been started and no connection requests have been received yet.

```
RO RA03,D TCPIP,TCPIPC,N,VCRT
D TCPIP,TCPIPC,N,VCRT
EZZ2500I NETSTAT CS V2R10 TCPIPC 068
DYNAMIC VIPA CONNECTION ROUTING TABLE:
DEST IPADDR      DPORT  SRC IPADDR       SPORT  DESTXCF ADDR
0 OF 0 RECORDS DISPLAYED


RO RA28,D TCPIP,TCPIPC,N,VCRT
D TCPIP,TCPIPC,N,VCRT
EZZ2500I NETSTAT CS V2R10 TCPIPC 862
DYNAMIC VIPA CONNECTION ROUTING TABLE:
DEST IPADDR      DPORT  SRC IPADDR       SPORT  DESTXCF ADDR
0 OF 0 RECORDS DISPLAYED


RO RA39,D TCPIP,TCPIPC,N,VCRT
D TCPIP,TCPIPC,N,VCRT
EZZ2500I NETSTAT CS V2R10 TCPIPC 731
DYNAMIC VIPA CONNECTION ROUTING TABLE:
DEST IPADDR      DPORT  SRC IPADDR       SPORT  DESTXCF ADDR
0 OF 0 RECORDS DISPLAYED
```

Figure 5-25 NETSTAT VCRT for RA03, RA28 and RA39 IP stacks

Figure 5-26 on page 146 shows a portion of the output from the NETSTAT VCRT command for the distributing IP stack during normal operation. The output of this command displays the dynamic VIPA CRT for the local stack. Because this is the distributing IP stack, it shows all the connections between the clients and the participating IP stacks. It includes all the XCF addresses(1). During Sysplex Distributor normal operation, this table could be very large. It contains one entry for each connection being distributed.

While NETSTAT VCRT on the distributing IP stack displays all the distributed connections, the same command on the other stacks will show only those connections established with the local server instance. Figure 5-27 on page 146 and Figure 5-28 on page 147 show the VCRT on RA28 and RA39. They show the destination IP address (2), the port destination (3), the source IP address (4), the source port (5), and the Dynamic XCF address of the stack processing this connection (1).

```
RO RA03,D TCPIP,TCPIPC,N,VCRT,MAX=400
D TCPIP,TCPIPC,N,VCRT,MAX=400
EZZ2500I NETSTAT CS V2R10 TCPIPC 050
DYNAMIC VIPA CONNECTION ROUTING TABLE:
DEST IPADDR      DPORT  SRC IPADDR       SPORT  DESTXCF ADDR
     1             2        3              4         5
172.16.251.3     00021  9.24.104.75      03445  172.16.233.3
172.16.251.3     00021  9.24.104.75      03448  172.16.233.3
172.16.251.3     00021  9.24.104.75      03451  172.16.233.3
.......
172.16.251.3     00020  9.24.104.75      03565  172.16.233.3
172.16.251.3     00020  9.24.104.75      03568  172.16.233.3
172.16.251.3     00020  9.24.104.75      03571  172.16.233.3
.......
172.16.251.3     00021  9.24.104.75      03443  172.16.233.28
172.16.251.3     00021  9.24.104.75      03446  172.16.233.28
172.16.251.3     00021  9.24.104.75      03449  172.16.233.28
.......
172.16.251.3     00020  9.24.104.75      03593  172.16.233.28
172.16.251.3     00020  9.24.104.75      03594  172.16.233.28
172.16.251.3     00020  9.24.104.75      03595  172.16.233.28
.......
172.16.251.3     00021  9.24.104.75      03536  172.16.233.39
172.16.251.3     00021  9.24.104.75      03539  172.16.233.39
172.16.251.3     00021  9.24.104.75      03542  172.16.233.39
.......
172.16.251.3     00020  9.24.104.75      03718  172.16.233.39
172.16.251.3     00020  9.24.104.75      03721  172.16.233.39
172.16.251.3     00020  9.24.104.75      03724  172.16.233.39
.......
```

Figure 5-26 NETSTAT VCRT from RA03 IP stack when there are several concurrent FTP sessions

```
RO RA28,D TCPIP,TCPIPC,N,VCRT,MAX=400
D TCPIP,TCPIPC,N,VCRT,MAX=400
EZZ2500I NETSTAT CS V2R10 TCPIPC 840
DYNAMIC VIPA CONNECTION ROUTING TABLE:
DEST IPADDR      DPORT  SRC IPADDR       SPORT  DESTXCF ADDR
     1             2        3              4         5
172.16.251.3     00020  9.24.104.75      03609  172.16.233.28
172.16.251.3     00020  9.24.104.75      03610  172.16.233.28
172.16.251.3     00020  9.24.104.75      03611  172.16.233.28
172.16.251.3     00020  9.24.104.75      03612  172.16.233.28
.......
```

Figure 5-27 NETSTAT VCRT from RA28 IP stack when there are several concurrent FTP sessions

```
RO RA39,D TCPIP,TCPIPC,N,VCRT,MAX=400
D TCPIP,TCPIPC,N,VCRT,MAX=400
EZZ2500I NETSTAT CS V2R10 TCPIPC 991
DYNAMIC VIPA CONNECTION ROUTING TABLE:
DEST IPADDR      DPORT  SRC IPADDR       SPORT  DESTXCF ADDR
     1             2        3              4         5
172.16.251.3     00020  9.24.104.75      03431  172.16.233.39
172.16.251.3     00020  9.24.104.75      03435  172.16.233.39
172.16.251.3     00020  9.24.104.75      03439  172.16.233.39
172.16.251.3     00020  9.24.104.75      03443  172.16.233.39
.......
```

Figure 5-28   NETSTAT VCRT from RA39 IP stack when there are several concurrent FTP sessions

Figure 5-29 displays the output from the NETSTAT VDPT DET command on the RA03 IP stack. In this figure we can appreciate how the WLM and W/Q weights have changed once the connection requirements start arriving at the RA03 IP stack. You can see the difference between the amount of FTP connections 1 for IP stacks RA03, RA28, and RA39. If you take the different WLM weights 2 into account ,then the distribution of workload works correctly. In this scenario no QoS policy was active.

```
RO RA03,D TCPIP,TCPIPC,N,VDPT,DET
D TCPIP,TCPIPC,N,VDPT,DET
EZZ2500I NETSTAT CS V2R10 TCPIPC 185
DYNAMIC VIPA DISTRIBUTION PORT TABLE:
DEST IPADDR     DPORT DESTXCF ADDR    RDY TOTALCONN
172.16.251.3    00020 172.16.233.3    000 0000000320
  WLM: 01  W/Q: 01
172.16.251.3    00020 172.16.233.28   000 0000001200
  WLM: 03  W/Q: 03
172.16.251.3    00020 172.16.233.39   000 0000001131
  WLM: 03  W/Q: 03
172.16.251.3    00021 172.16.233.3    001 0000000004  1
  WLM: 01  W/Q: 01  2
172.16.251.3    00021 172.16.233.28   001 0000000016  1
  WLM: 03  W/Q: 03  2
172.16.251.3    00021 172.16.233.39   001 0000000017  1
  WLM: 03  W/Q: 03  2
6 OF 6 RECORDS DISPLAYED
```

Figure 5-29   NETSTAT VDPT DET from RA03 IP stack with multiple concurrent FTP sessions

Figure 5-30 on page 148 shows the output from NETSTAT VDPT after 100 concurrent FTP sessions in which each session requested 80 data connections. The NETSTAT VDPT command displays how these FTP sessions were distributed among the different IP stacks participating in the sysplex cluster.

```
RO RA03,D TCPIP,TCPIPC,N,VDPT
D TCPIP,TCPIPC,N,VDPT
EZZ2500I NETSTAT CS V2R10 TCPIPC 499
DYNAMIC VIPA DISTRIBUTION PORT TABLE:
DEST IPADDR     DPORT DESTXCF ADDR    RDY TOTALCONN
172.16.251.3    00020 172.16.233.3    000 0000001360
172.16.251.3    00020 172.16.233.28   000 0000003280
172.16.251.3    00020 172.16.233.39   000 0000003360
172.16.251.3    00021 172.16.233.3    001 0000000017
172.16.251.3    00021 172.16.233.28   001 0000000041
172.16.251.3    00021 172.16.233.39   001 0000000042
6 OF 6 RECORDS DISPLAYED
```

Figure 5-30 NETSTAT VDPT from RA03 IP stack after 100 FTP sessions

Figure 5-31 shows how these connections are distributed taking the WLM weights into account (as was done in Figure 5-29 on page 147). We start distributing the connections to the stack on the highest WLM weight. In our case, assume we start with RA28. Then we distribute the first three connections to RA28. We then choose RA39 for the next three connections and then send one connection to RA03. At the eighth new connection, the Distributor starts sending the connections to RA28 again.

WLM and QoS weights are refreshed by the Distributor stack after one minute or when we add a new stack. After WLM weights are refreshed and the target stacks possibly reordered, connection distribution resumes at the first target stack.

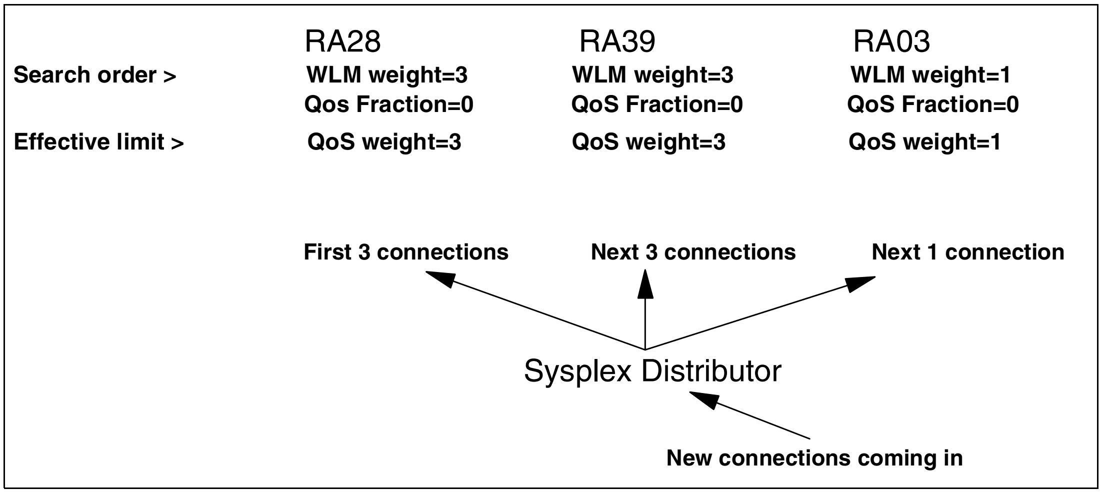


Figure 5-31 Using WLM/QoS data to select a target stack

5.6.2 Scenario 2: VIPA takeover and takeback with Sysplex Distributor

This second example, as shown in Figure 5-32 on page 149, illustrates the functionality of automatic VIPA takeover and takeback. We use the same environment as before: three LPARs, each running IBM Communications Server for OS/390 V2R10 IP and having only one IP stack configured. There are three physical connections to an IBM 2216 router, one per IP stack. The 2216 router connections have been configured as MPC+ and OSPF is the routing protocol selected to work in this scenario. There is one FTP server running in each IP stack using the ports 20 and 21.

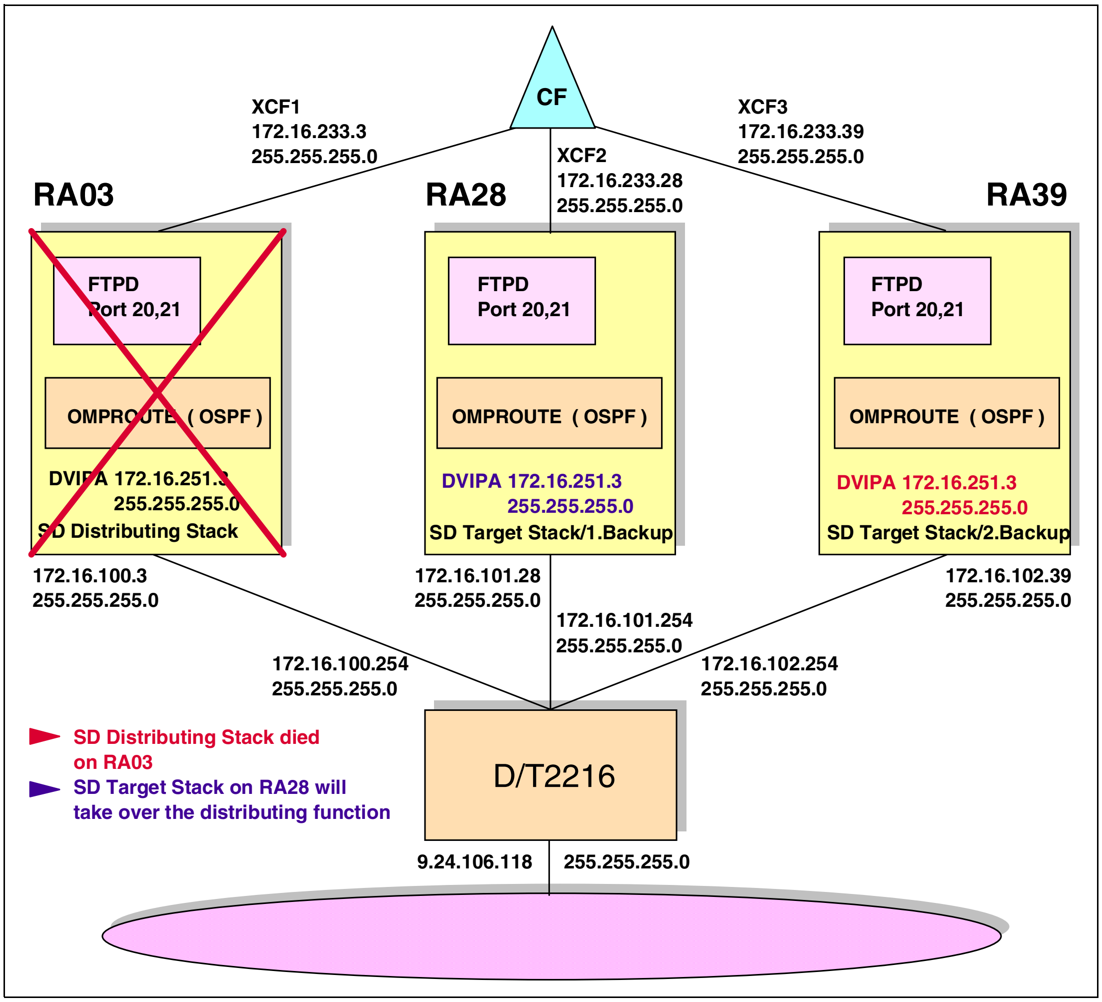


Figure 5-32 RA28 IP stack takes over the distributing function of RA03

In this example, we monitor only one FTP session. We FTPed a large file from our FTP client to the sysplex. We used a large file just to have enough time to take down the RA03 IP stack and show how DVIPA is switched over to the RA28 IP stack. At the same time, the FTP connection remains active. Some time later, we brought up the RA03 IP stack, which subsequently caused the DVIPA to be taken back by the RA03 IP stack without any disruption to the FTP session. Finally, after some minutes, the transfer completed successfully.

Figure 5-33 on page 150 displays the Dynamic VIPA distribution port table. We see three FTP servers listening on port 20 and 21, one in each IP stack. There is one session currently active, the large file FTP session (1). According to this output, the session has been distributed to RA39 stack (2).

```
RO RA03,D TCPIP,TCPIPC,N,VDPT
D TCPIP,TCPIPC,N,VDPT
EZZ2500I NETSTAT CS V2R10 TCPIPC 008
DYNAMIC VIPA DISTRIBUTION PORT TABLE:
DEST IPADDR     DPORT DESTXCF ADDR     RDY TOTALCONN
172.16.251.3    00020 172.16.233.3     000 0000000000
172.16.251.3    00020 172.16.233.28    000 0000000000
172.16.251.3    00020 172.16.233.39 2  000 0000000001  1
172.16.251.3    00021 172.16.233.3     001 0000000000
172.16.251.3    00021 172.16.233.28    001 0000000000
172.16.251.3    00021 172.16.233.39 2  001 0000000001  1
6 OF 6 RECORDS DISPLAYED
```

Figure 5-33 D TCPIP,TCPIPC,N,VDPT on RA03 IP stack after establishing one FTP session

Figure 5-34 displays the Dynamic VIPA connection routing table. We see that the connection is being routed through the XCF link associated with RA39 (3). Please take note of the source port number 4, since we later compare this value after takeover and takeback.

```
RO RA03,D TCPIP,TCPIPC,N,VCRT
D TCPIP,TCPIPC,N,VCRT
EZZ2500I NETSTAT CS V2R10 TCPIPC 010
DYNAMIC VIPA CONNECTION ROUTING TABLE:
DEST IPADDR      DPORT  SRC IPADDR       SPORT    DESTXCF ADDR
172.16.251.3     00020  9.24.104.75      01997 4   172.16.233.39 3
172.16.251.3     00021  9.24.104.75      01996 4   172.16.233.39 3
2 OF 2 RECORDS DISPLAYED
```

Figure 5-34 D TCPIP,TCPIPC,N,VCRT on RA03 IP stack after establishing one FTP session

Figure 5-35 shows the resulting messages at the moment when the RA03 stack was stopped and all the links associated with this stack were terminated. We see message EZZ8301, which tells us that the Dynamic VIPA has been taken over. This message 1 is displayed on the stack's joblog that takes over the Dynamic VIPA. In our case, this was RA28, since it was our first backup.

```
P TCPIPC
BPXF207I ROUTING INFORMATION HAS BEEN DELETED FOR TRANSPORT DRIVER
TCPIPC.
IUT5002I  TASK FOR ULPID TCPIPC   USING TRLE RA28M    TERMINATING
IUT5002I  TASK FOR ULPID TCPIPC   USING TRLE RA39M    TERMINATING
IUT5002I  TASK FOR ULPID TCPIPC   USING TRLE IUTSAMEH TERMINATING
IUT5002I  TASK FOR ULPID TCPIPC   USING TRLE M032216B TERMINATING
EZZ4201I TCP/IP TERMINATION COMPLETE FOR TCPIPC
.......
EZZ8301I VIPA 172.16.251.3 TAKEN OVER FROM TCPIPC ON RA03   1
```

Figure 5-35 Stopping RA03 IP stack

Because the FTP session was established with a target stack different from the distributing one, the session is not lost. The Dynamic VIPA is switched over to the backup IP stack and this DVIPA is recognized as the distributing VIPA. In our case, RA28 takes the ownership of the DVIPA and becomes the distributing IP stack. The RA28 IP stack informs the RA39 IP stack that it is now the distributing stack.

Figure 5-36 displays the Dynamic VIPA distribution port table of RA28 after it has become the distributing stack. This table was built based on saved information about RA03 and the information it received from RA39.

```
RO RA28,D TCPIP,TCPIPC,N,VDPT
D TCPIP,TCPIPC,N,VDPT
EZZ2500I NETSTAT CS V2R10 TCPIPC 334
DYNAMIC VIPA DISTRIBUTION PORT TABLE:
DEST IPADDR     DPORT DESTXCF ADDR    RDY TOTALCONN
172.16.251.3    00020 172.16.233.28   000 0000000000
172.16.251.3    00020 172.16.233.39   000 0000000001
172.16.251.3    00021 172.16.233.28   001 0000000000
172.16.251.3    00021 172.16.233.39   001 0000000001
4 OF 4 RECORDS DISPLAYED
```

Figure 5-36 D TCPIP,TCIPC,N,VDPT on RA28 IP stack after DVIPA takeover

Figure 5-37 displays the Dynamic VIPA connection routing table for stack RA28. We can see that it is the same connection that has remained active even after the RA03 stack has failed. The source port numbers here (1) can be compared with those in Figure 5-34 on page 150.

```
RO RA28,D TCPIP,TCPIPC,N,VCRT
D TCPIP,TCPIPC,N,VCRT
EZZ2500I NETSTAT CS V2R10 TCPIPC 332
DYNAMIC VIPA CONNECTION ROUTING TABLE:
DEST IPADDR      DPORT  SRC IPADDR       SPORT    DESTXCF ADDR
172.16.251.3     00020  9.24.104.75      01997 1  172.16.233.39
172.16.251.3     00021  9.24.104.75      01996 1  172.16.233.39
2 OF 2 RECORDS DISPLAYED
```

Figure 5-37 D TCPIP,TCPIPC,N,VCRT on RA28 IP stack after DVIPA takeover

Figure 5-38 shows the output of the SYSPLEX VIPADYN command and shows us that stack RA28 is now distributor and destination 3. The DIST column was added in IBM Communications Server for OS/390 V2R10 as expanded output to this display command.

```
RO RA28,D TCPIP,TCPIPC,SYSPLEX,VIPADYN
D TCPIP,TCPIPC,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V2R10 339
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA28
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPABACKUP
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA28     ACTIVE      255.255.255.0   172.16.251.0    BOTH 3
  TCPIPC   RA39     BACKUP 100                                  DEST
2 OF 2 RECORDS DISPLAYED
```

Figure 5-38 D TCPIP,TCPIP,SYSPLEX VIPADYN on RA28 IP stack after DVIPA takeover

Figure 5-39 on page 152 displays the home list of stack RA28 and shows us that the I flag for the Dynamic VIPA address defined on the RA03 stack has been removed because RA28 is now the owner of this DVIPA.

```
RO RA28,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 342
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.101.28    M282216B         P
172.16.233.28    EZASAMEMVS
172.16.233.28    EZAXCF39
172.16.233.28    EZAXCF03
172.16.251.3     VIPLAC10FB03
127.0.0.1        LOOPBACK
6 OF 6 RECORDS DISPLAYED
```

Figure 5-39 D TCPIP,TCPIPC,N,HOME on RA28 IP stack after DVIPA takeover

Figure 5-40 shows the resulting messages at the moment the RA03 IP stack is started again. Once the IP stack is active in RA03, the process of taking the DVIPA back is started. Additionally, we see message EZZ8302, which indicates that the Dynamic VIPA has been taken back to stack RA03. This message 1 is displayed on the stack's joblog, which takes back the Dynamic VIPA. In our case, this is stack RA03, which was our originally defined distributor stack.

```
S TCPIPC
IEF403I TCPIPC - STARTED - TIME=17.49.18
IEE252I MEMBER CTIEZB01 FOUND IN SYS1.PARMLIB
EZZ7450I FFST SUBSYSTEM IS NOT INSTALLED
EZZ0300I OPENED PROFILE FILE DD:PROFILE
EZZ0309I PROFILE PROCESSING BEGINNING FOR DD:PROFILE
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE DD:PROFILE
EZZ0641I IP FORWARDING NOFWDMULTIPATH SUPPORT IS ENABLED
EZZ8302I VIPA 172.16.251.3 TAKEN FROM TCPIPC ON RA28
EZZ0335I ICMP WILL IGNORE REDIRECTS
EZZ0350I SYSPLEX ROUTING SUPPORT IS ENABLED
EZZ8303I VIPA 172.16.251.3 GIVEN TO TCPIPC ON RA03     1
EZZ0352I VARIABLE SUBNETTING SUPPORT IS ENABLED
EZZ0345I STOPONCLAWERROR IS ENABLED
EZZ0624I DYNAMIC XCF DEFINITIONS ARE ENABLED
.......
```

Figure 5-40 Starting RA03 IP stack

Because the VIPADEFINE is coded with MOVE IMMED in the PROFILE data set, as shown in Figure 5-13 on page 135, the RA03 IP stack takes ownership of the DVIPA without waiting for existing connections to finish. This process is nondisruptive for the FTP session because the RA28 stack sends the connection routing table to the RA03 stack. The RA03 stack can continue distributing the following packets to the proper target FTP server.

Figure 5-41 on page 153 shows the display of the Dynamic VIPA destination port table of the RA03 stack once it has been re-activated. Note that RA03 updated the port table according to the information it received from RA28 (1).

```
RO RA03,D TCPIP,TCPIPC,N,VDPT
D TCPIP,TCPIPC,N,VDPT
EZZ2500I NETSTAT CS V2R10 TCPIPC 188
DYNAMIC VIPA DISTRIBUTION PORT TABLE:
DEST IPADDR     DPORT DESTXCF ADDR    RDY TOTALCONN
172.16.251.3    00020 172.16.233.3    000 0000000000
172.16.251.3    00020 172.16.233.28   000 0000000000
172.16.251.3    00020 172.16.233.39   000 0000000001  1
172.16.251.3    00021 172.16.233.3    001 0000000000
172.16.251.3    00021 172.16.233.28   001 0000000000
172.16.251.3    00021 172.16.233.39   001 0000000001  1
6 OF 6 RECORDS DISPLAYED
```

Figure 5-41   D TCPIP,TCPIPC,N,VDPT on RA03 IP stack after DVIPA takeback

Figure 5-42 shows the display of the Dynamic VIPA connection routing table for the RA03 IP stack just started. Note that the FTP connection is still active and is still using the same port numbers 2 since it has not been disrupted. After some minutes, the file transfer finished successfully.

```
RO RA03,D TCPIP,TCPIPC,N,VCRT
D TCPIP,TCPIPC,N,VCRT
EZZ2500I NETSTAT CS V2R10 TCPIPC 190
DYNAMIC VIPA CONNECTION ROUTING TABLE:
DEST IPADDR      DPORT  SRC IPADDR       SPORT    DESTXCF ADDR
172.16.251.3     00020  9.24.104.75      01997 2  172.16.233.39
172.16.251.3     00021  9.24.104.75      01996 2  172.16.233.39
2 OF 2 RECORDS DISPLAYED
```

Figure 5-42   D TCPIP,TCPIP,N,VCRT on RA03 IP stack after DVIPA takeback

Finally, the SYSPLEX VIPADYN command shows us that the status of our participating sysplex stacks is the same as when we started. See Figure 5-43.

```
RO RA03,D TCPIP,TCPIPC,SYSPLEX,VIPADYN
D TCPIP,TCPIPC,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V2R10 184
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.251.0    BOTH
  TCPIPC   RA28     BACKUP 200                                  DEST
  TCPIPC   RA39     BACKUP 100                                  DEST
3 OF 3 RECORDS DISPLAYED
```

Figure 5-43   D TCPIP,TCPIPC,SYSPLEX,VIPADYN on RA03 IP stack after DVIPA takeback

5.6.3 Scenario 3: Distributing multiple IP services

Figure 5-44 depicts our third scenario. We have three LPARs, each running IBM Communications Server for OS/390 V2R10 IP and each having only one IP stack configured. There are three physical connections to an IBM 2216 router, one per IP stack. The 2216 router connections have been configured as MPC+ and OSPF is the routing protocol selected to work in this scenario. There is one FTP server running in each IP stack using ports 20 and 21. The TN3270E server is also running in each stack using port 23. In addition we have implemented a server, RSSERVER, in RA28 and RA39 using port 1492. See Figure 5-44.

RSSERVER and RSCLIENT are a pair of programs that provide an example of how to use REXX/SOCKETS to implement a service. These are provided by IBM and can be found in the TCPIP.SEZAINST library. The RSSERVER program runs on a dedicated TSO user ID. It returns a number of data lines as requested to the client. The RSCLIENT program is used to request a number of arbitrary data lines from the server. One or more clients can access the server until it is terminated.

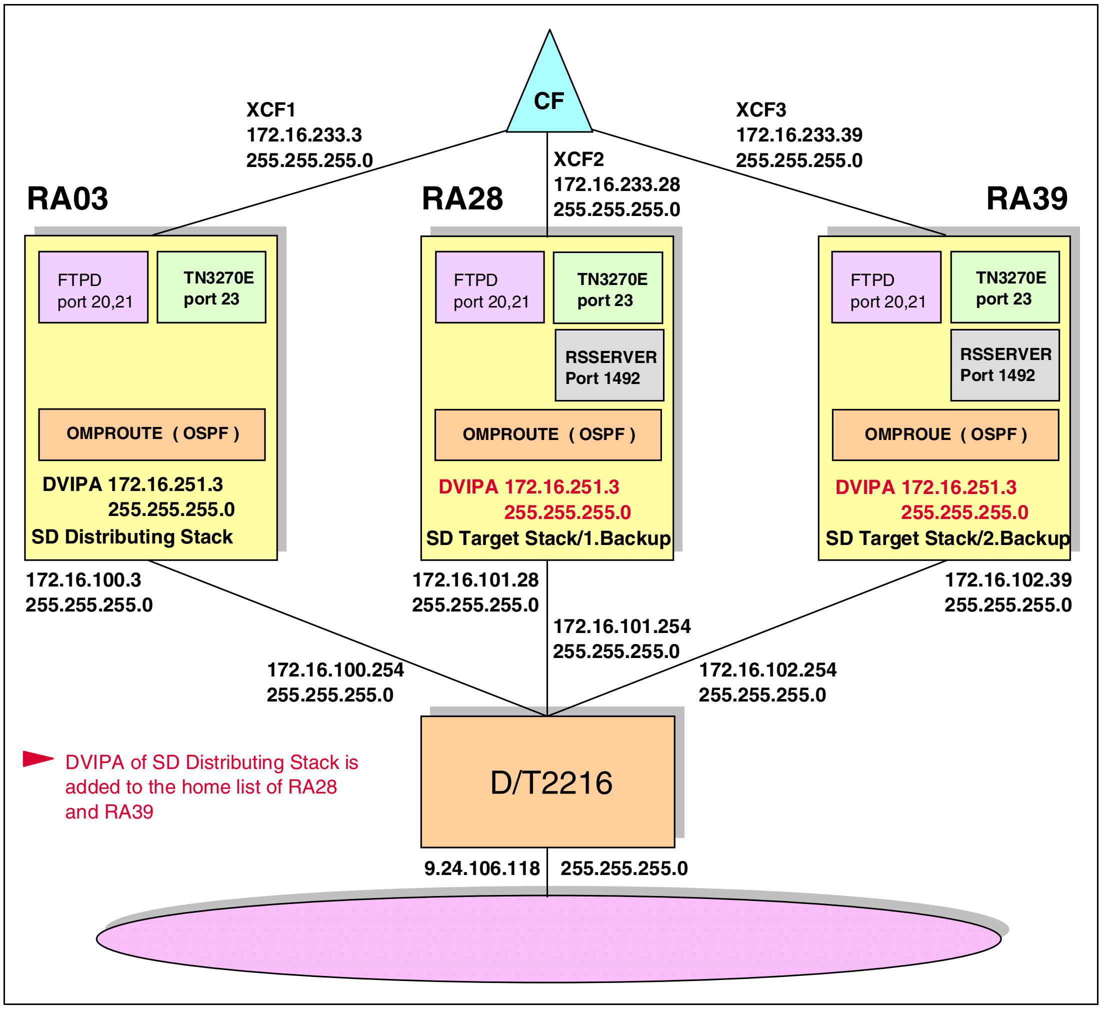


Figure 5-44 Distributing multiple IP services

Figure 5-45 on page 156, Figure 5-46 on page 157, and Figure 5-47 on page 158 show the PROFILE data set being used for this scenario. There are some differences from those shown in Figure 5-13 on page 135, Figure 5-14 on page 136, and Figure 5-15 on page 137:

- In AUTOLOG we included T03CSM **11** (the name we assigned the RSSERVER) for RA28 and RA39 IP stacks.
- We added to the already existing VIPADISTRIBUTE statement for FTP the port 23 for TN3270E **12**. In addition we added a new VIPADISTRIBUTE **13** statement to reflect that T03CSM (RSSERVER) instances are running on RA28 and RA39 IP stacks.
- We have included on each stack the TELNETPARMS/ENDTELNETPARMS **14** and BEGINVTAM/ENDVTAM **15** blocks, as well as PORT 23 INTCLIEN (port reservation) to support TN3270E server in these stacks.

The RSCLIENT and the FTP clients were executed from an OS/390 system attached to the network with IP address 9.24.104.75. The TN3270E client was executed from a PC running Windows NT with IP address 9.24.106.247.

```
; SYSPLEX DISTRIBUTOR: RA03 ( DISTRIBUTOR )  1

IPCONFIG
 DATAGRAMFWD          6
 DYNAMICXCF 172.16.233.3 255.255.255.0 1    5
 SYSPLEXROUTING     4
 IGNOREREDIRECT
 VARSUBNETTING

PORT
  ...
  20  TCP OMVS      NOAUTOLOG ; FTP SERVER
  21  TCP FTPDC1              ; FTP SERVER
  23  TCP INTCLIEN      10
  ...

TELNETPARMS           14
  PORT 23
  INACTIVE 0
ENDTELNETPARMS

AUTOLOG 5
 FTPDC JOBNAME FTPDC1
 OMPROUTC
 ENDAUTOLOG

 DEVICE M032216B MPCPTP AUTORESTART
 LINK   M032216B MPCPTP M032216B

HOME
 172.16.100.3 M032216B

VIPADYNAMIC
  VIPADEFINE MOVE IMMED 255.255.255.0 172.16.251.3
  VIPADISTRIBUTE 172.16.251.3 PORT 20 21 23 DESTIP ALL  12
  VIPADISTRIBUTE 172.16.251.3 PORT 1492                          13
     DESTIP 172.16.233.28 172.16.233.39
ENDVIPADYNAMIC

BEGINVTAM             15
  PORT 23
  DEFAULTLUS RA03TV21..RA03TV29 ENDDEFAULTLUS
  ALLOWAPPL *
  USSTCP USSVLAD1
ENDVTAM

START M032216B
```

Figure 5-45   PROFILE data set for RA03 IP stack

```
 SYSPLEX DISTRIBUTOR TARGET STACK: RA28    2

IPCONFIG
  DATAGRAMFWD          6
  DYNAMICXCF 172.16.233.28 255.255.255.0 1      5
  SYSPLEXROUTING     4
  IGNOREREDIRECT
  VARSUBNETTING

PORT
  ...
  20  TCP OMVS      NOAUTOLOG ; FTP SERVER
  21  TCP FTPDC1              ; FTP SERVER
  23  TCP INTCLIEN      10
  ...

TELNETPARMS          14
  PORT 23
  INACTIVE 0
ENDTELNETPARMS

AUTOLOG 5
  FTPDC JOBNAME FTPDC1
  OMPROUTC
  T03CSM                     11
ENDAUTOLOG

  DEVICE M282216B MPCPTP AUTORESTART
  LINK   M282216B MPCPTP M282216B

HOME
  172.16.101.28 M282216B

VIPADYNAMIC
  VIPABACKUP 200 172.16.251.3
ENDVIPADYNAMIC

BEGINVTAM               15
  PORT 23
  DEFAULTLUS RA28TV21..RA28TV29 ENDDEFAULTLUS
  ALLOWAPPL *
  USSTCP USSVLAD1
ENDVTAM

START M282216B
```

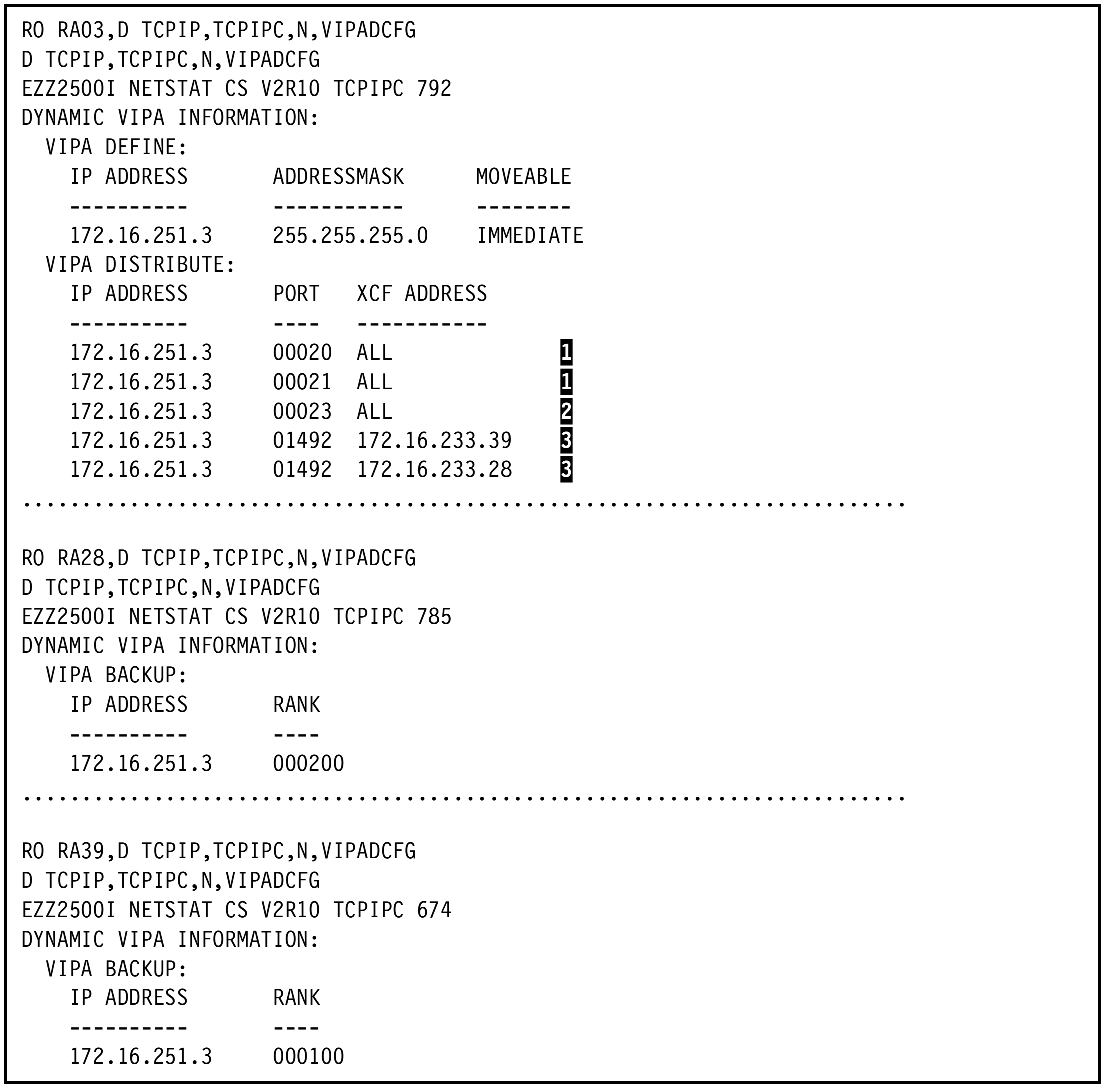
Figure 5-46   PROFILE data set for RA28 IP stack

```
;SYSPLEX DISTRIBUTOR : RA39 (2.BACKUP)

IPCONFIG
 DATAGRAMFWD         6
 DYNAMICXCF 172.16.233.39 255.255.255.0 1   5
 SYSPLEXROUTING     4
 IGNOREREDIRECT
 VARSUBNETTING

PORT
 ...
 20  TCP OMVS      NOAUTOLOG ; FTP SERVER
 21  TCP FTPDC1              ; FTP SERVER
 23  TCP INTCLIEN      10
 ...

TELNETPARMS          14
 PORT 23
 INACTIVE 0
ENDTELNETPARMS

AUTOLOG 5
 FTPDC JOBNAME FTPDC1
 OMPROUTC
 T03CSM                     11
ENDAUTOLOG

 DEVICE M392216B MPCPTP AUTORESTART
 LINK   M392216B MPCPTP M392216B

HOME
 172.16.102.39 M392216B

VIPADYNAMIC
 VIPABACKUP 100 172.16.251.3
ENDVIPADYNAMIC

BEGINVTAM              15
 PORT 23
 DEFAULTLUS
    RA39TV31..RA39TV39
 ENDDEFAULTLUS
 ALLOWAPPL *
 USSTCP USSVLAD1
ENDVTAM

START M392216B
```

Figure 5-47   PROFILE data set for RA39 IP stack

Figure 5-48 on page 159 shows the NESTAT VIPADCFG output for all the IP stacks participating in the sysplex cluster. Comparing this output with the one shown in Figure 5-19 on page 140, we see that only the information for the RA03 IP stack has changed. This is because we did not modify Dynamic VIPA definitions for the PROFILE data sets for RA28 and RA39.

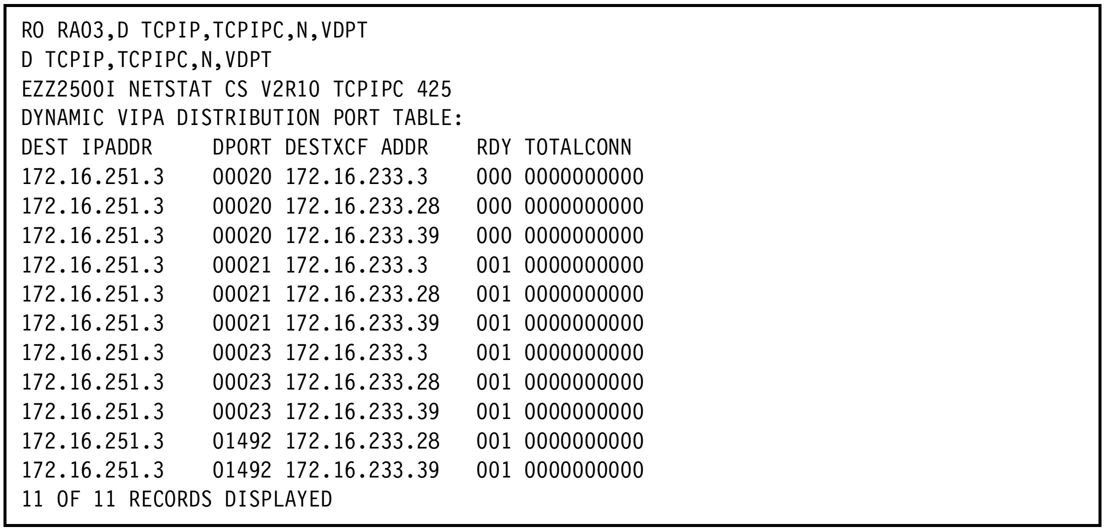
The way that this new configuration will distribute the workload is displayed in Figure 5-48. The three stacks RA03, RA28, and RA39 will be considered for FTP 1 and TN3270E 2 connection requests. The connection requests for RSSERVER 3 will be distributed only to RA28 and RA39.



```
RO RA03,D TCPIP,TCPIPC,N,VIPADCFG
D TCPIP,TCPIPC,N,VIPADCFG
EZZ2500I NETSTAT CS V2R10 TCPIPC 792
DYNAMIC VIPA INFORMATION:
  VIPA DEFINE:
    IP ADDRESS       ADDRESSMASK      MOVEABLE
    ----------       -----------      --------
    172.16.251.3     255.255.255.0    IMMEDIATE
  VIPA DISTRIBUTE:
    IP ADDRESS       PORT   XCF ADDRESS
    ----------       ----   -----------
    172.16.251.3     00020  ALL               1
    172.16.251.3     00021  ALL               1
    172.16.251.3     00023  ALL               2
    172.16.251.3     01492  172.16.233.39     3
    172.16.251.3     01492  172.16.233.28     3
...........................................................................

RO RA28,D TCPIP,TCPIPC,N,VIPADCFG
D TCPIP,TCPIPC,N,VIPADCFG
EZZ2500I NETSTAT CS V2R10 TCPIPC 785
DYNAMIC VIPA INFORMATION:
  VIPA BACKUP:
    IP ADDRESS       RANK
    ----------       ----
    172.16.251.3     000200
...........................................................................

RO RA39,D TCPIP,TCPIPC,N,VIPADCFG
D TCPIP,TCPIPC,N,VIPADCFG
EZZ2500I NETSTAT CS V2R10 TCPIPC 674
DYNAMIC VIPA INFORMATION:
  VIPA BACKUP:
    IP ADDRESS       RANK
    ----------       ----
    172.16.251.3     000100
```

Figure 5-48 D TCPIP,TCPIPC,N,VIPACDFG for RA03, RA28 and RA39 IP stack

Figure 5-49 on page 160 displays the NETSTAT VDPT output for the distributing stack. The output for the same command on target stacks does not give any information. Here we see that the distributing IP stack already notices the existence of servers listening on ports 21, 23, and 1492.



```
RO RA03,D TCPIP,TCPIPC,N,VDPT
D TCPIP,TCPIPC,N,VDPT
EZZ2500I NETSTAT CS V2R10 TCPIPC 425
DYNAMIC VIPA DISTRIBUTION PORT TABLE:
DEST IPADDR     DPORT DESTXCF ADDR    RDY TOTALCONN
172.16.251.3    00020 172.16.233.3    000 0000000000
172.16.251.3    00020 172.16.233.28   000 0000000000
172.16.251.3    00020 172.16.233.39   000 0000000000
172.16.251.3    00021 172.16.233.3    001 0000000000
172.16.251.3    00021 172.16.233.28   001 0000000000
172.16.251.3    00021 172.16.233.39   001 0000000000
172.16.251.3    00023 172.16.233.3    001 0000000000
172.16.251.3    00023 172.16.233.28   001 0000000000
172.16.251.3    00023 172.16.233.39   001 0000000000
172.16.251.3    01492 172.16.233.28   001 0000000000
172.16.251.3    01492 172.16.233.39   001 0000000000
11 OF 11 RECORDS DISPLAYED
```

Figure 5-49   D TCPIP,TCPIP,N,VDPT for RA03 IP stack after distributing multiple IP services

After this configuration has been running for a while, we can use the NETSTAT VCRT command to display the current Dynamic VIPA connection routing table. Figure 5-50 on page 161 displays only a portion of this output for the distributing stack. Please remember that this output can be very large, because it shows all active connections between the client and the participating IP stacks.

```
RO RA03,D TCPIP,TCPIPC,N,VCRT,MAX=500
D TCPIP,TCPIPC,N,VCRT,MAX=500
EZZ2500I NETSTAT CS V2R10 TCPIPC 397
DYNAMIC VIPA CONNECTION ROUTING TABLE:
DEST IPADDR      DPORT  SRC IPADDR       SPORT  DESTXCF ADDR
172.16.251.3     00020  9.24.104.75      03304  172.16.233.3
172.16.251.3     00020  9.24.104.75      03306  172.16.233.3
.......
172.16.251.3     00020  9.24.104.75      03251  172.16.233.28
172.16.251.3     00020  9.24.104.75      03252  172.16.233.28
.......
172.16.251.3     00020  9.24.104.75      03301  172.16.233.39
172.16.251.3     00020  9.24.104.75      03303  172.16.233.39
.......
172.16.251.3     00021  9.24.104.75      03809  172.16.233.3
172.16.251.3     00021  9.24.104.75      03976  172.16.233.3
.......
172.16.251.3     00021  9.24.104.75      03799  172.16.233.28
172.16.251.3     00021  9.24.104.75      03961  172.16.233.28
.......
172.16.251.3     00021  9.24.104.75      01868  172.16.233.39
172.16.251.3     00021  9.24.104.75      01909  172.16.233.39
.......
172.16.251.3     00023  9.24.106.247     04850  172.16.233.3
172.16.251.3     00023  9.24.106.247     04858  172.16.233.3
.......
172.16.251.3     00023  9.24.106.247     04847  172.16.233.28
172.16.251.3     00023  9.24.106.247     04848  172.16.233.28
.......
172.16.251.3     00023  9.24.106.247     04849  172.16.233.39
172.16.251.3     00023  9.24.106.247     04853  172.16.233.39
.......
172.16.251.3     01492  9.24.104.75      03501  172.16.233.28
172.16.251.3     01492  9.24.104.75      03502  172.16.233.28
.......
172.16.251.3     01492  9.24.104.75      03475  172.16.233.39
172.16.251.3     01492  9.24.104.75      03505  172.16.233.39
.......
```

Figure 5-50 D TCPIP,TCPIPC,N.VCRT for RA03 IP stack after distributing multiple IP services

While we were establishing connections using the Sysplex Distributor to balance the workload in conjunction with Workload Manager, we did two displays of the distributing port table of RA03, our distributing stack. Figure 5-51 on page 162 and Figure 5-52 on page 163 show the resulting display. These displays show that the WLM/QoS weight for RA03 1 is always 01, while the WLM/QoS weight of RA28 2 is always 02. The WLM/QoS weight for RA39 3 periodically switches from 02 to 01 and back. That actually explains why we have a connection rate for RA03 of 21%, RA28 of 42%, and RA38 of 37%. When we compare these values with the actual CPU load of these stacks, the distribution of the workload in conjunction with WLM works very well.

Finally, the number of connections that have been distributed since the Sysplex Distributor function was started is also displayed with the NETSTAT VDPT command.

```
RO RA03,D TCPIP,TCPIPC,N,VDPT,DET
D TCPIP,TCPIPC,N,VDPT,DET
EZZ2500I NETSTAT CS V2R10 TCPIPC 862
DYNAMIC VIPA DISTRIBUTION PORT TABLE:
DEST IPADDR     DPORT DESTXCF ADDR    RDY TOTALCONN
172.16.251.3    00020 172.16.233.3    000 0000000258
  WLM: 01  W/Q: 01    1
172.16.251.3    00020 172.16.233.28   000 0000000455
  WLM: 02  W/Q: 02    2
172.16.251.3    00020 172.16.233.39   000 0000000350
  WLM: 02  W/Q: 02    3
172.16.251.3    00021 172.16.233.3    001 0000000007
  WLM: 01  W/Q: 01    1
172.16.251.3    00021 172.16.233.28   001 0000000011
  WLM: 02  W/Q: 02    2
172.16.251.3    00021 172.16.233.39   001 0000000009
  WLM: 02  W/Q: 02    3
172.16.251.3    00023 172.16.233.3    001 0000000180
  WLM: 01  W/Q: 01    1
172.16.251.3    00023 172.16.233.28   001 0000000360
  WLM: 02  W/Q: 02    2
172.16.251.3    00023 172.16.233.39   001 0000000360
  WLM: 02  W/Q: 02    3
172.16.251.3    01492 172.16.233.28   001 0000000229
  WLM: 02  W/Q: 02    2
172.16.251.3    01492 172.16.233.39   001 0000000192
  WLM: 02  W/Q: 02    3
11 OF 11 RECORDS DISPLAYED
```

Figure 5-51 D TCPIP,TCPIPC,N,VDPT,DET on RA03 IP stack after running multiple IP services for some time

```
RO RA03,D TCPIP,TCPIPC,N,VDPT,DET
D TCPIP,TCPIPC,N,VDPT,DET
EZZ2500I NETSTAT CS V2R10 TCPIPC 273
DYNAMIC VIPA DISTRIBUTION PORT TABLE:
DEST IPADDR     DPORT DESTXCF ADDR    RDY TOTALCONN
172.16.251.3    00020 172.16.233.3    000 0000001600
  WLM: 01  W/Q: 01
172.16.251.3    00020 172.16.233.28   000 0000003097
  WLM: 02  W/Q: 02
172.16.251.3    00020 172.16.233.39   000 0000002791
  WLM: 01  W/Q: 01    3
172.16.251.3    00021 172.16.233.3    001 0000000023
  WLM: 01  W/Q: 01
172.16.251.3    00021 172.16.233.28   001 0000000044
  WLM: 02  W/Q: 02
172.16.251.3    00021 172.16.233.39   001 0000000040
  WLM: 01  W/Q: 01    3
172.16.251.3    00023 172.16.233.3    001 0000000392
  WLM: 01  W/Q: 01
172.16.251.3    00023 172.16.233.28   001 0000000779
  WLM: 02  W/Q: 02
172.16.251.3    00023 172.16.233.39   001 0000000629
  WLM: 01  W/Q: 01    3
172.16.251.3    01492 172.16.233.28   001 0000000242
  WLM: 02  W/Q: 02
172.16.251.3    01492 172.16.233.39   001 0000000207
  WLM: 01  W/Q: 01    3
11 OF 11 RECORDS DISPLAYED
```

Figure 5-52 D TCPIP,TCPIPC,N,VDPT,DET on RA03 IP stack after running multiple IP services for some time

5.6.4 Scenario 4: Deleting and adding a VIPADISTRIBUTE statement

With the DELETE parameter of the VIPADISTRIBUTE statement, we can delete a previous designation of a dynamic VIPA as distributable. This gives you the ability to stop distribution for a certain application/port. In this scenario (depicted in Figure 5-53 on page 164), we stop the distribution of the RSSERVER on port 1492. After deleting the VIPADISTRIBUTE for this port 1492, we add a VIPADISTRIBUTE statement for the Web server using port 80.

If the VIPADISTRIBUTE DELETE statement is defined with the keyword DESTIP ALL, then it is not possible to use the VIPADISTRIBUTE with the DESTIP <dynxcfip> syntax. You must use the VIPADISTRIBUTE DELETE with the keyword DESTIP ALL and then you can use VIPADISTRIBUTE DEFINE with keyword DESTIP <dynxcfip> to specify the new specific stacks for distribution consideration.

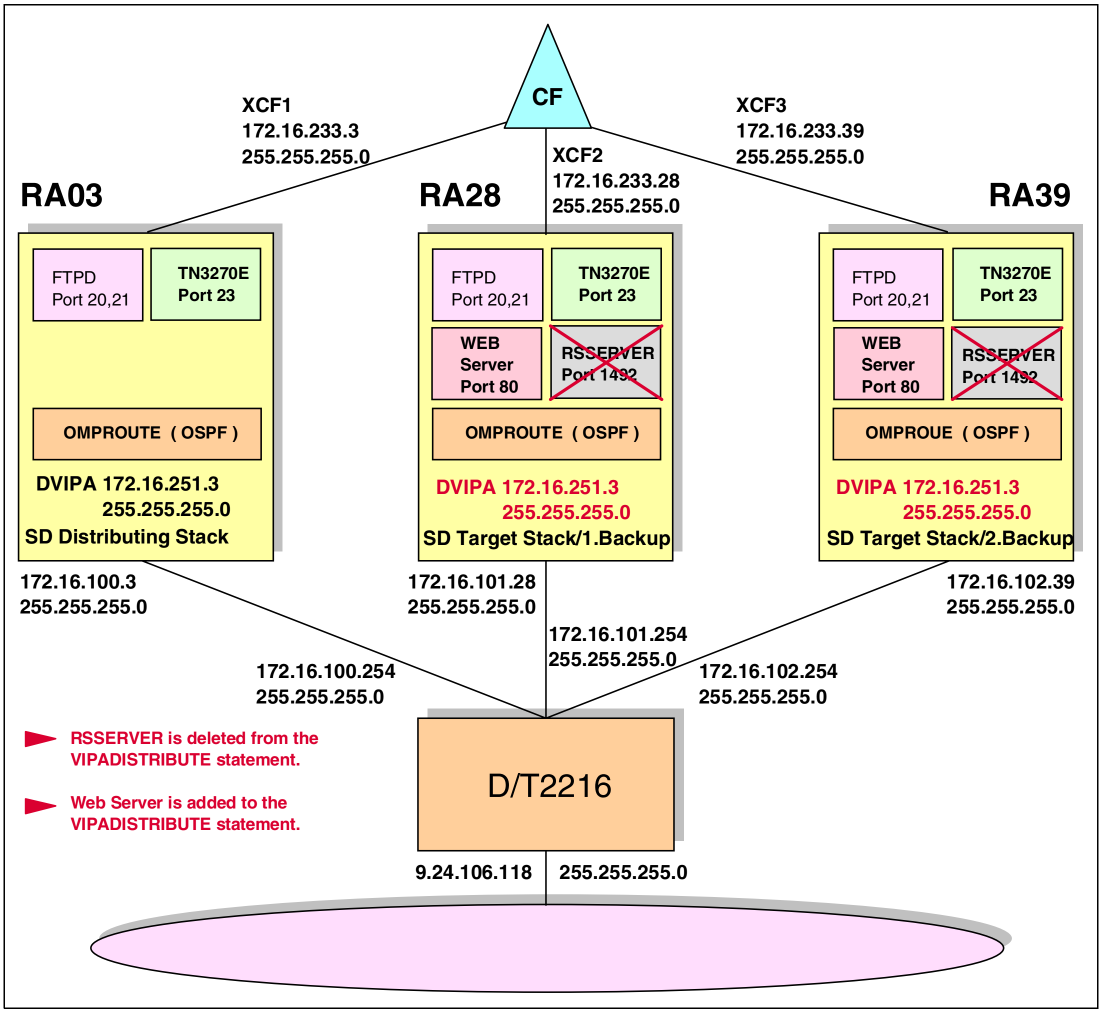


Figure 5-53 Deleting RSSERVER distribution and adding Web Server distribution dynamically

We used the VARY OBEY file shown in Figure 5-54 to delete the RSSERVER from distribution. In our case, we could use the VIPADISTRIBUTE DELETE statement with the parameter DESTIP <dynxcfip>, since we were only distributing the port 1492 1 for the RSSERVER to the RA28 IP stack 2 and the RA39 IP stack 2. Note that existing connections are maintained even after processing the VIPADISTRIBUTE DELETE.

```
VIPADYNAMIC
 VIPADISTRIBUTE DELETE 172.16.251.3 PORT 1492  1
     DESTIP 172.16.233.28 172.16.233.39        2
ENDVIPADYNAMIC
```

Figure 5-54 OBEY file for VIPADISTRIBUTE DELETE of port 1492

Figure 5-55 on page 165 displays the Dynamic VIPA destination port table of RA03 stack after deleting the VIPADISTRIBUTE statement for the RSSERVER port 1492. At this point, we see that the port 1492 is no longer being distributed.

```
RO RA03,D TCPIP,TCPIPC,N,VDPT
D TCPIP,TCPIPC,N,VDPT
EZZ2500I NETSTAT CS V2R10 TCPIPC 661
DYNAMIC VIPA DISTRIBUTION PORT TABLE:
DEST IPADDR     DPORT DESTXCF ADDR    RDY TOTALCONN
172.16.251.3    00020 172.16.233.3    000 0000002240
172.16.251.3    00020 172.16.233.28   000 0000004480
172.16.251.3    00020 172.16.233.39   000 0000004080
172.16.251.3    00021 172.16.233.3    001 0000000031
172.16.251.3    00021 172.16.233.28   001 0000000061
172.16.251.3    00021 172.16.233.39   001 0000000055
172.16.251.3    00023 172.16.233.3    001 0000000452
172.16.251.3    00023 172.16.233.28   001 0000000899
172.16.251.3    00023 172.16.233.39   001 0000000749
9 OF 9 RECORDS DISPLAYED
```

Figure 5-55 D TCPIP,TCPIPC,N,VDPT on RA03 IP stack after deleting port 1492 from distribution

We then started the Web server on stacks RA28 and RA39 and verified that the server was listening on port 80. After that, we added the VIPADISTRIBUTE DEFINE statement for port 80 with the XCF IP addresses for RA28 and RA39 with the VARY OBEY file shown in Figure 5-56.

```
VIPADYNAMIC
 VIPADISTRIBUTE 172.16.251.3 PORT 80
     DESTIP 172.16.233.28 172.16.233.39
ENDVIPADYNAMIC
```

Figure 5-56 VARY OBEY file to distribute the Web server on port 80

Figure 5-57 shows the Dynamic VIPA destination port table of RA03 stack after adding the distribution of the Web server.

```
RO RA03,D TCPIP,TCPIPC,N,VDPT
D TCPIP,TCPIPC,N,VDPT
EZZ2500I NETSTAT CS V2R10 TCPIPC 736
DYNAMIC VIPA DISTRIBUTION PORT TABLE:
DEST IPADDR     DPORT DESTXCF ADDR    RDY TOTALCONN
172.16.251.3    00020 172.16.233.3    000 0000002240
172.16.251.3    00020 172.16.233.28   000 0000004480
172.16.251.3    00020 172.16.233.39   000 0000004080
172.16.251.3    00021 172.16.233.3    001 0000000031
172.16.251.3    00021 172.16.233.28   001 0000000061
172.16.251.3    00021 172.16.233.39   001 0000000055
172.16.251.3    00023 172.16.233.3    001 0000000452
172.16.251.3    00023 172.16.233.28   001 0000000899
172.16.251.3    00023 172.16.233.39   001 0000000749
172.16.251.3    00080 172.16.233.28   001 0000000000
172.16.251.3    00080 172.16.233.39   001 0000000000
11 OF 11 RECORDS DISPLAYED
```

Figure 5-57 D TCPIP,TCPIPC,N,VDPT on RA03 after VIPADISTRIBUTE DEFINE for port 80

Figure 5-58 on page 166 shows only a part of the connection table after establishing connections to the Web server using the Sysplex Distributor to balance the workload between the IP stacks RA28 and RA39.

```
RO RA03,D TCPIP,TCPIPC,N,VCRT
D TCPIP,TCPIPC,N,VCRT
EZZ2500I NETSTAT CS V2R10 TCPIPC 895
DYNAMIC VIPA CONNECTION ROUTING TABLE:
DEST IPADDR      DPORT  SRC IPADDR       SPORT  DESTXCF ADDR
.......
172.16.251.3     00080  9.24.106.32      01083  172.16.233.28
172.16.251.3     00080  9.24.106.102     01182  172.16.233.28
172.16.251.3     00080  9.24.106.252     01130  172.16.233.28
172.16.251.3     00080  9.24.106.247     03550  172.16.233.39
172.16.251.3     00080  9.24.106.252     01131  172.16.233.39
.......
```

Figure 5-58 D TCPIP,TCPIPC,N,VCRT on RA03 after establishing connection to Web server

Finally, to show the number of connections that have been distributed since the Sysplex Distributor function was started, we issue the command as shown in Figure 5-59.

```
RO RA03,D TCPIP,TCPIPC,N,VDPT
D TCPIP,TCPIPC,N,VDPT
EZZ2500I NETSTAT CS V2R10 TCPIPC 898
DYNAMIC VIPA DISTRIBUTION PORT TABLE:
DEST IPADDR     DPORT DESTXCF ADDR    RDY TOTALCONN
172.16.251.3    00020 172.16.233.3    000 0000002240
172.16.251.3    00020 172.16.233.28   000 0000004480
172.16.251.3    00020 172.16.233.39   000 0000004080
172.16.251.3    00021 172.16.233.3    001 0000000031
172.16.251.3    00021 172.16.233.28   001 0000000061
172.16.251.3    00021 172.16.233.39   001 0000000055
172.16.251.3    00023 172.16.233.3    001 0000000452
172.16.251.3    00023 172.16.233.28   001 0000000899
172.16.251.3    00023 172.16.233.39   001 0000000749
172.16.251.3    00080 172.16.233.28   001 0000000032
172.16.251.3    00080 172.16.233.39   001 0000000024
11 OF 11 RECORDS DISPLAYED
```

Figure 5-59 D TCPIP,TCPIPC,N,VDPT on RA03 after some time running the distribution

5.6.5 Scenario 5: Removing a target stack from distribution

There are a few reasons why an IP stack may need to be removed from distribution. One example is the event of a planned maintenance update for this stack, which should affect the least number of users as possible. Removing the stack from distribution will ensure that all the new connections are sent to the other participating stacks in the sysplex. Existing connections on the removed stack will still be maintained until their normal completion.

Let’s assume we still have the same environment as we had after 5.6.4, “Scenario 4: Deleting and adding a VIPADISTRIBUTE statement” on page 163. In this case, we are running the FTP and Telnet server on all three IP stacks (RA03,RA28, and RA39). The Web server is running on IP stacks RA28 and RA39. Now we remove the target IP stack RA39 from distribution. There is also new status information for the DVIPAs. We redefine the IP stack RA39 just as a target stack and remove the VIPABACKUP definition.

Figure 5-60 on page 167 shows you all the participating IP stacks in the sysplex and their status.

```
RO RA03,D TCPIP,TCPIPC,SYSPLEX,VIPADYN
D TCPIP,TCPIPC,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V2R10 147
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.251.0    BOTH
  TCPIPC   RA28     BACKUP 200                                  DEST
  TCPIPC   RA39     BACKUP 100                                  DEST
3 OF 3 RECORDS DISPLAYED
```

Figure 5-60   D TCPIP,TCPIPC,SYSPLEX,VIPADYN

Now we change the IP stack RA39 to a target stack without any backup capabilities. Remember that we have defined RA39 as our second backup for RA03. This change is necessary because of the new status of the DVIPAs that were introduced with IBM Communications Server for OS/390 V2R10. If RA39 would still be defined as backup, the new DVIPA status is never displayed; it would show only the BACKUP status. We do this with the VARY OBEY file on IP stack RA39 as shown in Figure 5-61. We also automatically delete the VIPBACKUP definition on RA39 by deleting the RA03 DVIPA on RA39.

```
VIPADYNAMIC
 VIPADELETE 172.16.251.3
ENDVIPADYNAMIC
```

Figure 5-61   VARY OBEY file for deleting VIPABACKUP definition on RA39

Note: This will not remove the DVIPA entry from the home list of IP stack RA39, because we are still a target for distribution.

Figure 5-62 shows you the participating IP stacks in the sysplex and their status after deleting the VIPABACKUP definition on RA39. RA39 now shows the status of ACTIVE 1 and a distribution status of DEST 2, which only shows target stacks without backup capabilities.

```
RO RA03,D TCPIP,TCPIPC,SYSPLEX,VIPADYN
D TCPIP,TCPIPC,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V2R10 151
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.251.0    BOTH
  TCPIPC   RA28     BACKUP 200                                  DEST
  TCPIPC   RA39   1 ACTIVE                                      DEST 2
3 OF 3 RECORDS DISPLAYED
```

Figure 5-62   D TCPIP,TCPIPC,SYSPLEX,VIPADYN after deleting VIPABACKUP definition

Figure 5-63 on page 168 and Figure 5-64 on page 168 show us that we are distributing to all three stacks for FTP 1 and Telnet 2. For the Web server 3 we are distributing to RA28 and RA39. At the moment we have established only four Telnet connections: two active connections to RA28 4 and two active connections to RA39 5.

```
RO RA03,D TCPIP,TCPIPC,N,VDPT
D TCPIP,TCPIPC,N,VDPT
EZZ2500I NETSTAT CS V2R10 TCPIPC 154
DYNAMIC VIPA DISTRIBUTION PORT TABLE:
DEST IPADDR     DPORT DESTXCF ADDR    RDY TOTALCONN
172.16.251.3    00020 172.16.233.3    000 0000001716
172.16.251.3    00020 172.16.233.28   000 0000003282
172.16.251.3    00020 172.16.233.39   000 0000003202
172.16.251.3    00021 172.16.233.3    001 0000000039 1
172.16.251.3    00021 172.16.233.28   001 0000000081 1
172.16.251.3    00021 172.16.233.39   001 0000000080 1
172.16.251.3    00023 172.16.233.3    001 0000000754 2
172.16.251.3    00023 172.16.233.28   001 0000000877 2
172.16.251.3    00023 172.16.233.39   001 0000000873 2
172.16.251.3    00080 172.16.233.28   001 0000000044 3
172.16.251.3    00080 172.16.233.39   001 0000000050 3
11 OF 11 RECORDS DISPLAYED
```

Figure 5-63 D TCPIP,TCPIPC,N,VDPT before removing target stack RA39 from distribution

```
RO RA03,D TCPIP,TCPIPC,N,VCRT
D TCPIP,TCPIPC,N,VCRT
EZZ2500I NETSTAT CS V2R10 TCPIPC 157
DYNAMIC VIPA CONNECTION ROUTING TABLE:
DEST IPADDR      DPORT  SRC IPADDR       SPORT  DESTXCF ADDR
172.16.251.3     00023  9.24.106.247     01740  172.16.233.28 4
172.16.251.3     00023  9.24.106.247     01741  172.16.233.28 4
172.16.251.3     00023  9.24.106.247     01742  172.16.233.39 5
172.16.251.3     00023  9.24.106.247     01743  172.16.233.39 5
4 OF 4 RECORDS DISPLAYED
```

Figure 5-64 D TCPIP,TCPIPC,N,VCRT before removing target stack RA39 from distribution

At this point, we use the VIDISTRIBUTE DELETE statement in a VARY OBEY file to delete all the ports defined for distribution to RA39. At the moment we are distributing ports 20, 21, 23 and 80 to IP stack RA39. Remember, ports 20, 21 and 23 were defined in the PROFILE using the keyword DESTIP ALL. To remove distribution for these ports, we first have to use the VIPADISTRIBUTE DELETE with DESTIP ALL 1 before we can use the VIPADISTRIBUTE DEFINE 2 for the special XCF addresses (review explanation in 5.6.4, “Scenario 4: Deleting and adding a VIPADISTRIBUTE statement” on page 163). Distribution for port 80 was done with a VARY OBEY file and we used the XCF addresses in the DESTIP keyword. To remove this distribution, we can use the VIPADISTRIBUTE DELETE 3 statement with the keyword DESTIP <dynxcfip>. Please refer to Figure 5-65.

```
VIPADYNAMIC
 VIPADISTRIBUTE DELETE 172.16.251.3 PORT 20 21 23
     DESTIP ALL  1
 VIPADISTRIBUTE DELETE 172.16.251.3 PORT 80
     DESTIP 172.16.233.39  3
 VIPADISTRIBUTE 172.16.251.3 PORT 20 21 23
     DESTIP 172.16.233.3  172.16.233.28  2
ENDVIPADYNAMIC
```

Figure 5-65 OBEY file to remove only the distribution to target IP stack RA39

Figure 5-66 and Figure 5-67 show us that the target IP stack RA39 is removed from distribution. There is no entry in the destination port table for this stack as a distribution target. But the existing 1 connections are still maintained (not broken).

```
RO RA03,D TCPIP,TCPIPC,N,VDPT
D TCPIP,TCPIPC,N,VDPT
EZZ2500I NETSTAT CS V2R10 TCPIPC 174
DYNAMIC VIPA DISTRIBUTION PORT TABLE:
DEST IPADDR     DPORT DESTXCF ADDR    RDY TOTALCONN
172.16.251.3    00020 172.16.233.3    000 0000001716
172.16.251.3    00020 172.16.233.28   000 0000003282
172.16.251.3    00021 172.16.233.3    001 0000000039
172.16.251.3    00021 172.16.233.28   001 0000000081
172.16.251.3    00023 172.16.233.3    001 0000000754
172.16.251.3    00023 172.16.233.28   001 0000000877
172.16.251.3    00080 172.16.233.28   001 0000000044
7 OF 7 RECORDS DISPLAYED
```

Figure 5-66 D TCPIP,TCPIPC,N,VDPT after removing target IP stack RA39 from distribution

```
RO RA03,D TCPIP,TCPIPC,N,VCRT
D TCPIP,TCPIPC,N,VCRT
EZZ2500I NETSTAT CS V2R10 TCPIPC 286
DYNAMIC VIPA CONNECTION ROUTING TABLE:
DEST IPADDR      DPORT  SRC IPADDR       SPORT  DESTXCF ADDR
172.16.251.3     00023  9.24.106.247     01740  172.16.233.28
172.16.251.3     00023  9.24.106.247     01741  172.16.233.28
172.16.251.3     00023  9.24.106.247     01742  172.16.233.39 1
172.16.251.3     00023  9.24.106.247     01743  172.16.233.39 1
4 OF 4 RECORDS DISPLAYED
```

Figure 5-67 D TCPIP,TCPIPC,N,VCRT after removing target IP stack RA39 from distribution

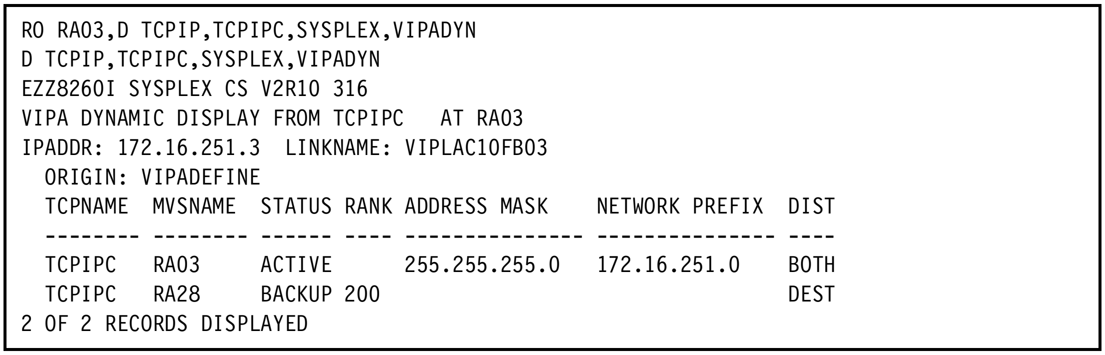
We now show the current dynamic VIPA information after removing the target IP stack RA39 from distribution. The DVIPA shows a new status of QUIESCING 1, which was added in IBM Communications Server for OS/390 V2R10. This state indicates that this DVIPA is no longer a target for distribution. Existing connections are still kept active and functional. The distribution status 2 is blank on the display, since we have actually removed the IP stack from distribution. Please refer to Figure 5-68.

```
RO RA03,D TCPIP,TCPIPC,SYSPLEX,VIPADYN
D TCPIP,TCPIPC,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V2R10 310
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.251.0    BOTH
  TCPIPC   RA28     BACKUP 200                                  DEST
  TCPIPC   RA39   1 QUIESC      255.255.255.0   0.0.0.0            2
3 OF 3 RECORDS DISPLAYED
```

Figure 5-68 D TCPIP,TCPIPC,SYSPLEX,VIPADYN after removing target IP stack RA39 from distribution

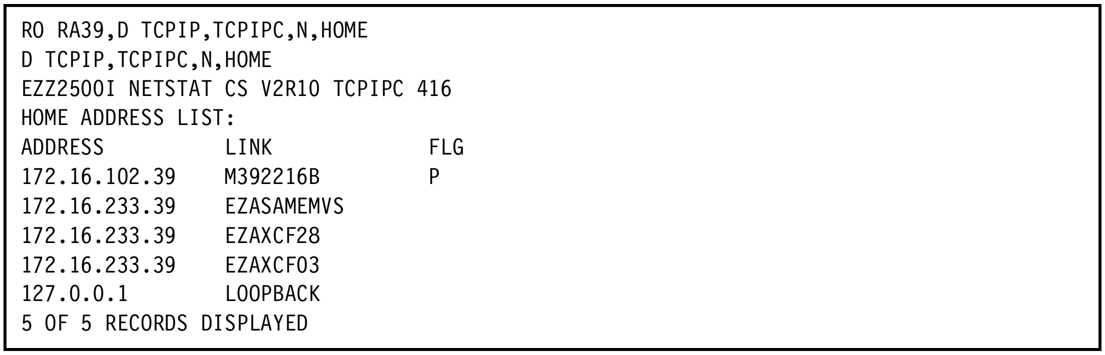
Note: If the target IP stack RA39 would be defined as a backup for RA03, the display would show the status BACKUP and not QUIESCING.

Figure 5-69 shows the current Dynamic VIPA information after disconnecting the connections to target IP stack RA39. The IP stack RA39 is no longer displayed on this command, since it is now completely removed from distribution. Also, the DVIPA entry of RA03 in the home list of RA39 is removed at this time. Please see Figure 5-70 for that display.



```
RO RA03,D TCPIP,TCPIPC,SYSPLEX,VIPADYN
D TCPIP,TCPIPC,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V2R10 316
VIPA DYNAMIC DISPLAY FROM TCPIPC   AT RA03
IPADDR: 172.16.251.3  LINKNAME: VIPLAC10FB03
  ORIGIN: VIPADEFINE
  TCPNAME  MVSNAME  STATUS RANK ADDRESS MASK    NETWORK PREFIX  DIST
  -------- -------- ------ ---- --------------- --------------- ----
  TCPIPC   RA03     ACTIVE      255.255.255.0   172.16.251.0    BOTH
  TCPIPC   RA28     BACKUP 200                                  DEST
2 OF 2 RECORDS DISPLAYED
```

Figure 5-69 D TCPIP,TCPIPC,SYSPLEX,VIPADYN after disconnecting all connections to RA39



```
RO RA39,D TCPIP,TCPIPC,N,HOME
D TCPIP,TCPIPC,N,HOME
EZZ2500I NETSTAT CS V2R10 TCPIPC 416
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.102.39    M392216B         P
172.16.233.39    EZASAMEMVS
172.16.233.39    EZAXCF28
172.16.233.39    EZAXCF03
127.0.0.1        LOOPBACK
5 OF 5 RECORDS DISPLAYED
```

Figure 5-70 D TCPIP,TCPIPC,N,HOME after disconnecting all connections to RA39

5.6.6 Scenario 4 - Fast connection reset demonstration

CS for z/OS V1R2 IP enhances the Sysplex Distributor by adding a new feature called *fast connection reset*. In the case of failure of any target stack, the distributor stack now has the ability to reset all the connections made with the failure target stack, making the client recovery faster. Now the clients will receive a reset packet (TCP layer) for each connection they have with the failing target stack.

We created the scenario illustrated in Figure 5-71 on page 171 to demonstrate how this new feature works.

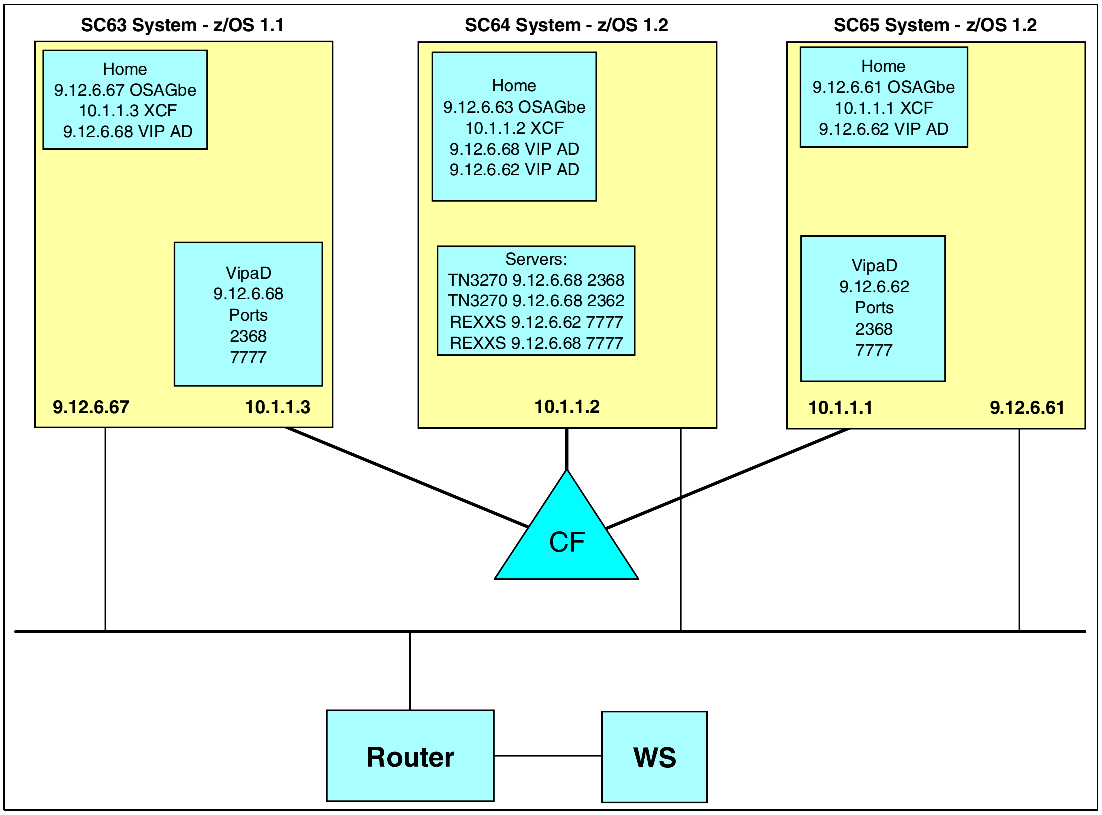


Figure 5-71 Fast connect reset demonstration scenario

We have two stacks distributing connections to the same target stack. Two of them, the system SC64 and SC65 are z/OS V1R2 systems with the fast connection reset feature and the system SC63 is a z/OS V1R1 (using CS for OS/390 V2R10 IP) system without the fast connection reset feature.

Each one of the distributing stacks distributes two applications: one is a TN3270E server on two different ports and the other is a REXX program acting as a server for a REXX client running on a workstation.

Note that the two distributed VIPA addresses belong to the same network subnet of the OSA cards. We use this configuration to facilitate the implementation. By using the subnet of the VIPA addresses on the same subnet of the OSA cards, you do not need to use a dynamic routing protocol and implement the OMPROUTE. The disadvantage of this approach is that in case of a failure in the OSA card, the z/OS image would no longer be reachable. Other images using the XCF connection would not be able to route packets to it, since they belong to the same network.

In order to simulate a stack failure, we issued a **`FORCE tcpipprocname,ARM`** command in the target stack SC64.

For each one of the distributing stacks, we started a packet trace to check how the system behaves in case of a failure in a target stack. We do not have another stack to distribute a new connection (a reconnect) coming from the clients. But, in a production environment, we should have more than one target stack to receive new connections

The profile configuration for system SC63 is as follows:

```
IPCONFIG        DATAGRAMFWD
```

```
                 DYNAMICXCF 10.1.1.3 255.255.255.252 1
TCPCONFIG        RESTRICTLOWPORTS
UDPCONFIG        RESTRICTLOWPORTS
DEVICE OSA22E0 MPCIPA
LINK   OSA22E0 IPAQGNET OSA22E0
HOME
   9.12.6.67 OSA22E0
BEGINROUTES
   ROUTE 9.12.6.0   255.255.255.0 = OSA22E0 MTU 1492
   ROUTE DEFAULT    9.12.6.75       OSA22E0 MTU 1492
ENDROUTES
VIPADYNAMIC
   VIPADEFINE MOVEABLE IMMEDIATE 255.255.255.0 9.12.6.68
   VIPADISTRIBUTE DEFINE 9.12.6.68 PORT 2368 7777
      DESTIP 10.1.1.2
ENDVIPADYNAMIC
ITRACE OFF
START OSA22E0
```

For system SC65, we used the following profile:

```
DATASETPREFIX TCPIPB
IPCONFIG         DATAGRAMFWD
                 DYNAMICXCF 10.1.1.1 255.255.255.252 1
TCPCONFIG        RESTRICTLOWPORTS
UDPCONFIG        RESTRICTLOWPORTS
DEVICE OSA22E0 MPCIPA
LINK   OSA22E0 IPAQGNET OSA22E0
HOME
   9.12.6.61 OSA22E0
BEGINROUTES
   ROUTE 9.12.6.0   255.255.255.0 = OSA22E0 MTU 1492
   ROUTE DEFAULT    9.12.6.75       OSA22E0 MTU 1492
ENDROUTES
VIPADYNAMIC
   VIPADEFINE MOVEABLE IMMEDIATE 255.255.255.0 9.12.6.62
   VIPADISTRIBUTE DEFINE 9.12.6.62 PORT 2362 7777
      DESTIP 10.1.1.2
ENDVIPADYNAMIC
ITRACE OFF
START OSA22E0
```

For system SC64, we used the following profile:

```
IPCONFIG         DATAGRAMFWD
                 DYNAMICXCF 10.1.1.2 255.255.255.252 1
TCPCONFIG        RESTRICTLOWPORTS
UDPCONFIG        RESTRICTLOWPORTS
DEVICE OSA22E0 MPCIPA
LINK   OSA22E0 IPAQGNET OSA22E0
HOME
   9.12.6.63 OSA22E0
BEGINROUTES
   ROUTE 9.12.6.0   255.255.255.0 = OSA22E0 MTU 1492
   ROUTE DEFAULT    9.12.6.75       OSA22E0 MTU 1492
ENDROUTES
PORT
   2362 TCP INTCLIEN BIND 9.12.6.62
   2368 TCP INTCLIEN BIND 9.12.6.68
   7777 TCP * SHAREPORT
ITRACE OFF
TELNETPARMS
   PORT 2362
ENDTELNETPARMS
TELNETPARMS
   PORT 2368
ENDTELNETPARMS
BEGINVTAM
  PORT 2362 2368
  DEFAULTLUS
     TCP64B00..TCP64B99
  ENDDEFAULTLUS
  ALLOWAPPL *
ENDVTAM
         START OSA22E0
```

Note the port definitions: we used two TN3270 servers in different IP ports and IP addresses using the BIND option in the PORT statement. For the REXX server, we are sharing the PORT with two servers. Each one of the servers will use a specific BIND to listen for connections using different IP addresses.

The REXX server program is as follows:

```
/* REXX */
trace i
parse arg vipa .
if vipa="" then do say "Vipa ????"; exit; end
id=MvsVar(SysClone)
signal on syntax
signal on halt
port = '7777'

rc=Socket('Initialize','rodolfi',50)
parse var rc rc rest
if rc<>0 then call Error rc 'Init' rest

rc=Socket('Socket')
parse var rc rc s rest
if rc<>0 then call Error rc 'Socket' s rest

rc=Socket('Bind',s,'AF_INET' port vipa)
parse var rc rc rest
if rc<>0 then call Error rc 'Bind' rest

rc=Socket('Listen',s,10)
parse var rc rc rest
if rc<>0 then call Error rc 'Listen' rest

do forever
   rc=Socket('Accept',s)
   parse var rc rc ns rest
   if rc<>0 then call Error rc 'Accept' ns rest
   call Socket 'SetSockOpt',ns,'SOL_SOCKET','So_ASCII','On'
   rc=Socket('Read',ns)
   parse var rc rc nbytes data
   if rc<>0 then call Error rc 'Read' nbytes data
   say 'SERVER:' nbytes 'bytes received:' data
   rc=Socket('Read',ns)
   parse var rc rc nbytes data
```

```
      if rc<>0 then call Error rc 'Read' nbytes data
      say 'SERVER:' nbytes 'bytes received:' data
      rc=Socket('Close',ns)
      if data='END' then leave
   end

   Flush:
   call Socket 'Shutdown',s,'Both'
   call Socket 'Terminate','Server'
   exit

   Error:
   parse arg rc api rest
   say 'Server: Return code' rc 'on api' api ':' rest
   signal Flush

   Syntax:
   signal Flush

   Halt:
   say 'Server halted ...'
   signal Flush
```

This program expects to receive an IP address as input and will issue the BIND on a specific port. The JCL used to start both servers are shown here.

The JCL to start the SC63 server:

```
//SRV63G   JOB  NOTIFY=RODOLFI,CLASS=A,MSGCLASS=A
//SERVER   EXEC PGM=IRXJCL,REGION=0M,PARM='SERVER 9.12.6.68'
//SYSTSPRT DD   SYSOUT=*
//SYSEXEC  DD   DSN=RODOLFI.WORK,DISP=SHR
//SYSTCPD  DD   DSN=TCPIPB.SC64.TCPPARMS(TCPDATB),DISP=SHR
```

The JCL to start the SC65 server:

```
//SRV65G   JOB  NOTIFY=RODOLFI,CLASS=A,MSGCLASS=A
//SERVER   EXEC PGM=IRXJCL,REGION=0M,PARM='SERVER 9.12.6.62'
//SYSTSPRT DD   SYSOUT=*
//SYSEXEC  DD   DSN=RODOLFI.WORK,DISP=SHR
//SYSTCPD  DD   DSN=TCPIPB.SC65.TCPPARMS(TCPDATB),DISP=SHR
```

For the client side we used a TN3270 client (Personal Communication) and a REXX program developed in Object REXX for Windows. The REXX client is shown here:

```
   /* */
 trace ?i
 rc1 = RxFuncAdd("SockLoadFuncs","rxsock","SockLoadFuncs")
 rc2 = SockLoadFuncs()
 call time(r)
 say cab() "Initializing sockets ..." rc1 rc2 time(e)
 say cab() "Socket Version" SockVersion() time(e)

 rcinit = SockInit()
 if rcinit <> 0 then call Error  "Error" rcinit "on sockinit function."
 socket = SockSocket("AF_INET","SOCK_STREAM","IPPROTO_TCP")
 if socket < 0 then call Error "Error on socksocket call" socket
 say cab() "Socket" socket "initialized ..." time(e)
 address.family = "AF_INET"
 address.port = "7777"
 address.addr = "9.12.6.68"
 rcconnect = SockConnect(socket,"address.")
 if rcconnect <> 0 then call Error "Error on connect" rcconnect
 say cab() "Connected to port" address.port "for socket" socket "to address" address.addr
"ok " rcconnect time(e)

 say cab() "Entre com os dados a serem enviados:"
 parse pull dados
 rcsend = SockSend(socket,dados)
 say cab() "Number of bytes sent on socket" socket":" rcsend
 rcreceive = SockRecv(socket,dados,1024)
 say rcreceive dados

 CLOSE:
 rcclose = SockSoClose(socket)
 say cab() "Closed socket" socket rcclose time(e)
 call SysDropFuncs
 exit 0

 ERROR:
 parse arg message
 errno = SockSock_Errno()
 call SockPSock_Errno
 say cab() "Error Number" errno
 say cab() message
 call SockPSock_Errno
 signal close

 CAB:
 return date() time()
```

We started the three stacks and the two servers in the SC64 system. After that, we started four clients in the workstation: two PCOMMs, one with IP address 9.12.6.68 and port 2368 and one with IP address 9.12.6.62 and port 2362; two REXX clients, one with IP address 9.12.6.68 and port 7777 and one with IP address 9.12.6.62 and port 7777. The output from the NETSTAT SOCKETS command in system SC64 is included here:

```
MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCPIPB           15:51:17
Sockets interface status:
Type   Bound to              Connected to          State    Conn
====   ========              ============          =====    ====
Name: SRV63G    Subtask: 008FF330
Stream 9.12.6.68..7777       9.24.106.46..4691     Establsh 00000029
Stream 9.12.6.68..7777       0.0.0.0..0            Listen   00000018
Name: SRV65G    Subtask: 008FF330
Stream 9.12.6.62..7777       0.0.0.0..0            Listen   0000001A
Stream 9.12.6.62..7777       9.24.106.46..4692     Establsh 0000002A
Name: TCPIPB    Subtask: 00000000
Stream 9.12.6.68..2368       9.24.106.46..4693     Establsh 0000002B
Stream 9.12.6.62..2362       9.24.106.46..4694     Establsh 0000002C
Name: TCPIPB    Subtask: 008CAE78
Stream 9.12.6.62..2362       0.0.0.0..0            Listen   00000011
Name: TCPIPB    Subtask: 008E01E0
Stream 9.12.6.68..2368       0.0.0.0..0            Listen   00000010
Name: TCPIPB    Subtask: 008E0CF8
Stream 127.0.0.1..1026       127.0.0.1..1025       Establsh 0000000E
Name: TCPIPB    Subtask: 008E1B50
Stream 127.0.0.1..1025       127.0.0.1..1026       Establsh 0000000F
Stream 127.0.0.1..1025       0.0.0.0..0            Listen   0000000B
```

Note that we have four clients connected. After having all clients connected we started a packet trace in both SC63 and SC65 systems to trace IP packets. With the trace, we see what happens when stack SC64 goes down. We compare how the fast connection reset feature from CS for z/OS V1R2 IP reacts to the target stack failure.

The commands to start the packet trace on the system console are:

```
v tcpip,tcpipb,pkt,on
trace ct,wtrstart=tcptrc (procedure to collect the trace records)
trace ct,on,comp=systcpda,sub=(tcpipb)
xxx,wtr=tcptrc,options=(internet,tcp),end (this is a reply from the previous command)
```

Now we start the applications to collect the trace packet records. In our test, we started the packet trace, started all the clients, and then we issued a **force TCPIPB, ARM** in system SC64. After all the client connections had terminated, we stopped the trace with the following commands:

```
v tcpip,tcpipb,pkt,off
trace ct,on,comp=systcpda,sub=(tcpipb)
xxx,wtr=disconnect,end (a reply from the previous command)
trace ct,off,comp=systcpda,sub=(tcpipb)
trace ct,wtrstop=tcptrc
```

The procedure used to collect the trace packets was:

```
//TCPTRC  PROC
//CTRACE   EXEC PGM=ITTTRCWR
//SYSPRINT DD SYSOUT=A
//TRCOUT01 DD DSNAME=RODOLFI.&SYSNAME..CTRACE1,DISP=OLD
```

We ran this JCL in both systems, SC63 and SC65, to format both trace packets. Then we formatted the output with the following IPCS JCL:

```
//TSOIPCS  JOB  NOTIFY=GIANCA,CLASS=A,MSGCLASS=A
//TSOBATCH EXEC PGM=IKJEFT01,REGION=0M
//STEPLIB  DD   DSN=TCPIP.SEZAMIG,DISP=SHR
//IPCSPARM DD   DSN=SYS1.IBM.PARMLIB,DISP=SHR
//SYSPROC  DD   DSN=SYS1.SBLSCLI0,DISP=SHR
//SYSTSPRT DD   SYSOUT=*
//IPCSTOC  DD   SYSOUT=*
//IPCSPRNT DD   SYSOUT=*
//SYSTSIN  DD   *
IPCS
CTRACE DSN('RODOLFI.SC65.CTRACE1') -
   COMP(SYSTCPDA) SUB((TCPIPB)) FULL
Y
END
/*
```

With both trace packets formatted, we see the following behavior in the SC65 (CS for z/OS V1R2 IP) stack:

```
SC65      PACKET    00000001  15:53:04.896118  Packet Trace
  TO LINK   =  OSA22E0          DEV = QDIO Ethernet        FULL
PKT 98
TCP SRC PORT = 2362 TCP DST PORT = 4694
   SEQ NUM =         0  ACK NUM =         0  FLAGS = ACK
```

This packet is a probe sent from the distributor stack to one of the clients to synchronize the connection it has with the failing stack. After receiving the answer from the client, the server sent the reset packet:

```
SC65      PACKET    00000001  15:53:05.017776  Packet Trace
TO LINK =  OSA22E0           DEV = QDIO Ethernet        FULL
TCP SRC PORT = 2362 TCP DST PORT = 4694
   FLAGS = RST
```

This reset operation is done for all the connections with the failing target.

5.7 Diagnosing Sysplex Distributor problems

Using the already described NETSTAT and DISPLAY WLM commands, you can have a clear picture of your sysplex environment. Diagnosing Sysplex Distributor problems may be complex, since the DVIPA is associated with more than one IP stack within the sysplex cluster.

Which command you use depends on the specific situation. If your Sysplex Distributor is not working as you expected, we suggest you follow this sequence of steps:

1. Review the configuration, going through the implementations steps listed in 5.4.3, “Implementation” on page 120.
2. Use the NETSTAT commands described in 5.6.1, “Scenario 1: Three IP stacks distributing FTP services” on page 133 to display the actual configuration and compare it against your expectations. A very good indication in regards to whether a connection can be distributed is the NETSTAT,VDPT command. It shows you in the RDY output field which application is ready to receive connections.
3. Verify that OSPF or RIP dynamic routing protocols have been implemented and the DVIPA is being advertised through this. In addition, ensure that the downstream routers in your network have learned from your OMPROUTE (or ORouteD) where to find the DVIPA.
4. For checking the distributing function, you could use the REXX samples we have added in Appendix B, “REXX EXECs to gather connection statistics” on page 261. The REXX program connects to a server on a given host name/port pair the specified number of times. With that utility, you can easily check if the Sysplex Distributor is distributing as you expect.

If this is not enough, we strongly suggest you refer to *z/OS V1R2.0 CS: IP Configuration Reference,* SC31-8726 for further information.

6

Sysplex Distributor with MNLB

This chapter introduces a load distribution solution using Communications Server for z/OS Sysplex Distributor in cooperation with Cisco MultiNode Load Balancing (MNLB) functions in routers. The load distribution process and the data flow are different from Sysplex Distributor and standard MNLB because the tasks are performed the IBM Sysplex Distributor and Cisco routers/switches operating cooperatively, rather than in Sysplex Distributor or MNLB independently. This involves:

- Determination of the optimal application server
- Forwarding IP packets from clients to the selected application server

The following sections describe:

- TCP connection and distribution process
- The advantages of the combined IBM Sysplex Distributor/Cisco MNLB solution
- The IBM Sysplex Distributor/Cisco MNLB configuration used in our tests
- The IBM Sysplex Distributor Service Manager implementation
- Cisco Forwarding Agent definitions
- The data flow between the IBM Sysplex Distributor Service Manager and Cisco Forwarding Agent
- Displays used to control the definitions
- The IBM Sysplex Distributor backup and recovery tasks and definitions
- The Cisco Forwarding Agent definitions and backup considerations
- The Generic Routing Encapsulation (GRE) Protocol, and why it is needed

6.1 Sysplex Distributor/MNLB joint solution overview

The MNLB/Sysplex Distributor distribution provides an attractive solution when clients need high-speed access to such TCP/IP services as:

- Hypertext Transfer Protocol (HTTP) for Web services
- 3270 Telnet Server (TN3270 or TN3270E)
- File Transfer Protocol (FTP)

Multiple homogeneous application servers should be organized in a server cluster within a sysplex. The IBM z/OS Sysplex Distributor uses information provided periodically by the z/OS Workload Manager (WLM) to keep track of the current optimal server within the sysplex. This information is used later to distribute TCP connection requests to the "best" server.

The Sysplex Distributor advertises the server cluster address for specific application services to the network. Clients use this cluster address rather than directly addressing the desired target server. This cluster address is an IP address known by the DNS. From the view of the client, it looks as if the Sysplex Distributor runs the application. However, the Sysplex Distributor only advises the router where to forward the packets for a specific connection. The real server is located on another LPAR/system in the sysplex.

When the client sends the connection request to the network, it will arrive at the Cisco MNLB Forwarding Agent. The Forwarding Agent sends the connection request directly to the Sysplex Distributor. The Sysplex Distributor searches for the "best" target server and sends the packet via the Cross Coupling Facility (XCF) link to the target system.

The target server processes the connection request, which is indicated by the synch (SYN) bit in the TCP header coming from the client. The target server responds to the client also with a SYN and an acknowledgment (ACK). This establishes the full-duplex TCP connection. The response packet from the server to the client flows directly to the network without necessarily having to traverse the XCF link to the Sysplex Distributor. The route used is calculated by the routing daemon. In our test environment, we used OMPROUTE within the sysplex and Cisco's EIGRP within the router-controlled network.

The client now will respond to the SYN, previously sent from the application server, with an ACK. Since the client has no destination IP address other than the one of the Sysplex Distributor, it believes that the desired application server runs in the stack of the Sysplex Distributor. Therefore it will continue sending the ACK packet and the following packets to the Sysplex Distributor, which acts as the representative of the cluster IP address for the application server.

But from now on when the client's packets arrive at the Cisco router before entering the sysplex, the router already knows the real location (IP address) of the target server. This would enable direct routing to the target server without touching the Sysplex Distributor.

In order to provide the router or switch with knowledge about a shorter path to the target systems, the Sysplex Distributor has to transfer the connection distribution information to the router.

In a pure Cisco MNLB configuration, this information is provided by the Cisco Service Manager, which runs in a Cisco LocalDirector unit, installed externally from the IBM sysplex. The Cisco Service Manager communicates with the Cisco Forwarding Agent (implemented in the Cisco router). A special communication protocol is used to exchange routing information on accessible application servers in target systems. This protocol is called Cisco Appliance Services Architecture (CASA).

Hence, with CS for z/OS V1R2 IP, Sysplex Distributor provides the Service Manager functionality. This enables the Sysplex Distributor to propagate the connection dispatching information to the Cisco Forwarding Agent. As a result, the Cisco Forwarding Agent is able to directly send all packets of an existing connection to the real target server.

6.2 Advantages of the solution

There are some significant advantages to implementing the Sysplex Distributor with the Service Manager function in an MNLB environment. These include:

- The inbound path for all TCP data to the target server now goes directly from the Forwarding Agent to the target server.
  - The inbound data path used in a pure Sysplex Distributor environment always passed through the distributing stack and XCF link to the target server.
  - In some cases the Sysplex Distributor could become a bottleneck for high-speed access used by Web services, regarding the load of traffic on links to the Sysplex Distributor, the CPU load needed for each IP packet on the IP layer, and the load of the traffic used for the XCF links.
- In a pure MNLB configuration, a separate Cisco LocalDirector unit (and a backup LocalDirector) containing the Cisco Service Manager was used to make the connection distribution. Now, with the Service Manager being in the Sysplex Distributor, the LocalDirector is no longer needed.
- A separate Cisco OS/390 Workload Agent, running in each LPAR of all TCP/IP application server target stacks, is no longer required. Also, using the Cisco Dynamic Feedback Protocol (DFP) is unnecessary, and thus traffic load from the Workload Agent to the Forwarding Agent is avoided.
- The Service Manager in the Sysplex Distributor provides load-balancing decisions based on Quality of Service (QoS) from the Policy Agent (PAGENT) definitions in the z/OS system, in addition to the WLM information.
- A backup distributing stack can provide for failure recovery of the Service Manager.

6.3 IP addresses used during our tests

Our test configuration consists of a sysplex with four logical partitions (LPARs) named as follows:

- MVS001, which is the Sysplex Distributor (the Service Manager for the Cisco MNLB configuration).
- MVS069, which is the backup Sysplex Distributor for MVS001.
- MVS062, which can be regarded as a target stack running TCP/IP application servers. For our tests, we used mainly the TN3270E server to access TSO, a Web server, and the FTP server. TCP connection requests are distributed to this host or to the MVS154 by the Sysplex Distributor/Service Manager based on z/OS Workload Manager (WLM) information.
- MVS154, which is another target stack like MVS062.

Each LPAR runs one TCP/IP stack. The LPARs are connected to each other via the Cross Coupling Facility (XCF).

All LPARs have connections to two Cisco routers controlling the network:

- Via a shared OSA-Express Gigabit Ethernet (GbE) adapter.
- Via ESCON director to Cisco router using Common Link Access to Workstation (CLAW) or the Multipath Channel Plus (MPC+) Protocol.
  - The CLAW connection is to the Cisco router 7206VX.
  - The MPC connection is to the Cisco 7507 router running Cisco MPC+ (CMPC+).

Our network is depicted in Figure 6-1 on page 183.

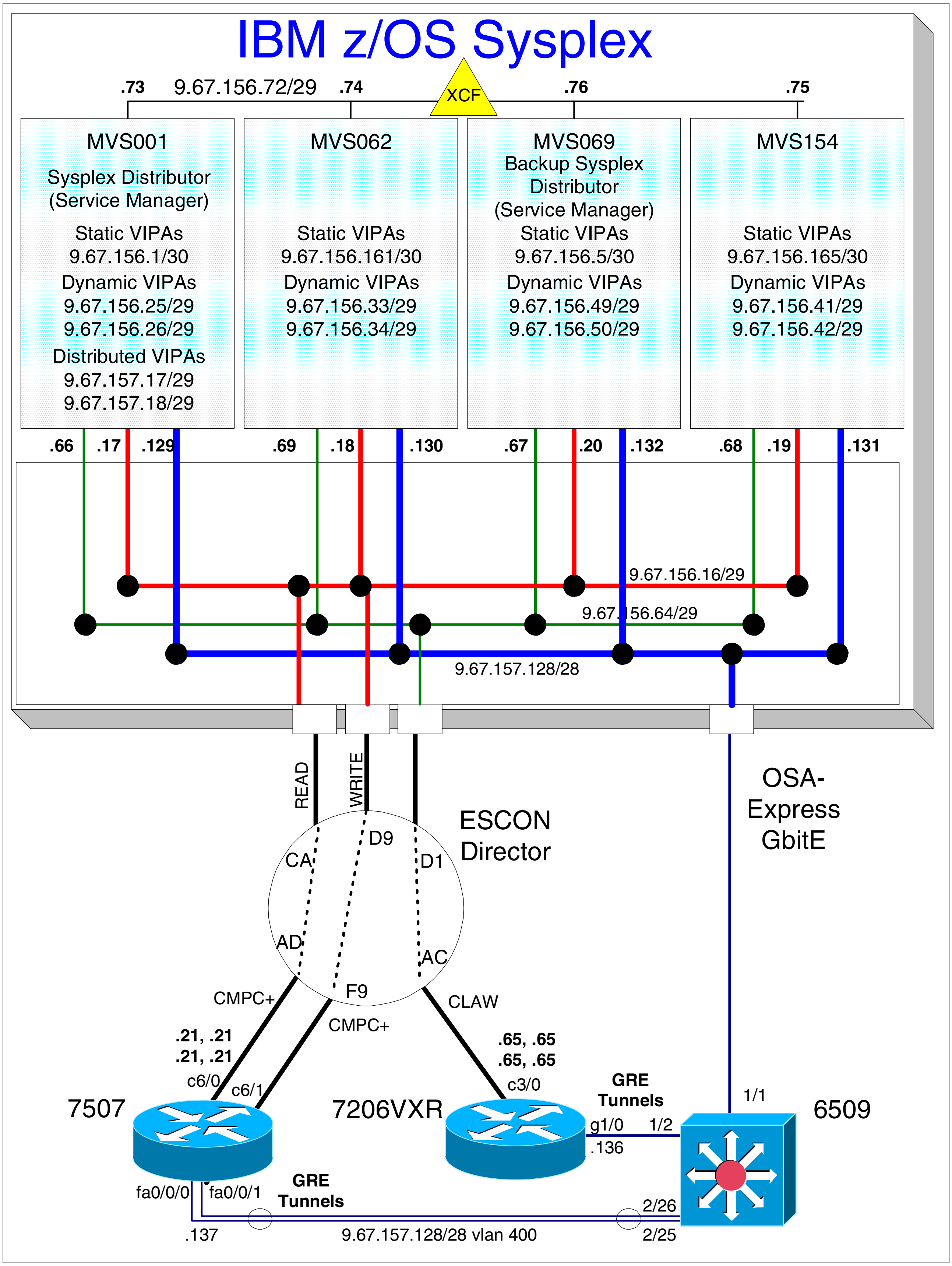


Figure 6-1 IBM z/OS Sysplex and Cisco MNLB network

XCF interface

Dynamic XCF links connect the four LPARs via the Cross Coupling Facility (XCF). The subnet used is 9.67.156.72 with mask 255.255.255.248.

The TCPIP.PROFILE XCF definition for the system MVS001 for this link is:

```
IPCONFIG DYNAMICXCF 9.67.156.73 255.255.255.248 2
```

The XCF definition for MVS062 is:

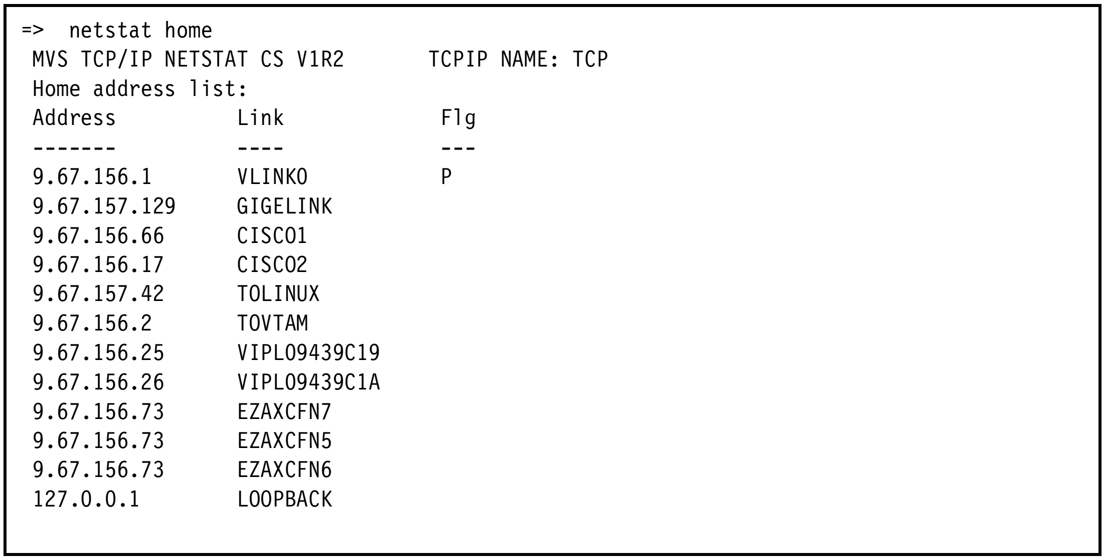
```
IPCONFIG DYNAMICXCF 9.67.156.74 255.255.255.248 2
```

The XCF definitions for the remaining systems MVS069 and MVS154 are:

```
IPCONFIG DYNAMICXCF 9.67.156.76 255.255.255.248 2

IPCONFIG DYNAMICXCF 9.67.156.75 255.255.255.248 2
```

Note: The format of the DYNAMICXCF statement shows the IP address of the XCF link, a subnet mask and a metric value for the link. This metric value will be used only for the Routing Information Protocol (RIP) in BSDROUTINGPARMS statements of the ORouteD daemon. Since we used Open Shortest Path First (OSPF) as the routing protocol for the sysplex with the OMPROUTE daemon, all definitions for the metric of each interface have been made in the OMPROUTE configuration file. The metric values defined in the DYNAMICXCF statement are therefore arbitrary and not used.

When the system MVS001 has XCF links to all other systems, the HOME list on MVS001 would look like the following screen.



```
=>  netstat home
 MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP
 Home address list:
 Address          Link             Flg
 -------          ----             ---
 9.67.156.1       VLINK0           P
 9.67.157.129     GIGELINK
 9.67.156.66      CISCO1
 9.67.156.17      CISCO2
 9.67.157.42      TOLINUX
 9.67.156.2       TOVTAM
 9.67.156.25      VIPL09439C19
 9.67.156.26      VIPL09439C1A
 9.67.156.73      EZAXCFN7
 9.67.156.73      EZAXCFN5
 9.67.156.73      EZAXCFN6
 127.0.0.1        LOOPBACK
```

Three HOME list statements with equal IP addresses but with different link names show the existing XCF links to the other systems. These HOME statements are created dynamically when the partner TCP/IP stack with its XCF interface is started. Also the DEVICE, LINK, and START statements are created automatically.

An example of the dynamic XCF DEVICE and LINK definitions is shown in the following screen.

```
DevName: NO7N             DevType: MPC       DevNum: 0000
   DevStatus: Ready
   LnkName: EZAXCFN7         LnkType: MPC         LnkStatus: Ready
     NetNum: 0   QueSize: 0
     BytesIn: 41536               BytesOut: 40822
   BSD Routing Parameters:
     MTU Size: 04472           Metric: 200
     DestAddr: 0.0.0.0         SubnetMask: 255.255.255.248
   Multicast Specific:
     Multicast Capability: Yes
     Group           RefCnt
     -----           ------
     224.0.0.5       0000000001
     224.0.0.1       0000000001
```

The screen shows the definitions to system N7. The value N7 is taken from the &SYSCLONE parameter of system MVS001. The DEVICE name is taken from the VTAM CP node name. The following shows our system symbol values.

```
290  IEA007I STATIC SYSTEM SYMBOL VALUES 016
090          &SYSALVL.  = "1"
090          &SYSCLONE. = "N7"
090          &SYSNAME.  = "MVS069"
090          &SYSPLEX.  = "LOCAL"
090          &SYSR1.    = "OS120"
090          &CONSOLID. = "MVS069"
.....
090          &DOMAIN.   = "NO7NV"
……… VTAM-Start…
090  IST093I NO7N ACTIVE
```

Static VIPA IP address

Usually, a static VIPA address is used for a device-independent destination endpoint that remains on a single stack. Whenever an interface of a device, such as the Ethernet or token-ring adapter, fails, a routing protocol would find an alternate path to the static VIPA address.

In our configuration, we used the static VIPA address (for example: 9.67.156.1 with subnet mask 255.255.255.252) as the endpoint for a *tunnel* between the Cisco routers 7507 and 7206VXR and the OSA-Express adapter. Tunnels are required only when the OSA-Express adapter is shared by multiple stacks. In our case, there is one OSA-Express adapter that is connected to all four LPARs - MVS001, MVS062, MVS069, and MVS154. The OSA-Express adapter is defined in share mode.

The tunnel carries Generic Routing Encapsulation (GRE) packets over the Gigabit Ethernet (GbE) to the OSA-Express adapter. More detailed information is found in 6.10, “Generic Routing Encapsulation (GRE) protocol” on page 231.

Though we originally defined our static VIPA in system MVS001 for Enterprise Extender, we re-used it for the tunnel, too. Its definition is given in Example 6-1 on page 186.

Example 6-1 Static VIPA definitions

```
DEVICE VIPA03   VIRTUAL 0
LINK   VLINK0   VIRTUAL 0   VIPA03
HOME  9.67.156.1    VLINK0
```

OSA-Express adapter interfaces

In our network, one OSA-Express GbE adapter is shared by our four LPARs - MVS001, MVS069, MVS062, and MVS154. The IP addresses of all stacks belong to the netid 9.67.157.128 with subnet mask 255.255.255.240. The TCPIP.PROFILE definitions for MVS001 are given in Example 6-2.

Example 6-2 OSA-Express adapter definitions for primary router

```
DEVICE  GIGE2F00  MPCIPA  PRIR
LINK  GIGELINK  IPAQGNET  GIGE2F00
START GIGE2F00
```

For MVS001, the OSA adapter interface is defined as the primary router (see parameter PRIR in the DEVICE statement). This indicates to the OSA adapter that it should forward packets with IP addresses not in the OSA Adapter Table (OAT) to this TCP/IP stack. The OAT learns all IP addresses when the devices are started. These are IP addresses defined in the HOME list either statically (such as the Ethernet or token-ring interfaces) or automatically (such as the dynamic XCF address).

The primary router definition “PRIR” for the OSA adapter is enabled for the TCP/IP stack of Sysplex Distributor. The definition of the OSA adapter interface for the TCP/IP stack of the backup Sysplex Distributor is therefore defined as the secondary router. This means that, in case of a failure of the TCP/IP stack on MVS001, the OSA adapter interface for the backup Sysplex Distributor takes over the function of receiving all IP packets with known and unknown IP addresses in the OAT.

The OSA-Express GbE definitions for MVS069 are shown in Example 6-3.

Example 6-3 OSA-Express adapter definitions for the secondary router

```
DEVICE  GIGE2F00  MPCIPA  SECR
LINK  GIGELINK  IPAQGNET  GIGE2F00
START GIGE2F00
```

The interfaces of the remaining LPARs are defined as non-router “NONR”. This is because these are target systems and do not route or forward IP packets. These systems are the endpoints of the TCP connections. They are the real application server systems only. The OSA device definitions are exactly the same as illustrated in Example 6-4.

Example 6-4 OSA-Express Adapter definitions for the target systems

```
DEVICE  GIGE2F00  MPCIPA  NONR
LINK  GIGELINK   IPAQGNET  GIGE2F00
START  GIGE2F00
```

The VTAM Transmission Resource Element (TRLE) defines the channel path for the OSA-Express adapter. See the following definitions.

Example 6-5 VTAM TRLE entries for OSA-Express adapter

```
N04GIG1  TRLE  LNCTL=MPC,
                READ=(2F14),
                WRITE=(2F15),
                DATAPATH=(2F16,2F17),
                PORTNAME=(GIGE2F00,0)
```

Common Link Access to Workstation (CLAW) Protocol

The CLAW interface provides channel attachment to a Cisco router or RS6000 through an ESCON director. In our network, all of the LPARs have a channel path to the Cisco router. If the OSA-Express fails, this might be an alternate path between the sysplex and the IP network.

The TCPIP.PROFILE definitions shown in Example 6-6 are used.

Example 6-6 TCPIP.PROFILE definitions for CLAW device

```
DEVICE  CIP1A  CLAW  D30  MVS001B  C7507A  PACKED  15 15  32768  32768
LINK  CISCO1 IP  0  CIP1A
START CIP1A
```

Subchannel addresses D30 and D31 are used for the receive and the send path from the MVS001 to the Cisco 7206 router. The host name is MVS001B. The workstation name is C7202A.

The channel program uses *packed* mode, which means that more than 4 KB buffer sizes for the channel-read or channel-write commands may be used. In packed mode, the size may be 32 KB or 60 KB. In our case we used 32768 bytes for the read and write buffer sizes, using 15 read buffers and 15 write buffers.

Multipath Channel (MPC) Protocol

In addition to CLAW, there is also an MPC channel attachment for all LPARs to a Cisco router. This attachment is to our Cisco router 7507.

The TCPIP.PROFILE definitions in Example 6-7 illustrate the attachment.

Example 6-7 TCPIP.PROFILE definitions for MPC device

```
DEVICE   N04CMPC   MPCPTP
LINK  CISCO2  MPCPTP   N04CMPC
START   N04CMPC
```

The device name N04CMPC has to match the VTAM TRLE name. The VTAM statement carries the information about the channel addresses, as shown in Example 6-8.

Example 6-8 VTAM TRLE definition for MPC device

```
N04CMPC  TRLE  LNCTL=MPC,MAXBFRU=255,REPLYTO=25.5,MAXREADS=8,
                         STORAGE=DS,MPCLEVEL=HPDT,
                         READ=(D20),WRITE=(D21)
```

Dynamic VIPA addresses

There are some Dynamic VIPA (DVIPA) addresses defined in our test network that we did not use for the tests illustrated in this chapter. These are shown in Example 6-9.

Example 6-9 dynamic VIPA definitions

```
VIPADYNAMIC
  VIPADEFINE 255.255.255.248 9.67.156.25 9.67.156.26
    VIPABACKUP  1  9.67.156.33 9.67.156.34
    VIPABACKUP  1  9.67.156.41 9.67.156.42
    VIPABACKUP 100 9.67.156.49 9.67.156.50
```

```
ENDVIPADYNAMIC
```

Distributed Dynamic VIPAs

There are also additional definitions for the distributed DVIPAs, which belong to the same VIAPDYNAMIC/ENDVIPADYNAMIC block. They are defined in the TCPIP.PROFILE of the Sysplex Distributor only. The backup Sysplex Distributor will take over these distributed DVIPAs if the primary Sysplex Distributor fails. You may view the definitions for the backup Sysplex Distributor in 6.9, “Sysplex Distributor backup” on page 220.

The TCPIP.PROFILE definitions for the target stacks with its real application servers may be viewed in 6.6.1, “Basic TCPIP.PROFILE definitions” on page 198.

The Sysplex Distributor uses the DVIPAs shown in Example 6-10 for the distribution to the target machines MVS062 and MVS154.

Example 6-10 Dynamic distributed VIPA definitions with no Service Manager

```
VIPADEFINE  MOVEABLE IMMED 255.255.255.248
            9.67.157.17
   VIPADIST 9.67.157.17 PORT 80 443 23 523
            DESTIP   9.67.156.74
                     9.67.156.75

VIPADEFINE  MOVEABLE IMMED 255.255.255.248
            9.67.157.18
   VIPADIST 9.67.157.18 PORT 20 21
            DESTIP   9.67.156.73
                     9.67.156.74
```

Three statements (or options as is the case with DESTIP) are used to define the distributed DVIPAs:

1. VIPADEFINE is used to define the VIPA. It also implies that this IP address may be taken over by another TCP/IP stack. This might occur when a backup Sysplex Distributor takes over the DVIPA because the primary Sysplex Distributor fails. In this case a VIPABACKUP statement has to be defined in the backup Sysplex Distributor’s TCPIP.PROFILE.

   See an example in 6.9, “Sysplex Distributor backup” on page 220.

2. VIPADIST is used to specify which DVIPA is going to be distributed to TCP/IP target stacks. This statement does not define to which stack it is distributed. It defines which server ports are associated with this DVIPA.

   In our example, the distributed DVIPA 9.67.157.17 will be the IP address for Web services on port 80 and 443. Port 80 is used for the HTTP protocol. Port 443 is the secure port used for the HTTPS protocol.

   On the same IP address, TN3270E connections will be opened on port 23, and secure TN3270 connections will be on port 523.

   The distributed DVIPA 9.67.157.18 is defined for File Transfer Protocol (FTP) connections on port 20 for the data transmission and on port 21 for the control connection.

3. DESTIP is used to specify to which TCP/IP stacks in other LPARs/systems within the sysplex the DVIPA may be distributed. The IP addresses of the TCP/IP stacks are the dynamic XCF addresses.

   Whenever the TCP/IP stacks in the target systems are active, the XCF link to the Sysplex Distributor’s XCF IP address will be used to exchange information between the stacks telling them to implement and activate a distributed DVIPA on the target stack.

In our example TCP connection requests will be distributed for Web services and the TN3270 services to TCP/IP stacks with XCF addresses 9.67.156.74 (MVS062) and 9.67.156.75 (MVS154).

The FTP services are distributed to 9.67.156.73 (MVS001) and to 9.67.156.74 (MVS062).

The following screen shows the dynamically created HOME statements on MVS062.

```
=> netstat home
MVS TCP/IP NETSTAT CS V1R2   TCPIP NAME: TCP
 Home address list:
 Address         Link             Flg
------------     -------------    ---
9.67.156.161     VLINK0           P
9.67.157.130     GIGELINK
9.67.156.69      CISCO1
9.67.156.18      CISCO2
9.67.157.243     CISCO3
9.67.156.162     TOVTAM
9.67.156.74      EZAXCFN6
9.67.156.33      VIPL09439C21
9.67.156.34      VIPL09439C22
9.67.156.74      EZAXCFN7
9.67.156.74      EZAXCFN4
9.67.157.17      VIPL09439D11     I
9.67.157.18      VIPL09439D12     I
127.0.0.1        LOOPBACK
***
```

The screen shows the two inserted (flag `I`) distributed DVIPAs, 9.67.157.17 and 9.67.157.18, along with special link names that are constructed from the hexadecimal values of each distributed DVIPA.

6.4 Data flow: Service Manager and Forwarding Agent

The data flow between the Service Manager in the IBM Sysplex Distributor and the Forwarding Agent in the Cisco router consists of two main messages. These are:

- Fixed affinity
- Wildcard affinity

Affinity

The term affinity describes the action of associating or coupling one thing with another.

In this context, information about the IP and TCP header is collected to build a unique identifier to distinguish and associate IP packets to a certain TCP connection. This identifier is used by Forwarding Agents to differentiate the IP packets from incoming TCP connection requests and from packets belonging to existing TCP connections.

Fixed affinity

A fixed affinity is one that matches specific information within a TCP connection identifier. This identifier is defined by its unique 5-tuple that spans the packet headers. These are:

- For the IP header:
  - Protocol type (TCP only = value 6)

  - Source IP address
  - Destination IP address
- For the TCP header:
  - Source port
  - Destination port

This composed information is used by the Forwarding Agent to differentiate incoming IP packets, and to associate the packets to a certain action. This may be, for example, forwarding the packet to a predetermined target system selected for this specific TCP connection.

In addition, a fixed affinity contains two other important pieces of information:

- The forward IP address, which specifies the real target server address for the specific TCP connection. The Sysplex Distributor provides the dynamic XCF link address of the selected target TCP/IP stack.
- A time-to-live (TTL) value telling what time the fixed affinity entry should be kept in the Forwarding Agent's cache so that the affinity expires and the resources it consumes can be freed. This value is provided by the Sysplex Distributor. The value is 15 minutes.

Wildcard affinity

A wildcard affinity is a 5-tuple piece of information as well. Compared with fixed affinity, it has different values in the source IP address and the source port.

- The source IP address contains a value of 0.0.0.0
- The source port contains a value of 0000

This allows the Forwarding Agent to accept all incoming packets from all IP addresses and ports for further processing.

Wildcard affinity is mainly used to accept TCP connection requests with a SYN-bit set on in the TCP header. This request will be forwarded by the Forwarding Agent to the Service Manager, which will distribute it to one of the target systems running the application server.

Wildcard affinity is created by and sent from the Service Manager to the Forwarding Agent. It also contains a forwarding IP address, which is the dynamic XCF link IP address of the Sysplex Distributor.

6.4.1 Wildcard affinity and processing

Wildcard affinities are used by the Forwarding Agent to know which IP packets should be sent to the Service Manager. Therefore it has to know which IP address and port are used as the cluster for a specific application service.

When a Forwarding Agent receives an IP packet, it looks at the IP header searching for the cluster address that is the destination IP address and the protocol value ‘6’, indicating the TCP protocol. It also checks the destination port in the TCP header for the requested application (for example 23, the Telnet port). The Forwarding Agent also checks the source IP address and source port. In a wildcard definition, these two values are always set to zero. This means the Forwarding Agent accepts TCP packets from any source. Only the destination entries and the protocol are important.

The wildcard affinity information is provided by the Service Manager in multicast packets at the initialization of the Sysplex Distributor stack and later periodically.

The following screen, obtained from one of two Forwarding Agents, shows the wildcard information sent from the Sysplex Distributor that we used in our tests.

```
NIVT7507#show ip casa wildcard
Source Address  Source Mask     Port  Dest Address    Dest Mask       Port  Prot
0.0.0.0         0.0.0.0         0     9.67.157.17     255.255.255.255 523   TCP
0.0.0.0         0.0.0.0         0     9.67.157.17     255.255.255.255 443   TCP
0.0.0.0         0.0.0.0         0     9.67.157.17     255.255.255.255 80    TCP
0.0.0.0         0.0.0.0         0     9.67.157.17     255.255.255.255 23    TCP
0.0.0.0         0.0.0.0         0     9.67.157.18     255.255.255.255 21    TCP
0.0.0.0         0.0.0.0         0     9.67.157.18     255.255.255.255 20    TCP
NIVT7507#
```

The screen sample shows the information used by the Forwarding Agents:

- They accept IP packets from any client to two destination IP addresses (9,67,157.17 and 9.67.157.18). These IP addresses were defined previously in the TCPIP.PROFILE of MVS001 under VIPADYNAMIC, VIPADEFINE, VIPADIST, DESTIP, and ENDVIPADYNAMIC. See “Distributed Dynamic VIPAs” on page 188.
- They accept packets with TCP connection requests where the SYN bit in the TCP header is set on for:
  - Port 23 for normal TN3270 connections
  - Port 523 for secure TN3270 connections
  - Port 80 for HTTP to Web services
  - Port 443 for HTTPS secure access to Web services
  - Port 21 for FTP server control connections
  - Port 20 for FTP server data connections

When the Forwarding Agent receives a TCP connection request from a client, it checks the destination IP address, protocol and port number. If the destination IP address, port number and TCP protocol match the wildcard affinity entry and the SYN bit is on, the request will be encapsulated in a CASA packet and sent to the Service Manager. This also occurs for data packets for which a wildcard affinity exists, but no fixed affinity does.

Further information may be obtained from “CASA information in the Forwarding Agent” on page 210.

6.4.2 Service Manager processes TCP connection request

The incoming CASA request for the TCP connection is processed by the Sysplex Distributor as follows:

- Unpacks the CASA packet.
- Checks the IP destination address, which is the cluster address of an application service.
- Selects the “best” target server, from a group defined under VIPADIST, PORT, and DESTIP, based on load-balancing information and rules.
- Forwards the initial IP packet to the target server via the XCF link.
- Sends a unicast packet, called fixed affinity, to the Forwarding Agent that sent the CASA packet with the SYN request. This unicast message contains the information about the forwarding IP address of the real target server for this particular TCP connection. The forwarding IP address is the XCF link IP address of the target stack.

- The XCF link address will be used by the Forwarding Agent to build a fixed affinity for the specific TCP connection. See “Fixed affinity” on page 189.

6.4.3 Continuation of the TCP connection establishment process

The connection establishment process continues between the application server and the client. The application server sends a SYN, ACK to the client. The client also returns an ACK to the application server. Since the Forwarding Agent sees all incoming packets, it is informed about the connection establishment process and updates the state of the connection.

6.4.4 Fixed affinity processing

When the TCP connection is established, the Forwarding Agent knows all information regarding the specific TCP connection. All TCP connections are kept in the Forwarding Agent’s table for fixed affinities.

A sample of a fixed affinity table is shown in the following output taken from a Cisco router’s Forwarding Agent.

```
NIVT7507#show ip casa affinities
Source Address  Port  Dest Address    Port  Prot
9.67.156.104    4970  9.67.157.17     23    TCP
9.67.156.104    4971  9.67.157.18     21    TCP
NIVT7507#
```

A table entry for a fixed affinity is valid for one connection only. It is defined by its unique 5-tuple information taken from the IP header and TCP header. Only IP packets with matching 5-tuples will be recognized by the Forwarding Agent as packets belonging to an existing TCP connection. These packets will be forwarded directly to the real application server without traversing the path via the Sysplex Distributor and the XCF link. Therefore, the fixed affinity entry also must contain a forward IP address for the real application server. This IP address will be provided by the Sysplex Distributor via a unicast fixed affinity-update message. The forwarding IP address is the dynamic XCF link address for the target stack.

In the screen above, the forwarding IP address is not shown. It is detailed information that is obtained using the command **`show ip casa aff det`**.

For more information, see “CASA information in the Forwarding Agent” on page 210.

6.4.5 Prerequisites for the CASA protocol exchange

The communication between the Service Manager and the Forwarding Agents is done using the Cisco Appliance Services Architecture (CASA) protocol. In order to send CASA multicast packets, the following definitions are required:

- In the Sysplex Distributor:
  - Define the Sysplex Distributor as the Service Manager.
  - Determine the multicast IP address, which is a class D address, such as 224.0.1.2. This address was used in our tests, as recommended in Cisco’s documentation. This is the destination IP address to reach all Forwarding Agents.
  - Determine the CASA port. This is the destination port to reach the CASA protocol in all Forwarding Agents. We used the Cisco default recommendation port number 1637.

- In the Forwarding Agents:
  - Define the multicast IP address used by the Service Manager to address the Forwarding Agent. In our configuration we used 224.0.1.2 as the multicast address.
  - Define the listening port (1637 in our configuration) in the Forwarding Agent.

You will find the required definitions for the Sysplex Distributor in Example 6-11 on page 197 and for the Forwarding Agent in 6.7, “Forwarding Agent definitions” on page 199.

6.4.6 Message flow of wildcard and fixed affinities, SYN, ACK, data

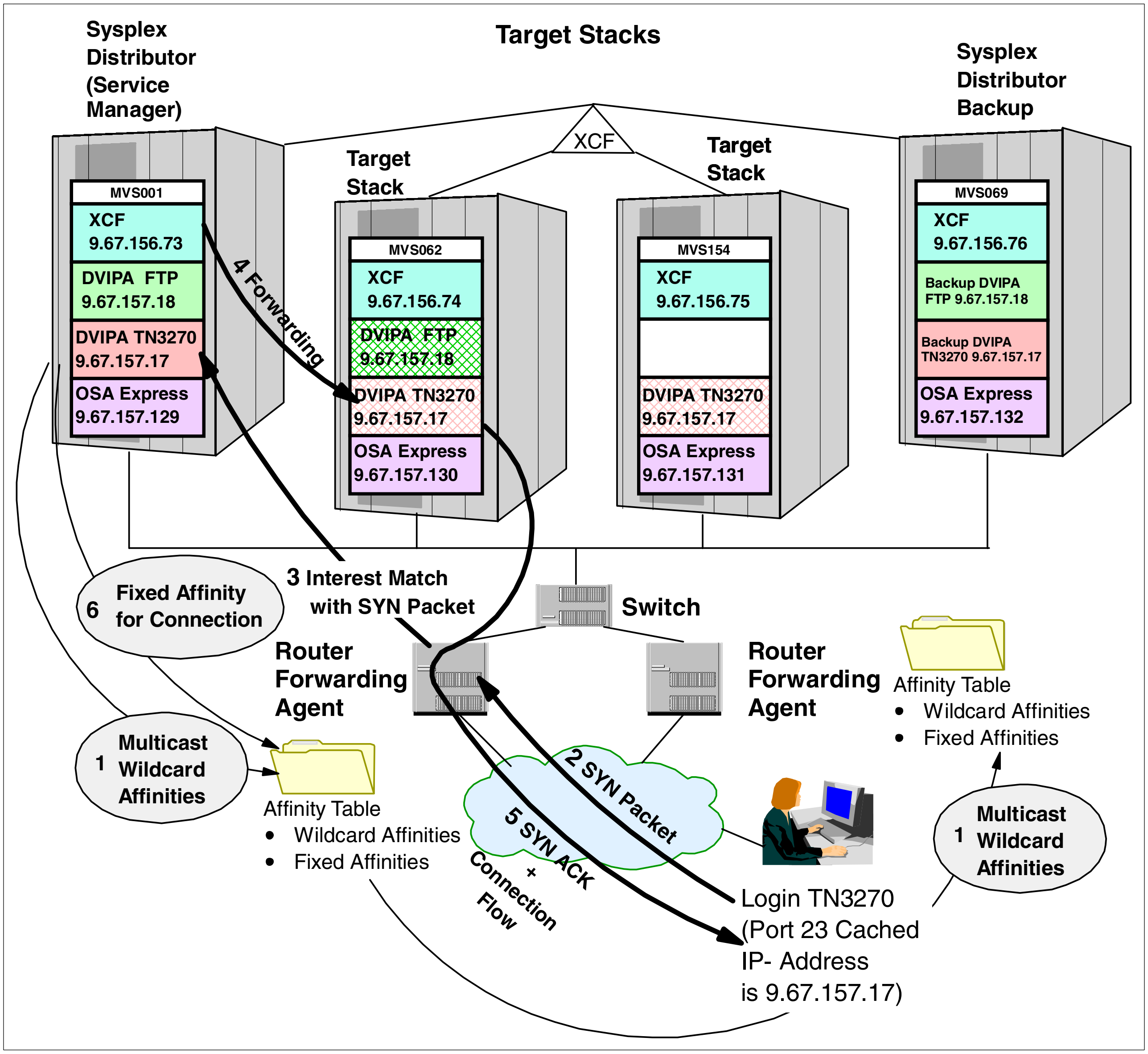


Figure 6-2 Message flow of wildcard and fixed affinities, SYN, ACK, and data

The following is an overview of the message flow needed to prepare the Forwarding Agent to accept client TCP connection requests and IP packets, and indicate to the Forwarding Agent where to send the received IP packets, as shown in Figure 6-2 on page 193:

1. The Sysplex Distributor configured as Service Manager multicasts a wildcard affinity update specifying that all new connection requests and existing connection packets, for which there is no fixed affinity entry available, should be sent to the Sysplex Distributor stack.
2. A connection request, represented as a SYN packet, is received by the Forwarding Agent.
3. The Forwarding Agent encapsulates the SYN packet in an Interest Match CASA packet and forwards it to the Service Manager in the Sysplex Distributor stack.
4. The Sysplex Distributor unpacks the SYN packet, makes the routing decision, and forwards the SYN packet to the selected application server. The IP address for the real target, running the application server, is the dynamic XCF link IP address.
5. The target server returns a SYN, ACK directly to the client without touching the Sysplex Distributor stack.
6. The Service Manager in the Sysplex Distributor sends a fixed affinity for this particular connection back to the Forwarding Agent that forwarded the SYN packet. The fixed affinity instructs the Forwarding Agent to send data for this connection via the most efficient route to the application server, identified by the dynamic XCF IP address of the target stack.
7. The subsequent data flow carrying the ACK from the client and data exchanged during the connection are not shown here. These packets travel the direct path from the Forwarding Agent to the target system known by the dynamic XCF link IP address, and no longer touch the Sysplex Distributor.

6.4.7 Message flow for connection data with no fixed affinity

This situation might occur when a Forwarding Agent other than the one that received the prior SYN request receives data. This might happen if the Forwarding Agent that has the fixed affinity for a particular connection fails. It also might happen when the path from the client to the application server target stack has changed and uses another Forwarding Agent. This Forwarding Agent, however, does not possess a fixed affinity yet.

This second Forwarding Agent only has a wildcard affinity with a forwarding address to the Sysplex Distributor. It is, of course, the dynamic XCF link IP address. If the Forwarding Agent would have no wildcard affinity, it would not be able to communicate with the Service Manager in the Sysplex Distributor's stack.

The following flow description indicates how this case will be managed:

1. The Sysplex Distributor configured as Service Manager multicasts a wildcard affinity update specifying that all new connection requests and existing connection packets, for which there is no fixed affinity entry available, should be sent to the Sysplex Distributor stack.
2. An IP packet for an existing TCP connection now arrives at a Forwarding Agent, but the Forwarding Agent does not have a fixed affinity entry for this particular TCP connection. This Forwarding Agent also did not receive the SYN request for this TCP connection. It only has wildcard affinities.
3. If a matching fixed affinity is not found, the Forwarding Agent compares the packet against the wildcard affinity.
   - If there is an entry available for the destination IP address and the destination port in the wildcard affinity table, then the Service Manager is known for the particular TCP connection. The destination IP address should be the cluster address the Sysplex

Distributor has multicast prior to all Forwarding Agents. The destination port should be an application service associated to the cluster address multicast by the Sysplex Distributor. A matching wildcard affinity causes the Forwarding Agent to create a CASA packet.

- If there is no matching wildcard affinity entry (which might not occur because the Service Manager multicasts wildcard affinities every 30 seconds to all Forwarding Agents), the IP packet will be returned to the router. The router will send the packet to the IP address named in the IP header. Since it is the cluster address, which is the distributed DVIPA address, the packet will be received due to the routing mechanism by the Sysplex Distributor stack, or if this stack is down, by the backup Sysplex Distributor.

4. When a matching wildcard affinity is found, the Forwarding Agent encapsulates the IP packet as an IP-only CASA message type, and sends it to the known forwarding IP address, which is the dynamic XCF link address of the Service Manager in the Sysplex Distributor stack.
5. The Sysplex Distributor unpacks this IP-only CASA packet and checks the DVIPA connection routing table. If the 5-tuple information matches an existing TCP connection, it forwards the IP packet to the application server running the current connection. The forwarding IP address for the target running the application server is taken from the DVIPA connection routing table. It is the dynamic XCF link IP address of the target stack.
6. The Service Manager in the Sysplex Distributor returns a fixed affinity update to the Forwarding Agent that previously sent the CASA IP-only packet.
7. The Forwarding Agent updates its fixed affinity table. For subsequent packets, it now has the information to forward IP packets directly to the correct target server without the assistance of the Service Manager.

6.4.8 Message flow for closing a TCP connection

The client closes a TCP connection by sending a packet with a FIN-bit set on in the TCP header. The connection closing procedure is similar compared to the connection establishment viewing the TCP flow only.

The client's FIN is sent to the application server. The server is notified that the connection will be closed. The server acknowledges the FIN with an ACK, and sends a FIN also to finish the other half of the full-duplex TCP connection.

The data path, however, from the client to the application server differs completely from the connection establishment path, because the Sysplex Distributor doesn't see the packets sent from the client to the server and vice versa.

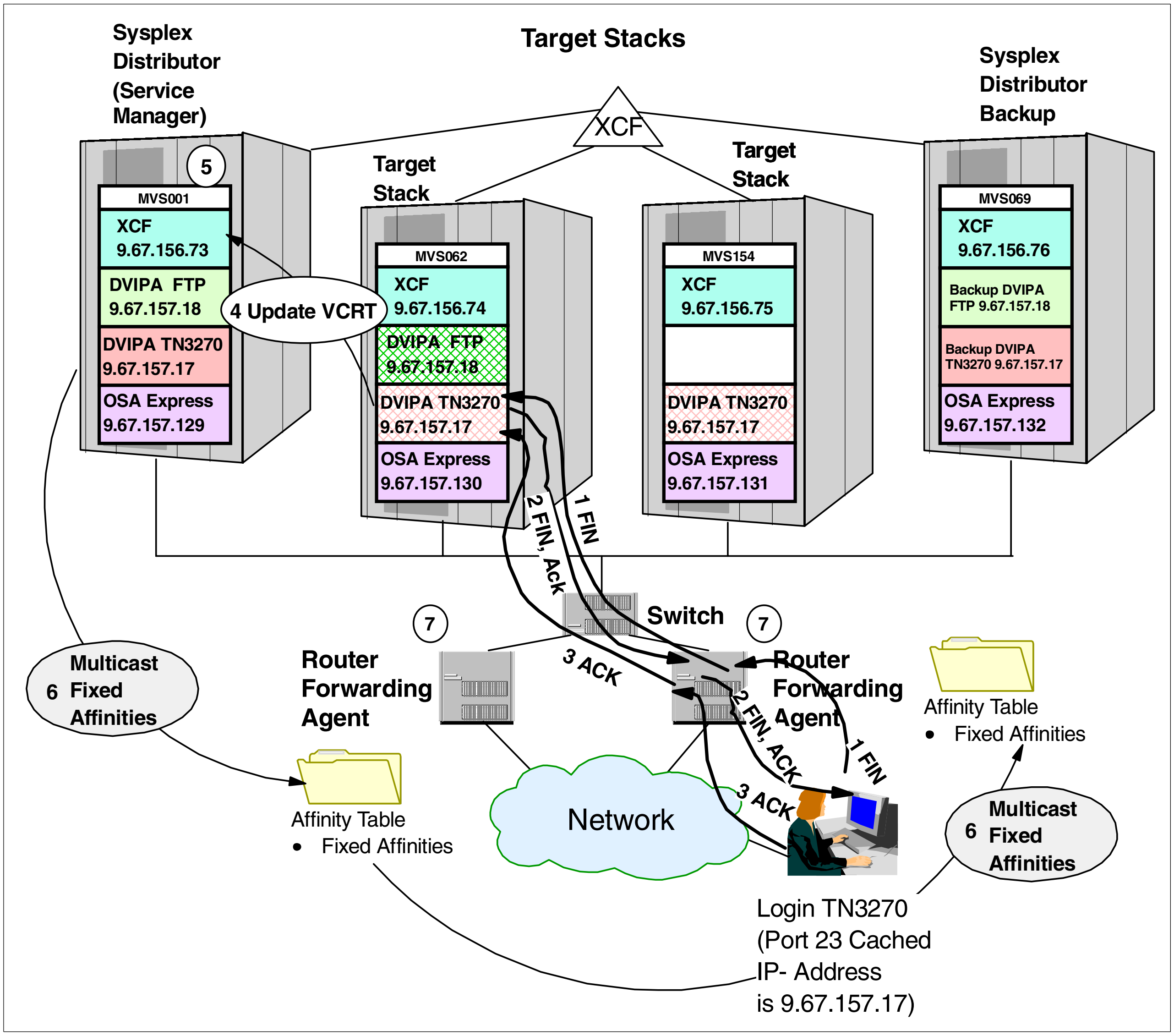


Figure 6-3 Message flow for shutting down a TCP connection

The following is the message flow needed to close a TCP connection with activities by the client, the Forwarding Agent, the application server stack, and the Sysplex Distributor:

1. The client starts closing his TCP connection, for example by typing `quit` within the application program. This creates an IP packet with a FIN-bit set on in the TCP header. The FIN packet is sent towards the application server via the network. The Forwarding Agent receives this packet and forwards it depending on the fixed affinity entry for this particular TCP connection to the target server based on the forwarding IP address. This forwarding IP address is the dynamic XCF link address of the target system. The path from the Forwarding Agent to the target system goes directly upstream without touching the Sysplex Distributor.
2. The TCP/IP stack of the target system receives the FIN request. It indicates that the client wants to close the existing TCP connection. The target stack acknowledges the received FIN with an ACK, and also sends a FIN to the client to close the second part of the full-duplex connection.
3. The client responds to the second FIN with an ACK also using again the direct path to the application server's stack.

4. The application server's TCP/IP stack informs the Sysplex Distributor about the closed TCP connection via the dynamic XCF link.
5. The Sysplex Distributor updates its cache of the dynamic VIPA connection routing table (VCRT) by deleting the entry for this connection. It recognizes the TCP connection by comparing the 5-tuple information. Remember, the 5-tuple consists of the information about the following information:
   - Protocol used, which always is TCP (protocol number 6 in the IP header)
   - Source IP address
   - Destination IP address
   - Source port
   - Destination port
6. The Sysplex Distributor sends a multicast CASA message to all Forwarding Agents using the IP address 224.0.1.2 and port 1637 as the destination, and its own dynamic XCF link IP address and port 1637 as source. This message is an Affinity-Delete type message with a time-to-live (TTL) value of 0 (zero). The TTL 0 causes all Forwarding Agents to delete existing entries for the particular TCP connection.
7. When the Forwarding Agents receive the multicast packet, they delete the fixed affinity for this TCP connection from their caches.

The process to close a TCP connection is now finished. The Sysplex Distributor will no longer show the previous TCP connection in its VIPA connection routing table.

The **`show IP CASA affinities`** command, issued at the Forwarding Agent, may display the previously existing TCP connection for a couple of seconds but the entry will soon disappear.

6.5 Service Manager implementation

As described in 6.4.5, "Prerequisites for the CASA protocol exchange" on page 192, the Service Manager has to be defined:

- For the desired distributed DVIPAs
- Enabling the multicast support with:
  - Multicast address
  - Service Manager port

These definitions have to be done in the TCPIP.PROFILE.

6.5.1 Service Manager new TCPIP.PROFILE definitions

In order to switch on Service Manager functions in the TCP/IP stack, in addition to Sysplex Distributor functions, several statements had to be defined within the VIPADYNAMIC and ENDVIPADYNAMIC scope.

The VIPADEFINE statement received a new parameter, called SERVICEMGR. This allows the Service Manager to propagate the associated IP cluster address to MNLB Forwarding Agents for further distribution by Cisco routers. Packets for a TCP connection with this specific destination cluster IP address for an application are now distributed by an MNLB Forwarding Agent according to the distribution information of the Sysplex Distributor's Service Manager.

Example 6-11 VIPADEFINE ... SERVICEMGR

```
VIPADEFINE  MOVEABLE IMMED SERVICEMGR 255.255.255.248
            9.67.157.17
   VIPADIST   9.67.157.17 PORT 80 443 23 523
```

```
    DESTIP    9.67.156.74
              9.67.156.75
```

In our sample the Service Manager is switched on for the DVIPA 9.67.157.17. A TCP connection with this cluster IP address will be distributed by the Forwarding Agent to the real target server 9.67.156.74 or 9.67.156.75, depending on the decision the Service Manager made when processing the SYN request.

The parameter SERVICEMGR is needed for each VIPADEFINE statement if the DVIPA address has to be distributed using VIPADIST and DESTIP. The parameter SERVICEMGR is valid only if VIPADIST and DESTIP statements follow. If no VIPADIST and DESTIP is defined for DVIPA, the SERVICEMGR does not work.

An additional statement, the VIPASMPARMS statement, has to be added. The statement (shown in Example 6-12) tells the Service Manager what multicast IP address and port has to be used to send out CASA messages.

Example 6-12 VIPASMPARMS

```
VIPASMPARMS SMMCAST 224.0.1.2 SMPORT 1637
```

At the time of multicasting wildcard affinities, the Service Manager uses the VIPASMPARMS parameter SMMCAST and SMPORT to address the Forwarding Agents.

A special authentication password may be defined as the parameter SMSPASSword. This restricts communication between the Service Manager and Forwarding Agent with matching values based on MD5 (message digest 5 protocol) only.

The sample used is based on Cisco router recommended values. The same multicast IP address and the port have to be defined in the Forwarding Agents.

6.6 TCP/IP stack of the target systems

The TCPIP.PROFILE definitions for applications in the target stacks are the same as in a pure Sysplex Distributor configuration.

6.6.1 Basic TCPIP.PROFILE definitions

Basically, the following IPCONFIG parameters have to be defined for the Sysplex Distributor besides the DYNAMICXCF parameter already mentioned:

DATAGRAMFWD Enables rerouting IP packets to another TCP/IP stack.

IGNOREREDIRECT Enabled automatically when OMPROUTE is used.

VARSUBNETTING To use variable subnet masks.

MULTIPATH Enables multipath selection for outbound traffic.

PATHMTUDISCOVERY Discovers dynamically the minimum transfer unit of each hop to the destination.

SYSPLEXROUTING Enables the TCP/IP stack to communicate with the WLM.

Other IPCONFIG parameters have to be considered, depending on the existing configuration.

6.7 Forwarding Agent definitions

The Forwarding Agent will accept multicast messages sent by the Service Manager with destination IP address 224.0.1.2, and listen to messages on port 1637, which is reserved for CASA UDP messages. These two definitions have to match the Service Manager. See 6.5.1, “Service Manager new TCPIP.PROFILE definitions” on page 197.

Our network used two Forwarding Agents. The Forwarding Agent is implemented in Cisco router 7507 and 7206VXR.

6.7.1 CASA definitions for Cisco 7507

There are three important lines only:

```
ip multicast-routing
ip casa 1.1.1.1 224.0.1.2
   forwarding-agent 1637
```

These lines indicate:

- Multicast routing is enabled.
- CASA is enabled using a device-independent IP address. In our implementation, we used 1.1.1.1 as the unicast IP address. The IP address for receiving multicast packets is 224.0.1.2.
- The Forwarding Agent listens on port 1637 for destined packets and uses the same port number as the source port. Basically, CASA uses UDP packets only for port 1637.

6.7.2 CASA definitions for Cisco router 7206VXR

There are similar definitions for the Cisco router 7206VXR.

```
ip multicast-routing
ip casa 1.1.1.2 224.0.1.2
 forwarding-agent 1637
```

In order to address unicast packets, this router needs another device-independent IP address. In our implementation for this router we used 1.1.1.2. Again it receives multicast packets with the IP address 224.0.1.2 and the listening port is 1637, which is also a source port.

6.8 Operations: control and displays

In order to control whether the Service Manager definitions are defined correctly and applied as desired, there are a variety of displays, which are explained in the following sections.

During our tests, we checked the requested information using the following displays and trace extracts. The displays are grouped as:

- CASA information in the Sysplex Distributor
- CASA information in the Cisco router
- Integrated CASA information for the Sysplex Distributor and Cisco router using a sample of closing a TCP connection

6.8.1 CASA information in the Sysplex Distributor

Multicast address and packet distribution flow

The defined multicast IP address and the port address may be checked by displaying the DVIPA configuration using the following commands:

```
netstat vipadyn

netstat vipadcfg
```

The display of the distributed VIPA port table shows which cluster addresses with associated port addresses the Sysplex Distributor will distribute to available target hosts. This display will be obtained using the command **`netstat vdpt`**.

The VIPA connection routing tables provides the information about currently distributed TCP connections. You get the information using the command **`netstat vcrt`**.

Further, we used to check the data flow of the multicast messages and the distribution flow through an IPCS trace. Excerpts of the trace follow.

Dynamic VIPA configuration, part 1

```
=> netstat vipadyn
 MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP             18:00:42
 IP Address      AddressMask     Status    Origination     DistStat
 ----------      -----------     ------    -----------     --------
9.67.156.25     255.255.255.248 Active    VIPADefine
9.67.156.26     255.255.255.248 Active    VIPADefine
9.67.156.33     255.255.255.248 Backup    VIPABackup
9.67.156.34     255.255.255.248 Backup    VIPABackup
9.67.156.41     255.255.255.248 Backup    VIPABackup
9.67.156.42     255.255.255.248 Backup    VIPABackup
9.67.156.49     255.255.255.248 Backup    VIPABackup
9.67.156.50     255.255.255.248 Backup    VIPABackup
9.67.157.17     255.255.255.248 Active    VIPADefine      Dist
9.67.157.18     255.255.255.248 Active    VIPADefine      Dist/Dest
 ***
```

The last two entries contain the cluster IP addresses that may be distributed. To determine to which available target system TCP connection requests may be distributed, use the **`netstat vdpt`** command.

The current VIPA configuration including the Service Manager definitions for the multicast IP address and the port number will be shown by using the command **`netstat vipadcfg`** (see the following screen output).

The multicast address and the port number have to match the definitions in the Cisco routers running the Forwarding Agent. See 6.5.1, “Service Manager new TCPIP.PROFILE definitions” on page 197.

Dynamic VIPA configuration part 2

```
=> netstat vipadcfg
MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP 18:06:48
Dynamic VIPA Information:
VIPA Backup:
  IP Address       Rank
  ----------       ----
  9.67.156.33      000001
  9.67.156.34      000001
  9.67.156.41      000001
  9.67.156.42      000001
  9.67.156.49      000100
  9.67.156.50      000100

VIPA Define:
  IP Address       AddressMask      Moveable  SrvMgr
  ----------       -----------      --------  ------
  9.67.156.25      255.255.255.248  Immediate No
  9.67.156.26      255.255.255.248  Immediate No
  9.67.157.17      255.255.255.248  Immediate Yes
  9.67.157.18      255.255.255.248  Immediate Yes

VIPA Distribute:
  IP Address       Port   XCF Address
  ----------       ----   -----------
  9.67.157.17      00023  9.67.156.75
9.67.157.17      00023  9.67.156.74
9.67.157.17      00080  9.67.156.75
9.67.157.17      00080  9.67.156.74
9.67.157.17      00443  9.67.156.75
9.67.157.17      00443  9.67.156.74
9.67.157.17      00523  9.67.156.75
9.67.157.17      00523  9.67.156.74
9.67.157.18      00020  9.67.156.74
9.67.157.18      00020  9.67.156.73
9.67.157.18      00021  9.67.156.74
9.67.157.18      00021  9.67.156.73

VIPA Service Manager:
  McastGroup: 224.0.1.2        Port: 01637  Pwd: No
```

VIPA distribution port table

The VIPA distribution port table obtained using the **`netstat vdpt`** command lists cluster IP addresses with the different port addresses the Sysplex Distributor is responsible for. Each cluster address is defined in the TCPIP.PROFILE using the statements VIPADEFINE, VIPADIST, and DESTIP.

```
==> netstat vdpt
MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP       14:50:09
Dynamic VIPA Distribution Port Table:
Dest IPaddr     DPort DestXCF Addr    Rdy TotalConn  WLM
-----------     ----- ------------    --- ---------  ---
9.67.157.17     00023 9.67.156.74     001 0000000000 01
9.67.157.17     00023 9.67.156.75     001 0000000000 01
9.67.157.17     00080 9.67.156.74     001 0000000000 01
9.67.157.17     00080 9.67.156.75     001 0000000000 01
9.67.157.17     00443 9.67.156.74     001 0000000000 01
9.67.157.17     00443 9.67.156.75     000 0000000000 01
9.67.157.17     00523 9.67.156.74     001 0000000000 01
9.67.157.17     00523 9.67.156.75     001 0000000000 01
9.67.157.18     00020 9.67.156.73     000 0000000001 01
9.67.157.18     00020 9.67.156.74     000 0000000000 01
9.67.157.18     00021 9.67.156.73     001 0000000001 01
9.67.157.18     00021 9.67.156.74     001 0000000000 01
```

The following fields are included in the display:

Rdy Counts the number of applications ready to receive connection requests.

TotalConn Lists the total number of connection already routed to the stack identified by the XCF link IP address.

WLM Is the WLM weight of the target stack. The lowest value is the first to be used for distribution.

Detailed information about the VIPA distribution port table may be obtained using the command **`netstat vdpt det`**.

```
===> netstat vdpt detail
MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP         20:17:50
Dynamic VIPA Distribution Port Table:
Dest IPaddr     DPort DestXCF Addr    Rdy TotalConn  WLM
-----------     ----- ------------    --- ---------  ---
9.67.157.17     00023 9.67.156.74     001 0000000000 01
  QosPlcAct:  *DEFAULT*                                   W/Q: 01
9.67.157.17     00023 9.67.156.75     001 0000000000 01
  QosPlcAct:  *DEFAULT*                                   W/Q: 01
9.67.157.17     00080 9.67.156.74     001 0000000000 01
  QosPlcAct:  *DEFAULT*                                   W/Q: 01
9.67.157.17     00080 9.67.156.75     001 0000000000 01
  QosPlcAct:  *DEFAULT*                                   W/Q: 01
9.67.157.17     00443 9.67.156.74     001 0000000000 01
  QosPlcAct:  *DEFAULT*                                   W/Q: 01
9.67.157.17     00443 9.67.156.75     000 0000000000 01
  QosPlcAct:  *DEFAULT*                                   W/Q: 01
9.67.157.17     00523 9.67.156.74     001 0000000000 01
  QosPlcAct:  *DEFAULT*                                   W/Q: 01
9.67.157.17     00523 9.67.156.75     001 0000000000 01
  QosPlcAct:  *DEFAULT*                                   W/Q: 01
9.67.157.18     00020 9.67.156.73     000 0000000000 01
  QosPlcAct:  *DEFAULT*                                   W/Q: 01
9.67.157.18     00020 9.67.156.74     000 0000000000 01
  QosPlcAct:  *DEFAULT*                                   W/Q: 01
9.67.157.18     00021 9.67.156.73     001 0000000000 01
  QosPlcAct:  *DEFAULT*                                   W/Q: 01
9.67.157.18     00021 9.67.156.74     001 0000000000 01
  QosPlcAct:  *DEFAULT*                                      W/Q: 01
```

The following fields are included in the display:

W/Q Is the WLM weight after modification using QoS information provided by the Policy Agent. This information is an indication of the network performance (TCP retransmissions and timeouts) for the display of a QoSPolicyAction.

QoSPlcAct Is the QoS Policy name configured to the Policy Agent.

VIPA connection routing table

```
=> netstat vcrt
 MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP        14:48:53
 Dynamic VIPA Connection Routing Table:
 Dest IPaddr      DPort    Src IPaddr       SPort    DestXCF Addr
 -----------      -----     ----------      -----    ------------
 9.67.157.18      00021    9.67.155.223     01034    9.67.156.73
 9.67.157.18      00021    9.67.156.104     03494    9.67.156.73
 9.67.157.17      00023    9.67.156.104     03395    9.67.156.74
 9.67.157.17      00023    9.67.156.104     03460    9.67.156.75
 9.67.157.17      00023    9.67.156.104     03472    9.67.156.74
 9.67.157.17      00080    9.67.156.104     03469    9.67.156.74
```

The `netstat vcrt` command shows current TCP connections distributed by Sysplex Distributor to the target systems identified through their XCF link IP address. These connections are addressed to the following:

- FTP server with its cluster address 9.67.157.18
- TN3270 server with its cluster address 9.67.157.17

TCP connections to FTP server

One connection was initiated by a client from a workstation with the IP address 9.67.155.223, and the other connection from workstation 9.67.156.104. Both connections were distributed to an FTP server running in the same TCP/IP stack as the Sysplex Distributor.

TCP connections to TN3270 server

All connections were initiated by a client from one workstation with IP address 9.67.156.104. Three connections were distributed to the TN3270 server running in a TCP/IP stack with IP address 9.67.156.74 and one connection to 9.67.156.75.

Detailed information may be obtained using the `netstat vcrt detail` command.

This display would also show Policy Agent information. In our test case, we did not define policy rules.

```
=> netstat vcrt detail
MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP     20:54:59
Dynamic VIPA Connection Routing Table:
Dest IPaddr      DPort  Src IPaddr       SPort  DestXCF Addr
-----------      -----  ----------       -----  ------------
9.67.157.17      00023  211.1.2.14       11033  9.67.156.74
  PolicyRule:    *NONE*
  PolicyAction:  *NONE*
***
```

Trace details

1. *Multicast messages from the Sysplex Distributor Service Manager*

Multicast of a wildcard affinity-update affinity:

This trace record shows multicast messages sent from the Service Manager to the network on each defined interface shown in the HOME list. This includes XCF links and IUTSAMEH links to stacks within the same LPAR.

Multicast messages will be sent in a cycle of 30 seconds. There is no statement in the TCPIP.PROFILE to change the value nor to block the interface from sending multicast packets.

```
 238 MVS001   PACKET   00000001 14:31:59.255980 Packet Trace
 To Link          : GIGELINK        Device: QDIO Ethernet    Full=96
  Tod Clock       : 2001/07/24 14:31:59.255978
  Lost Records    : 0               Flags: Pkt Ver2 Out
  Source Port     : 1637    3       Dest Port: 1637 Asid: 0036 TCB:0000
 IpHeader: Version : 4              Header Length: 20
  Tos             : 00              QOS: Routine Normal Service
  Packet Length   : 96              ID Number: 626B
  Fragment        :                 Offset: 0
  TTL             : 1               Protocol: UDP      CheckSum: CF5B
  Source          : 9.67.157.129  1
  Destination     : 224.0.1.2     2 SGI-Dogfight

 UDP   4
  Source Port     : 1637  ()  3     Destination Port:  1637  ()
  Datagram Length : 76              CheckSum: 049B FFFF

IP Header         : 20     IP: 9.67.157.129, 224.0.1.2
000000 45000060 626B0000 0111CF5B 09439D81  E0000102
Protocol Header    : 8      Port:  1637,  1637
000000 06650665 004C049B

Data    5         : 68     Data Length: 68
000000 00010101 81010024 00000006 00000000 !....a........... .......$..
000010 00000000 09439D11 FFFFFFFF 00000050 !...............& .....C.P!
000020 00000000 003C0000 05050008 05040000 !................ .....<..!
000030 85040014 09439C49 06658100 81000000 !e.........a.a... .....C..!
000040 00000000                            !....             ....    !
```

Remarks on the trace record:

1 Source IP address 9.67.157.129 is the OSA-Express GbE interface with link name GIGELINK.

2 Destination IP address 224.0.1.2 is the multicast IP address specified in the TCPIP.PROFILE statement VIPASMPARMS parameter SMMCAST. This multicast address has to match the definition in the Cisco router's Forwarding Agent. See 6.7.1, "CASA definitions for Cisco 7507" on page 199.

3 Port address for the CASA application. The same port address will be used in the Sysplex Distributor's Service Manager and in the Forwarding Agent. The port is defined in the TCPIP.PROFILE statement VIPASMPARMS parameter SMPORT. It has to match the definition in the Cisco router's Forwarding Agent. See 6.7.1, "CASA definitions for Cisco 7507" on page 199.

4 UDP is used for the port-to-port communication.

5 Trace Data of the wildcard affinity message:

Line 000000:

00010101 8101	Is a CASA packet, wildcard affinity update
00000006	Supported protocol is TCP (value in the IP header)
00000000	Is the source IP address
00000000	Is the source subnet mask

Line 000010:

09439D11	Is the destination IP address 9.67.157.17
FFFFFFFF	Is the destination subnet mask 255.255.255.255
00000050	Is the destination port 80 for Web services

Line 000020:

00000000 Is the forwarding destination IP address (not known yet by the Sysplex Distributor)

003C Is the time-to-live (TTL) value for the wildcard affinity entry for the Forwarding Agent's table. It means: 60 seconds.

2. ***SYN sent from the Forwarding Agent to the Service Manager***

   This message is sent by the Forwarding Agent to the Service Manager or vice versa in special situations, for example an IP packet matching the wildcard affinity entry. This is the case when the Forwarding Agent received an IP packet from a client containing a SYN request.

   Trace data of the IP packet only:

   1 Source and destination IP addresses are tunnel addresses that are used by the Generic Routing Encapsulation (GRE) protocol. This will be discussed later in 6.10, “Generic Routing Encapsulation (GRE) protocol” on page 231.

   2 GRE header will be discussed later in 6.10, “Generic Routing Encapsulation (GRE) protocol” on page 231.

   3 The source address 1.1.1.2 is the IP address used by the Forwarding Agent to send CASA packets to the Service Manager's destination IP address 9.67.156.73. This is the XCF link address provided in the multicast CASA wildcard packet to the Forwarding Agent.

   4 Source and destination port 1637 will be used for CASA communication.

   5 Trace data of the CASA packet:

   Line 000000:

   `00010601 0601` Is a CASA message of the type IP packet only

   Line 000010:

   `09439C68` The packet comes from source IP address 9.67.156.104, a client workstation

   `09439D11` Destination IP address is 9.67.157.17, the cluster IP address for the application service

   `0AD3` Source port is 2771

   `0017` Destination port is 23 TN3270 services

```
299 MVS001   PACKET   00000001 17:54:26.608493 Packet Trace
 From Link         : GIGELINK        Device: QDIO Ethernet    Full=108
  Tod Clock        : 2001/07/16 17:54:26.608492
  Lost Records     : 0               Flags: Pkt Ver2 Gre
  Source Port      : 0               Dest Port: 0     Asid: 0025 TCB: 0000000
 IpHeader: Version : 4               Header Length: 20
  Tos              : 00              QOS: Routine Normal Service
  Packet Length    : 108             ID Number: B9EA
  Fragment         :                 Offset: 0
  TTL              : 254             Protocol: GRE           CheckSum: B668
  Source           : 9.67.157.136  1
  Destination      : 9.67.156.1    1

 Generic Routing Encapsulation Header   2
  GRE Options      :
  Version          : 0               Protocol: IP
  Recursion        : 0
  Gre header size  : 4


IP Header          : 20
000000 4500006C B9EA0000 FE2FB668 09439D88  09439C01

GRE Header         : 4
000000 00000800
 IpHeader: Version : 4               Header Length: 20
  Tos              : 00              QOS: Routine Normal Service
  Packet Length    : 84              ID Number: 0275
  Fragment         :                 Offset: 0
  TTL              : 255             Protocol: UDP           CheckSum: 1195
  Source           : 1.1.1.2       3
  Destination      : 9.67.156.73   3

 UDP
  Source Port      : 1637  ()        Destination Port:  1637  ()     4
  Datagram Length  : 64              CheckSum: 0000 74FF

IP Header          : 20     IP: 1.1.1.2, 9.67.156.73    3
000000 45000054 02750000 FF111195 01010102  09439C49
Protocol Header    : 8      Port:  1637,  1637
000000 06650665 00400000

Data        5      : 56     Data Length: 64
000000 00010601 06010034 45300030 B66E4000 !.............> . .......4EO.O.n§.!
000010 7F06F929 09439C68 09439D11 0AD30017 !".9..........L.. ...).C.h.C......!
000020 76748623 00000000 70024000 EF9D0000 !..f....... ..... vt.#....p.§.....!
000030 020405B4 01010402                   !........         ........        !
```

3. SYN sent from Sysplex Distributor request to target system

```
300 MVS001   PACKET   00000001 17:54:26.609824 Packet Trace
 To Link           : EZAXCFN5        Device: MPCPTP           Full=48
  Tod Clock        : 2001/07/16 17:54:26.609824
  Lost Records     : 0               Flags: Pkt Ver2 Out
  Source Port      : 2771            Dest Port: 23   Asid: 0025 TCB:000
 IpHeader: Version : 4               Header Length: 20
  Tos              : 30                QOS: Priority MinimumDelay
  Packet Length    : 48              ID Number: B66E
   Fragment         : DontFragment  Offset: 0
   TTL               : 126             Protocol: TCP      CheckSum: FA29
   Source           : 9.67.156.104
   Destination      : 9.67.157.17

  TCP
   Source Port      : 2771  ()       Destination Port: 23   telnet)
   Sequence Number  : 1987348003     Ack Number: 0
   Header Length    : 28             Flags: Syn
   Window Size      : 16384          CheckSum: EF9D FFFF Urgent Data P
    Option          : Max Seg Size Len: 4 MSS: 1460
    Option          : NOP
    Option          : NOP
    Option          : SACK Permitted

 IP Header          : 20     IP: 9.67.156.104, 9.67.157.17
 000000 45300030 B66E4000 7E06FA29 09439C68  09439D11
 Protocol Header    : 28     Port:  2771,    23
 000000 0AD30017 76748623 00000000 70024000  EF9D0000 020405B4 01010402
```

4. Fixed affinity update sent from Sysplex Distributor to Forwarding Agent

A unicast message is used to send a CASA packet from XCF link address of the Sysplex Distributor, which is 9.67.156.73, to the Forwarding Agent with its IP address 1.1.1.2. Only the Forwarding Agent that sent the SYN to the Service Manager will receive the CASA affinity update.

.

```
301 MVS001   PACKET   00000001 17:54:26.610488 Packet Trace
 To Link          : GIGELINK        Device: QDIO Ethernet    Full=80
  Tod Clock       : 2001/07/16 17:54:26.610487
  Lost Records    : 0            Flags: Pkt Ver2 Out
  Source Port     : 1637         Dest Port: 1637  Asid: 0025 TCB:0000
 IpHeader: Version : 4           Header Length: 20
  Tos             : 00           QOS: Routine Normal Service
  Packet Length   : 80           ID Number: 512D
  Fragment        :              Offset: 0
  TTL             : 1            Protocol: UDP       CheckSum: C0E1
  Source          : 9.67.156.73
  Destination     : 1.1.1.2

 UDP
  Source Port     : 1637  ()        Destination Port:  1637  ()
  Datagram Length  : 60             CheckSum: 97E9 FFFF

IP Header          : 20     IP: 9.67.156.73, 1.1.1.2
000000 45000050 512D0000 0111C0E1 09439C49  01010102
Protocol Header    : 8      Port:  1637,  1637
000000 06650665 003C97E9

Data               : 52     Data Length: 52
000000 00010102 8102001C 00020006 09439C68 !....a........ .........C.h!
000010 09439D11 0AD30017 09439C4A 03840000 !.....L..Ä.d.. .C...C.J....!
000020 85040014 09439C49 06650002 00020000 !e........... .....e......!
000030 00000000                            !....           ....
```

Trace data of the CASA fixed affinity update packet:

Line 000000:

00010102 Defines a CASA message fixed affinity-update affinity

8102001C Specifies the message type

00020006 Specifies flags and the protocol, which is x’06’ = TCP in the IP header

09439C68 Defines the source IP address 9.67.156.104 which is the workstation as sender of the SYN

Line 000010:

09439D11 Defines the destination IP address is 9.67.157.17, the distributed DVIPA, the cluster IP address of the application service

0AD3 Source port of the client in the workstation

0017 Destination port for the TCP connection, port 23 = TN3270 services

09439C4A Forward IP address of the real target system running this specific TCP connection; it is the XCF link IP address defined in TCPIP.PROFILE statement VIPADEFINE, parameter VIPADIST, parameter DESTIP

0384 Defines the time-to-live (TTL) value of this fixed affinity-update affinity for this TCP connection; the value is 15 minutes

6.8.2 CASA information in the Forwarding Agent

Some displays help you to determine whether the Forwarding Agent is communicating with the Sysplex Distributor Service Manager. During our tests we noticed that this communication did not work as we thought it would. In certain cases, the Service Manager did not send wildcard affinities. These displays allowed us to understand what was really happening.

You may control the Forwarding Agent by issuing several displays (`show` commands) at the Cisco router. These are for example:

`show ip casa ?` Display what parameters are available

- **With no parameter** To display a list of subfunctions named in the next lines
- **With parameter `affinities`** To display fixed affinities, which are the current TCP connections
- **With parameter `oper`** To display operational information of CASA
- **With parameter `stats`** To display statistical information about fixed affinities
- **With parameter `wildcard`** To display information on wildcard blocks

```
NIVT7507>show ip casa ?
  affinities  display info on fa affinities
  oper        operational information for casa
  stats       statistical information for fa
  wildcard    display info on wildcard blocks
```

Display of CASA operational information

This display shows that the CASA is defined with the corresponding multicast address and the listen port number. These have to match the definitions of the Sysplex Distributor TCPIP.PROFILE statement VIPASMPARMS with parameters SMMCAST and SMPORT.

In addition, a CASA control address has to be defined to be used as IP address for unicasts to the Service Manager or vice versa. This address has to be unique for all Forwarding Agents. We used IP address 1.1.1.1 for the Forwarding Agent in the router 7507 and 1.1.1.2 in router 720VXR.

This display also shows that CASA is operating in the Cisco router.

```
NIVT7507>show ip casa oper
Casa is active:
  Casa control address is 1.1.1.1/32
  Casa multicast address is 224.0.1.2
  Listening on ports:
    Forwarding Agent Listen Port: 1637
      Current passwd: NONE Pending passwd: NONE
      Passwd timeout: 180 sec (Default)
```

The next step should be to check if the Service Manager propagates wildcard affinities correctly.

Display of wildcard affinities

We experienced a situation where the Service Manager did not function properly. When we wanted to see the wildcard affinities propagated by the Service Manager, we received the following display result.

```
NIVT7507>show ip casa affinities
No matching entries in affinity cache
NIVT7507>
```

After fixing the problem we received the expected result. The following display shows the wildcard affinities multicasted by the Service Manager to the Forwarding Agents as depicted in the Cisco router 7507.

```
NIVT7507>show ip casa wild
Source Address Source Mask Port Dest Address Dest Mask       Port  Prot
0.0.0.0        0.0.0.0     0    9.67.157.17  255.255.255.255 523   TCP
0.0.0.0        0.0.0.0     0    9.67.157.17  255.255.255.255 443   TCP
0.0.0.0        0.0.0.0     0    9.67.157.17  255.255.255.255 80    TCP
0.0.0.0        0.0.0.0     0    9.67.157.17  255.255.255.255 23    TCP
0.0.0.0        0.0.0.0     0    9.67.157.18  255.255.255.255 21    TCP
0.0.0.0        0.0.0.0     0    9.67.157.18  255.255.255.255 20    TCP
```

The source IP address of 0.0.0.0 allows the Forwarding Agent to accept any incoming IP packet with any port number of the protocol TCP. The destination IP address represents the cluster address for the application service defined by the port number, for example:

- Port 23 for TN3270E service
- 523 for secure TN3270E service
- 80 for Web services
- 443 for secure Web services
- 21 and 20 for FTP service

The subnet mask limits the match to the whole 32-bit IP address.

This wildcard affinity block is sent to the Forwarding Agents from the Service Manager every 30 seconds over each interface of the Sysplex Distributor.

Display of wildcard affinities detailed information

This display is an excerpt of the block displayed in the previous screen. It provides more detailed information. For example:

- Interest address

  The interest address is the IP address of the Sysplex Distributor's XCF link address 9.67.156.73.

- Port

  The port 1637 is the CASA listening port of the Sysplex Distributor.

- Interest packet

  All packets including fragmented IP packets are accepted by the Forwarding Agent.

- Dispatch address

  The dispatch address is the IP address for a target server running the desired application. At the moment no dispatch address is known. A dispatch address will be determined after processing a SYN request in the Sysplex Distributor. The dispatch address will be determined based on load-balancing rules. This dispatch address is sent via unicast messages by the Sysplex Distributor to the Forwarding Agent when the target system is known.

```
NIVT7507>show ip casa wild det
Source Address  Source Mask     Port  Dest Address    Dest Mask       Port  Prot
0.0.0.0         0.0.0.0         0     9.67.157.17     255.255.255.255 523   TCP
  Service Manager Details:
    Manager Addr:           9.67.156.17       Insert Time: 01:22:57 UTC 07/11/01
  Affinity Statistics:
    Affinity Count:         0                 Interest Packet Timeouts: 0
  Packet Statistics:
    Packets:                0                 Bytes: 0
 Advertise Dest Address: NO                   Match Fragments:  YES
 Affinity TTL: 30
  Action Details:
    Interest Addr:          9.67.156.73       Interest Port: 1637
    Interest Packet: 0x8100 FRAG ALLPKTS
    Interest Tickle: 0x0000
    Dispatch (Layer 2):     NO                Dispatch Address: 0.0.0.0
Source Address  Source Mask     Port  Dest Address    Dest Mask       Port  Prot
0.0.0.0         0.0.0.0         0     9.67.157.17     255.255.255.255 443   TCP
  Service Manager Details:
    Manager Addr:           9.67.156.17       Insert Time: 01:22:57 UTC 07/11/01
  Affinity Statistics:
    Affinity Count:         0                 Interest Packet Timeouts: 0
  Packet Statistics:
    Packets:                0                 Bytes: 0
 Advertise Dest Address: NO                   Match Fragments:  YES
 Affinity TTL: 30
  Action Details:
    Interest Addr:          9.67.156.73       Interest Port: 1637
    Interest Packet: 0x8100 FRAG ALLPKTS
    Interest Tickle: 0x0000
    Dispatch (Layer 2):     NO                Dispatch Address: 0.0.0.0

Source Address  Source Mask     Port  Dest Address    Dest Mask       Port  Prot
0.0.0.0         0.0.0.0         0     9.67.157.17     255.255.255.255 80    TCP
  Service Manager Details:
    Manager Addr:           9.67.156.17       Insert Time: 01:22:57 UTC 07/11/01
  Affinity Statistics:
    Affinity Count:         0                 Interest Packet Timeouts: 0
  Packet Statistics:
............
```

CASA fixed affinity

This display shows the TCP connections currently established and known by the Forwarding Agent. As you can see, only the source and destination IP addresses and ports of the IP and TCP header are shown, not the real target system’s IP address. If you want to see the target system’s address, you have to use the **`show ip casa det`** (detail) command.

```
NIVT7507>show ip casa aff
Source Address  Port  Dest Address    Port  Prot
9.67.156.104    3879  9.67.157.17     23    TCP
9.67.156.104    3890  9.67.157.18     21    TCP
NIVT7507>
```

Our sample shows two established TCP connections:

1. A telnet connection from workstation 9.67.156.104 to the cluster address 9.67.157.17. This connection is distributed to the target system with XCF link address (dispatch address) 9.57.156.74.
2. An FTP connection from workstation 9.67.156.104 to the cluster address 9.67.157.18. This connection is distributed to the target system with XCF link address (dispatch address) 9.57.156.73.

```
NIVT7507>show ip casa aff det
Source Address  Port  Dest Address    Port Prot
9.67.156.104    3879  9.67.157.17     23   TCP
  Action Details:
    Interest Addr:          9.67.156.73    Interest Port: 1637
    Interest Packet: 0x0002 SYN
    Interest Tickle: 0x0000
    Dispatch (Layer 2):     YES            Dispatch Address: 9.67.156.74
Source Address  Port  Dest Address    Port Prot
9.67.156.104    3890  9.67.157.18     21   TCP
  Action Details:
    Interest Addr:          9.67.156.73    Interest Port: 1637
    Interest Packet: 0x0002 SYN
    Interest Tickle: 0x0000
     Dispatch (Layer 2):  YES       Dispatch Address: 9.67.156.73
```

The detailed fixed affinity information in the Forwarding Agent may be compared with the information of the Sysplex Distributor obtained using the command **`netstat vcrt`**.

CASA statistics

The following screen provides information about the activities of the CASA protocol.

```
NIVT7507>show ip casa stats
Casa is active:
  Wildcard Stats:
    Wildcards:       6           Max Wildcards:            6
    Wildcard Denies: 0           Wildcard Drops:           0
    Pkts Throughput: 0           Bytes Throughput:         0
  Affinity Stats:
    Affinities:      0           Max Affinities:           0
    Cache Hits:      0           Cache Misses:             0
    Affinity Drops:  0           Interest Packet Timeouts: 0
  Casa Stats:
    Int Packet:      0           Int Tickle:               0
    Casa Denies:     0           Drop Count:               0
    Security Drops:  0
```

6.8.3 Integrated CASA information

This section provides information about the control of operations at the time a TCP connection is closed. The displays are intentionally not separated as in the two previous sections, where the focus was mainly on describing fundamentals of each system. This section describes control situations viewed from a time aspect at both systems, the Sysplex Distributor and the router, at one time.

The section includes displays issued at the Sysplex Distributor and at the router to control a process over a time period. It also provides a trace flow to understand the activities. We selected as a sample the process of finishing a TCP connection.

A sample connection

A TN3270 connection was established between a client on workstation 9.67.156.104 and TN3270 server cluster address 9.67.157.17. The Sysplex Distributor on system MVS001 distributed the connection to target system 9.67.156.17 on system MVS154.

The following screens show the steps in closing the TN3270 connection:

- Display the connection via **`netstat con`** on MVS154
- Display the distributed VIPA connection routing table via **`netstat vcrt`** on Sysplex Distributor on system MVS001
- Display the fixed affinity table in router 7206 using the **`show ip casa affinities det`** command
- Analyze trace records obtained from system MVS154
  - FIN sent from client to server, to signal the start of the connection process
  - ACK sent from server to client, to acknowledge the reception of the FIN
  - FIN sent from server to client, to signal that second part of closing the connection is started
  - ACK sent from client server, to finally finish the connection flow and thus the connection is closed
- Check if the connection no longer exists using the **`netstat con`** command on MVS154
- Check if the VIPA connection routing table is cleared from the connection using the **`netstat vcrt`** command
- Check if fixed affinity is deleted in the router using the **`show ip casa affinities det`** command

Display the existing connection

The display shows the current connection in the target system MVS154. The information of the TCP connection between client and server is highlighted.

```
=> netstat conn
MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP
User Id  Conn     Local Socket      Foreign Socket       State
-------  ----     ------------      --------------       -----
BPXOINIT 0000001D 0.0.0.0..10007    0.0.0.0..0           Listen
ENDPT40  0000003C 0.0.0.0..10115    0.0.0.0..0           Listen
FTPMVS1  00000023 0.0.0.0..21       0.0.0.0..0           Listen
HODSERV5 0000002B 0.0.0.0..8999     0.0.0.0..0           Listen
................
OMPROUTE 00000022 127.0.0.1..1027   127.0.0.1..1028      Establsh
OSNMPD   00000021 0.0.0.0..1029     0.0.0.0..0           Listen
TCP      0000004E 9.67.157.17..23   9.67.156.104..4202   Establsh
TCP      00000010 0.0.0.0..523      0.0.0.0..0           Listen
TCP      0000000B 127.0.0.1..1025   0.0.0.0..0           Listen
TCP      0000000E 127.0.0.1..1026   127.0.0.1..1025      Establsh
TCP      00000011 0.0.0.0..23       0.0.0.0..0           Listen
TCP      0000001C 127.0.0.1..1028   127.0.0.1..1027      Establsh
TCP      0000000F 127.0.0.1..1025   127.0.0.1..1026      Establsh
WEBSDB2  00000019 0.0.0.0..80       0.0.0.0..0           Listen
ENDPT40  0000003B 0.0.0.0..10115    *..*                 UDP
INETD1   00000041 0.0.0.0..7        *..*                 UDP
INETD1   0000003F 0.0.0.0..19       *..*                 UDP
INETD1   00000040 0.0.0.0..9        *..*                 UDP
..............
```

The VIPA connection routing table in the Sysplex Distributor stack of system MVS001 also shows this connection.

```
===> netstat vcrt
 MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP
Dynamic VIPA Connection Routing Table:
Dest IPaddr      DPort  Src IPaddr       SPort  DestXCF Addr
-----------      -----  ----------       -----  ------------
9.67.157.17      00023  9.67.156.104     04202  9.67.156.75
***
```

The router has a fixed affinity for the connection pointing to the same IP address for the target system, the MVS154.

The interest address points to the Sysplex Distributor's Service Manager. The dispatch address points to the target system MVS154.

```
7200-Z55>show ip casa aff det

Source Address  Port  Dest Address    Port  Prot
9.67.156.104    4202  9.67.157.17     23    TCP
 Action Details:
   Interest Addr:         9.67.156.73 Interest Port: 1637
   Interest Packet: 0x0002 SYN
   Interest Tickle: 0x0000
   Dispatch (Layer 2):     YES         Dispatch Address: 9.67.156.75
--More--
C7200-Z55>
```

Trace of closing the TN3270 connection

This trace shows the actions between the client and the target stack on system MVS154 when the client starts to finish the TCP connection with a TN3270 server.

The trace is recorded at system MVS154, at the endpoint of the TCP connection. All trace records have record numbers starting with record 503. The direction of the data flow is indicated as follows:

Inbound traffic to MVS154	From Link:	GIGELINK
Outbound traffic to router	To :	GIGELINK

Inbound traffic records are shortened intentionally, in order to take some complexity from the shown trace. Inbound traffic requires in addition a Generic Routing Encapsulation (GRE) protocol between router and target stack. Trace data for this part is not shown here. You may read more about GRE in 6.10, “Generic Routing Encapsulation (GRE) protocol” on page 231.

Trace record 503

The trace starts with record 503, listing a FIN sent from a client to the application server to start closing the connection.

Client IP address	Source address: 9.67.156.104
Client port number	Port: 4202
TN3270 server IP address	Destination address: 9.67.157.17 (cluster IP address)
TN3270 port	Port: 23 (telnet)
Client sequence number	1118528899 for the FIN to server
ACK sequence number	1627793504 for the last data from server

```
 503 MVS154   PACKET   00000001 16:31:46.482764 Packet Trace
From Link         : GIGELINK        Device: QDIO Ethernet    Full=64
  Tod Clock        : 2001/07/27 16:31:46.482763
------------------------------------------------------------------------------
GRE part is deleted here, to avoid the complexity of the trace
------------------------------------------------------------------------------
IpHeader: Version : 4               Header Length: 20
 Tos              : 30              QOS: Priority MinimumDelay
 Packet Length    : 40              ID Number: 1591
 Fragment         : DontFragment    Offset: 0
 TTL              : 127             Protocol: TCP          CheckSum: 9A0F
 Source           : 9.67.156.104
 Destination      : 9.67.157.17

TCP
  Source Port      : 4202  ()        Destination Port: 23    (telnet)
 Sequence Number  : 1118528899      Ack Number: 1627793504
  Header Length    : 20              Flags: Ack Fin
 Window Size      : 17149           CheckSum: DEC0 FFFF Urgent Data Pointer:

IP Header          : 20     IP: 9.67.156.104, 9.67.157.17
000000 45300028 15914000 7F069A0F 09439C68  09439D11
Protocol Header    : 20     Port:  4202,    23
000000 106A0017 42AB6583 61062860 501142FD  DEC00000
```

Figure 6-4 Trace record 503: FIN sent from client

Trace record 504

Trace record 504 lists an acknowledgment (ACK) only from the server as response to the previous FIN from the client. PSH means *PUSH,* to force the IP stack to send this packet immediately to the network and not wait for other data to fill the output buffer before sending the packet to the client.

TN3270 server IP address	Source address: 9.67.157.17 (again cluster IP address)
TN3270 port	Port: 23 (telnet)
Client IP address	Destination address: 9.67.156.104
Client port number	Port: 4202
Server sequence number	1627793504 (remains the same because no data was sent)
ACK sequence number	1118528900 for the ACK to client (1 byte higher than sent from the client)

```
504 MVS154   PACKET   00000001 16:31:46.482962 Packet Trace
 To Link          : GIGELINK        Device: QDIO Ethernet    Full=40
  Tod Clock       : 2001/07/27 16:31:46.482962
  Lost Records    : 0               Flags: Pkt Ver2 Out
  Source Port     : 23              Dest Port: 4202  Asid: 0034 TCB: 0000000
 IpHeader: Version : 4              Header Length: 20
  Tos             : 30              QOS: Priority MinimumDelay
  Packet Length   : 40              ID Number: 0B06
  Fragment        : DontFragment    Offset: 0
  TTL             : 64              Protocol: TCP           CheckSum: E39A
  Source          : 9.67.157.17
  Destination     : 9.67.156.104

  TCP
   Source Port      : 23    (telnet)   Destination Port: 4202  ()
   Sequence Number  : 1627793504       Ack Number: 1118528900
   Header Length    : 20               Flags: Ack Psh
   Window Size      : 32760            CheckSum: A1BD FFFF Urgent Data Pointer:

 IP Header          : 20     IP: 9.67.157.17, 9.67.156.104
 000000 45300028 0B064000 4006E39A 09439D11  09439C68
 Protocol Header    : 20     Port:    23,  4202
 000000 0017106A 61062860 42AB6584 50187FF8  A1BD0000
```

Figure 6-5 Trace record 504: ACK sent from server

Trace record 505

Trace record 505 lists a FIN from the server to close the second part of the TCP connection. PSH means *PUSH,* to force the IP stack to send this packet immediately to the network and not wait for other data to fill the output buffer before sending the packet to the client.

TN3270 server IP address	Source address: 9.67.157.17 (again cluster IP address)
TN3270 port	Port: 23 (telnet)
Client IP address	Destination address: 9.67.156.104
Client port number	Port: 4202
Server sequence number	1627793504 (remains the same because no data was sent)
ACK sequence number	1118528900 for the ACK to client (remains the same because no data came from the other end of the connection)

```
505 MVS154   PACKET   00000001 16:31:46.484090 Packet Trace
  To Link          : GIGELINK        Device: QDIO Ethernet    Full=40
   Tod Clock       : 2001/07/27 16:31:46.484089
   Lost Records    : 0               Flags: Pkt Ver2 Out
   Source Port     : 23              Dest Port: 4202  Asid: 0034 TCB: 007D42E
  IpHeader: Version : 4              Header Length: 20
     Tos            : 30               QOS: Priority MinimumDelay
   Packet Length   : 40              ID Number: 0B07
   Fragment        : DontFragment    Offset: 0
   TTL             : 64              Protocol: TCP           CheckSum: E399
   Source          : 9.67.157.17
   Destination     : 9.67.156.104

  TCP
   Source Port     : 23    (telnet)   Destination Port: 4202  ()
   Sequence Number  : 1627793504      Ack Number: 1118528900
   Header Length   : 20              Flags: Ack Psh Fin
   Window Size     : 32760           CheckSum: A1BC FFFF Urgent Data Pointer:

 IP Header         : 20     IP: 9.67.157.17, 9.67.156.104
 000000 45300028 0B074000 4006E399 09439D11  09439C68
 Protocol Header   : 20     Port:    23,  4202
 000000 0017106A 61062860 42AB6584 50197FF8  A1BC0000
```

Figure 6-6 Trace record 505: FIN sent from server

Trace record 506

Trace record 506 lists an ACK sent from the client to the application server to respond to the FIN and thus the TCP connection is totally closed.

Client IP address	Source address: 9.67.156.104
Client port number	Port: 4202
TN3270 server IP address	Destination address: 9.67.157.17 (cluster IP address)
TN3270 port	Port: 23 (telnet)
Client sequence number	1118528900 for the ACK to server (remains the same, because no data was sent)
ACK number	1627793505 (is increased by 1, caused by having received the FIN from the server)

```
506 MVS154   PACKET   00000001 16:31:46.489254 Packet Trace
 From Link         : GIGELINK        Device: QDIO Ethernet    Full=64
  Tod Clock        : 2001/07/27 16:31:46.489253
  Lost Records     : 0               Flags: Pkt Ver2 Gre
  ------------------------------------------------------------------------------
GRE part is deleted here, to avoid the complexity of the trace
------------------------------------------------------------------------------
 IpHeader: Version : 4               Header Length: 20
  Tos              : 30              QOS: Priority MinimumDelay
  Packet Length    : 40              ID Number: 1592
  Fragment         : DontFragment    Offset: 0
  TTL              : 127             Protocol: TCP           CheckSum: 9A0E
  Source           : 9.67.156.104
  Destination      : 9.67.157.17

 TCP
  Source Port      : 4202  ()        Destination Port: 23    (telnet)
  Sequence Number  : 1118528900      Ack Number: 1627793505
  Header Length    : 20              Flags: Ack
  Window Size      : 17149           CheckSum: DEBF FFFF Urgent Data Pointer:

 IP Header          : 20     IP: 9.67.156.104, 9.67.157.17
 000000 45300028 15924000 7F069A0E 09439C68  09439D11
 Protocol Header    : 20     Port:  4202,    23
 000000 106A0017 42AB6584 61062861 501042FD  DEBF0000

 ------------------------------------------------------------------------------
End of  FIN trace
```

Figure 6-7 Trace record 506: ACK sent from client

Display target system, SD, and router after closing connection

There is no connection on MVS154.

```
===> netstat conn

MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP
User Id  Conn     Local Socket       Foreign Socket    State
-------  ----     ------------       --------------    -----
BPXOINIT 0000001D 0.0.0.0..10007     0.0.0.0..0        Listen
ENDPT40  0000003C 0.0.0.0..10115     0.0.0.0..0        Listen
FTPMVS1  00000023 0.0.0.0..21        0.0.0.0..0        Listen
HODSERV5 0000002B 0.0.0.0..8999      0.0.0.0..0        Listen
INETD1   0000004B 0.0.0.0..623       0.0.0.0..0        Listen
INETD1   00000044 0.0.0.0..19        0.0.0.0..0        Listen
OMPROUTE 00000022 127.0.0.1..1027    127.0.0.1..1028   Establsh
OSNMPD   00000021 0.0.0.0..1029      0.0.0.0..0        Listen
TCP      00000010 0.0.0.0..523       0.0.0.0..0        Listen
TCP      0000000B 127.0.0.1..1025    0.0.0.0..0        Listen
TCP      0000000E 127.0.0.1..1026    127.0.0.1..1025   Establsh
TCP      00000011 0.0.0.0..23        0.0.0.0..0        Listen
TCP      0000001C 127.0.0.1..1028    127.0.0.1..1027   Establsh
TCP      0000000F 127.0.0.1..1025    127.0.0.1..1026   Establsh
WEBSDB2  00000019 0.0.0.0..80        0.0.0.0..0        Listen
OSNMPD   00000020 0.0.0.0..161       *..*              UDP
```

No entry is in the VIPA connection routing table in the Sysplex Distributor of MVS001.

```
===> netstat vcrt

MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP
Dynamic VIPA Connection Routing Table:
Dest IPaddr      DPort  Src IPaddr       SPort  DestXCF Addr
-----------      -----  ----------       -----  ------------
***
```

The router has also cleared the fixed affinity table.

```
C7200-Z55>show ip casa aff
No matching entries in affinity cache
C7200-Z55>
```

6.9 Sysplex Distributor backup

When the primary TCP/IP stack running the Sysplex Distributor fails, the backup stack with a secondary Sysplex Distributor automatically takes over all defined DVIPAs, including the distribution definitions such as VIPADIST and DESTIP. Prerequisite for this takeover process is that VIPABackup is defined on the backup stack(s).

The stack with the highest defined rank automatically activates the DVIPA in a backup situation. Ranks may be defined from 1 (default) to 254; 1 is the lowest order in a backup chain.

6.9.1 TCPIP.PROFILE definitions

These definitions in Example 6-13 were used on MVS069.

Example 6-13 Sysplex Distributor DVIPA backup definitions

```
VIPADYNAMIC
;----------------------------------------------------------
; DISTRIBUTED VIPA BACKUP FOR WEB AND TN3270
;----------------------------------------------------------
   VIPABACKUP 200 9.67.157.17
;----------------------------------------------------------
;  DISTRIBUTED VIPA BACKUP FOR FTP
;----------------------------------------------------------
   VIPABACKUP 200 9.67.157.18
;
ENDVIPADYNAMIC
```

The backup stack will distribute new TCP connection requests based on the VIPADISTRIBUTE and DESTIP parameters of the original stack.

Additional IPCONFIG statements in the backup Sysplex Distributor have to be defined, as shown in the Example 6-14.

Example 6-14 IPCONFIG statements in the backup Sysplex Distributor

```
DYNAMICXCF  9.67.156.76 255.255.255.248
DATAGRAMFWD
SYSPLEXROUTING
```

```
VARSUBNETTING
PATHMTUDISCOVERY
```

6.9.2 Sysplex Distributor backup procedures

This section gives an overview of the following:

- Activities of the backup Sysplex Distributor in a backup sequence when the primary Sysplex Distributor fails.
- Activities of the primary Sysplex Distributor in a recovery sequence when the primary Sysplex Distributor is restarted.

Figure 6-8 shows our failure scenario.

Sysplex Distributor backup sequence

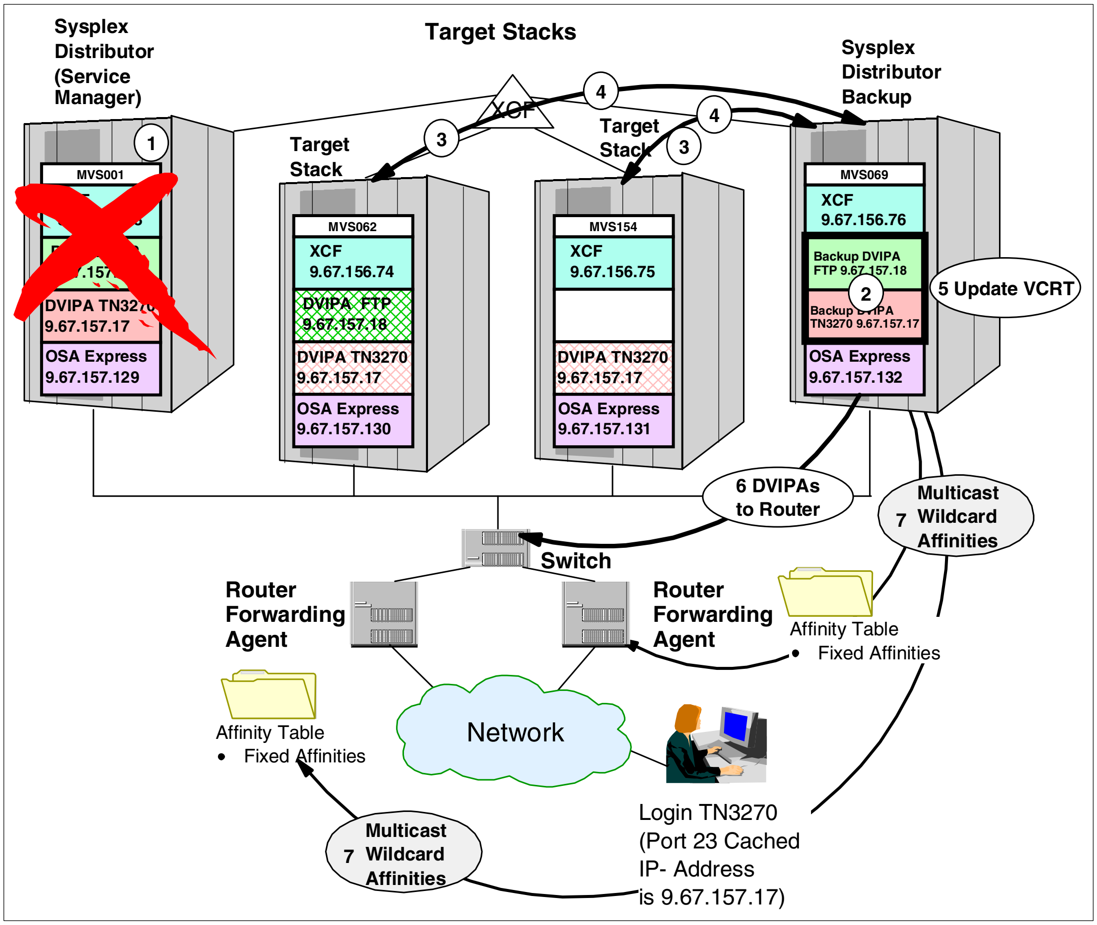


Figure 6-8 Sysplex Distributor takeover procedure

In our figure:

1. The Distribution Stack (MVS001) fails.
2. The backup Distribution Stack (MVS069) automatically activates DVIPAs for which it is backup.
3. The backup Distribution Stack (MVS069) informs target stacks (MVS062 and MVS154).

4. Target stacks (MVS062 and MVS154) inform the backup Distribution Stack (MVS069) about application server status on ports defined for distribution.
5. The backup Distribution Stack (MVS069) builds and maintains its destination port table (DPT) and its current routing table (CRT).
6. The backup Distribution Stack (MVS069) advertises to the network DVIPAs that it took over.
7. Wildcard affinities are multicasted to the Forwarding Agents propagating the new forward IP address, which is now the dynamic XCF link address 9.67.156.76 of the backup Sysplex Distributor (MVS069).

   Fixed affinities need not to be updated in the Forwarding Agents.

Backup SD MVS069 displays before the primary SD MVS001 failed

The following display shows that the backup Sysplex Distributor has not activated yet the DVIPAs 9.67.156.17 and 9.67.156.18 of the primary Sysplex Distributor.

```
===> netstat vipadcfg
MVS TCP/IP NETSTAT CS V1R2    TCPIP NAME: TCP 17:39:21
Dynamic VIPA Information:
  VIPA Backup:
    IP Address       Rank
    ----------       ----
    9.67.156.25      000100
    9.67.156.26      000100
    9.67.156.33      000010
    9.67.156.34      000010
    9.67.156.41      000010
    9.67.156.42      000010

  VIPA Define:
    IP Address       AddressMask      Moveable  SrvMgr
    ----------       -----------      --------  ------
    9.67.156.49      255.255.255.248  Immediate No
    9.67.156.50      255.255.255.248  Immediate No
***
```

The display of the VIPA configuration table of the primary Sysplex Distributor MVS001 shows that this stack is still Service Manager.

```
===> netstat vipadcfg

MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP
Dynamic VIPA Information:
 VIPA Backup:
   IP Address       Rank
   ----------       ----
   9.67.156.33      000001
   9.67.156.34      000001
   9.67.156.41      000001
   9.67.156.42      000001
   9.67.156.49      000100
   9.67.156.50      000100

 VIPA Define:
   IP Address       AddressMask      Moveable  SrvMgr
   ----------       -----------      --------  ------
   9.67.156.25      255.255.255.248  Immediate No
   9.67.156.26      255.255.255.248  Immediate No
   9.67.157.17      255.255.255.248  Immediate Yes
   9.67.157.18      255.255.255.248  Immediate Yes

VIPA Distribute:
   IP Address       Port   XCF Address
   ----------       ----   -----------
   9.67.157.17      00023  9.67.156.75
   9.67.157.17      00023  9.67.156.74
   9.67.157.17      00080  9.67.156.75
   9.67.157.17      00080  9.67.156.74
   9.67.157.17      00443  9.67.156.75
   9.67.157.17      00443  9.67.156.74
   9.67.157.17      00523  9.67.156.75
   9.67.157.17      00523  9.67.156.74
   9.67.157.18      00020  9.67.156.74
   9.67.157.18      00020  9.67.156.73
   9.67.157.18      00021  9.67.156.74
   9.67.157.18      00021  9.67.156.73

  VIPA Service Manager:
   McastGroup: 224.0.1.2        Port: 01637  Pwd: No
```

The following screen shows that the Sysplex Distributor MVS001 currently has two TCP connections.

```
MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP
Dynamic VIPA Connection Routing Table:
Dest IPaddr      DPort  Src IPaddr       SPort  DestXCF Addr
-----------      -----  ----------       -----  ------------
9.67.157.17      00023  9.67.156.104     04219  9.67.156.74
9.67.157.17      00023  9.67.156.104     04220  9.67.156.75
```

Because the backup Sysplex Distributor is not the current Sysplex Distributor, it does not have a distributed VIPA port table as shown in the following screen.

```
===> netstat vdpt

MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP
Dynamic VIPA Distribution Port Table:
Dest IPaddr     DPort DestXCF Addr    Rdy TotalConn  WLM
----------      ----- ------------    --- ---------  ---
***
```

The backup Sysplex Distributor does not have any TCP connections. The next screen shows output of the dynamic VIPA connection routing table of MVS06.

```
===> netstat vcrt

MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP
Dynamic VIPA Connection Routing Table:
Dest IPaddr      DPort  Src IPaddr       SPort  DestXCF Addr
-----------      -----  ----------       -----  ------------
***
```

The command **`netstat vipadyn`** issued at the backup Sysplex Distributor MVS069 also does not show any distributed DVIPAs. The screen output has no distributed VIPAs 9.67.157.17 and 9.67.156.18.

```
MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP
IP Address      AddressMask     Status    Origination  DistStat
----------      -----------     ------    -----------  --------
9.67.156.25     255.255.255.248 Backup    VIPABackup
9.67.156.26     255.255.255.248 Backup    VIPABackup
9.67.156.33     255.255.255.248 Backup    VIPABackup
9.67.156.34     255.255.255.248 Backup    VIPABackup
9.67.156.41     255.255.255.248 Backup    VIPABackup
9.67.156.42     255.255.255.248 Backup    VIPABackup
9.67.156.49     255.255.255.248 Active    VIPADefine
9.67.156.50     255.255.255.248 Active    VIPADefine
9.67.157.17     255.255.255.248 Backup    VIPABackup
9.67.157.18     255.255.255.248 Backup    VIPABackup
***
```

Primary Sysplex Distributor fails

The backup Sysplex Distributor starts the takeover phase. It immediately activates the DVIPAs that are defined as VIPABACKUP in the TCPIP.PROFILE. See the next screen output, obtained using the **`netstat vipadyn`** command. The distributed DVIPAs 9.76.157.17 and 9.67.157.18 are taken over as VIPABACKUP DISTributed.

```
MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP
IP Address      AddressMask     Status    Origination     DistStat
----------      -----------     ------    -----------     --------
9.67.156.25     255.255.255.248 Active    VIPABackup
9.67.156.26     255.255.255.248 Active    VIPABackup
9.67.156.33     255.255.255.248 Backup    VIPABackup
9.67.156.34     255.255.255.248 Backup    VIPABackup
9.67.156.41     255.255.255.248 Backup    VIPABackup
9.67.156.42     255.255.255.248 Backup    VIPABackup
9.67.156.49     255.255.255.248 Active    VIPADefine
9.67.156.50     255.255.255.248 Active    VIPADefine
9.67.157.17     255.255.255.248 Active    VIPABackup      Dist
9.67.157.18     255.255.255.248 Active    VIPABackup      Dist
***
```

The command **netstat vipadcfg** command issued at the backup Sysplex Distributor shows the following result.

Now the backup Sysplex Distributor is responsible for the taken-over distributed DVIPAs. All ports and the dynamic XCF IP addresses of the available target stacks are displayed.

```
===> netstat vipadcfg
MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP          17:49:26
Dynamic VIPA Information:
  VIPA Backup:
    IP Address       Rank
    ----------       ----
    9.67.156.33      000010
    9.67.156.34      000010
    ..........       ......
  VIPA Define:
    IP Address       AddressMask      Moveable  SrvMgr
    ----------       -----------      --------  ------
    9.67.156.49      255.255.255.248  Immediate No
    9.67.156.50      255.255.255.248  Immediate No
    9.67.157.17      255.255.255.248  Immediate Yes
    9.67.157.18      255.255.255.248  Immediate Yes
  VIPA Distribute:
    IP Address       Port   XCF Address
    ----------       ----   -----------
    9.67.157.17      00023  9.67.156.75
    9.67.157.17      00023  9.67.156.74
    ..........       .....  ...........
    9.67.157.18      00020  9.67.156.74
    9.67.157.18      00020  9.67.156.73
    9.67.157.18      00021  9.67.156.74
    9.67.157.18      00021  9.67.156.73
   VIPA Service Manager:
     McastGroup: 224.0.1.2        Port: 01637  Pwd: No
```

Backup SD informs the target stacks about DVIPAs and ports

The port table obtained with the **netstat vdpt** command is filled now.

```
MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP
Dynamic VIPA Distribution Port Table:
Dest IPaddr     DPort DestXCF Addr    Rdy TotalConn  WLM
-----------     ----- ------------    --- ---------  ---
9.67.157.17     00023 9.67.156.74     001 0000000000 01
9.67.157.17     00023 9.67.156.75     001 0000000000 01
9.67.157.17     00080 9.67.156.74     001 0000000000 01
9.67.157.17     00080 9.67.156.75     001 0000000000 01
9.67.157.17     00443 9.67.156.74     001 0000000000 01
9.67.157.17     00443 9.67.156.75     000 0000000000 01
9.67.157.17     00523 9.67.156.74     001 0000000000 01
9.67.157.17     00523 9.67.156.75     001 0000000000 01
9.67.157.18     00020 9.67.156.74     000 0000000000 01
9.67.157.18     00021 9.67.156.74     001 0000000000 01
```

In the meantime, some more connections were established.

Target stacks provided connection information

```
MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP
Dynamic VIPA Connection Routing Table:
Dest IPaddr      DPort  Src IPaddr       SPort  DestXCF Addr
-----------      -----  ----------       -----  ------------
9.67.157.17      00023  9.67.156.104     04623  9.67.156.74
9.67.157.17      00023  9.67.156.104     04632  9.67.156.74
9.67.157.17      00023  9.67.156.104     04622  9.67.156.75
9.67.157.17      00023  9.67.156.104     04631  9.67.156.75
***
```

Backup SD stack propagates to the network

A router display obtained through the **`show ip casa wildcard det`** command shows the wildcard affinity table contents. It shows the new interest address of the backup Sysplex Distributor. This address is 67.156.76. All packets, even fragmented packets, are accepted by the Sysplex Distributor. The interest address is the dynamic XCF link IP address of the backup Sysplex Distributor that was propagated to the network via a multicast CASA packet on behalf of the takeover process. This means that, from now on, the Forwarding Agents have to send CASA requests per unicast to the dynamic XCF link IP address 9.67.156.76 and port 1637. See the highlighted lines in the following display. A dispatch address is not available for wildcard affinities.

Do not be confused by the Service Manager's IP address 9.67.156.67. This is the IP address of the CLAW interface.

```
Source Address  Source Mask Port  Dest Address  Dest Mask       Port  Prot
0.0.0.0         0.0.0.0     0     9.67.157.17   255.255.255.255 523   TCP
 Service Manager Details:
   Manager Addr:      9.67.156.67      Insert Time: 11:36:27 UTC 07/27/01
 Affinity Statistics:
   Affinity Count:        0                Interest Packet Timeouts: 0
 Packet Statistics:
   Packets:               0                Bytes: 0
Advertise Dest Address: NO                 Match Fragments:  YES
Affinity TTL: 30
 Action Details:
   Interest Addr:      9.67.156.76       Interest Port: 1637
   Interest Packet: 0x8100 FRAG ALLPKTS
   Interest Tickle: 0x0000
   Dispatch (Layer 2):     NO              Dispatch Address: 0.0.0.0
Source Address  Source Mask  Port  Dest Address Dest Mask       Port  Prot
0.0.0.0         0.0.0.0      0     9.67.157.17  255.255.255.255 443   TCP
 Service Manager Details:
   Manager Addr:      9.67.156.67       Insert Time: 11:36:27 UTC 07/27/01
 Affinity Statistics:
   Affinity Count:        0            Interest Packet Timeouts: 0
 Packet Statistics:
   Packets:               0                Bytes: 0
Advertise Dest Address: NO                 Match Fragments:  YES
Affinity TTL: 30
 Action Details:
   Interest Addr:      9.67.156.76       Interest Port: 1637
   Interest Packet: 0x8100 FRAG ALLPKTS
   Interest Tickle: 0x0000
   Dispatch (Layer 2):     NO              Dispatch Address: 0.0.0.0

Source Address  Source Mask  Port  Dest Address  Dest Mask       Port  Prot
0.0.0.0         0.0.0.0      0     9.67.157.17   255.255.255.255 80    TCP
 Service Manager Details:
   Manager Addr:      9.67.156.67       Insert Time: 11:36:27 UTC 07/27/01
 Affinity Statistics:
   Affinity Count:        0            Interest Packet Timeouts: 0
Packets:                0              Bytes: 0
Advertise Dest Address: NO                 Match Fragments:  YES
Affinity TTL: 30
 Action Details:
   Interest Addr:      9.67.156.76       Interest Port: 1637
    Interest Packet: 0x8100 FRAG ALLPKTS
...........
```

On router 7206, nothing has been changed concerning existing connections. The four TCP connections may be viewed by issuing the command **`show ip affinities`**. This is the short form for the display.

```
Source Address  Port  Dest Address    Port  Prot
9.67.156.104    4622  9.67.157.17     23    TCP
9.67.156.104    4623  9.67.157.17     23    TCP
9.67.156.104    4632  9.67.157.17     23    TCP
9.67.156.104    4631  9.67.157.17     23    TCP
C7200-Z55>
```

The command **`show ip affinities det`** will let you see that the backup Sysplex Distributor now maintains this connection. This is indicated through the interest address. See the highlighted line. Note, the following display is an excerpt only.

```
C7200-Z55> show ip casa aff det
Source Address  Port  Dest Address    Port  Prot
9.67.156.104    4622  9.67.157.17     23    TCP
Action Details:
 Interest Addr:       9.67.156.76  Interest Port: 1637
 Interest Packet: 0x0002 SYN
Interest Tickle: 0x0000
Dispatch (Layer 2):     YES        Dispatch Address: 9.67.156.75
Source Address  Port  Dest Address    Port  Prot
9.67.156.104    4623  9.67.157.17     23    TCP
Action Details:
 Interest Addr:       9.67.156.76  Interest Port: 1637
Interest Packet: 0x0002 SYN
Interest Tickle: 0x0000
Dispatch (Layer 2):     YES        Dispatch Address: 9.67.156.74
```

The takeover process also deposits footprints in the z/OS console log. See the following display excerpt of the system MVS069, the backup Sysplex Distributor.

```
EZZ8301I VIPA 9.67.156.25 TAKEN OVER FROM TCP ON MVS001
EZZ8301I VIPA 9.67.156.26 TAKEN OVER FROM TCP ON MVS001
EZZ8301I VIPA 9.67.157.17 TAKEN OVER FROM TCP ON MVS001
EZZ8301I VIPA 9.67.157.18 TAKEN OVER FROM TCP ON MVS001
```

Sysplex Distributor recovery sequence

When the primary Sysplex Distributor is restarted, the recovery process defined for the DVIPAs will take back the resources depending on the VIPADEFINE parameter, MOVEABLE IMMEDIATE or MOVEABLE WHENIDLE.

MOVEABLE IMMEDIATE will enable the primary Sysplex Distributor to take back immediately the current TCP connection maintenance previously taken over from the backup Sysplex Distributor. This takeback happens without interruption of the TCP connection between the client and the application running on a target system. After the takeback phase, the connections are no longer viewed in the backup Sysplex Distributor's distributed VIPA connection routing table. They are now maintained in the recovered Sysplex Distributor.

All new connection requests are directed to the recovered Sysplex Distributor.

MOVEABLE WHENIDLE indicates that this DVIPA can be moved to another stack, when there are no connections for this DVIPA on the current stack. While there are existing connections, any new connection request continues to be directed to the current stack.

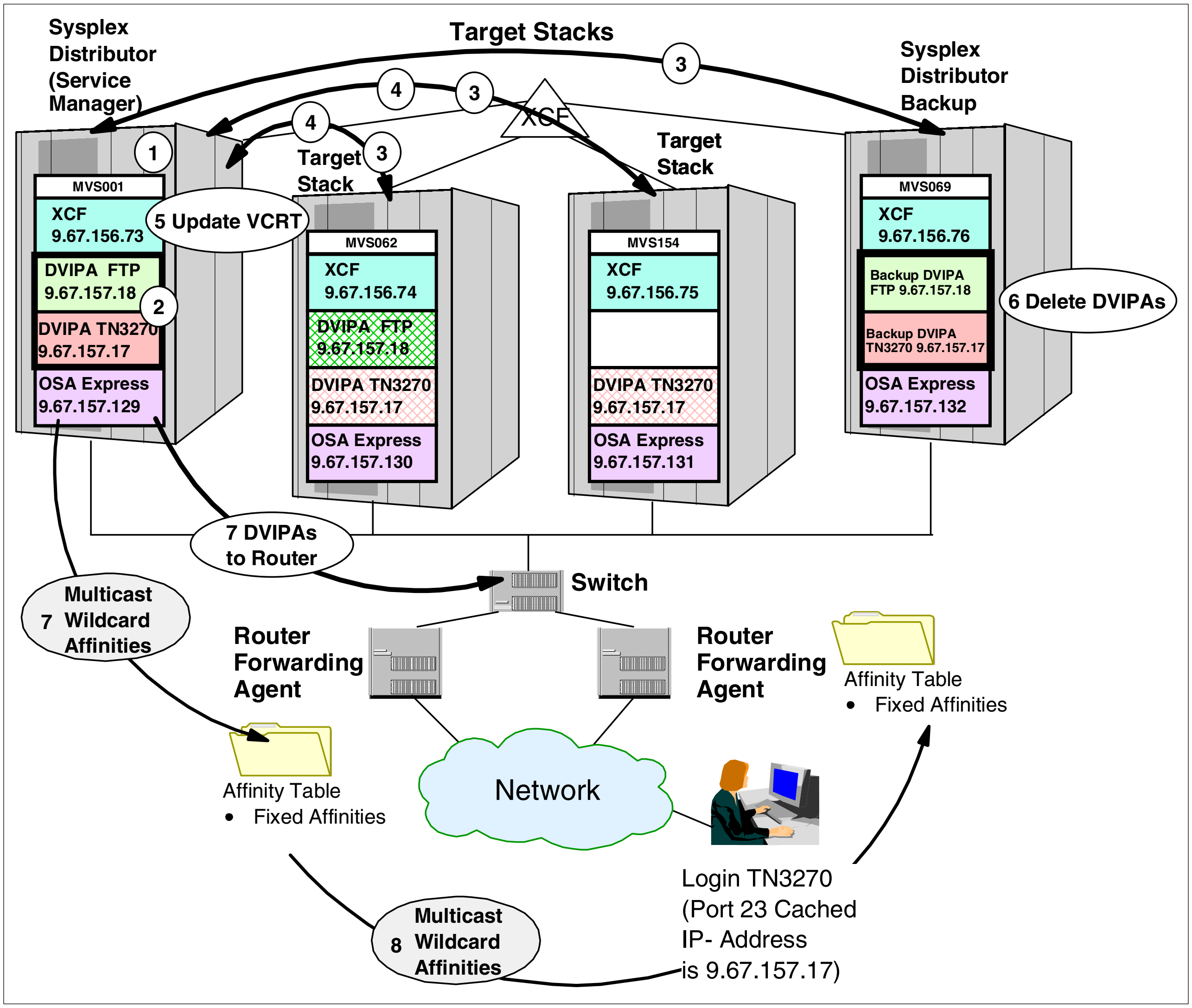


Figure 6-9 Sysplex Distributor recovery sequence

Figure 6-9 shows a recovery sequence:

1. The Distribution Stack (MVS001) is restarted.
2. The Distribution Stack (MVS001) activates DVIPAs.
3. The Distribution Stack (MVS001) informs the backup Distribution Stack (MVS069) and the target stacks (MVS062 and MVS0154).
4. The target stacks (MVS062 and MVS0154) inform the Distribution Stack (MVS001) about application server status on ports defined for distribution.
5. The backup Distribution Stack (MVS069) sends a table to the distribution stack (MVS001) containing all connections currently routed.
6. The backup Distribution Stack (MVS069) cleans up its tables and deletes the DVIPAs given back.
7. The Distribution Stack (MVS001) advertises DVIPAs to the network.

> **Note:** This procedure only works if DVIPAs in MVS001 are defined as MOVEable IMMEDiate (the default) on the VIPADefine statement.

Display documentation of the recovery sequence

The z/OS console of MVS069, the backup Sysplex Distributor, indicates the DVIPAs are returned to the originator MVS001.

```
EZZ8303I VIPA 9.67.156.25 GIVEN TO TCP ON MVS001
EZZ8303I VIPA 9.67.156.26 GIVEN TO TCP ON MVS001
EZZ8303I VIPA 9.67.157.17 GIVEN TO TCP ON MVS001
EZZ8303I VIPA 9.67.157.18 GIVEN TO TCP ON MVS001
```

The z/OS console of MVS001, the primary Sysplex Distributor, indicates that the DVIPAs returned.

```
EZZ8302I VIPA 9.67.156.25 TAKEN FROM TCP ON MVS069
EZZ8302I VIPA 9.67.156.26 TAKEN FROM TCP ON MVS069
EZZ8302I VIPA 9.67.157.17 TAKEN FROM TCP ON MVS069
EZZ8302I VIPA 9.67.157.18 TAKEN FROM TCP ON MVS069
```

DVIPAs are reactivated

DVIPAs are activated as shown in “Backup SD MVS069 displays before the primary SD MVS001 failed” on page 222.

Target stacks provide connection state information

The recovered Sysplex Distributor takes back the maintenance of the four TCP connections.

```
===> netstat vcrt
MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP
Dynamic VIPA Connection Routing Table:
Dest IPaddr      DPort  Src IPaddr       SPort  DestXCF Addr
-----------      -----  ----------       -----  ------------
9.67.157.17      00023  9.67.156.104     04623  9.67.156.74
9.67.157.17      00023  9.67.156.104     04632  9.67.156.74
9.67.157.17      00023  9.67.156.104     04622  9.67.156.75
9.67.157.17      00023  9.67.156.104     04631  9.67.156.75
***
```

Backup Sysplex Distributor stack cleans up port table

No ports are available because the backup stack is no longer the Service Manager. See the display in “Backup SD MVS069 displays before the primary SD MVS001 failed” on page 222.

```
===> netstat vdpt

MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP
Dynamic VIPA Distribution Port Table:
Dest IPaddr     DPort DestXCF Addr    Rdy TotalConn  WLM
-----------     ----- ------------    --- ---------  ---
***
```

Backup Sysplex Distributor stack cleans up the VIPA connection routing table

```
===> netstat vcrt

MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCP
Dynamic VIPA Connection Routing Table:
Dest IPaddr      DPort  Src IPaddr       SPort  DestXCF Addr
-----------      -----  ----------       -----  ------------
***
```

Recovered SD stack advertises DVIPAs to the network

Fixed affinities are still viewable at the Forwarding Agent. In the detailed display, the new interest address is shown.

```
C7200-Z55>show ip casa aff
Source Address  Port  Dest Address    Port  Prot
9.67.156.104    4622  9.67.157.17     23    TCP
9.67.156.104    4623  9.67.157.17     23    TCP
9.67.156.104    4632  9.67.157.17     23    TCP
9.67.156.104    4631  9.67.157.17     23    TCP
C7200-Z55>
```

The interest IP address of the recovered Sysplex Distributor is highlighted.

```
C7200-Z55>show ip casa aff det
Source Address  Port  Dest Address    Port  Prot
9.67.156.104    4631  9.67.157.17     23    TCP
 Action Details:
   Interest Addr:         9.67.156.73  Interest Port: 1637
   Interest Packet: 0x0002 SYN
   Interest Tickle: 0x0000
   Dispatch (Layer 2):    YES          Dispatch Address: 9.67.156.75
```

6.10 Generic Routing Encapsulation (GRE) protocol

There may be situations where IP packets have to be encapsulated with an additional IP header before they are sent over the network. This situation is applicable, for example, when there are multiple logical partitions (LPARs) in a z/OS sysplex environment, each running a TCP/IP stack and sharing one OSA-Express adapter, and each TCP/IP stack has defined distributed dynamic virtual IP addresses (DVIPAs). These DVIPAs are defined in the Sysplex Distributor, and in the backup Sysplex Distributor, for target stacks running various application services.

6.10.1 The need for GRE

Such a situation arises when multiple z/OS LPARs, each running a TCP/IP stack, share one OSA-Express adapte, and a distributed dynamic virtual IP (DVIPA). An MNLB Forwarding Agent distributes IP packets from clients of TCP connections to target application servers. For each connection, a specific target server was selected by the Sysplex Distributor from a group of servers within the sysplex based on load-balancing rules.

Remember, a route between the client and the server located in a z/OS sysplex will have a destination IP address that is a distributed dynamic VIPA (DVIPA). The distributed DVIPA is the cluster address representing the application server. The distributed DVIPA is dynamically defined by the Sysplex Distributor also in other target stacks for load-balancing purposes. Thus, there are multiple equal distributed DVIPAs known in TCP/IP stacks in a sysplex.

The following display is taken from the target system MVS062. It shows the dynamically defined DVIPAs. For example, the IP address 9.67.157.17 for TN3270 and Web services also appears in the HOME list of the application hosts in the target systems MVS062 and MVS154.

```
=> netstat home
 MVS TCP/IP NETSTAT CS V1R2 TCPIP NAME: TCP
 Home address list:
 Address          Link             Flg
-------          ----             ---
9.67.156.161     VLINK0           P
9.67.157.130     GIGELINK
9.67.156.69      CISCO1
9.67.156.18      CISCO2
9.67.157.243     CISCO3
9.67.156.162     TOVTAM
9.67.156.74      EZAXCFN6
9.67.156.33      VIPL09439C21
9.67.156.34      VIPL09439C22
9.67.156.74      EZAXCFN7
9.67.156.74      EZAXCFN4
9.67.157.17      VIPL09439D11     I
9.67.157.18      VIPL09439D12     I
127.0.0.1        LOOPBACK
```

A problem arises when a destination IP address, representing this DVIPA, has to be routed by the OSA-Express GbE adapter to the real application server's IP address. Although the distributed DVIPAs, defined in the Sysplex Distributor with VIPADIST and DESTIP, also appear in the HOME list of all target stacks, they are "hidden" from the network.

What does "hidden" mean? Distributed DVIPAs are not propagated by a router daemon and, most importantly, they are not downloaded into the OSA-Express adapter's address table (OAT). Thus they cannot be mapped with the 48-bit ISO/OSI layer-2 medium access control (MAC) address.

In our test environment, all LPARs use the OSA-Express adapter in a shared mode. Thus the OSA-Express adapter knows the distributed DVIPAs, but is owned by the Sysplex Distributor stack only. In our test case the Sysplex Distributor would be the receiver for packets with cluster addresses 9.67.157.17, and 9.67.157.18. This means that, since all packets carrying the cluster address for existing connections would always be sent to the Sysplex Distributor and not the destination address of the real target server, there has to be some additional work by the Forwarding Agent to address the real target server rather than the Sysplex Distributor.

Before we try to explain how the target addressing problem will be solved, we first review how the OAT is created in the OSA-Express adapter. It differs completely from the procedure you might follow when an OSA-2 OAT is defined.

When a START DEVICE command is executed by the TCP/IP stack for an OSA-Express adapter that runs in MPCIPA mode (also known as *IP assist mode*), information is passed from the TCP/IP stack to the adapter, including:

- HCD definitions (channel path ID, control unit, device address, and operation mode (shr)).
- TRLE definitions with a subchannel address to access the LPAR (read, write, data path, and port name that maps the device name of the DEVICE statement guiding to the LINK name).
- DEVICE and LINK statement information.
- HOME statements, which list all IP addresses with associated link names known to the stack.

A sample of the OAT shows which IP addresses are downloaded from the TCP/IP stacks. You will discover the differences in the entries for:

- MVS001 as the Sysplex Distributor stack
- MVS062 as one of the target stacks
- MVS069 as the backup Sysplex Distributor

The backup Sysplex Distributor will receive entries for the VIPABACKUP addresses when the backup occurs. The entries for the other target stack MVS154 are not shown because they are similar to MVS062.

We have edited the samples in Example 6-15 through Example 6-17 on page 234 for a better understanding. Our additional text is shown in *italics*.

The OAT entries are also not shown in the real sequence. The real sequence is sorted by the subchannel addresses of the read, write, and data path addresses defined in the TRLE.

OAT for MVS001 (Sysplex Distributor)

Example 6-15 OSA-Express adapter address table (OAT) of Sysplex Distributor

```
THERE IS DATA FOR 55 OAT(s) ON CHPID F5
START OF OSA ADDRESS TABLE
  UA(Dev) Mode     Port     Entry specific information      Entry  Valid
                             LP 01 (RALVM9)  is the VM system which carries the four
z/OS LPARs
                                                       MVS001, MVS069, MVS062, MVS154
MVS001 Sysplex Distributor (node name: N04)
14(2F14) MPC        n/a     N04GIG1    (QDIO control)         SIU    ALL
15(2F15) MPC        n/a     N04GIG1    (QDIO control)         SIU    ALL
16(2F16) MPC        00   PRI N04GIG1    (QDIO data)        SIU    ALL
                            9.67.156.17    -> MPC CISCO2  MVS001
                            9.67.156.66    -> CLAW CISCO1
                            9.67.156.2     ->  TOVTAM
                            9.67.156.73    ->  DXCF
                            9.67.157.129   ->  GIGELINK
                            9.67.156.1     ->  static VIPA
                            9.67.156.25    ->  DVIPA
                            9.67.156.26    ->  DVIPA
                            9.67.157.17    ->  distributed DVIPA
                            9.67.157.18    ->  distributed DVIPA
```

This display shows that z/OS subchannel address 2F14 and 2F15 are the read and write subchannel addresses for the QDIO control data. The real data transmission goes over the channel path 2F16, defined in the TRLE. This line also shows that the path to MVS001 is defined as the primary router (PRI). Compare the equivalent line for the backup Sysplex

Distributor. There the path is defined as the secondary router (SEC). You will also not find the distributed DVIPAs 9.67.157.18 and 9.67.157.19 in the backup Sysplex Distributor, since they are not displayed after using the **netstat home** command. Remember, the distributed DVIPAs are activated in a backup case only.

The name between the router specification and the QDIO issues represents the OSA name. It is the name of the TRLE entry. For example, N04GIG1 is the OAT entry that points to the TRLE name (see "OSA-Express adapter interfaces" on page 186).

When you look at the subchannel addresses of all LPARs, you will notice that they are unique. This is because the OSA-Express adapter is defined to one system only. This system is a VM system, called RALVM9. The four z/OS systems are guest machines running under VM.

In a pure z/OS environment, without VM, you have four independent LPARs. This allows you to define three equal subchannel addresses for all LPARs. For example:

2F14 for the read control subchannel
2F15 for the write control subchannel
2F16 for the data path

If the port name of the TRLE maps the device name in the TCPIP.PROFILE, then one TRLE may be defined, but copied to the VTAMLST of the four different systems.

OAT for MVS062 (target system)

Example 6-16 OSA-Express adapter address table (OAT) of target stack MVS062

```
MVS062 Target system (node name: N05)
04(2F04) MPC        n/a     N05GIG1    (QDIO control)        SIU    ALL
05(2F05) MPC        n/a     N05GIG1    (QDIO control)        SIU    ALL
06(2F06) MPC        00  No  N05GIG1    (QDIO data)         SIU    ALL
                            9.67.156.18    -> MPC CISCO2 MVS062
                            9.67.157.243   -> CLAW CISCO3
                            9.67.156.162   ->  TOVTAM
                            9.67.156.74     -> XCF
                            9.67.157.130   ->  GIGELINK
                            9.67.156.161   ->  static VIPA
                            9.67.156.33    ->  DVIPA
                            9.67.156.34  ->  DVIPA
```

OAT for MVS069 (backup Sysplex Distributor)

Example 6-17 OSA-Express adapter address table (OAT) of backup Sysplex Distributor

```
MVS069 Backup Sysplex Distributor (node name: N07)
18(2F18) MPC        n/a     N07GIG1    (QDIO control)        SIU    ALL
19(2F19) MPC        n/a     N07GIG1    (QDIO control)        SIU    ALL
1A(2F1A) MPC        00  SEC N07GIG1    (QDIO data)       SIU    ALL
                            9.67.156.20     -> MPC CISCO2  MVS069
                            9.67.157.245   -> CLAW CISCO3
                            9.67.156.5      -> TOVTAM
                            9.67.156.76     -> DXCF
                            9.67.157.132   -> GIGELINK
                            9.67.156.5     ->  static VIPA
                            9.67.156.49   ->  DVIPA
                            9.67.156.50    ->  DVIPA
--------------------------------------------------------------------------------
***                     Legend for abbreviations              ***
* Entry column             Valid column
* ------------             ------------
* S   - Started           OSA - Does not exist in IOCDS, but is on OSA
* NS  - Not started       CSS - Exists only in Channel Subsystem (IOCDS)
* SIU - Started & in use   ALL - Exists on the OSA and in IOCDS
* N/A - Not Applicable
* R   - Rejected (see messages following this legend)
* Entry Specific Information
* -----------------------
* Passthru           - Default entry, Home IP address & Netmask
* SNA                - VTAM IDNUM (if port number is FF)
* MPC (IP or IPX)    - OSA name
* MPC (QDIO Control) - OSA name
* MPC (QDIO Data)    - Default entry & OSA name

CHPID F5 RETURNED THE FOLLOWING OAT REASON MESSAGES
  None
****************************  End of Query data  ************************
```

IP assist uses the downloaded information to create the OSA Address Table (OAT). This enables the OSA adapter to recognize IP addresses that are mapped to the MAC address of the adapter.

IP assist will also use the referenced information such as all IP addresses of other TCP/IP stack's HOME lists if the OSA-Express GbE adapter runs in a shared mode.

Now, we return to the addressing situation initiated by the fact that the destination IP for all TCP connection data to the server is the cluster address. The only choice for the OSA-Express could be to send the packet to the Sysplex Distributor, which would check the 5-tuple information and by scanning the VIPA connection routing table (VCRT) would determine to which TCP connection this packet belongs, and forward it to the correct target system. But this would end being the non-MNLB Sysplex Distributor solution. It is not the preferred approach for a sysplex environment with MNLB. Therefore, another solution has to be made available to overcome the shared OSA adapter, distributed DVIPA problem.

6.10.2 Search for a shared OSA-Express solution

One approach could be to encapsulate the IP packet received by the Forwarding Agent into another IP packet with its destination IP address of the real target system. The true destination address is known by the Forwarding Agent for every IP packet associated with the TCP connection. The target IP address was provided by the Sysplex Distributor’s Service Manager through a fixed affinity at connection establishment through the CASA packet carrying the SYN sent to the Service Manager.

The protocol of encapsulating the original IP packet into a delivery IP packet is called the Generic Routing Encapsulation (GRE) Protocol. This protocol is used for routing IP packets from the Forwarding Agent to the correct target system in a z/OS sysplex with multiple LPARs sharing one OSA-Express GbE adapter. Without using GRE, a packet arriving at the OSA adapter would be forwarded to a default mapped IP address, which might not be the correct target system.

Note: In our case, GRE is required only for inbound packets from the Forwarding Agent to the target system. The path back to the client doesn't need GRE, since the route to the client can be determined correctly because the destination address is the unique client.

6.10.3 Generic Routing Encapsulation (GRE) overview

A general description of GRE is found in RFC 1701. RFC 1702 is about using GRE over IPv4 networks.

GRE is used in cases where a system has a packet that needs to be encapsulated and delivered to some destination. In our case, it is the MNLB Forwarding Agent that encapsulates and delivers the packet to the real target system.

The original packet received by the Forwarding Agent is called a payload packet. The packet around the payload packet is called the delivery packet.

Supported payload protocols

GRE was designed to encapsulate any protocol of the many networking protocols known as the Ethernet protocol types shown in Table 6-1.

Table 6-1 List of supported protocol types (excerpt only)

Protocol Family	Protocol Type
SNA	0004
OSI network layer	0FE
XNS	0600
IP	0800
RFC 826 ARP	806
Frame Relay ARP	808
VINES	0BAD
DECnet (Phase IV)	6003
Ethertalk (Appletalk)	809B
Novell IPX	8137
RFC 1144 TCP/IP compression	876B
Secure Data	876D

Overall packet structure

An additional header, the GRE header, has to be added to let the receiver of the delivery packet know what Ethernet type of protocol is encapsulated. The structure of the entire encapsulated packet is shown in Figure 6-10.

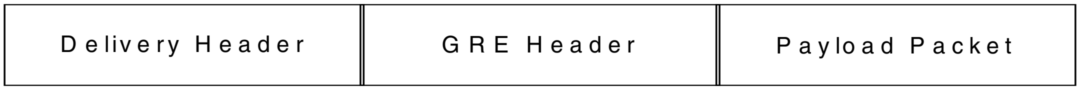


Figure 6-10 Overall packet structure

The delivery header consists of the 20 bytes IPv4 header. The length of the GRE header is dependent on the functions used. For our tests we used protocol type IP (0x0800) only. The length of the payload packet depends on the data to be transmitted.

GRE header structure

Figure 6-11 on page 237 shows the structure of the GRE header.

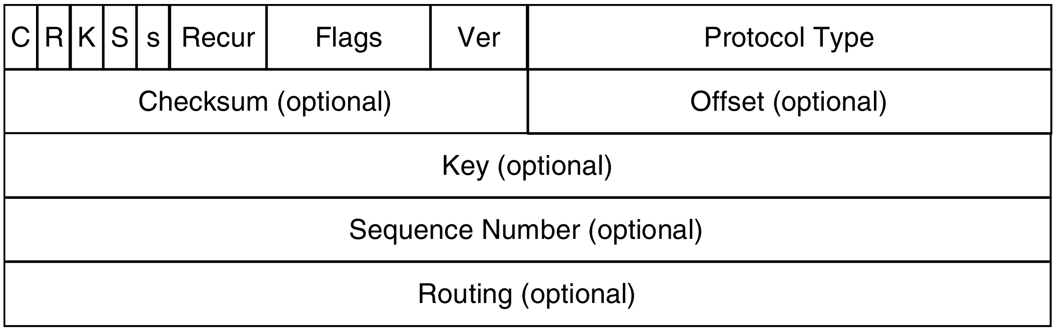


Figure 6-11 GRE header structure

The descriptions of the GRE fields may be obtained from RFC 1701. The length of the GRE depends on the functions used by switching on the flag bits. Functions are:

Checksum Calculated checksum of the GRE and payload packet

Key Used for authentication of the sender

Sequence number To control the sequence of incoming GRE and payload packets

Routing To determine a source route path

Payload packet structure

The payload packet structure is the usual IP packet containing the IPv4 header, TCP or UDP header, and data. In our case, we used the TCP header only, since the Service Manager in the Sysplex Distributor supports the TCP protocol only.

Forwarding and processing of GRE packets

The MNLB Forwarding Agent, in our test case implemented in Cisco routers 7507 and 7206VXR, implements RFC 1701 and RFC 1702 to encapsulate IP packets received from the client and forwards an IPv4 delivery packet, if the path goes outbound over the Gigabit Ethernet interfaces. This means that GRE tunnels have to be defined for the GbE interfaces in both routers. The GRE tunnels have to be defined in the Cisco routers for outbound traffic to the sysplex only. Within the sysplex there are no tunnel definitions required on any TCP/IP stack, either in the Sysplex Distributor or in the target stacks. For definitions in the Cisco routers, see 6.10.4, “Definitions in the Cisco routers 7507 and 7206” on page 240.

The payload packet has an Ethernet type 0x'0800'. In our case it is the IP packet received by the Forwarding Agent, sent from the client. So GRE will be used to encapsulate an IP packet by another IP packet according to RFC 1702.

When a GRE packet is received by a system, such as our z/OS system, it must recognize that an encapsulated IP packet has been received. z/OS V1.2 supports GRE and is thus able to unpack the delivery packet.

Once the delivery packet is recognized, the GRE contents is processed. For example:

- If the C-bit is set on, a checksum is recalculated and compared.
- If the R-bit is set on, the routing fields and the offset are available.
- If the K-bit is set on, the authentication key of the sender is checked.
- If the S-bit is set on, a sequence number field is present and may be checked with previous sequence numbers.
- If the s-bit is set on, a strict source routing path is defined.

Further information may be obtained from the RFCs 1701 and 1702.

As you will see later, all referenced bits in the GRE header were not switched on by the Forwarding Agent in our test case.

What destination IP address in the delivery header?

This is a question that has to be answered before defining the tunnel in the two Cisco routers. The tunnel end determines the destination IP address for the delivery header.

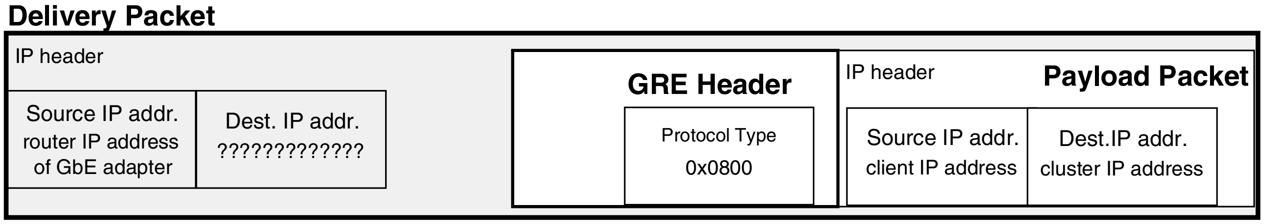


Figure 6-12 What destination IP address for the delivery packet?

Because the description in the current documentation does not indicate what IP address for the tunnel end is most suitable, we had to find it out through tests. We tested:

- The IP address of the dynamic XCF link address. This did not work. A ping timed out. OSPF has routes via the XCF to other stacks only.
- The IP address of any hardware interface showed the same characteristics. A ping timed out.
- A DVIPA seems not to be suitable.
- A good choice was a static VIPA. The ping worked successfully.

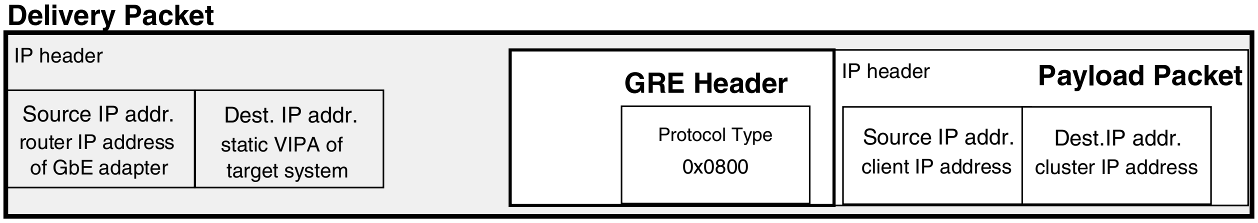


Figure 6-13 Delivery packet

The test clarified that the tunnel end works with the static VIPA of the target system. This means a tunnel has to be defined in each Cisco router with:

- Source IP address of the GbE interface of the router
- Destination IP address of the static VIPA of the system to be reached

This means that for our tests four tunnels were needed from router 7507 to each host in the sysplex. Four tunnels are also needed from router 7206.

The delivery packet is sent via a tunnel from the router over the Gigabit Ethernet, a switch, the OSA-Express adapter, to the TCP/IP stack containing the static VIPA address of the desired host.

Figure 6-14 on page 239 shows the four tunnels from router 7507. The other four tunnels from router 7206 are indicated only.

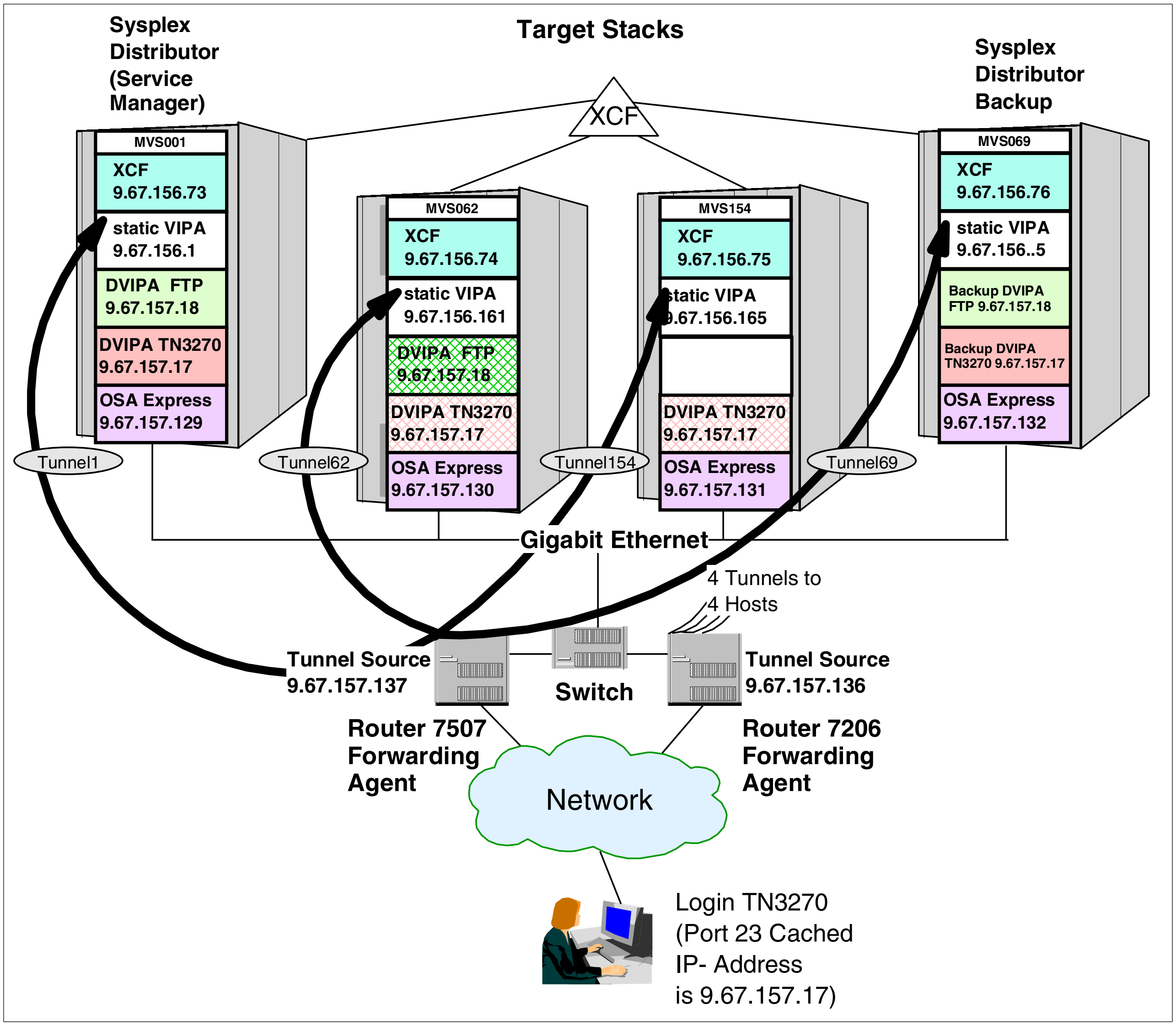


Figure 6-14 Tunnels over the shared OSA-Express

One question is still open. How does the Forwarding Agent know if it should take the destination IP address of tunnel1, or tunnel62, or tunnel69, or tunnel154?

The answer could only be that the Forwarding Agent knows the forwarding address for the TCP connection, which is the dynamic XCF link IP address. If a static route is defined in the router pointing to the dynamic XCF link IP address using the tunnel name, such as tunnel1, as the first hop, then the delivery packet will reach the correct destination.

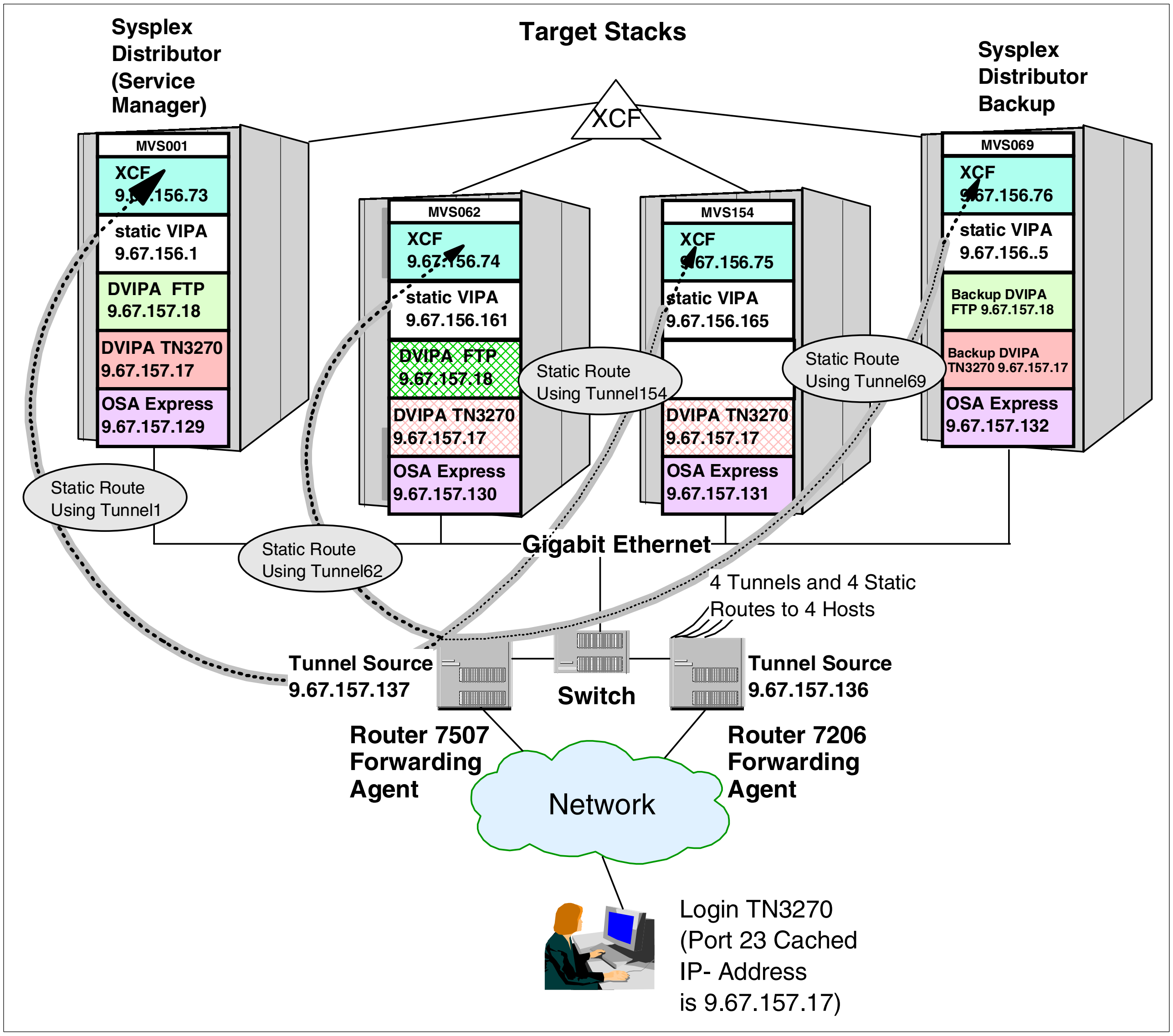

Sysplex Distributor (Service Manager)
Target Stacks
Sysplex Distributor Backup
XCF
MVS001
XCF 9.67.156.73
static VIPA 9.67.156.1
DVIPA FTP 9.67.157.18
DVIPA TN3270 9.67.157.17
OSA Express 9.67.157.129
MVS062
XCF 9.67.156.74
static VIPA 9.67.156.161
DVIPA FTP 9.67.157.18
DVIPA TN3270 9.67.157.17
OSA Express 9.67.157.130
MVS154
XCF 9.67.156.75
static VIPA 9.67.156.165
DVIPA TN3270 9.67.157.17
OSA Express 9.67.157.131
MVS069
XCF 9.67.156.76
static VIPA 9.67.156..5
Backup DVIPA FTP 9.67.157.18
Backup DVIPA TN3270 9.67.157.17
OSA Express 9.67.157.132
Static Route Using Tunnel1
Static Route Using Tunnel62
Static Route Using Tunnel154
Static Route Using Tunnel69
Gigabit Ethernet
4 Tunnels and 4 Static Routes to 4 Hosts
Tunnel Source 9.67.157.137
Switch
Tunnel Source 9.67.157.136
Router 7507 Forwarding Agent
Router 7206 Forwarding Agent
Network
Login TN3270 (Port 23 Cached IP- Address is 9.67.157.17)

Figure 6-15 Static routes using GRE tunnels

Another question may arise: What destination IP address will be taken by the Forwarding Agent when a SYN request is received or if there is no fixed affinity for an existing connection? In both cases, the Forwarding Agent has the forwarding address of the Sysplex Distributor, which is the dynamic XCF link IP address:

- In the wildcard affinity for the SYN
- In the wildcard affinity for cluster addresses and port numbers

Therefore, tunnels and static routes also have to be defined from each Forwarding Agent to the Sysplex Distributor with Service Manager and the corresponding backup Sysplex Distributor.

6.10.4 Definitions in the Cisco routers 7507 and 7206

- One tunnel is needed for each LPAR sharing the OSA adapter:
  - The source IP address of the tunnel is the Cisco GbE adapter address

  – The destination IP address of the tunnel end is the static VIPA address of the z/OS host

- For each dynamic XCF link IP address a static route is needed to the XCF link address using the tunnel.

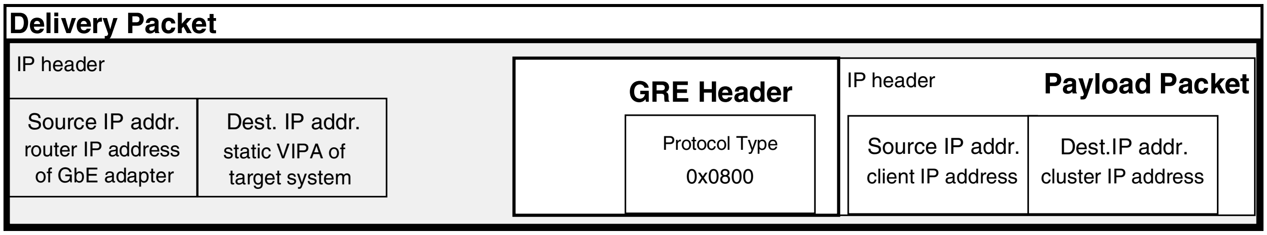


Figure 6-16 Complete delivery packet

Definitions in router 7507

Example 6-18 Tunnel definitions

```
interface Tunnel1
 description GRE tunnel to MVS001
 ip address 7.7.7.1 255.255.255.252
 no ip route-cache cef
 no ip route-cache
 no ip mroute-cache
 tunnel source 9.67.157.137
 tunnel destination 9.67.156.1
!
interface Tunnel62
 description GRE tunnel to MVS062
 ip address 8.8.8.1 255.255.255.252
 tunnel source 9.67.157.137
 tunnel destination 9.67.156.161
!
interface Tunnel69
 description GRE tunnel to MVS069
 ip address 11.11.11.1 255.255.255.252
 tunnel source 9.67.157.137
 tunnel destination 9.67.156.5
!
interface Tunnel154
 description GRE tunnel to MVS154
 ip address 12.12.12.1 255.255.255.252
 tunnel source 9.67.157.137
 tunnel destination 9.67.156.165
```

The tunnel is defined through an **`interface`** command with parameters shown in Table 6-2. We did not use tunnel checksum, tunnel key, tunnel mode (GRE is default), and tunnel sequence-datagrams, in order to keep the test simple.

Table 6-2 Tunnel definition through interface command

Command / Parameter	Description
interface tunnel1	Describes a tunnel name, which we used later for the static route. In our case it was tunnel1 from the router to MVS001.
description	Allows text for documentation.

Command / Parameter	Description
ip address	This line is not used for this kind of tunnel to the z/OS hosts. In other situations, for example for VPNs, it is required.
tunnel source	Defines the IP address where the tunnel starts. This address is used as the source address in the delivery packet.
tunnel destination	Defines the host name or IP address of the end of the tunnel. This address is used as the destination address in the delivery packet.
tunnel checksum	A checksum is created by the sender and transmitted in the GRE header to the receiver at the other end of the tunnel. The receiving IP stack recalculates the checksum and compares it with the sender's. If there is a transmission error, the GRE header and the payload packet will not be forwarded.
tunnel key	Is a kind of password, consisting only of numbers from 0 to 429467295.
tunnel mode	Describes the protocol used for the tunnel. In our case it was GRE.
tunnel sequence-datagrams	The sender sets sequence numbers in the GRE header. The receiver may control out-of-order packets with this information.

The tunnel source IP address will become the source IP address in the delivery packet. The tunnel destination IP address will become the destination IP address in the delivery packet.

Example 6-19 Static routes in router 7507

```
ip route 9.67.156.73 255.255.255.255 Tunnel1
ip route 9.67.156.74 255.255.255.255 Tunnel62
ip route 9.67.156.75 255.255.255.255 Tunnel154
ip route 9.67.156.76 255.255.255.255 Tunnel69
```

The definition of the static routes from router 7507 to the dynamic XCF link addresses of the four stacks use the tunnel names that have as the tunnel endpoint the static VIPA, defined in each z/OS host stack.

Example 6-20 Routing table in router 7507

```
NIVT7507#show ip route

Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
* - candidate default, U - per-user static route, o - ODR
P - periodic downloaded static route
Gateway of last resort is not set
     1.0.0.0/32 is subnetted, 1 subnets
C       1.1.1.1 is directly connected, CASA1
9.0.0.0/8 is variably subnetted, 51 subnets, 4 masks
C       9.67.156.144/28 is directly connected, Serial4/0/0
O E2    9.67.156.128/28 [110/20] via 9.67.156.149, 01:47:31, Serial4/0/0
C       9.67.157.128/28 is directly connected, Port-channel1
C       9.67.156.164/30 is directly connected, Tunnel154
O E1    9.67.156.165/32 [110/6] via 9.67.157.131, 01:47:31, Port-channel1
```

```
C       9.67.156.160/30 is directly connected, Tunnel62
O E1    9.67.156.161/32 [110/6] via 9.67.157.130, 01:47:32, Port-channel1
S       9.67.156.174/32 [1/0] via 9.67.156.174, Channel6/2
S       9.67.156.172/32 [1/0] via 9.67.156.172, Channel6/2
S       9.67.156.173/32 [1/0] via 9.67.156.173, Channel6/2
S       9.67.156.170/32 [1/0] via 9.67.156.170, Channel6/2
S       9.67.156.171/32 [1/0] via 9.67.156.171, Channel6/2
        9.67.156.68/32 [110/5] via 9.67.157.131, 01:52:32, Port-channel1
        9.67.156.69/32 [110/5] via 9.67.157.130, 01:52:32, Port-channel1
        9.67.156.66/32 [110/5] via 9.67.157.129, 01:52:33, Port-channel1
        9.67.156.67/32 [110/5] via 9.67.157.132, 01:52:33, Port-channel1
        9.67.156.65/32 [110/5] via 9.67.157.136, 01:52:34, Port-channel1
S       9.67.156.76/32 is directly connected, Tunnel69
S       9.67.156.74/32 is directly connected, Tunnel62
S       9.67.156.75/32 is directly connected, Tunnel154
O E1    9.67.156.72/29 [110/6] via 9.67.157.130, 01:52:36, Port-channel1
[110/6] via 9.67.157.131, 01:52:36, Port-channel1
[110/6] via 9.67.157.132, 01:52:37, Port-channel1
S       9.67.156.73/32 is directly connected, Tunnel1
```

The display shows that the static route, for example to 9.67.156.76, is directly connected, using tunnel69. Tunnel69 is the name used for the tunnel definition.

Definitions in router 7206

These definitions are similar to the definitions in router 7507. The only difference is the source IP address of the tunnel, which makes the source IP address of the delivery packet.

Example 6-21 Tunnel definitions of router 7206

```
interface Tunnel1
 description GRE tunnel to MVS001
 ip address 3.3.3.1 255.255.255.252
 tunnel source 9.67.157.136
 tunnel destination 9.67.156.1
!
interface Tunnel62
 description GRE tunnel to MVS062
 ip address 6.6.6.1 255.255.255.252
 tunnel source 9.67.157.136
 tunnel destination 9.67.156.161
!
interface Tunnel69
 description GRE tunnel to MVS069
 ip address 4.4.4.1 255.255.255.252
 tunnel source 9.67.157.136
 tunnel destination 9.67.156.5
!
interface Tunnel154
 description GRE tunnel to MVS154
 ip address 5.5.5.5 255.255.255.252
 tunnel source 9.67.157.136
 tunnel destination 9.67.156.165
!
```

The definitions in Example 6-22 do not differ from the ones in router 7507.

Example 6-22 Static routes in the router 7206

```
ip route 9.67.156.73 255.255.255.255 Tunnel1
ip route 9.67.156.74 255.255.255.255 Tunnel62
ip route 9.67.156.75 255.255.255.255 Tunnel154
ip route 9.67.156.76 255.255.255.255 Tunnel69
```

Trace examples

The following trace samples were taken from an IPCS packet trace of system MVS069. The trace shows:

- First the GRE header
- Then the delivery header
- Finally all together including the payload packet

GRE header analysis

```
 Generic Routing Encapsulation Header
  GRE Options      :
  Version          : 0              Protocol: IP
  Recursion        : 0
  Gre header size  : 4


GRE Header         : 4
000000 00000800
```

Figure 6-17 GRE header

In analyzing this header, you see that the Cisco router did not build a checksum for the GRE header, no source routing was used, no authentication was provided by the router, etc.

The only information in this header is the definition of the payload protocol with its Ethernet type value x'0800', which stands for IP.

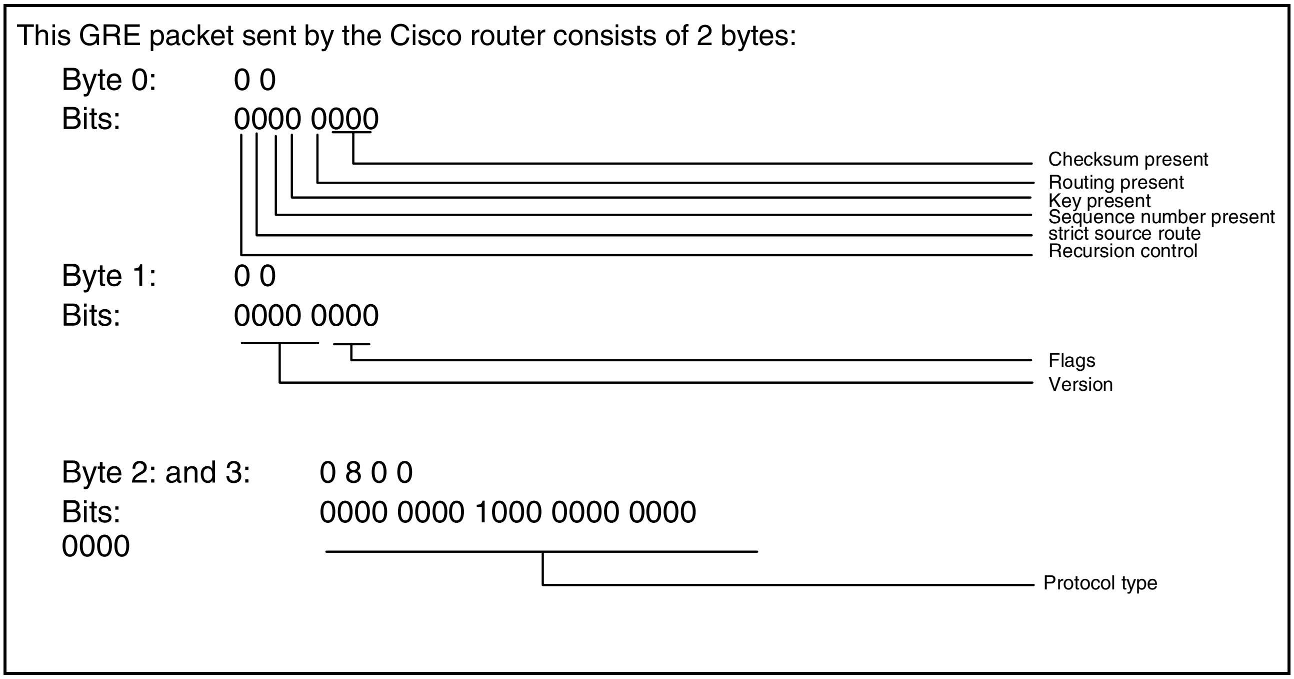
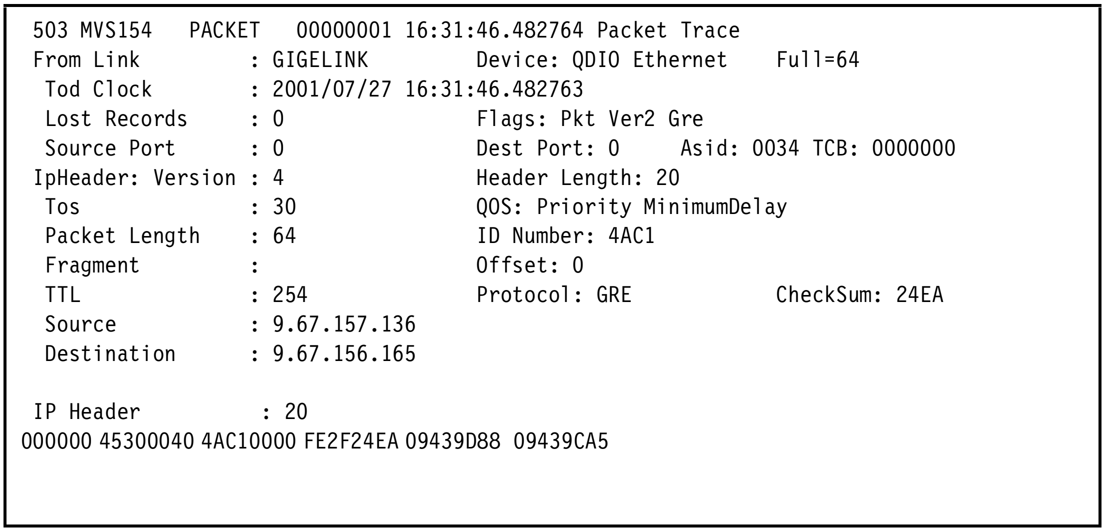


Figure 6-18 GRE header analysis

Delivery header



```
 503 MVS154   PACKET   00000001 16:31:46.482764 Packet Trace
 From Link         : GIGELINK        Device: QDIO Ethernet    Full=64
  Tod Clock        : 2001/07/27 16:31:46.482763
  Lost Records     : 0               Flags: Pkt Ver2 Gre
  Source Port      : 0               Dest Port: 0     Asid: 0034 TCB: 0000000
 IpHeader: Version : 4               Header Length: 20
  Tos              : 30              QOS: Priority MinimumDelay
  Packet Length    : 64              ID Number: 4AC1
  Fragment         :                 Offset: 0
  TTL              : 254             Protocol: GRE           CheckSum: 24EA
  Source           : 9.67.157.136
  Destination      : 9.67.156.165

 IP Header          : 20
000000 45300040 4AC10000 FE2F24EA 09439D88  09439CA5
```

Figure 6-19 Delivery header

This delivery packet was received at host MVS154, a target system. The packet was sent over the GbE link from router 7206.

- Tunnel source was 9.67.157.136, which is the IP address of the GbE interface of router 7206.
- Tunnel destination was 9.67.156 165, which is the static VIPA address of the z/OS host MVS154.
- Protocol in the IP header is GRE (byte 10 in the IP header with value x'2f').
- The length of the payload packet is 64 bytes, consisting of:
  - 20 bytes delivery header
  - 4 bytes GRE header
  - 40 bytes payload, consisting of:
    - 20 bytes IP header
    - 20 bytes TCP header
    - No data, because only an ACK, FIN was transmitted to initiate closing a TCP connection by the client

Entire delivery packet

The following trace record shows the IPCS output in a different order as provided by IPCS.

It shows in the first part the delivery header with its source address 9.67.157.136, which is the router's GbE adapter address, and the destination address 9.67.156.165, which is the static VIPA address of the target system MVS154.

The next block is the GRE header indicating the IP protocol only, since we did not define other GRE parameters.

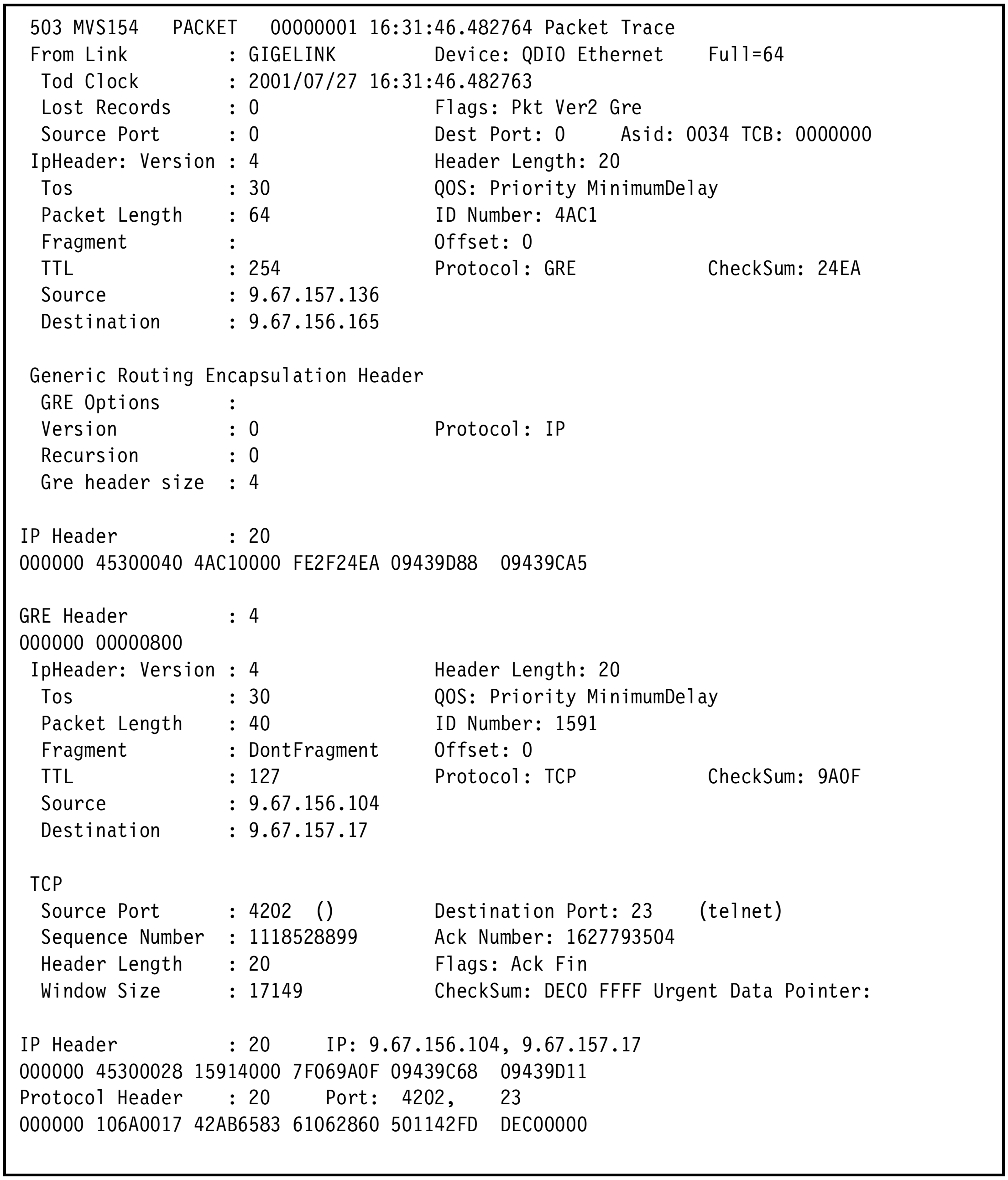
The last part is the payload with:

IP header Source: 9.67.156.104, which is the client

Destination: 9.67.157.17, which is the cluster address for the application

TCP header Source port, which is the client port

Destination port, which is the port of the TN3270 server



```
 503 MVS154   PACKET   00000001 16:31:46.482764 Packet Trace
 From Link         : GIGELINK        Device: QDIO Ethernet    Full=64
  Tod Clock        : 2001/07/27 16:31:46.482763
  Lost Records     : 0               Flags: Pkt Ver2 Gre
  Source Port      : 0               Dest Port: 0     Asid: 0034 TCB: 0000000
 IpHeader: Version : 4               Header Length: 20
  Tos              : 30              QOS: Priority MinimumDelay
  Packet Length    : 64              ID Number: 4AC1
  Fragment         :                 Offset: 0
  TTL              : 254             Protocol: GRE            CheckSum: 24EA
  Source           : 9.67.157.136
  Destination      : 9.67.156.165

 Generic Routing Encapsulation Header
  GRE Options      :
  Version          : 0               Protocol: IP
  Recursion        : 0
  Gre header size  : 4

IP Header          : 20
000000 45300040 4AC10000 FE2F24EA 09439D88  09439CA5

GRE Header         : 4
000000 00000800
 IpHeader: Version : 4               Header Length: 20
  Tos              : 30              QOS: Priority MinimumDelay
  Packet Length    : 40              ID Number: 1591
  Fragment         : DontFragment    Offset: 0
  TTL              : 127             Protocol: TCP            CheckSum: 9A0F
  Source           : 9.67.156.104
  Destination      : 9.67.157.17

 TCP
  Source Port      : 4202  ()        Destination Port: 23    (telnet)
  Sequence Number  : 1118528899      Ack Number: 1627793504
  Header Length    : 20              Flags: Ack Fin
  Window Size      : 17149           CheckSum: DEC0 FFFF Urgent Data Pointer:

IP Header          : 20     IP: 9.67.156.104, 9.67.157.17
000000 45300028 15914000 7F069A0F 09439C68  09439D11
Protocol Header    : 20     Port:  4202,    23
000000 106A0017 42AB6583 61062860 501142FD  DEC00000
```

Figure 6-20   Entire delivery packet

7

Performance and tuning

The performance of Communications Server for z/OS can be improved significantly by proper tuning. TCP/IP performance is influenced by a number of parameters that can be tailored for the specific operating environment. This includes not only TCP/IP stack performance that benefits all applications using the stack, but also applications that Communications Server for z/OS IP ships, such as TN3270 and FTP.

Because every TCP/IP environment is different, optimum performance can only be achieved when the system is tuned to match its specific environment. This chapter highlights the most important tuning parameters and recommends parameter values that have been found to maximize performance in customer installations.

7.1 Tuning the stack for performance

This section introduces the configuration files commonly used to improve performance, and describes some of the performance tuning options.

7.1.1 TCP/IP configuration files

The most important configuration files in Communications Server for z/OS IP from a performance point of view are:

- PROFILE.TCPIP

  The PROFILE.TCPIP file contains TCP buffer sizes, LAN controller definitions, server ports, home IP addresses, gateway and route statements, VTAM LUs for Telnet use, etc.

  TCP/IP in CS OS/390 V2R5 and later releases have been simplified by removing TCP/IP buffer pool and control block definitions from PROFILE.TCPIP, that is, no tuning is needed for TCP/IP buffers in the PROFILE.TCPIP. Buffers are dynamically allocated by the Communications Storage Manager (CSM).

- FTP.DATA

  The FTP.DATA file is used by the FTP server and FTP client to establish the initial configuration options. FTP.DATA contains items such as LRECL, BLOCKSIZE, RECFM, and CHKPTINT. Some parameters, such as EXTRATASKS, BUFNO, and NCP that were used in TCP/IP V3R1/V3R2 are eliminated from FTP.DATA for CS OS/390 V2R5 and later releases.

- TCPIP.DATA

  TCPIP.DATA contains host name, domain origin, and nsinteraddr (name server) definitions. The content of TCPIP.DATA is the same as for previous releases of TCP/IP for MVS. For a sample TCPIP.DATA, please see *z/OS V1R2.0 CS: IP Configuration Reference*, SC31-8776 or refer to the sample shipped with the product.

  One important recommendation is to keep the statement TRACE RESOLVER commented out to avoid complete tracing of all name server queries. This trace should be used for debugging purposes only.

7.1.2 Setting the appropriate MTU for devices

The maximum transmission unit (MTU) specifies the largest sized packet that TCP/IP will transmit over a given interface. Make sure that you specify a packet size explicitly instead of using the word DEFAULTSIZE. The word DEFAULTSIZE requests that TCP/IP supply a default value of 576 bytes, which might not be optimal in your configuration.

We recommend using the following sizes instead of DEFAULTSIZE as the packet size for the specified networks:

- 1492 bytes for Ethernet 802.3
- 1500 bytes for Ethernet Version 2 IEEE
- 1500, 2000 or 4000 bytes (or greater if your environment permits) for token-ring
- 4000 or 2000 bytes for FDDI (use larger value as permitted by your environment)
- 65527 bytes for CTC
- 4096 bytes for CLAW

Examples of static routing:

```
GATEWAY
;                               Packet  Subnet        Subnet
; Network    First hop   Driver size    mask          value
  9          9.67.115.43 FDDI1  4000    255.255.255.0 0.67.115.0
  DEFAULTNET 9.67.115.1  FDDI1  4000    0

BEGINROUTES
;                    Subnet
;       Destination  Mask     First hop       Link    MTU
 ROUTE  9.67.115.0/24       9.67.115.43    FDDI1   4000
 ROUTE    default           9.67.115.1     FDDI1   4000
ENDROUTES
```

Example of ORouteD dynamic routing:

```
BSDROUTINGPARMS TRUE
;                              Subnet        Subnet
; Driver  Packet size  metric  mask          value
  FDDI1   4000         0       255.255.255.0 9.67.115.0
ENDBSDROUTINGPARMS
```

Please note the true/false option on the BSDROUTINGPARMS statement. If you specify FALSE, all IP packets that are sent to a destination that is more than one hop away will not use the packet size of the interface, but will use a default packet size of 576 bytes. If you specify TRUE, all IP packets that are sent over an interface will use the interface's packet size, regardless of whether the destination is one or multiple hops away.

7.1.3 Devices and links

The DEVICE and LINK statements are used in the PROFILE.TCPIP. Other parameters, such as Ack Length and Blk Timer, are specific to the device configuration.

- 2216 LCS

  It is important to tune the 2216 LCS device parameters in the 2216 configuration - Blk Timer to 5 msec and Ack length to 10 bytes. For details, please refer to the *NWays Multiprotocol Access Services Software User's Guide Version 3 Release 1*, SC30-3886-03. Performance recommendations are described in Chapter 33 "Configuring and Monitoring ESCON Adapter".

- 3172 LCS

  For the 3172 LCS controller, we recommend that you set the delay timer to 10 msec and max response length to 500 for each LAN adapter in the 3172 configuration.

- CLAW devices

  If you are using CLAW devices, we recommend setting the number of read and write buffers to a value that is higher than the default of 20. In most situations you may be able to use 26 for read and 26 for write buffers. Some CLAW devices may permit 50 or an even higher number for read and write buffers. For example:

  ```
  DEVICE CLAWDEV CLAW 6B2 HOST PSCA NONE 26 26 4096 4096
   LINK  CLAWLINK IP 0 CLAWDEV
  ```

7.1.4 Tracing

From a performance perspective, turning off all tracing is recommended. Tracing does have an impact on performance. To disable tracing, include the following in your TCPIP.PROFILE:

```
ITRACE OFF
```

If tracing is to be used, change the parameters on the ITRACE statement. If, for example, you want to trace the configuration component, specify the ITRACE parameters as follows:

```
ITRACE ON CONFIG 1
```

7.2 z/OS UNIX System Services tuning

1. Follow the z/OS UNIX System Services performance tuning guidelines in *z/OS V1R2.0 UNIX System Services Planning*, GA22-7800 or at this URL:

   `http://www.s390.ibm.com/oe/bpxa1tun.html`
2. Make sure that the UNIXMAP RACF class is populated and cached.
3. Region size(s) and dispatching priority: It is highly recommended that you set the region size to 0K or 0M for the TCP/IP stack address space and for started tasks such as the FTP server, the SMTP/NJE server, the Web server, the ADSM server, etc. If your environment permits, set the dispatching priority for TCP/IP and VTAM equivalent and keep servers, such as FTP and ADSM, slightly lower than TCP/IP and VTAM. If you are using Workload Manager, follow the above recommendations when your installation defines performance goals in a service policy. Service policies are defined through an ISPF application and they set goals for all types of MVS managed work.
4. Update your PROFILE.TCPIP, TCPIP.DATA and FTP.DATA files with the applicable recommendations discussed in this chapter.
5. Estimate how many z/OS UNIX System Services users, processes, PTYs, sockets and threads would be needed for your z/OS UNIX installation. Update your BPXPRMxx member in SYS1.PARMLIB.
6. Spread z/OS UNIX user HFS data sets over more DASD volumes for optimal performance.
7. Monitor your z/OS UNIX resources with RMF and/or system commands (DISPLAY ACTIVE, and DISPLAY OMVS, etc.).

7.2.1 BPXPRMxx (SYS1.PARMLIB) tuning

1. Optimally set Max... parms.
   - Make sure MAXPROCSYS, MAXPROCUSER, MAXUIDS, MAXFILEPROC, MAXPTYS, MAXTHREADTASKS and MAXTHREADS are optimally set.
   - If these parms are not optimally set, your z/OS UNIX performance may be degraded. For more information, refer to *z/OS V1R2.0 UNIX System Services Planning*, GA22-7800.
2. Set MAXSOCKETS(n) to a high number to avoid shortage.
   - Make sure the MAXSOCKETS(n) parm for the AF_INET domain is set high enough to avoid running out of z/OS UNIX sockets.
   - As an example, each z/OS UNIX Telnet session would require one z/OS UNIX socket and each FTP session would require one z/OS UNIX socket. Once the MAXSOCKETS

limit is reached, no more Telnet, FTP sessions or other applications that require z/OS UNIX sockets would be allowed to start.

7.3 Storage requirements

You may need to estimate the storage requirement for your environment and tune CSM, CSA and SQA accordingly. We have provided storage usage summaries for typical applications, such as Telnet, FTP, CICS socket, and Web server. This may serve as a rule of thumb in determining the CSA, SQA and CSM storage requirements.

7.3.1 TCP and UDP send/receive buffer sizes

When send/recv buffer size(s) are not specified in the PROFILE, a default size of 16 KB will be used for send/recv buffers and a default of 32 KB will be used for the TCP window size. If send/recv buffer size(s) are specified, they will be used as specified and the TCP window size will be set to twice the TCP recv buffer size up to a maximum of 65535.

> **Note:** RFC 1323, which supports larger than 64 KB-1 TCP window sizes, is supported by CS for z/OS IP.

Customers can specify the send/recv buffer size(s) on the TCPCONFIG and UDPCONFIG statements:

```
TCPCONFIG     TCPSENDBFRSIZE    65535
              TCPRCVBUFRSIZE    65535
UDPCONFIG     UDPSENDBFRSIZE    65535
              UDPRCVBUFRSIZE    65535
```

Socket applications can override these values for a specific socket by using the setsockopt call:

`setsockopt(SO_SNDBUF)` or
`setsockopt(SO_RCVBUF)` in socket application.

> **Note:** The FTP server and client application override the default settings and use 64 KB-1 as the TCP window size and 180 KB for send/recv buffers. Therefore there is no change required in the TCPCONFIG statement for FTP server and client.

7.3.2 CSM storage usage

Table 7-1 shows a summary of the CSM usage pf the performance benchmark data from pre-GA measurements of CS for z/OS V1R2.

Table 7-1 CSM usage

Workload	# Users / Clients	TPUT	MAX CSM (ECSA)	MAX CSM (Data Space)	Max CSM (FIXED)
Web Server	80	5425 c/s	2.97 MB	8.0 MB	11.2 MB
CICS Sockets	84	409 c/s	0.736 MB	1.96 MB	3.8 MB

Workload	# Users / Clients	TPUT	MAX CSM (ECSA)	MAX CSM (Data Space)	Max CSM (FIXED)
TN3270 (Echo's)	4000 8000 16000 32000 64000	399.1 tr/sec 798.5 tr/sec 1591.5 tr/sec 3115.0 tr/sec 5732.5 tr/sec	0.85 MB 1.40 MB 1.34 MB 8.10 MB 17.90 MB	5.1 MB 11.1 MB 8.1 MB 12.6 MB 27.4 MB	12.5 MB 13.6 MB 17.5 MB 24.2 MB 50.0 MB
FTP Server	16 Inbound 16 Outbound	49.8 MB/S 68.9 MB/S	4.5 MB 0.54 MB	6.8 MB 4.4 MB	9.6 MB 6.4 MB

Based on the above application workload, we recommend the following definitions in SYS1.PARMLIB(IVTPRM00):

- FIXED MAX(60M), which includes 23 MB of additional storage for expected growth in the workloads.
- ECSA MAX(40M), which includes 18 MB of additional storage for expected growth in the workloads.

The above information may help in providing a guideline for deriving CSM requirements for a typical environment. With a different application mix, CSM requirements need to be adjusted accordingly. Our benchmark data for Telnet, FTP, CICS socket, etc., may help you in deriving your initial settings. You should monitor CSM and VTAM buffer usage using the following commands and fine tune the CSA, SQA and CSM requirements for your environments.

You may use the following display commands for monitoring storage:

```
D NET,BFRUSE,BUFFER=SHORT
D NET,STORUSE
```

For CSM usage, you may use the following commands:

```
D NET,CSM
D NET,CSM,ownerid=all
```

7.3.3 VTAM buffer settings

For a large number of Telnet (TN3270) sessions, we recommend that users change the default VTAM buffer settings using VTAM start options:

- IOBUF
- LFBUF
- CRPLBUF
- TIBUF
- CRA4BUF

Table 7-2 shows sample application buffer usage. We recommend using it as a guideline.

Table 7-2 VTAM buffer max usage

Workload	# Users / Clients	TPUT	VTAM Buffer (IO00)	VTAM Buffer (LF00)	VTAM Buffer (CRPL)	VTAM Buffer (TI00)	VTAM Buffer (CRA4)
Web Server	80	5425 c/s	6	4	2	4	4
CICS Sockets	84	409 c/s	26	5	54	29	6

Workload	# Users / Clients	TPUT	VTAM Buffer (IO00)	VTAM Buffer (LF00)	VTAM Buffer (CRPL)	VTAM Buffer (TI00)	VTAM Buffer (CRA4)
TN3270 (Echo's)	4000	399.1 tr/sec	112	4005	8007	5	25
	8000	798.5 tr/sec	112	8005	16012	5	41
	16000	1591.5 tr/sec	345	16005	32019	5	49
	32000	3115.0 tr/sec	2005	32003	64018	5	65
	64000	5732.5 tr/sec	11000	64005	128018	5	170
FTP Server	16 Inbound	49.8 MB/Sec	5	4	2	3	4
	16 Outbound	68.9 MB/Sec	5	4	2	3	4

To verify your settings, you may use the following display commands:

```
    D NET,BFRUSE,BUFFER=SHORT
    D NET,STORUSE
Telnet (TN3270) storage utilization
```

Table 7-3 shows benchmark data from the Telnet TN3270 performance test of 4000 to 64000 sessions using pre-GA level code for CS for z/OS V1R2. This data will be useful in deriving the rule of thumb for CSA and SQA storage requirements.

Table 7-3 Telnet storage utilization

# of Sessions	0	4000	8000	16000	32000	48000	64000
TCP/IP Below	0.548 M	0.6 M	0.616 M	0.648 M	0.696 M	0.792 M	0.824 M
TCP/IP Above	4.063 M	4.208 M	4.26 M	4.368 M	4.528 M	4.848 M	4.956 M
TCP/IP LSQA /SWA/229/ 230 Below	0.200 M	0.212 M	0.228 M	0.276 M	0.276 M	0.296 M	0.296 M
TCP/IP LSQA /SWA/229/ 230 Above	11.5 M	28.8 M	41.5 M	67.3 M	118 M	169 M	220 M
CSM Data Space	1.56 M	4 M	10.2 M	8.08 M	12.5 M	26.1 M	26.1 M
System CSA Below	1.14 M	1.14 M	1.15 M	1.15 M	1.15 M	1.15 M	1.15 M
System CSA Above	43.6 M	56.5 M	69.3 M	89.2 M	141 M	197 M	252 M
System SQA Below	0.756 M	0.836 M	0.836 M	0.836 M	0.836 M	0.836 M	0.836 M
System SQA Above	10.4 M	12.1 M	12.5 M	12.9 M	14 M	15 M	16.1 M
Total Below	2.65 M	2.79 M	2.83 M	2.91 M	2.96 M	3.07 M	3.106 M
Total Above	71.12 M	105.61M	138.7 M	181.85M	290.03M	411.95M	520.5M

# of Sessions	0	4000	8000	16000	32000	48000	64000
Total	73.77 M	108.4M	141.53M	184.76M	293M	415.0 M	523.6M
Delta Per User Total (KB)		8.66 KB	8.47 KB	6.94 KB	6.85 KB	7.11 KB	7.03 KB

Delta Per User represents the storage (below+above) in addition to initial storage requirements.

The above information will be useful for adjusting the CSA and SQA requirements for the TN3270 workload discussed in this chapter. For example, for 16,000 Telnet sessions, you need to adjust the CSA and SQA parameters in SYS1.PARMLIB such that 89.2 MB of CSA above the line and 12.9 MB of SQA above the line are available for Telnet TN3270. Note, however, that initial storage can be different for different environments.

7.4 Application performance and capacity

The following information provides the methodology for capacity planning for Telnet and FTP.

7.4.1 Telnet (TN3270)

The TCPIP.PROFILE is used for tuning Telnet parameters.

Telnet tuning

For best performance, use the following parameters in your TELNETPARMS section:

```
    TELNETPARMS
      INACTIVE 3600
      TIMEMARK 1200
      SCANINTERVAL 30
#######      DISABLESGA
    ENDTELNETPARMS
```

Where:

SCANINTERVAL Specifies the periodic time (in seconds) that the Telnet Server will scan the entire list of sessions. The default is 120 seconds.

TIMEMARK Specifies how often the Telnet Server will send a heartbeat to clients. Clients that do not respond to three consecutive probes are labeled inactive.

INACTIVE Specifies how long a client will remain in inactive state without communication until it is completely disconnected.

DISABLESGA Permits the transmission of GO AHEAD by the Telnet Server. This is negotiated by the client and server. Using this parameter increases the overhead for a full-duplex terminal using a full-duplex connection. The recommended default is to disable GO AHEAD.

Telnet capacity

In order to determine the capacity planning requirements, one needs to determine the CPU cost of performing a transaction of interest on a specific machine type. For example, we have used the Telnet TN3270 workload where 100 bytes of data is sent from the Telnet client to the Echo application running on z/OS Telnet TN3270 Server and 800 bytes of data is sent by the Echo application as a reply to the Telnet client. From our benchmarks, we have derived CPU cost in terms of milliseconds per Telnet TN3270 transaction using IBM 2064-108 (4 CP LPAR).

For example, we want to determine the capacity planning requirements for 4000 users, each performing six transactions per user per minute on an IBM 2064-108 (4 CP LPAR).

```
 # trans/user  x # users x CPU secs/tran      CPU secs
 -------------------------------------   =  ---------
           # of Elap secs                   Elap secs

Example:   CS/390 R12,  4000 users,   6 tr/min/user

 6 tr/u x 4000u x 0.000262 CPU secs/tr           cpu sec
 ------------------------------------- =  0.105   --------
          60  elap. sec                             elap sec
```

Figure 7-1 Sample Telnet CPU requirements

If the CPU secs/Elap sec ratio is greater than one, more than one CPU processor capacity would be required to drive the workload.

```
CPU secs/Elap Sec
-----------------  *  100 %   =  CPU Util %
 # of processors

# of processors:  4 ( This will be equal to number of
                  390 processors used for the LPAR)

Example:   Therefore, Percentage of CPU utilization on Four CP
LPAR to drive 6 tran/user/minute for 4,000 users will be


0.‘05 CPU secs/Elap sec
-----------------------  *  100 %  =  2.63 %
     4 processors
```

Figure 7-2 Sample Telnet CPU utilization

Thus, the total CPU requirement for 4000 TN3270 users would require 2.63% of a four-processor IBM 2064-108 machine. LSPR can be used to adjust for other processor types. Consult your IBM marketing representative or systems engineer to get the correct relationship in processing power between the different IBM processors.

7.4.2 FTP

The FTP.DATA file is used by the FTP server and client to establish the initial configuration options for an FTP session. The search order for FTP.DATA varies depending on the execution environment. For a detailed description of the search order, refer to *z/OS V1R2.0 CS: IP Configuration Reference*, SC31-8776.

FTP tuning

Several parameters can be changed during an FTP session by use of the SITE or LOCSITE end-user commands.

Data set attributes play a significant role in FTP performance. If your environment permits it, tune both BLOCKSIZE and LRECL according to the following recommendations.

We generally recommend using half a track as the block size. For IBM 3380 DASD, use 23424 as the block size with an LRECL of 64 bytes. For IBM 3390 or IBM 9334, use 27968 as the block size with an LRECL of 64 bytes. Use FB as the data set allocation format. It is also important to use cached DASD controllers. If your environment permits it, use a pre-allocated data set for FTP transfer operations into MVS.

Sample definitions in FTP.DATA:

```
PRIMARY   15
SECONDARY 20
LRECL     64
BLKSIZE   23424
RECFM     FB
CHKPTINT  0
DIRECTORY 27
SQLCOL    ANY
INACTIVE  0
```

If your installation supports System Determined Blocksize (SDB), you can leave the block size determination to DFSMS by specifying the block size as zero:

```
BLKSIZE 0
```

FTP capacity

The methodology for FTP capacity planning is very similar to Telnet. The key difference is that you need to determine the average cost for transferring a kilobyte of data on a given setup, that is machine type, channel attached device, type of workstation, type of transfer, etc.. You can derive the CPU cost using the selected setup by repeating FTP transfers of large files. CPU cost per kilobyte must include TCP/IP, VTAM and FTP address space usage.

Total (TCP/IP+VTAM+FTP) CPU requirements for FTP transfer:

```
Max KB         CPU  secs        CPU secs
---------   *  ---------   =   ----------
Elap secs         KB           Elap  secs
```

Figure 7-3 General formula to calculate FTP CPU requirements

For example, if we are interested in driving aggregate effective FTP throughput of 10000 KBps for transferring data from workstations to MVS using 2216 as the channel-attached device and FDDI as the LAN media on 2064-108 CP LPAR, from our FTP measurements of binary PUT, 20 Mbps file transfer, we have determined that CPU cost per Kb is 0.0000347 CPU seconds.

```
50995 KB       .0000127         .648    CPU secs
---------   *  ------------   =         ----------
Elap secs         KB                    Elap  secs
```

Figure 7-4   Sample formula to calculate FTP CPU requirements

If the CPU secs/Elap sec ratio is 1 or greater, you would need more than one processor capacity to drive the desired throughput.

```
 CPU secs/Elap Sec
 -----------------  *  100 %  =  CPU Util %
  # of processors

 # of processors:  4 ( This will be equal to number of
                   390 processors used for the LPAR)

Example: For our example we are using ES9672-RX5 4 processor
LPAR.  Therefore, Total CPU utilization will be


 0.648 CPU secs/Elap sec
 -----------------------   *   100 %   = 16.2 %
      4 processor
```

Figure 7-5   FTP CPU requirements example

Thus, Total CPU (TCP/IP+VTAM+FTP) requirements for driving 10000 KBps throughput on 2064-108 (4 CP LPAR) will be 16.2%.

If you are using a different type of mainframe but keeping the other setup the same, you can use the LSPR information to adjust the above CPU utilization. Consult your IBM marketing representative or systems engineer to get the correct relationship in processing power between the different IBM processors. The above methodology can be used for deriving the capacity planning for different types of workload.

7.5 TCP/IP performance checklist

Here is a quick summary of a performance checklist, which may help in confirming performance settings for your environment.

- MVS dispatching priority of TCP/IP, FTP or other servers. Recommendation: Set TCP/IP equal to VTAM or one lower than VTAM. For other started tasks, such as FTP and ADSM, set slightly lower than the TCP/IP task.
- Make sure client/server TCP Window size is set to the allowed maximum. Recommendation: Set the TCP window size on MVS to the allowed maximum by setting TCPRCVBUFRSIZE to 32 KB or larger and, if the client workstation permits, use 65535 as the client window size. If the installation is storage constrained, however, use the default TCPRCVBUFRSIZE of 16 KB.

Note: RFC 1323, which supports larger than 64 KB-1 TCP window sizes, is available in V2R7.

- Make sure the client and server MTU/packet size are equal. Follow the recommendations given in 7.1.1, “TCP/IP configuration files” on page 248.
- Routers: Make sure buffers are set appropriately so that packets are not being dropped.
- 3172: Make sure delay timer and maximum response length are set correctly for each LAN adapter:

  Recommendation: Delay timer = 10 ms
  maximum response length = 500 bytes
- 2216: Make sure Blk timer and Ack length are set correctly for each LCS or MPC+ definition:

  Recommendation: Blk Timer = 5 ms
  Ack Length = 10 bytes
- RS/6000 ESCON: Use MPC instead of CLAW for attachment.
- FTP: Use large data set block sizes on MVS. Recommendation: Data set block size = 1/2 DASD track. Follow the recommendations discussed in 7.4.2, “FTP” on page 255.
- Telnet: Follow recommendations described in 7.4.1, “Telnet (TN3270)” on page 254.
- Sockets: Use large msg sizes (> 1 KB) for better performance.
- Follow BPXPRM recommended settings shown in 7.2.1, “BPXPRMxx (SYS1.PARMLIB) tuning” on page 250.
- PTFs: Make sure you have the latest TCP/IP performance PTFs. We continually improve performance of z/OS releases and it is recommended that you keep track of PSP buckets for important PTFs.
- Make sure TCP/IP and all other traces are turned off for optimal performance. Trace activity does create an extra processing overhead.

A

Dump of T28ATCP name server - single-path network

```
$ORIGIN itso.ral.ibm.com.
ralplex1INSOAmvs03a.ralplex1.itso.ral.ibm.com. garthm@mvs03a.ralplex1.itso.ral.ibm.com. (
       1999040102 7200 3600 604800 3600 );Cl=5
       IN NS mvs03a.ralplex1.itso.ral.ibm.com.;Cl=5
       IN A  172.16.250.3;Cl=5
       IN A  172.16.252.28;Cl=5
       IN A  172.16.232.39;Cl=5
$ORIGIN ralplex1.itso.ral.ibm.com.
FTPRAL IN A  172.16.232.39;Cl=5
       IN A  172.16.250.3;Cl=5
TN28   IN A  172.16.252.28;Cl=5
mvs03a IN A  172.16.250.3;Cl=5
TN03   IN A  172.16.250.3;Cl=5
mvs03c IN A  172.16.251.5;Cl=5
mvs28a IN A  172.16.252.28;Cl=5
TNTSO  IN A  172.16.250.3;Cl=5
TNRAL  IN A  172.16.250.3;Cl=5
       IN A  172.16.252.28;Cl=5
       IN A  172.16.232.39;Cl=5
ralplex1INCNAMEralplex1.itso.ral.ibm.com.;Cl=5
TN39   IN A  172.16.232.39;Cl=5
mvs39a IN A  172.16.232.39;Cl=5
$ORIGIN FTPRAL.ralplex1.itso.ral.ibm.com.
MVS03A IN A  172.16.250.3;Cl=5
MVS39A IN A  172.16.232.39;Cl=5
$ORIGIN TN28.ralplex1.itso.ral.ibm.com.
MVS28A IN A  172.16.252.28;Cl=5
$ORIGIN TN03.ralplex1.itso.ral.ibm.com.
MVS03A IN A  172.16.250.3;Cl=5
$ORIGIN TNTSO.ralplex1.itso.ral.ibm.com.
MVS03A IN A  172.16.250.3;Cl=5
$ORIGIN TNRAL.ralplex1.itso.ral.ibm.com.
MVS39A IN A  172.16.232.39;Cl=5
MVS28A IN A  172.16.252.28;Cl=5
MVS03A IN A  172.16.250.3;Cl=5
```

```
$ORIGIN TN39.ralplex1.itso.ral.ibm.com.
MVS39A IN A  172.16.232.39;Cl=5
$ORIGIN 172.in-addr.arpa.
16     IN SOAmvs03a.ralplex1.itso.ral.ibm.com. garthm@mvs03a.16.172.in-addr.arpa. (
       1999040101 7200 3600 604800 3600 );Cl=4
       IN NS mvs03a.ralplex1.itso.ral.ibm.com.;Cl=4
$ORIGIN 100.16.172.in-addr.arpa.
3      IN PTRmvs03a.ralplex1.itso.ral.ibm.com.;Cl=4
$ORIGIN 101.16.172.in-addr.arpa.
28     IN PTRmvs28a.ralplex1.itso.ral.ibm.com.;Cl=4
$ORIGIN 232.16.172.in-addr.arpa.
39     IN PTRmvs39a.ralplex1.itso.ral.ibm.com.;Cl=4
$ORIGIN 102.16.172.in-addr.arpa.
39     IN PTRmvs39a.ralplex1.itso.ral.ibm.com.;Cl=4
$ORIGIN 233.16.172.in-addr.arpa.
28     IN PTRmvs28a.ralplex1.itso.ral.ibm.com.;Cl=4
39     IN PTRmvs39a.ralplex1.itso.ral.ibm.com.;Cl=4
3      IN PTRmvs03a.ralplex1.itso.ral.ibm.com.;Cl=4
$ORIGIN 250.16.172.in-addr.arpa.
3      IN PTRmvs03a.ralplex1.itso.ral.ibm.com.;Cl=4
$ORIGIN 251.16.172.in-addr.arpa.
5      IN PTRmvs03c.ralplex1.itso.ral.ibm.com.;Cl=4
$ORIGIN 252.16.172.in-addr.arpa.
28     IN PTRmvs28a.ralplex1.itso.ral.ibm.com.;Cl=4
$ORIGIN 0.127.in-addr.arpa.
0      IN SOAmvs03a.ralplex1.itso.ral.ibm.com. garthm@mvs03a.0.0.127.in-addr.arpa. (
       1999040101 7200 3600 604800 3600 );Cl=5
       IN NS mvs03a.ralplex1.itso.ral.ibm.com.;Cl=5
$ORIGIN 0.0.127.in-addr.arpa.
1      IN PTRloopback.;Cl=5
$ORIGIN 24.9.in-addr.arpa.
104    IN SOAmvs03a.ralplex1.itso.ral.ibm.com. garthm@mvs03a.104.24.9.in-addr.arpa. (
       1999040101 7200 3600 604800 3600 );Cl=5
       IN NS mvs03a.ralplex1.itso.ral.ibm.com.;Cl=5
$ORIGIN 104.24.9.in-addr.arpa.
42     IN PTRmvs28a.ralplex1.itso.ral.ibm.com.;Cl=5
149    IN PTRmvs39a.ralplex1.itso.ral.ibm.com.;Cl=5
113    IN PTRmvs03a.ralplex1.itso.ral.ibm.com.;Cl=5
; --- Hints ---
$ORIGIN .
.  3600IN NS mvs25o.buddha.ral.ibm.com.;Cl=0
$ORIGIN buddha.ral.ibm.com.
mvs25o3600INA9.24.104.125;Cl=0
```

B

REXX EXECs to gather connection statistics

In this appendix we show the three REXX EXECs used to invoke server functions in the sysplex for the purpose of gathering statistics on load balancing.

32-Bit Windows EXEC to issue repeated pings

```
/*Rexx - Exec to perform TCP/IP Sysplex validation and tracing */
call RxFuncAdd 'SysLoadFuncs','rexxutil','SysLoadFuncs'
call SysLoadFuncs
'@ECHO OFF'
/*                                                             */
/*    Syntax:  SYSPLEXW appl_name num_pings ping_delay         */
/*                                                             */
/*             appl_name  The name of the application as       */
/*                        registered in WLM.                   */
/*                                                             */
/*             num_pings  How many times do you want to        */
/*                        ping the application.                */
/*                        default = 20                         */
/*                                                             */
/*             ping_delay How many seconds to wait between     */
/*                        pings.  default = 0                  */
/*                                                             */
Parse Arg p1 p2 p3 .
/*                                                             */
/*  Which application to ping                                  */
/*                                                             */
If p1 <> '' Ã p1 = '?' Then pingname = p1
Else
  Do
    Say
    Say  'Syntax:  SYSPLEXW appl_name num_pings ping_delay       '
    Say  '                                                       '
    Say  '         appl_name  The name of the application as     '
    Say  '                    registered in WLM.                 '
    Say  '                                                       '
    Say  '         num_pings  How many times do you want to      '
    Say  '                    ping the application.              '
    Say  '                    default = 20                       '
    Say  '                                                       '
    Say  '         ping_delay How many seconds to wait between   '
    Say  '                    pings.  default = 0                '
    Say  '                                                       '
    Exit
  End
If p2 <> '' Then pingloop = p2
Else             pingloop = 10

If p3 <> '' Then sleeptime = p3
Else             sleeptime = 0
/*                                                             */
/*  Define some working files and variables and headings       */
/*                                                             */
fred64 = time(L)
Parse Var fred64  hhv ':' mmv ':' ssv '.' therest
datafile = '\pf' ÃÃ mmv ÃÃ ssv ÃÃ Substr(therest,1,2) ÃÃ '.dat'
pingfile = '\pf' ÃÃ mmv ÃÃ ssv ÃÃ Substr(therest,1,2) ÃÃ '.wrk'
'Erase 'pingfile

Call disp_prt_null
dataline = Left('Application or Host Name',40)  Left('IP Address',15) Left('Time',10)
Say dataline
fred = lineout(datafile,dataline,1)
Call disp_prt_null
```

```
goodpings = 0
lostpings = 0
maxaddrs = 0
/*                                                             */
/*  Now for the main loop                                      */
/*                                                             */
Do i = 1 to pingloop
  linectr = 0
  starttime = Time('R')
  fred2 = SysSleep(sleeptime)
  lostpingflag = 'NO'
  'Ping 'pingname ÃÃ ' -l 10 -n 1 > 'pingfile          /* Send the PING and response to a
temp file */
  endtime = Time('E')

  pingline.linectr = Linein(pingfile,,0)               /* Open the input file  */
  pinglineend = 'NO'
  /*      */
  Do until pinglineend = 'YES'                         /* Read lines into array  */
    linectr = linectr + 1
    pingline.linectr = Linein(pingfile)
    If pingline.linectr = '' & linectr > 8 Then
      Do
        pinglineend = 'YES'
        linectr = linectr -1
      End
  End /* do */
  Call pingparse                                        /* Parse the lines */
  fred = Lineout(pingfile)                              /* Closes the file  */
End /* Do */
dataline = ' '
Call disp_prt
dataline = 'Summary of Ping responses'
Call disp_prt
dataline = ' '
Call disp_prt
dataline = 'Good Responses : 'goodpings
Call disp_prt
dataline = 'Lost Responses : 'lostpings
Call disp_prt
dataline = 'Total Responses: 'goodpings + lostpings
Call disp_prt
dataline =  ' '
Call disp_prt
dataline = 'Hits by Canonical Addresses'
Call disp_prt
dataline = ' '
Call disp_prt
dataline =  Left('Number    ',10) Left('IP Address',15) Left('Application or Host Name',40)
Left('Time',10)
Call disp_prt
Do n = 1 to maxaddrs
  dataline =  Left(canoncnt.n,10) Left(canon.n,15) Left(pingname.n,40) Left(rsptime.n /
canoncnt.n,10)
  Call disp_prt
End /* do */
fred = lineout(datafile)                                 /* Close the file */
'erase 'pingfile
'Edit 'datafile
```

```
Say ''
Say '????????????????????????????????????????????????????????????????'
Say ''
Say 'A report file, 'datafile' has been created on your hard disk.'
Say ''
Say "             Reply 'y' to erase it"
Say "                   'r' to rename it"
Say '                    anything else to quit'
Say ''
Parse Upper Pull ans .
If ans = 'Y' Then 'erase 'datafile
If ans = 'R' Then
  Do
    Say 'Please enter new fn.ft for 'datafile'.'
    Parse Pull newname
    'rename 'datafile newname
  End
Exit
/*                                                       */
/* Now to parse and consolidate the output from PING     */
/* Lines have already been read in                       */
/*                                                       */
pingparse:
Do j = 1 to linectr                                      /* Last line is '' anyway */
  Parse Var pingline.j w1 w2 w3 w4 w5 w6 w7 w8 w9 w10
/*  Say 'w1='w1 'w2='w2 'w3='w3 */
  Select
     When w1 = 'Pinging' Then
       Do
         thispingname = w2
       End
     When w1 = 'Reply' & w2 = 'from' Then
       Do
         thisaddr = Substr(w3,1,length(w3)-1)
         goodpings = goodpings + 1
       End
     When w1 w2 w3 = 'Request' 'timed' 'out.' Then
       Do
         lostpingflag = 'YES'    h
         lostpings = lostpings + 1
         dataline = Left(pingname,40)  Left('no response',15) Left(endtime,10)
         Call disp_prt
       End
     When w1 w2 w3 = 'Bad' 'IP' 'address' Then
       Do
         lostpingflag = 'YES'    h
         lostpings = lostpings + 1
         dataline = Left(pingname,40)  Left('no response',15) Left(endtime,10)
         Call disp_prt
       End
    Otherwise NOP /* lostpingflag = 'YES' */
  End  /* select */
End /*Do*/
If lostpingflag = 'NO' Then
  Do
    dataline = Left(thispingname,40) Left(thisaddr,15) Left(endtime,10) /* create audit
record */
    Call disp_prt
    Call sortping
  End
```

```
Return
/*                                                       */
/* Check if same onsolidate the output from PING         */
/*                                                       */
/*     There are 4 arrays                                */
/*                                                       */
/*     canon.m    : canonical address                    */
/*     pingname.m : text name for this canonical addr    */
/*     canoncnt.m : number of times this addr used       */
/*     rsptime.m  : total response time for this addr    */
/*                                                       */
/*                                                       */
sortping:
If maxaddrs = 0 Then     /*First time thru*/
  Do
    maxaddrs   = 1
    canon.1    = thisaddr
    pingname.1 = pingname
    canoncnt.1 = 0
    rsptime.1  = 0
  End
newcanon = 'YES'                    /* assume it is a new canonical address */
Do m = 1 to maxaddrs
  If canon.m = thisaddr & pingname.m = pingname Then
    Do
      canoncnt.m = canoncnt.m + 1
      rsptime.m  = rsptime.m + endtime
      newcanon = 'NO'               /* flag we have it already */
      m = maxaddrs                  /* get out of loop         */
    End
End /* do */
If newcanon = 'YES' Then
  Do
    maxaddrs = maxaddrs + 1
    canon.maxaddrs    = thisaddr
    pingname.maxaddrs = pingname
    canoncnt.maxaddrs = 1
    rsptime.maxaddrs  = endtime
  End
Return
/*                                                            */
/* Routine to display records on screen and also file data    */
/*                                                            */
disp_prt_null:
dataline = ''
disp_prt:
Say dataline
fred = Lineout(datafile,dataline)
Return
```

EXEC to connect to server using TCP

```
/**********************************************************************/
/* sysplex2.cmd                                                       */
/*                                                                    */
/*   sysplex2 hostname port <-c num_connects> <-t conn_time>          */
/*                          <-b betw_time>                            */
/*                                                                    */
/* This Rexx program connects to a server on a given hostname/portname*/
/* pair the specified number of times. On each connection it will     */
/* read 4 bytes from the server - this value is interpreted as the    */
/* IP address of the server we have actually connected to             */
/* (irrespective of the server we *requested* to connect to).         */
/*                                                                    */
/* conn_time is a specified time to stay connected to server over and */
/* above the time needed for exchange of data. This is measured in    */
/* in seconds and defaults to 0. This option is required if you are   */
/* connecting to the multi-tasking server. If you fail to specify -t  */
/* when connecting to the multi-tasking server, the client program    */
/* will appear to hang.                                               */
/*                                                                    */
/* num_connects is the number of time you wish to connect to the      */
/* server. This defaults to a value of 10.                            */
/*                                                                    */
/* betw_time is a specified time to pause between connections to the  */
/* server. This is measured in seconds and defaults to 0.             */
/*                                                                    */
/* Overall statistics on the number of times the hostname was resolved*/
/* to a given IP address by the DNS server and the number of times we */
/* connected to a given TCP/IP stack are reported at the end of       */
/* execution                                                          */
/*                                                                    */
/**********************************************************************/

  parse arg arg1 arg2 '-'opt.3 arg.3 '-'opt.4 arg.4 '-'opt.5 arg.5

  if(arg1 = '' | arg1 = '?' | arg2 = '') then
  do
    say 'Usage: sysplex2 hostname port <-c num_connects> <-t conn_time> <-b betw_time>'
    return
  end


  connectToName = arg1                /* Hostname to connect to       */
  connectToPort = arg2                /* Port to connect to           */
                                      /* Set defaults:                */
  numConns = 10                       /* Default value for numConns   */
  connTime = 0                        /* Default value for connTime   */
  betwTime = 0                        /* Default value for betwTime   */
  do a = 3 to 5
    select
      when opt.a = 'c' | opt.a = 'C' then numConns = arg.a
      when opt.a = 't' | opt.a = 'T' then
      do
        connTime = arg.a
        connTimeSpecified = true
      end
      when opt.a = 'b' | opt.a = 'B' then betwTime = arg.a
      otherwise
    end          /* end of select select */
  end a

  printHeader = true                  /* Only print headers on the    */
                                      /* first iteration of the loop  */
/**********************************************************************/
/* Load the rexx sockets functions if not already loaded              */
/**********************************************************************/
  rc = RxFuncQuery("SockLoadFuncs")
  if(rc <> 0) then
  do
    rc = RxFuncAdd("SockLoadFuncs","rxsock","SockLoadFuncs")
    rc = SockLoadFuncs()
  end

/**********************************************************************/
/* Loop around for numConns                                           */
/**********************************************************************/
  do i = 1 to numConns

    call time('R')                    /* Reset timer                  */
/**********************************************************************/
/* Use DNS to resolve name to IP address                              */
/**********************************************************************/
    rc = SockGetHostByName(connectToname, "resolvedHost.!")

    resolvedTime = time('E')          /* Record time taken to resolve */
                                      /* name to an IP address        */
    if(rc = 0) then
    do
      say "Error resolving hostname: " errno
      return
    end

/**********************************************************************/
/* Create a socket                                                    */
/**********************************************************************/
    socket = SockSocket("AF_INET", "SOCK_STREAM", 0 )
    if(socket = -1) then
    do
      say "Error creating socket: " errno
      return
    end

/**********************************************************************/
/* Wait for specified time before connecting to the server.          */
/* If the pause is greater than or equal to 1 second, a CPU friendly  */
/* sleep call is performed.  If the pause is less than one second, a  */
/* CPU intensive loop is entered.  This is to be avoided.             */
/**********************************************************************/
    if betwTime >= 1 then
    do
      rc = RxFuncAdd("SysLoadFuncs","RexxUtil","SysLoadFuncs")
      rc = SysLoadFuncs()
      betwTime = betwTime % 1;          /* Discard fractional part    */
      call SysSleep( betwTime )
    end
    else
    do
      call time('R')                    /* Reset timer                */
      elapsedTime = time('E')
      do while elapsedTime < betwTime
        elapsedTime = time('E')
      end
    end

/**********************************************************************/
/* Connect to the server                                              */
/**********************************************************************/
    server.!family = "AF_INET"
    server.!port   = connectToPort
    server.!addr   = resolvedHost.!addr

    call time('R')                    /* Reset timer                  */
    rc = SockConnect(socket, "server.!")
    if(rc = -1) then
    do
      say "Error on connecting socket to '" || server.!addr || "':" errno

/**********************************************************************/
/* If we failed to connect we should log the resolved name for        */
/* statistics processing but the connected name isn't applicable      */
/**********************************************************************/
      savedResName.i = resolvedHost.!addr
      savedConName.i = errno

/**********************************************************************/
/* Close the socket as the connect failed                             */
/**********************************************************************/
      rc = SockSoClose(socket)
      if(rc = -1) then
      do
        say "Error closing socket:" errno
      end
      iterate                         /* Go back to beginning of loop */
    end
/**********************************************************************/
/* Before we receive the IP address from the server, we send it the   */
/* time specified by the user to stay connected.                      */
/**********************************************************************/
    if(connTimeSpecified = true) then
    do
      drop buffer
      buffer = d2c(connTime*10)
      rc = SockSend(socket, buffer, 4)
    end

/**********************************************************************/
/* The recv will block until the server has performed the sleep and   */
/* sent some data through the socket.  It should be a 4 byte IP addr. */
/* Since we use buffer each time we go round this loop we must 'drop' */
/* it so that it gets set correctly by recv                           */
/**********************************************************************/
    drop buffer
    rc = SockRecv(socket, buffer, 4)

    if(rc < 1) then
    do
      say "Error on receive:" errno
      return
    end
```

```
/**********************************************************************/
/* Convert the IP address to decimal and record who we really         */
/* connected to                                                       */
/**********************************************************************/
    byte1 = c2d(substr(buffer,1,1))
    byte2 = c2d(substr(buffer,2,1))
    byte3 = c2d(substr(buffer,3,1))
    byte4 = c2d(substr(buffer,4,1))

    connectedTo = byte1 || '.' || byte2 || '.' || byte3 || '.' || byte4

/**********************************************************************/
/* Close the socket                                                   */
/**********************************************************************/
    rc = SockSoClose(socket)
    if(rc = -1) then
    do
      say "Error closing socket:" errno
      return
    end
    connectedTime = time('E')         /* Record time spent connected  */

    if(printheader = true) then
    do
      say '+----------------------+------------------+----------------+--------+--------+'
      say '| Hostname             | Resolved Addr    | Connected Addr | Resolv | Connec |'
      say '+----------------------+------------------+----------------+--------+--------+'
      printheader = false             /* Do not print header next time*/
    end

    say '|' left(connectToName, 20) '|' left(resolvedHost.!addr, 16),
        '|' left(connectedTo, 14) '|' left(resolvedTime, 6),
        '|' left(connectedTime, 6) '|'


/**********************************************************************/
/* Save the resolved and connected names for this run to allow        */
/* processing of connection statistics                                */
/**********************************************************************/
    savedResName.i = resolvedHost.!addr
    savedConName.i = connectedTo

  end /* do i = 1 to numConns */

call sysstats numConns, savedResName., "Resolved "
call sysstats numConns, savedConName., "Connected"
```

REXX statistics subroutine

```
/**********************************************************************/
/* sysstats.cmd                                                       */
/*                                                                    */
/* Called from sysplex2.cmd                                           */
/*                                                                    */
/* Overall statistics on the number of times we connected to a        */
/* given TCP/IP stack are reported in this subroutine.                */
/*                                                                    */
/**********************************************************************/

  use arg numConns, savedAddr., Text

/**********************************************************************/
/* Loop around for numConns - this time for statistics processing.    */
/* In here we scan through the array of saved addresses and each time */
/* we find a new (non-blank) one we stop and count how many more of   */
/* this same address there subsequently are in the table, blanking    */
/* them out as we count them so they won't be counted more than once  */
/**********************************************************************/
  Count = 0                           /* Set counter to 0             */
  do forever
    biggest = 0.0.0.0                 /* Smallest possible IP address */
    do i = 1 to numConns              /* Find biggest non-blank name  */
      if(savedAddr.i <> '') then
      do                              /* Separate out domain levels   */
        parse value biggest with b.1 '.' b.2 '.' b.3 '.' b.4
        parse value savedAddr.i with c.1 '.' c.2 '.' c.3 '.' c.4
        do n = 1 to 4
          select                      /* Compare one level at a time. */
            when c.n > b.n then
              do
                biggest = savedAddr.i
                leave
              end
            when c.n = b.n then iterate
            when c.n < b.n then leave
          end /* select */
        end n
      end
    end i
    if(biggest = 0.0.0.0) then leave  /* No more left to sort         */
    else                              /* Else: we found one           */
    do
      Count = Count + 1               /* Increment counter            */
      statsAddr.Count = biggest       /* Store address away           */
      statsTotal.Count = 0            /* Set count for this addr to 0 */
      do j = 1 to numConns            /* and then count them up.      */
        if(savedAddr.j = biggest) then
        do
          savedAddr.j = ''            /* don't count this name again  */
          statsTotal.Count = statsTotal.Count + 1
        end
      end j
    end /* if(biggest <> 0) */
  end /* do forever */

  say ''
  say '+----------------+----------------+'
  say '|' left(Text,9) 'Addr |' left(Text,9) 'Count|'
  say '+----------------+----------------+'
  do i = Count to 1 by -1
    say '|' left(statsAddr.i, 14) '|' left(statsTotal.i, 14) '|'
  end
```

C

Sample applications and programs

In Chapter 2, “DNS/WLM (connection optimization)” on page 17, we introduced a simple REXX EXEC to enable us to determine how well load balancing was working across the images in a sysplex. Using DNS/WLM in OS/390 V2R5 IP as we were, this EXEC was perfectly adequate. However, with the increasing sophistication of later releases and the Network Dispatcher, we need more than that. Indeed, newer functions such as dynamic VIPA, sysplex sockets, and finally, Sysplex Distributor required more coding effort to demonstrate their operation effectively.

In this appendix, therefore, we introduce some simple coding that we used to good effect in subsequent tests:

- A WLM registration program, WLMREG.

  This works with our new servers (or indeed, with any server) to register the address space or process in which the server runs with WLM.

- A new REXX EXEC, SYSPLEX2.

  The simple EXECs in Chapter 2, “DNS/WLM (connection optimization)” on page 17 used ping to exchange data with their target servers. Network Dispatcher will not work with ping, because it dispatches only TCP and UDP traffic. Ping is ICMP and is echoed by the Network Dispatcher itself, proving nothing about its load-balancing abilities.

  Our EXEC uses TCP to connect to a server; we have written two versions of this server (a single-threading and a multithreading version). Our server provides the needed statistics to our REXX EXEC, and also shows how to register an application to a dynamic VIPA address.

- We have written a WLM query program to display what applications have registered and their status. Using this program is somewhat easier than poring over a trace or a dump, although it does not provide as much information.

- We have also written a program to utilize the sysplex sockets interface. This allows an application to query the sysplex environment in which it runs, and to take appropriate action depending on that environment.

WLMREG, a sample registration program

Telnet and FTP provide parameters that allow you to specify the group name that they will register under to use DNS/WLM workload balancing. If you want your own TCP/IP applications to benefit from connection workload balancing, then they must manually register with WLM.

Fortunately this is a simple thing to do. All that is required is an address space to call the IWMSRSRG macro or the IWMDNREG function from C. It is important to realize that it is the MVS *address space* the application is running in that is registered with WLM, not the application itself. This actually makes life a little easier, since we can simply call a generic registration program before starting a server and benefit from DNS/WLM connection balancing very easily.

The sample registration program has been designed to allow it to be used for almost any TCP/IP application that runs in a single address space. You simply call this program with the relevant parameters before you start your own application, and the address space will be registered with WLM; thus, work destined for the WLM group name will be routed to it. If you are running your programs in UNIX System Services, then it is the process, not the address space, that is registered with WLM.

The program we have provided is based upon the sample shipped with TCP/IP and located in /usr/lpp/tcpip/samples/wlmreg.c on HFS. The complete source for the sample is available in "WLMREG registration sample" on page 287.

The following section shows how to set up and use a simple user TCP/IP server application written in C.

The registration call

The IWMDNREG C function is documented in *Communications Server for z/OS IP Configuration*, SC31-8513. It provides a simple way to register with WLM, associating the IP addresses of the local TCP/IP stack with a given WLM group name. This association also includes the address space that performed the registration call, allowing WLM to deregister us automatically if our address space should terminate without manually deregistering.

Under the covers, the IWMDNREG C function calls the MVS Workload Manager's IWMSRSRG service.

The C function is invoked as follows:

```
extern long IWMDNREG( char *group_name,  char *host_name,  char *server_name,  char *netid,
char *wlm_user_data,  long *diag_code);
```

The first parameter, `group_name`, should be the group name under which this application will register. Effectively, this will become a virtual host name. For example, if we are running on host `ralplex1.buddha.ral.ibm.com` and we register with group name `fred`, then users will be able to connect to our application, or any other application registered with the same group name, through the virtual host name of `fred.ralplex1.buddha.ral.ibm.com`.

The second parameter, `host_name`, should contain the TCP/IP host name of the stack our application is listening on and should generally be obtained through the `gethostname()` call from the registration program.

The third parameter, `server_name`, should be a name that will identify uniquely this instance of the application and must be different for each server registering under a given group name.

The next two parameters are not required and should be set to NULL.

The `diag_code` is a variable passed by reference to give us extra information, if the registration call should fail.

If you make the IWMDNREG call from C, then you need to make sure you define the call with OS linkage, as is done in the sample TCP/IP header file iwmwdnsh.h, and link edit with the stub in your SYS1.CSSLIB data set.

If you are running a sockets program in UNIX System Services and cannot alter the program, you can still use the WLMREG program to register the process in which the program runs. This is achieved by using the `execvp()` system call after the IWMDNREG completes. `execvp()` loads a program from an ordinary executable file into the current process, replacing the current program. Since the process is still running, socket connections can be routed to the newly loaded program.

To deregister, or not to deregister?

Once our address space is registered, we will have work for our group name routed to us until either we manually deregister or our address space terminates. Typically, a long-running server program will probably want to have work routed to it until it terminates, in which case we do not strictly need to deregister manually unless some other program is going to run in our address space after our server has terminated. However, it is worth being aware of a potential scenario where we might want to deregister even though our server is still active.

Provided the TCP/IP stack registers itself with WLM, the status of its IP interfaces will be communicated to WLM and thus to DNS. Therefore, if all the interfaces through which our server can be reached are inactive, WLM will not route work to the server. However, if the stack fails or is brought down, DNS loses track of the interface status and will continue to route work to our server (which is still registered to WLM and therefore included in the data sent by WLM to DNS). Our server program's TCP/IP calls will fail with return codes indicating that the TCP/IP service is not available. Depending on how our server application has been written, we might either terminate or wait for TCP/IP to return. If we quit then, WLM will be informed and will no longer route work to us. If we hang around waiting for our TCP/IP stack to restart (which could take some time), then we will remain registered and WLM will attempt to route some work to us. WLM will have no way of knowing that the clients being routed to us via the now inactive network interface are being rejected while other connection requests that are routed to other members of our WLM group name are probably succeeding.

To prevent this, our application will need to deregister manually from WLM if a TCP/IP failure (or indeed any other transient failure preventing us from being able to process clients' requests) is detected. Obviously, once we are able to process work again we need to reregister with WLM.

Unfortunately, this causes problems when we want to use our sample WLM/DNS registration program to register an existing server application without having to modify it.

If you have a server application that does not terminate as the result of a temporary failure then your best option, if available, is indeed to modify the server to manually register and deregister from WLM as required.

If you cannot modify the server (it might be a third-party application), then you might have to live with the potential consequences. These may be acceptable if, for example, the client applications will continually try to reestablish a connection, since they will eventually be routed to an available server.

Another possibility to investigate might be the ability of the IWMSRSRG macro version of the call to register on behalf of *another* address space. Thus, you might have some other application that monitors resource availability on an MVS image and registers/deregisters the applications for which it is responsible as necessary. This version of the WLM registration call is available only if you use the macro call; it is not available from the C language version. We do not explore this scenario in this redbook.

Waiting for WLM to Update DNS

While at any given moment WLM is aware of exactly what servers are registered for each group name, this information is queried by the DNS only periodically (every 60 seconds by default). This means that there are periods during which DNS may incorrectly resolve, or even fail to resolve, group names to IP addresses.

This is most obvious when you start a server that is the first to register a particular group name. For up to 60 seconds (or whatever your DNS/WLM refresh rate is), the DNS will be unable to resolve this name. Conversely, where an application has deregistered (either manually or automatically at MVS end-of-memory processing) there is a period where clients will be routed to a server that is no longer available. You might like to consider this possibility when designing a server application: instead of deregistering and not accepting any more inbound connections, you might deregister and allow a grace period where clients can still connect while you wait a reasonable period of time for the WLM to update DNS. While this is far from ideal, it might alleviate some of the connection problems associated with a server shutdown.

Collecting statistics using REXX

In 2.6.5, “Observing the effects of WLM and DNS” on page 47 we introduced a REXX EXEC that repeatedly pings the sysplex and gathers up the number of successful pings made. We want to gather similar statistics using the Network Dispatcher (ND). However, the ND dispatches only TCP and UDP protocols and ping uses ICMP, so we provide a new client application that uses TCP.

We have written a REXX client program to connect to the SOCSRVR program. This program gathers statistics on the number of successful connects, and the IP address it was connected to each time.

The SYSPLEX2 EXEC is executed by the following command:

```
REXX SYSPLEX2 hostname port -c num_connects -b between_time
```

where `hostname` and `port` are the pair you wish to connect to, `num_connects` is the number of times you wish to connect to them and `between_time` is the time in seconds to pause between connections to the server. It should be noted that if the pause is greater than or equal to one second, a CPU-friendly `SysSleep()` call is performed. If the pause is less than one second, a CPU-intensive loop is entered. This should be avoided if possible. You could reimplement it in an alternative fashion if your need is great enough.

You can omit the parameters `num_connects` and `between_time`. They default to 10 and 0 respectively.

The client program starts by calling `SockGetHostByName()` to resolve the host name to an IP address as shown in Figure C-1 on page 277:

```
rc = SockGetHostByName(connectToName, "resolvedHost.!")
if(rc = 0) then
do
  say "Error resolving hostname: " errno
  return
end
```

Figure C-1 Resolving host name in REXX sockets API

It then allocates a standard stream socket and connects to it as shown in the code excerpt in Figure C-2:

```
socket = SockSocket("AF_INET", "SOCK_STREAM", 0 )
if(socket = -1) then
do
  say "Error creating socket: " errno
  return
end
server.!family = "AF_INET"
server.!port   = connectToPort
server.!addr   = resolvedHost.!addr

rc = SockConnect(socket, "server.!")
if(rc = -1) then
do
  say "Error on connecting socket to '" || server.!addr || "':" errno
end
```

Figure C-2 Allocate socket and connect to it

Then it receives the server's IP address from the server, and finally it closes the socket. This is shown in Figure C-3:

```
rc = SockRecv(socket, buffer, 4)
if(rc < 1) then
do
  say "Error on receive:" errno
  return
end

rc = SockSoClose(socket)
if(rc = -1) then
do
  say "Error closing socket:" errno
  return
end
```

Figure C-3 Receive data from server and close

This process is repeated for the number of connects specified and then the results are counted and printed.

The complete source for the REXX client is in “EXEC to connect to server using TCP” on page 266. The subroutine that is called at the end of the REXX program to sort and output the statistics, `sysstats`, is in “REXX statistics subroutine” on page 270. This was done in a separate file since it is less interesting from a socket programming point of view.

If you wish to store the results in a file, then use the following command:

```
REXX SYSPLEX2 hostname port -c num_connects -b between_time > output.txt
```

where `output.txt` is the name of your output file.

WLMQ, a WLM query program

We have a program that allows us to query the server programs that have already been registered to WLM. When called without parameters, a list of all registered server programs is displayed. If parameters are present, only servers registered with groups named by the parameters are shown.

The program uses the C function IWMDNGRP (which maps to the IWMSRDNS macro) to obtain a list of groups and then calls IWMDNSRV (which maps to the IWMSRSRS macro) for each group to obtain a list of registered servers.

The C function to query a list of group names is invoked as follows:

```
extern long IWMDNGRP( struct grpinfo_block *grp_array,  long * entry_count,  long *
diag_code );
```

The first parameter, `grp_array`, is an array of structures that can hold information on a group. Currently, the only information in the structure is the name of the group. Due to the asynchronous nature of deregistration, a group may still be present in the output list even though all server programs using that group have deregistered. Conversely, some registered servers may not appear for this same reason.

The second parameter, `entry_count`, is used on input to specify the maximum number of entries that the program can receive safely (how much storage we chose to allocate). On output it contains the number of currently registered groups, and hence how many `grpinfo_block` structures have been filled in. The WLMQ program first calls IWMDNGRP with a `group_count` of zero. This gives us the number of groups currently registered with WLM. We then allocate space for two more groups than are currently registered and call IWMDNGRP again. Having the extra two groups allows us to consider new groups being registered between the two calls.

The `diag_code` is a variable passed by reference to provide additional diagnostic information if the call does not complete successfully.

The C function to query information for a group of servers is invoked as follows:

```
extern long IWMDNSRV( char *group_name,  struct sysinfo_block *sys_array,  long
*entry_count,  long *diag_code );
```

The first parameter, `group_name`, identifies the group we are querying.

The second parameter, `sys_array`, is an array of structures that can hold information on a server program. The structure is defined as follows:

```
struct sysinfo_block {  char netid[WLMSIZE_OF_NETID];  char server[WLMSIZE_OF_SERVER];
unsigned char weight;  char user_data[64];  char host_nameid[WLMSIZE_OF_HOSTNAME];  };
```

For most server programs the `netid` and `user_data` fields will be blank. For the TCP/IP stack, the `user_data` field contains the list of addresses for the active IP interfaces.

The `weight` field contains the relative weighting for each server. This value tells a caller the relative number of requests to send to each server.

The third parameter to the IWMDNSRV call, `entry_count`, is again used on input to specify the maximum number of entries that the program has storage to receive. On output it contains the number of programs currently serving the group, and thus how many `sysinfo_block` structures have been populated. This value is set to 10 on input. If your environment has more programs serving a single group, edit the sample program and give the `#define`d symbol `WLMQ_MAX_SERVERS` a larger value.

The `diag_code` is a variable passed by reference to provide additional diagnostic information if the call does not complete successfully.

See Figure C-4 for some example output:

```
-----------------------------------------------------------
#    Group          Server     HostName   NetId      Weight
-----------------------------------------------------------
1    TESTRAL        SERVER28   MVS28A                 32
1    TESTRAL        SERVER03   MVS03A                 31
2    TNRAL          MVS03A     MVS03A                 21
2    TNRAL          MVS28A     MVS28A                 21
2    TNRAL          MVS39A     MVS39A                 21
3    TCPIP          T03ATCP    MVS03A     MVS03A     10
UserData: 172.16.250.3    9.24.104.113    172.16.100.3    172.16.233.3
          172.16.233.3
3    TCPIP          T03CTCP    MVS03C     MVS03C     10
UserData: 172.16.251.4    9.24.104.33     172.16.233.4    172.16.233.4
3    TCPIP          T28CTCP    MVS28C     MVS28C     10
UserData: 172.16.100.99   9.24.104.43     172.16.253.29   172.16.233.29
          172.16.233.29   172.16.233.29
3    TCPIP          T28ATCP    MVS28A     MVS28A     10
UserData: 172.16.252.28   9.24.104.42     172.16.101.28   172.16.104.28
          172.16.233.28   172.16.233.28   172.16.233.28   172.16.240.28
          172.16.240.193
3    TCPIP          T39ATCP    MVS39A     MVS39A     21
UserData: 172.16.232.39   9.24.104.149    172.16.105.39   172.16.233.39
          172.16.240.39
```

Figure C-4 Example output from the WLM query program

Note: The ability to query WLM registration status is planned for implementation in a future release of CS for OS/390.

SOCSRVR, a simple socket server program

Here we introduce a simple TCP/IP server program, SOCSRVR, that we use throughout this book to test workload balancing. The program is started with a single argument specifying the TCP/IP port number on which it should listen. It begins by calling `gethostname()` and `gethostid()` to find the host name and IP address of the TCP/IP stack it will use, as shown in Figure C-5 on page 280:

```
    if(gethostname(hostName, 64) !=0)
    {
        tcperror("Gethostname()");
        exit(2);
    }

    if((hostId = gethostid()) == 0)
    {
        tcperror("Gethostid()");
        exit(2);
    }
```

Figure C-5 Find hostname and IP address

It then allocates a standard stream socket and listens for inbound connections on the port specified at startup as shown in Figure C-6:

```
    if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        tcperror("Socket()");
        exit(3);
    }

    server.sin_family = AF_INET;
    server.sin_port   = htons(port);
    server.sin_addr.s_addr = INADDR_ANY;

    if (bind(s, (struct sockaddr *)&server, sizeof(server)) < 0)
    {
        tcperror("Bind()");
        exit(4);
    }

    if (listen(s, 255) != 0)
    {
        tcperror("Listen()");
        exit(5);
    }
```

Figure C-6 Allocate socket and listen

When a client connects we simply send it four bytes containing our IP address, close the connection, and go back to waiting for the next client connection request. This is shown in Figure C-7 on page 281.

```
while(1)
{
    namelen = sizeof(client);
    if ((ns = accept(s, (struct sockaddr *)&client, &namelen)) == -1)
    {
        tcperror("Accept()");
        exit(6);
    }

    if (send(ns, (char*) &hostId, sizeof(hostId), 0) < 0)
    {
        tcperror("Send()");
        exit(7);
    }
    close(ns);
}
```

Figure C-7   Accept conversation, send data and close

The complete source for the sample is available in “SOCSRVR single threading server” on page 294.

See Figure C-8 for some example JCL that runs the WLMREG and SOCSRVR programs in the same address space:

```
//GOWLMRG1 JOB (NEIL,D1111),'NEILJ',MSGCLASS=H
//REG EXEC PGM=WLMREG,REGION=0M,
// PARM='TESTRAL SERVER1'
//STEPLIB  DD DISP=SHR,DSN=NEIL.UTIL.LOAD
//         DD DISP=SHR,DSN=CEE.SCEERUN
//SYSPRINT DD SYSOUT=*
//SYSTCPD  DD DISP=SHR,DSN=TCP.TCPPARMS(TDATA03A)
//*
//SRV EXEC PGM=SOCSRVR,REGION=0M,TIME=NOLIMIT,
// PARM='1234'
//STEPLIB  DD DISP=SHR,DSN=NEIL.UTIL.LOAD
//         DD DISP=SHR,DSN=CEE.SCEERUN
//SYSPRINT DD SYSOUT=*
//SYSTCPD  DD DISP=SHR,DSN=TCP.TCPPARMS(TDATA03A)
```

Figure C-8   Sample JCL to run the WLMREG then the SOCSRVR programs

Modifying SOCSRVR for Dynamic VIPA

Modifying the SOCSRVR sample (see “SOCSRVR, a simple socket server program” on page 279) to exploit dynamic VIPA is very simple. First, we change the parameter checking so that two arguments are required, as shown in Figure C-9.

```
if (argc != 3)
{
  fprintf(stderr, "Usage: %s port VIPA\n", argv[0]);
  exit(1);
}
```

Figure C-9   Modifying the parameter checking on SOCSRVR for Dynamic VIPA

Secondly, we change the `sock_addr_in` structure so that instead of using INADDR_ANY to bind to all addresses we bind to the VIPA passed as the second argument, as shown in Figure C-10.

```
    server.sin_addr.s_addr = inet_addr( argv[2] );

    if (bind(s, (struct sockaddr *)&server, sizeof(server)) < 0)
    {
        tcperror("Bind()");
        exit(4);
    }
```

Figure C-10 Binding to a specific VIPA

Sysplex sockets

Socket applications are written generally to communicate with a partner on any platform. This means that the improved performance and scalability of the z/OS sysplex is not exploited, unless some application-specific protocol is used; this is not always possible.

The sysplex sockets function provides a standard way to discover information about the connected partner, which can then be used to make decisions that can exploit the value of the z/OS sysplex where applicable.

Discovering partner information

This is done by way of a new option on the socket call `getsockopt()`, which is described in more detail in *Communications Server for z/OS IP Application Programming Interface Guide*, SC31-8516. This new option is `SO_CLUSTERCONNTYPE` and is coded as shown in Figure C-11.

```
    if (getsockopt(s, SOL_SOCKET, SO_CLUSTERCONNTYPE, (char *)&type, &typelen) < 0)
    {
        tcperror("GetSockOpt()");
        exit(5);
    }
```

Figure C-11 getsockopt() call

The call can return any of the values shown in Table C-1. In this context a *cluster* is a sysplex.

Table C-1 Returned values

Returned Value	Description
SO_CLUSTERCONNTYPE_NOCONN	No connection active.
SO_CLUSTERCONNTYPE_NONE	Active connection and the partner is not in the same cluster.
SO_CLUSTERCONNTYPE_SAME_CLUSTER	Active connection and the partner is in the cluster.
SO_CLUSTERCONNTYPE_SAME_IMAGE	Active connection and the partner is in the same (MVS) image.

Returned Value	Description
SO_CLUSTERCONNTYPE_INTERNAL	Active connection and the partner is in the cluster and the link is either loopback, MPCPTP, CTC or XCF.

These returned values are tested for as shown in the code excerpt in Figure C-12.

```
if (!(type & SO_CLUSTERCONNTYPE_NOCONN))
{
    if (type & SO_CLUSTERCONNTYPE_NONE)
    {
        /* The connection is not in the same cluster */
    }
    if (type & SO_CLUSTERCONNTYPE_INTERNAL)
    {
        /* The connection is an internal type of connection */
    }
    if (type & SO_CLUSTERCONNTYPE_SAME_IMAGE)
    {
        /* The connection is in the same (MVS) image */
    }
    if (type & SO_CLUSTERCONNTYPE_SAME_CLUSTER)
    {
        /* The connection is in the same cluster */
    }
}
```

Figure C-12   getsockopt() call

SSOCCLNT, a sample sysplex sockets program

The sample sysplex sockets program very simply connects to the SOCSRVR program introduced in “SOCSRVR, a simple socket server program” on page 279 and, before receiving the data from the server, issues the `getsockopt()` call with the new option `SO_CLUSTER_CONNTYPE`. It then prints out the type of the connection. The output will look something like Figure C-13:

```
Connection Type :
 Same Image
 Same Cluster
Server's IP address : 9.24.104.113
```

Figure C-13   Output from SSOCCLNT

The complete source for the sample is available in “SSOCCLNT sysplex sockets sample” on page 296.

A useful application of the sysplex sockets function would be to make some decisions regarding, for example, security or data conversion, depending on the information returned about the partner.

Loading the system

For our more advanced tests, we needed to load our sysplex systems beyond what a single client could provide by way of traffic. Therefore, we modified our server to handle multiple clients. In other words, we converted it to a multi-threading server.

MTCSRVR, a multitasking socket program

Here we introduce our multitasking server, MTCSRVR, and the subtask program it schedules, MTCSUBT. The server is the Multitasking C Socket Sample Program provided in the *Communications Server for z/OS IP Application Programming Interface Guide*, SC31-8516, with some minor changes so that it no longer handles the simplest case. These changes remove a couple of lines that say:

```
/*** do simplified situation first ***/
```

and add code to allow the number of subtasks to be specified as the second parameter. The first parameter is the port on which to listen.

The source for this sample can be found in “MTCSRVR multitasking sockets program” on page 298, with the lines that are changed from the original highlighted in bold.

We have written a new subtask program based upon the sample subtask provided to interact with our REXX client program.

The subtask starts by using the information passed as parameters to fill in the `clientid` structure and take the socket from the calling program as shown in Figure C-14.

```
memset(&cid, 0, sizeof(cid));
memcpy(cid.name,        tskname,  8);
memcpy(cid.subtaskname, tsksname, 8);
cid.domain = AF_INET;

socket = takesocket(&cid, *clsock);
if (socket < 0)
{
  tcperror("Csub: Error from takesocket");
}
```

Figure C-14 Obtaining the socket from the calling program

If the `takesocket` is successful, we receive data from the socket into a local byte. This controls how long the server program sleeps before responding to the client. The value is the number of tenths of a second to sleep. See Figure C-15 on page 285.

```
recvbytes = recv(socket, data, sizeof(data), 0);
if (recvbytes < 0)
{
  tcperror("Csub: Recv()");
}
else
{
  printf("Sleeping for %d seconds\n", (*data)/10 );
  sleeptime = (*data) / 10;
  sleep(sleeptime);
}
```

Figure C-15 Receiving data from the socket and sleeping for a time

The subtask then queries the local IP address and sends it back to the client program as shown in Figure C-16.

```
if((hostId = gethostid()) == 0)
{
  tcperror("Csub: Gethostid()");
}
sendbytes = send(socket, (char*) &hostId, sizeof(hostId), 0);
if (sendbytes < 0)
{
  tcperror("Csub: Send()");
}
```

Figure C-16 Querying the host ID and sending the value to the client

When getting the subtask to work, you must link-edit it correctly as described in the *OS/390 C/C++ Programming Guide*, SC09-2362, with the following linkage editor control statements:

```
INCLUDE SYSLIB(EDCMTFS)
ENTRY   CEEESTART
```

The source for this sample can be found in “MTCSUBT subtask for the multitasking sockets program” on page 304.

Extra option for the REXX client program

The REXX client program has an extra option that causes it to send the time to sleep to the server program.

To connect to the multitasking server, SYSPLEX2 should be invoked in the following way:

```
REXX SYSPLEX2 hostname port -c num_connects -t time_connected -b between_conns
```

The -c and -b parameters are the same as in “Collecting statistics using REXX” on page 276. The new parameter is -t. If the -t parameter is omitted, then the sleep time is not sent to the server program. Care must be taken to ensure that the presence or absence of the -t parameter matches the server program. If you are connecting to the simple server, you should omit the -t parameter, since the simple server does not expect to receive any data. Specifying -t will not do any harm, but the simple server will not perform a sleep. If you fail to specify -t when connecting to the multitasking server, then your client program will appear to hang. This is because the multitasking server program has called recv(), expecting to receive a sleep time, but the client program has not sent a sleep time. The source for the REXX client program can be found in “EXEC to connect to server using TCP” on page 266.

D

Sample C program source code

In this section we list the C programs we used to exercise the sysplex load balancing functions, the dynamic VIPA and the sysplex sockets feature.

WLMREG registration sample

```
/**********************************************************************/
/* A program to register the address space with WLM. If this is to be */
/* used in Open Edition, execvp() is called to execute another        */
/* program in the same process.                                       */
/**********************************************************************/

/**********************************************************************/
/* If you are building the OE version, uncomment the next line        */
/**********************************************************************/
/* #define BUILD_OE_VERSION */

#if defined( BUILD_OE_VERSION )
  #define MIN_PARAMETERS 3
#else
  #define MIN_PARAMETERS 2
  #include <manifest.h>
#endif

#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <iwmwdnsh.h>

int main(int argc, char** argv)
{
    char *groupName;
    char *serverName;
    char hostName[64];
    long diagCode = 0;
    long rc = 0;
    if (argc < MIN_PARAMETERS)
    {
#if defined( BUILD_OE_VERSION )
        fprintf(stderr, "Usage: %s groupname servername "
                "program_to_start <program parameters>\n", argv[0]);
#else
        fprintf(stderr, "Usage: %s groupname servername\n", argv[0]);
#endif
        exit(1);
    }
/**********************************************************************/
/* Extract the groupname and servername from the command line         */
/**********************************************************************/
    groupName = argv[1];
    serverName = argv[2];

    rc = gethostname(hostName, WLMSIZE_OF_HOSTNAME);
    if (rc != 0)
    {
        fprintf(stderr, "gethostname failure errno = %d \n", errno);
        exit(2);
    }
/**********************************************************************/
/* Register a server                                                  */
/**********************************************************************/
    rc = IWMDNREG(groupName
                 ,hostName
                 ,serverName
                 ,NULL
                 ,NULL
                 ,&diagCode
                 );
    if (rc != 0)
    {
        fprintf(stderr, "IWMDNREG failure, error = %d \n", diagCode);
        exit(3);
    }
    printf("Registered a server successfully:\n");
    printf("  Groupname  = %s\n", groupName);
    printf("  Hostname   = %s\n", hostName);
    printf("  Servername = %s\n", serverName);

/**********************************************************************/
/* In Open Edition, start the sockets server program in the same      */
/* process, as we're deregistered when the process ends               */
/* If execvp() succeeds, the call never returns as the new program    */
/* overwrites the old one.                                            */
/**********************************************************************/
#if defined( BUILD_OE_VERSION )
    rc = execvp(argv[3], &argv[3]);
    /* if the execvp() call succeeds, the call never returns */
    fprintf(stderr, "execvp returned %d\n", rc);
    exit(rc);
#endif
    exit(0);
}
```

WLM query program

```
/*********************************************************************/
/* wlmq: A WLM Query Program                                         */
/*                                                                   */
/* When called without parameters, a list of all registered server   */
/* programs is displayed.  If you only want information on specific  */
/* groups, supply those group names as parameters.                   */
/*                                                                   */
/*********************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <manifest.h>                /* not required in Open Edition */
#include <bsdtypes.h>                /* not required in Open Edition */
#include <socket.h>
#include <in.h>
#include <iwmwdnsh.h>    /* in /usr/lpp/tcpip/samples in Open Edition */

/*********************************************************************/
/* Specify defined values                                            */
/*********************************************************************/
#define CHARS_PER_LINE 16
#define WLMQ_MAX_SERVERS 10

/*********************************************************************/
/* Function prototypes for local fnctions                            */
/*********************************************************************/
static struct grpinfo_block * allocateGroupArray( int group_count );
static int  userDataEmpty( char * user_data );
static int  looksLikeIPAddresses( char * buffer );
static void showIPAddresses( char * buffer );
static void showTableHeader( void );
static void hexDumpBuffer( char * buffer, int length );

/* ----------------------------------------------------------------- */
/* Start of program                                                  */
/* ----------------------------------------------------------------- */
int main( int argc, char** argv )
{
    long diagCode = 0;                    /* Diagnostic code        */
    long group_count;                     /* Num groups registered  */
    long server_count = WLMQ_MAX_SERVERS;  /* Max servers acceptable */
    int i, j;                             /* for loop counters      */
    int alreadyShownHeader = 0;           /* Flag used for output   */
    long rc = 0;                          /* rc for IWMDN* calls    */
    struct grpinfo_block * grp_array;      /* Ptr to array of groups */
    char * currentGroup;                  /* Working group pointer  */
    struct sysinfo_block  * currentServer;  /* Working server pointer */
    struct sysinfo_block sys_array[ WLMQ_MAX_SERVERS ];  /* Array of  */
                         /* structures describing server programs */

/*********************************************************************/
/* If present, extract the group names from the command line         */
/*********************************************************************/
    if( argc > 1 )
    {
      group_count = argc - 1;
      grp_array = allocateGroupArray( group_count );
      for( i = 1; i < argc; i++ )
      {
        strcpy( grp_array[i-1].cluster, argv[i] );
      }
    }
/**********************************************************************/
/* Else find out how many groups there are and query the list         */
/**********************************************************************/
    else
    {
      group_count = 0;                  /* Force the 0x040a diagCode */
      rc = IWMDNGRP( grp_array,
                     &group_count,
                     &diagCode
                    );

/**********************************************************************/
/* We expect a diagCode of 0x040a (IwmRsnCodeOutputAreaTooSmall) as   */
/* we set group_count to 0 before the call.  group_count will contain */
/* the number of registered groups after the call has completed       */
/**********************************************************************/
      if( diagCode == 0x040a )
      {
/**********************************************************************/
/* Allocate space for 2 extra groups to allow for new groups          */
/* registering between the IWMDNGRP calls                             */
/**********************************************************************/
        group_count += 2;
        grp_array = allocateGroupArray( group_count );

        rc = IWMDNGRP( grp_array,
                       &group_count,
                       &diagCode
                     );
        if( rc )
        {
          fprintf( stderr, "IWMDNGRP failure, error = %d \n",
                   diagCode );
          exit( 2 );
        }
      }
/**********************************************************************/
/* Unexpected return from the first IWMDNGRP                          */
/**********************************************************************/
      else
      {
        fprintf( stderr, "IWMDNGRP failure, error = %d \n",
                 diagCode );
        exit( 3 );
      }
    }

/**********************************************************************/
/* Query the servers that are available for each group                */
/**********************************************************************/
    for( i = 0; i < group_count; i++ )
    {
      server_count = WLMQ_MAX_SERVERS;      /* Reset the input value  */

      currentGroup = grp_array[i].cluster;  /* Assign working pointer */
      rc = IWMDNSRV( currentGroup,
                     sys_array,
                     &server_count,
                     &diagCode
                   );

      if( rc )
      {
/**********************************************************************/
/* IWMDNSRV returned IwmRsnCodeNoServersRegistered                    */
/**********************************************************************/
        if( diagCode == 0x040b )
        {
          fprintf( stderr, "No servers registered for group '%s'\n",
                   currentGroup );
          continue;                         /* Process the next group */
        }
/**********************************************************************/
/* IWMDNSRV returned IwmRsnCodeOutputAreaTooSmall                     */
/**********************************************************************/
        else if( diagCode == 0x040a )
        {
          fprintf( stderr, "Too many servers registered for "
                   "group %s\n", currentGroup );
          fprintf( stderr, "Increase value of WLMQ_MAX_SERVERS to "
                   "at least %d and recompile\n", server_count );
          exit( 4 );
        }
/**********************************************************************/
/* IWMDNSRV returned something unexpected                             */
/**********************************************************************/
        else
        {
          fprintf( stderr, "IWMDNSRV failure, error = %d \n",
                   diagCode );
          exit( 5 );
        }
      }

/**********************************************************************/
/* Display information on each server in each group                   */
/**********************************************************************/
      for( j = 0; j < server_count; j++ )
      {
        if( alreadyShownHeader == 0 )
        {
          showTableHeader( );            /* Display the table header  */
          alreadyShownHeader = 1;        /* Only show the header once */
        }

        currentServer = &sys_array[j];   /* Assign working pointer    */

        printf( "%-2d   %-12.12s   %-8.8s   %-8.8s   %-8.8s   %d\n",
                i+1,
                currentGroup,
                currentServer->server,
                currentServer->host_nameid,
                currentServer->netid,
                currentServer->weight
```

```
              );

/**********************************************************************/
/* If user_data field is not empty, show the contents.                */
/**********************************************************************/
        if( ! userDataEmpty( currentServer->user_data ) )
        {
          printf( "UserData: " );

          if( looksLikeIPAddresses( currentServer->user_data ) )
          {
            showIPAddresses( currentServer->user_data );
          }
          else
          {
            hexDumpBuffer( currentServer->user_data, 64 );
          }
          printf( "\n" );
        }
      }
    }
    return 0;
}

/* ------------------------------------------------------------------ */
/* Function to allocate space for the group array                     */
/* ------------------------------------------------------------------ */
static struct grpinfo_block * allocateGroupArray( int group_count )
{
void * ptr;

  ptr = malloc( group_count * sizeof( struct grpinfo_block ) );
  if( ptr == NULL )
  {
    printf( "Couldn't allocate space for %d groups\n",
            group_count );
    exit( 1 );
  }
  return (struct grpinfo_block *)ptr;

}

/* ------------------------------------------------------------------ */
/* Function to see if there is anything in the user_data field        */
/* ------------------------------------------------------------------ */
static int userDataEmpty( char * user_data )
{
int k;

  for( k = 0; k < 64; k++ )
  {
    if( user_data[k] != '\0' )
      return 0;
  }
  return 1;
}

/* ------------------------------------------------------------------ */
/* Function to see if the user_data contains IP addresses             */
/* ------------------------------------------------------------------ */
```

```
static int looksLikeIPAddresses( char * buffer )
{
  if( buffer[0] == '\0' &&
      buffer[1] <  16   &&    /* 15 IP addresses can fit in user_data */
      buffer[2] == '\0' &&
      buffer[3] == '\0' )
  {
    return 1;
  }

  return 0;
}


/* ------------------------------------------------------------------ */
/* Function to display a list of dotted decimal IP addresses          */
/* ------------------------------------------------------------------ */
static void showIPAddresses( char * buffer )
{
int i;
int numAddresses;
int * anAddress = (int *)( &(buffer[4]) );

  numAddresses = buffer[1];     /* 2nd byte contains num of addresses */

  for( i = 0; i < numAddresses; i++ )
  {
    printf( "%-15.15s ", inet_ntoa( anAddress[i] ) );

    if( ( ( i+1 ) % 4 == 0 ) && i < numAddresses -1 )
      printf( "\n          " );
  }
}

/* ------------------------------------------------------------------ */
/* Function to display the column headings for the table of servers   */
/* ------------------------------------------------------------------ */
static void showTableHeader( void )
{
  printf( "-------------------------------------------"
          "-----------------\n" );
  printf( "#    Group          Server     HostName    "
          "NetId      Weight\n" );
  printf( "-------------------------------------------"
          "-----------------\n" );
}

/* ------------------------------------------------------------------ */
/* Function to dump a buffer in hex and string format                 */
/* ------------------------------------------------------------------ */
static void hexDumpBuffer( char * buffer, int buflen )
{
int i = 0;                                    /* loop counter         */
int j = 0;                                    /* another loop counter */
int  ch = 0, line_number = 0;
char line_text[CHARS_PER_LINE + 1];
int  chars_this_line = 0;
int  lines_printed   = 0;
int  page_number     = 1;
```

```
  do
  {
    chars_this_line = 0;

    printf( "\n%08X: ", line_number );

    while( ( chars_this_line < CHARS_PER_LINE ) &&
           ( ch < buflen ) )
    {
      if( ch % 2 == 0 )
        printf( " " );

      printf( "%02X", buffer[ch] );

      line_text[chars_this_line] =
              isprint( buffer[ch] ) ? buffer[ch] : '.';

      chars_this_line++;

      ch++;
      line_number++;
    }

/**********************************************************************/
/* pad with blanks to format the last line correctly                  */
/**********************************************************************/
    if( chars_this_line < CHARS_PER_LINE )
    {
      for( ; chars_this_line < CHARS_PER_LINE; chars_this_line++ )
      {
        if( chars_this_line % 2 == 0 )
          printf( " " );
        printf( "  " );
        line_text[chars_this_line] = ' ';
      }
    }

    line_text[chars_this_line] = '\0';

    printf( " '%s'", line_text );

    lines_printed ++;

    if( lines_printed == 32 )
    {
      lines_printed = 0;
      printf( "\n " );
    }
  }
  while( ch < buflen );
  printf( "\n" );
}
```

SOCSRVR single threading server

```
#include <manifest.h>          /* not required in Open Edition        */
#include <bsdtypes.h>          /* not required in Open Edition        */
#include <socket.h>
#include <in.h>
```

```
#include <stdio.h>

int main(int argc, char** argv)
{
    unsigned short port;       /* port server binds to                */
    struct sockaddr_in client; /* client address information          */
    struct sockaddr_in server; /* server address information          */
    char buf[256];             /* buffer for sending & receiving data */
    char hostName[64];         /* space to discover our hostname      */
    unsigned long hostId;      /* For the server's IP address         */
    int s;                     /* socket for accepting connections    */
    int ns;                    /* socket connected to client          */
    int namelen;               /* length of client name               */

    if (argc != 2)
    {
        fprintf(stderr, "Usage: %s port\n", argv[0]);
        exit(1);
    }

/**********************************************************************/
/* First argument should be the port.                                 */
/**********************************************************************/
    port = (unsigned short) atoi(argv[1]);

/**********************************************************************/
/* Extract our hostname                                               */
/**********************************************************************/
    if(gethostname(hostName, 64) !=0)
    {
        tcperror("Gethostname()");
        exit(2);
    }

/**********************************************************************/
/* Extract our IP address                                             */
/**********************************************************************/
    if((hostId = gethostid()) == 0)
    {
        tcperror("Gethostid()");
        exit(2);
    }

/**********************************************************************/
/* Get a socket for accepting connections.                            */
/**********************************************************************/
    if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        tcperror("Socket()");
        exit(3);
    }

/**********************************************************************/
/* Bind the socket to the server address.                             */
/**********************************************************************/
    server.sin_family = AF_INET;
    server.sin_port   = htons(port);
    server.sin_addr.s_addr = INADDR_ANY;

    if (bind(s, (struct sockaddr *)&server, sizeof(server)) < 0)
    {
        tcperror("Bind()");
        exit(4);
    }

/*********************************************************************/
/* Listen for connection requests - backlog of 255                   */
/*********************************************************************/
    if (listen(s, 255) != 0)
    {
        tcperror("Listen()");
        exit(5);
    }

/*********************************************************************/
/* This server will continually loop responding to inbound requests   */
/*********************************************************************/
    while(1)
    {
/*********************************************************************/
/* Accept the conversation                                           */
/*********************************************************************/
        namelen = sizeof(client);
        if ((ns = accept(s, (struct sockaddr *)&client,
             &namelen)) == -1)
        {
            tcperror("Accept()");
            exit(6);
        }
/*********************************************************************/
/* Send our IP address so the client knows who he connected to       */
/*********************************************************************/
        if (send(ns, (char*) &hostId, sizeof(hostId), 0) < 0)
        {
            tcperror("Send()");
            exit(7);
        }
/*********************************************************************/
/* Close the connection to the client and loop back to accept the    */
/* next one.                                                         */
/*********************************************************************/
        close(ns);
    }
}
```

SSOCCLNT sysplex sockets sample

```
#include <manifest.h>
#include <bsdtypes.h>
#include <socket.h>
#include <in.h>
#include <stdio.h>
#include <iwmwdnsh.h>
/*#include <netdb.h>*/

struct  hostent                 /* This structure is in netdb.h       */
{                               /* Included here due to header file   */
                                /* problems.                          */
    char    *h_name;            /* official name of host              */
    char    **h_aliases;        /* alias list                          */
    int     h_addrtype;         /* host address type                   */
    int     h_length;           /* length of address                   */
    char    **h_addr_list;      /* list of addresses from name server */
#define h_addr  h_addr_list[0]  /* address, for backward compatiblity */
};
struct hostent  *gethostbyname();

int main(int argc, char** argv)
{
    struct hostent *hostName;  /* Server's IP address                 */
    unsigned short port;       /* port server binds to                */
    int s;                     /* socket for accepting connections    */
    struct sockaddr_in server; /* server address information          */
    int type;                  /* type of cluster connection          */
    int typelen;               /* length of connection type           */
    char buf[256];             /* buffer for sending & receiving data */

    if (argc != 3)
    {
        fprintf(stderr, "Usage: %s hostname port\n", argv[0]);
        exit(1);
    }

/**********************************************************************/
/* First argument should be hostname. Use it to get server address.   */
/**********************************************************************/
    hostName = gethostbyname(argv[1]);
    if (hostName == (struct hostent *) 0)
    {
        tcperror("Gethostbyname()");
        exit(2);
    }

/**********************************************************************/
/* Second argument should be the port.                                */
/**********************************************************************/
    port = (unsigned short) atoi(argv[2]);

/**********************************************************************/
/* Create a socket.                                                   */
/**********************************************************************/
    if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        tcperror("Socket()");
        exit(3);
    }

/**********************************************************************/
/* Connect to the server.                                             */
/**********************************************************************/
    server.sin_family = AF_INET;
    server.sin_port   = htons(port);
    server.sin_addr.s_addr = *((unsigned long *)hostName->h_addr);

    if (connect(s, (struct sockaddr *)&server, sizeof(server)) < 0)
    {
        tcperror("Connect()");
        exit(4);
    }
```

```
/**********************************************************************/
/* Discover our socket cluster connection type                        */
/**********************************************************************/
    if (getsockopt(s, SOL_SOCKET, SO_CLUSTERCONNTYPE,
                   (char *)&type, &typelen) < 0)
    {
        tcperror("GetSockOpt()");
        exit(5);
    }
    if (!(type & SO_CLUSTERCONNTYPE_NOCONN))
    {
        printf("Connection Type : \n");
        if (type & SO_CLUSTERCONNTYPE_NONE)
          printf("\tNone\n");
        if (type & SO_CLUSTERCONNTYPE_INTERNAL)
          printf("\tInternal\n");
        if (type & SO_CLUSTERCONNTYPE_SAME_IMAGE)
          printf("\tSame Image\n");
        if (type & SO_CLUSTERCONNTYPE_SAME_CLUSTER)
          printf("\tSame Cluster\n");
    }

/**********************************************************************/
/* Recv IP address from the server.                                   */
/**********************************************************************/
    if (recv(s, (char*) &buf, sizeof(buf), 0) < 0)
    {
        tcperror("Recv()");
        exit(6);
    }
    printf("Server's IP address : %d.%d.%d.%d\n",
           buf[0], buf[1], buf[2], buf[3]);

/**********************************************************************/
/* Close the connection to the server.                                */
/**********************************************************************/
    close(s);
}
```

MTCSRVR multitasking sockets program

```
/*** IBMCOPYR ********************************************************/
/*                                                                    */
/* Component Name: MTCSRVR (alias EZAEC047)                           */
/*                                                                    */
/*                                                                    */
/* Copyright:    Licensed Materials - Property of IBM                 */
/*                                                                    */
/*               "Restricted Materials of IBM"                        */
/*                                                                    */
/*               5647-A01                                             */
/*                                                                    */
/*               (C) Copyright IBM Corp. 1977, 1998                   */
/*                                                                    */
/*               US Government Users Restricted Rights -              */
/*               Use, duplication or disclosure restricted by         */
/*               GSA ADP Schedule Contract with IBM Corp.             */
/*                                                                    */
/* Status:       CSV2R6                                              */
/*                                                                   */
/*  SMP/E Distribution Name: EZAEC049                                */
/*                                                                   */
/*                                                                   */
/*** IBMCOPYR ********************************************************/

/*********************************************************************/
/* C socket Server Program                                           */
/*                                                                   */
/* This code performs the server functions for multitasking, which   */
/* include                                                           */
/*      . creating subtasks                                          */
/*      . socket(), bind(), listen(), accept()                       */
/*      . getclientid                                                */
/*      . givesocket() to TCP/IP in preparation for the subtask      */
/*                      to do a takesocket()                         */
/*      . select()                                                   */
/*                                                                   */
/* There are three test tasks running:                               */
/*      . server master                                              */
/*      . server subtask - separate TCB within server address space  */
/*      . client                                                     */
/*                                                                   */
/*********************************************************************/

static char ibmcopyr()=
   "MTCSRVR - Licensed Materials - Property of IBM. "
   "This module is \"Restricted Materials of IBM\" "
   "5647-A01 (C) Copyright IBM Corp. 1994, 1996. "
   "See IBM Copyright Instructions.";

#include <manifest.h>
#include <bsdtypes.h>
#include <in.h>
/* #include <netdb.h> */
#include <socket.h>
#include <inet.h>
#include <fcntl.h>
#include <errno.h>
#include <tcperrno.h>
#include <bsdtime.h>
#include <mtf.h>
#include <stdio.h>

int  dotinit(int numsubs);
void getsock(int *s);
int  dobind(int *s, unsigned short port);
int  dolisten(int *s);
int  getname(char *myname, char *mysname);
int  doaccept(int *s);
int  testgive(int *s);
int  dogive(int *clsocket, char *myname);

/*
 * Server Main.
 */
int main(int argc, char **argv)
{
    unsigned short port;       /* port server for bind                */
```

```
int s;                     /* socket for accepting connections    */
int rc;                    /* return code                         */
int count;                 /* counter for number of sockets       */
int clsocket;              /* client socket                       */
char myname[8];            /* 8 char name of this address space   */
char mysname[8];           /* my subtask name
int numsubtasks;           /* Number of subtasks                       */

/*
 * Check arguments. Should be only one: the port number to bind to.
 * Added another, the number of subtasks.
 */
if (argc != 3) {
    fprintf(stderr, "Usage: %s port subtasks\n", argv[0]);
    exit(1);
}

/*
 * First argument should be the port.
 */
port = (unsigned short) atoi(argv[1]);
fprintf(stdout, "Server: port = %d \n", port);
/*
 * Second argument should be the number of subtasks.
 */
numsubtasks = atoi(argv[2]);
fprintf(stdout, "Server: numsubtasks = %d \n", numsubtasks);
/*
 * Create subtasks
 */
rc = dotinit(numsubtasks);
if (rc < 0)
    perror("Srvr: error for tinit");
printf("rc from tinit is %d\n", rc);

getsock(&s);
printf("Srvr: socket = %d\n", s);

rc = dobind(&s, port);
if (rc < 0)
    tcperror("Srvr: error for bind");
printf("Srvr: rc from bind is %d\n", rc);

rc = dolisten(&s);
if (rc < 0)
    tcperror("Srvr: error for listen");
printf("Srvr: rc from listen is %d\n", rc);

/***************************************
 * To do nonblocking mode,
 *  uncomment out this code.
 *
rc = fcntl(s, F_SETFL, FNDELAY);
if (rc != 0)
    tcperror("Error for fcntl");
printf("rc from fcntl is %d\n", rc);

***************************************/

rc = getname(myname, mysname);
```

```
    if (rc < 0)
        tcperror("Srvr: error for getclientid");
    printf("Srvr: rc from getclientid is %d\n", rc);

    /*----------------------------------------------------------------*/
    /* . issue accept(), waiting for client connection               */
    /* . issue givesocket() to pass client's socket to TCP/IP        */
    /* . issue select(), waiting for subtask to complete takesocket() */
    /* . close our local socket associated with client's socket      */
    /* . loop on accept(), waiting for another client connection     */
    /*----------------------------------------------------------------*/
    rc    = 0;
    count = 0;          /* number of sockets */
    while (rc == 0) {
        clsocket = doaccept(&s);
        printf("Srvr: clsocket from accept is %d\n", clsocket);
        count = count + 1;
        printf("Srvr: ###number of sockets is %d\n", count);
        if (clsocket != 0) {
            rc = dogive(&clsocket, myname);
            if (rc < 0)
                tcperror("Srvr: error for dogive");
            printf("Srvr: rc from dogive is %d\n", rc);
            if (rc == 0) {
                rc = tsched(MTF_ANY,"csub", &clsocket,
                                myname, mysname);
                if (rc < 0)
                    perror("error for tsched");
                printf("Srvr: rc from tsched is %d\n", rc);

                rc = testgive(&clsocket);
                printf("Srvr: rc from testgive is %d\n", rc);
             /* sleep(60);  *** do simplified situation first ***/
                printf("Srvr: closing client socket %d\n", clsocket);
                rc = close(clsocket);   /* give back this socket */
                if (rc < 0)
                    tcperror("error for close of clsocket");
                printf("Srvr: rc from close of clsocket is %d\n", rc);
                /****************************************************/
             /* exit(0);  *** do  this simplified situation first ***/
                /****************************************************/
            } /** end of if (rc == 0)       ****/
        }   /**** end of if (clsocket != 0) ****/
    }   /******** end of while (rc == 0)    ****/
}   /************ end of main           ********/

/*--------------------------------------------------------------------*/
/*    dotinit()                                                       */
/*    Call tinit() to ATTACH subtask and fetch() subtask load module  */
/*--------------------------------------------------------------------*/
int dotinit(int numsubs)
{
    int rc;
 /* int numsubs = 1; */
    printf("Srvr: calling __tinit\n");
    rc = __tinit("mtccsub", numsubs);
    return rc;
}

/*--------------------------------------------------------------------*/
```

```
/*    getsock()                                                       */
/*    Get a socket                                                    */
/*--------------------------------------------------------------------*/
void getsock(int *s)
{
    int temp;
    temp = socket(AF_INET, SOCK_STREAM, 0);
    *s = temp;
    return;
}

/*--------------------------------------------------------------------*/
/*    dobind()                                                        */
/*    Bind to all interfaces                                          */
/*--------------------------------------------------------------------*/
int dobind(int *s, unsigned short port)
{
    int rc;
    int temps;
    struct sockaddr_in tsock;
    memset(&tsock, 0, sizeof(tsock));   /* clear tsock to 0's */
    tsock.sin_family      = AF_INET;
    tsock.sin_addr.s_addr = INADDR_ANY; /* bind to all interfaces */
    tsock.sin_port        = htons(port);

    temps = *s;
    rc = bind(temps, (struct sockaddr *)&tsock, sizeof(tsock));
    return rc;
}

/*--------------------------------------------------------------------*/
/*    dolisten()                                                      */
/*    Listen to prepare for client connections.                       */
/*--------------------------------------------------------------------*/
int dolisten(int *s)
{
    int rc;
    int temps;
    temps = *s;
    rc = listen(temps, 10);     /* backlog of 10 */
    return rc;
}

/*--------------------------------------------------------------------*/
/*    getname()                                                       */
/*    Get the identifiers by which TCP/IP knows this server.          */
/*--------------------------------------------------------------------*/
int getname(char *myname, char *mysname)
{
    int rc;
    struct clientid cid;
    memset(&cid, 0, sizeof(cid));
    rc = getclientid(AF_INET, &cid);
    memcpy(myname,  cid.name,        8);
    memcpy(mysname, cid.subtaskname, 8);
    return rc;
}

/*--------------------------------------------------------------------*/
/*    doaccept()                                                      */
/*    Select() on this socket, waiting for another client connection. */
/*    If connection is pending, issue accept() to get client's socket */
/*--------------------------------------------------------------------*/
int doaccept(int *s)
{
    int temps;
    int clsocket;
    struct sockaddr clientaddress;
    int addrlen;
    int maxfdpl;
    struct fd_set readmask;
    struct fd_set writmask;
    struct fd_set excpmask;
    int rc;
    struct timeval time;

    temps = *s;
    time.tv_sec  = 1000;
    time.tv_usec = 0;
    maxfdpl = temps + 1;

    FD_ZERO(&readmask);
    FD_ZERO(&writmask);
    FD_ZERO(&excpmask);

    FD_SET(temps, &readmask);

    rc = select(maxfdpl, &readmask, &writmask, &excpmask, &time);
    printf("Srvr: rc from select is %d\n", rc);
    if (rc < 0) {
        tcperror("error from select");
        return rc;
    }
    else if (rc == 0) {  /* time limit expired */
        return rc;
    }
    else {              /* this socket is ready */
        addrlen = sizeof(clientaddress);
        clsocket = accept(temps, &clientaddress, &addrlen);
        return clsocket;
    }
}

/*--------------------------------------------------------------------*/
/*    testgive()                                                      */
/*    Issue select(), checking for an exception condition, which      */
/*    indicates that takesocket() by the subtask was successful.      */
/*--------------------------------------------------------------------*/
int testgive(int *s)
{
    int temps;
    struct sockaddr clientaddress;
    int addrlen;
    int maxfdpl;
    struct fd_set readmask;
    struct fd_set writmask;
    struct fd_set excpmask;
    int rc;
    struct timeval time;
    temps = *s;
    time.tv_sec  = 1000;
    time.tv_usec = 0;
    maxfdpl = temps + 1;

    FD_ZERO(&readmask);
    FD_ZERO(&writmask);
    FD_ZERO(&excpmask);

 /* FD_SET(temps, &readmask); */
 /* FD_SET(temps, &writmask); */
    FD_SET(temps, &excpmask);

    rc = select(maxfdpl, &readmask, &writmask, &excpmask, &time);
    printf("Srvr: rc from select for testgive is %d\n", rc);
    if (rc < 0) {
        tcperror("Srvr: error from testgive");
    }
    else
        rc = 0;

    return rc;
}

/*--------------------------------------------------------------------*/
/*    dogive()                                                        */
/*    Issue givesocket() for giving client's socket to subtask.       */
/*--------------------------------------------------------------------*/
int dogive(int *clsocket, char *myname)
{
    int rc;
    struct clientid cid;
    int temps;

    temps = *clsocket;
    memset(&cid, 0, sizeof(cid));
    cid.domain = AF_INET;

    memcpy(cid.name,       myname,     8);
    memcpy(cid.subtaskname,"        ", 8);
    printf("Srvr: givesocket socket is %d\n", temps);
    printf("Srvr: givesocket name is %s\n", cid.name);

    rc = givesocket(temps, &cid);
    return rc;
}
```

MTCSUBT subtask for the multitasking sockets program

```
#pragma runopts(noargparse,plist(mvs),noexecops)

#include <manifest.h>
#include <bsdtypes.h>
#include <in.h>
#include <socket.h>
#include <inet.h>
#include <fcntl.h>
#include <errno.h>
#include <tcperrno.h>
```

```
#include <bsdtime.h>
#include <stdio.h>

/*
 * Server subtask
 */
csub(int *clsock,         /* address of socket passed  */
     char *tskname,       /* address of caller's name  */
     char *tsksname)      /* address of caller's sname */
{
    struct clientid cid;       /* Information needed to take socket   */
    int socket;                /* socket taken                        */
    int sendbytes;             /* # bytes sent                        */
    int recvbytes;             /* # bytes received                    */
    unsigned long hostId;      /* For the server's IP address         */
    char data[1];
    int sleeptime;

/*********************************************************************/
/* Take the socket given by the server.                              */
/*********************************************************************/
    memset(&cid, 0, sizeof(cid));
    memcpy(cid.name,        tskname,  8);
    memcpy(cid.subtaskname, tsksname, 8);
    cid.domain = AF_INET;

    socket = takesocket(&cid, *clsock);
    if (socket < 0)
    {
        tcperror("Csub: Error from takesocket");
    }
    else
    {
/*********************************************************************/
/* Receive data from the client.  This will be a time in tenths of   */
/* seconds to sleep before closing the socket.  Perform the sleep.   */
/*********************************************************************/
        recvbytes = recv(socket, data, sizeof(data), 0);
        if (recvbytes < 0)
        {
            tcperror("Csub: Recv()");
        }
        else
        {
            printf("Sleeping for %d seconds\n", (*data)/10 );
            sleeptime = (*data) / 10;
            sleep(sleeptime);
        }
/*********************************************************************/
/* Extract our IP address                                            */
/*********************************************************************/
        if((hostId = gethostid()) == 0)
        {
            tcperror("Csub: Gethostid()");
        }
/*********************************************************************/
/* Send our IP address so the client knows who he connected to       */
/*********************************************************************/
        sendbytes = send(socket, (char*) &hostId, sizeof(hostId), 0);
        if (sendbytes < 0)
        {
            tcperror("Csub: Send()");
        }
/***********************************************************************/
/* Close the socket.                                                   */
/***********************************************************************/
        close(socket);
    }
    fflush(stdout);
}
```

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 308.

- *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 1: Base and TN3270 Configuration,* SG24-5227
- *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 2: UNIX Applications,* SG24-5228
- *OS/390 eNetwork Communications Server for V2R7 TCP/IP Implementation Guide Volume 3: MVS Applications*, SG24-5229
- *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 4: Connectivity and Routing*, SG24-6516
- *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 6: Policy and Network Management*, SG24-6839
- *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 7: Security*, SG24-6840
- *TCP/IP in a Sysplex*, SG24-5235
- *Managing OS/390 TCP/IP with SNMP*, SG24-5866
- *Secure e-business in TCP/IP Networks on OS/390 and z/OS*, SG24-5383
- *TCP/IP Tutorial and Technical Overview*, GG24-3376
- *Migrating Subarea Networks to an IP Infrastructure Using Enterprise Extender*, SG24-5957
- *Networking with z/OS and Cisco Routers: An Interoperability Guide*, SG24-6297
- *zSeries HiperSockets*, SG24-6816

Other resources

These publications are also relevant as further information sources:

- *z/OS V1R2.0 UNIX System Services Planning,* GA22-7800
- *z/OS V1R2.0 UNIX System Services User’s Guide*, GA22-7801
- *z/OS V1R1.0-V1R2.0 MVS Initialization and Tuning Guide*, SA22-7591
- *z/OS V1R2.0 C/C++ Programming Guide*, SC09-4765
- *z/OS V1R2.0 C/C++ Run-Time Library Reference*, SA22-7821
- *z/OS V1R2.0 CS: IP Migration*, GC31-8773
- *z/OS V1R2.0 CS: IP Configuration Guide*, SC31-8775
- *z/OS V1R2.0 CS: IP Configuration Reference*, SC31-8776
- *z/OS V1R2.0 CS: IP User’s Guide and Commands*, SC31-8780

- *z/OS V1R2.0 CS: IP System Administrator’s Commands*, SC31-8781
- *z/OS V1R2.0 CS: IP Diagnosis*, GC31-8782
- *z/OS V1R2.0 CS: IP Messages Volume 1 (EZA)*, SC31-8783
- *z/OS V1R2.0 CS: IP Messages Volume 2 (EZB)*, SC31-8784
- *z/OS V1R2.0 CS: IP Messages Volume 3 (EZY)*, SC31-8785
- *z/OS V1R2.0 CS: IP Messages Volume 4 (EZZ-SNM)*, SC31-8786
- *z/OS V1R2.0 CS: IP Application Programming Interface Guide*, SC31-8788

Referenced Web sites

These Web sites are also relevant as further information sources:

- z/OS UNIX Performance

  `http://www.s390.ibm.com/oe/bpxa1tun.html`
- The z/OS Web pages

  `http://www-1.ibm.com/servers/eserver/zseries/zos/installation/installz12.html`

How to get IBM Redbooks

You can order hardcopy Redbooks, as well as view, download, or search for Redbooks at the following Web site:

ibm.com/redbooks

You can also download additional materials (code samples or diskette/CD-ROM images) from that site.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

Index

IBM

Redbooks

Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 5: Availability, Scalability, and Performance

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

Communications Server for z/OS V1R2 TCP/IP Implementation Guide

Volume 5: Availability, Scalability, and Performance

Covers load balancing with DNS/WLM, Sysplex Distributor and MNLB

Describes high availability with dynamic VIPA scenarios

Details z/OS TCP/IP settings to increase performance

The Internet and enterprise-based networks have led to a rapidly increasing reliance upon TCP/IP implementations. The zSeries platform provides an environment in which critical business applications flourish. The demands placed on these systems are growing and require a solid, scalable, highly available, and highly performing operating system and TCP/IP component. z/OS and Communications Server for z/OS provide for such a requirement with a TCP/IP stack that is robust and rich in functionality. The *Communications Server for z/OS TCP/IP Implementation Guide* series provides a comprehensive, in-depth survey of CS for z/OS.

Volume 5 concentrates on the availability and scalability of z/OS TCP/IP. We cover load-balancing solutions including DNS/WLM, Sysplex Distributor, and the preferred Sysplex Distributor/MNLB joint solution. We further describe mechanisms by which availability is increased on z/OS systems with dynamic VIPA and Automatic VIPA Takeover. Finally, we provide a survey of tuning exercises that can be employed to further enhance the performance of your z/OS TCP/IP system.

Because of the varied scope of CS for z/OS, this volume is not intended to cover all aspects of it. The main goal of this volume is to provide an insight into the different functions available in CS for z/OS to increase availability, scalability, and performance through the use of VIPAs, load-balancing mechanisms, and performance tuning. For more information, including applications available with CS for z/OS IP, please refer to the other volumes in the series.

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

SG24-6517-00 ISBN 0738424161