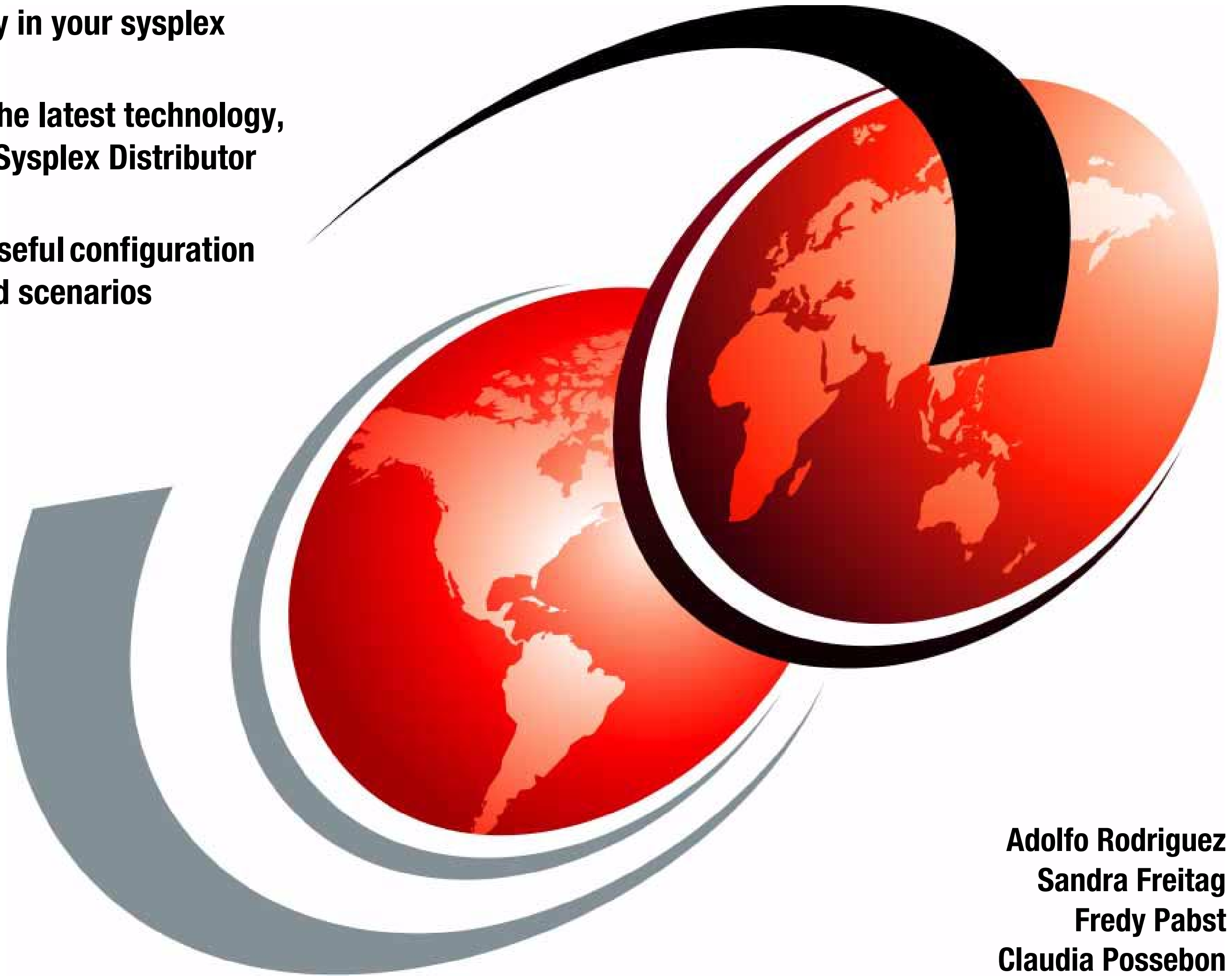
TCP/IP in a Sysplex

Implement load balancing and high availability in your sysplex

Contains the latest technology, including Sysplex Distributor

Includes useful configuration details and scenarios



Adolfo Rodriguez
Sandra Freitag
Fredy Pabst
Claudia Possebon

Redbooks

ibm.com/redbooks

International Technical Support Organization

SG24-5235-02

TCP/IP in a Sysplex

March 2001

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix F, “Special notices” on page 315.

Third Edition (March 2001)

This edition applies to Version 2 Release 10 of IBM Communications Server for OS/390.

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

Preface

The sysplex environment provides a unique setting for the creation of high performing application servers. Quite simply, the main goals of a sysplex are high availability and load balancing. That is, the sysplex provides the ability to maintain a service as highly available and the ability to add resources so that service performance can scale with growing number of client requests. This redbook demonstrates how these goals can be achieved in the particular environment of IBM Communications Server for OS/390 and the TCP/IP applications that function with it.

In this redbook, we describe the objectives of a sysplex and include a discussion of three sysplex-specific solutions that help to meet these demands. All of these three solutions make use, to some extent, of the MVS Workload Manager (WLM). Because of this, we describe the benefits of WLM-aware solutions and include source application code that can be used to work with WLM. Additionally, because the sysplex notion of high availability is so closely tied together with the Virtual IP Addressing (VIPA) concept, we discuss in detail the advantages of VIPA and the necessary configuration that needs to take place when using VIPA. This includes a detailed routing discussion as we deal with VIPAs in the sysplex.

We highlight the solution discussion with the description of Sysplex Distributor, a new function available on the sysplex as of IBM Communications Server for OS/390 V2R10 IP. Sysplex Distributor is the state-of-the-art technology of achieving efficient load balancing and high availability within the sysplex. We compare the issues associated with Sysplex Distributor with those of the Domain Name System solution called DNS/WLM and the Network Dispatcher solution. We provide detailed descriptions of these as well.

To this end, this redbook will help you design your OS/390-based IP network to gain the maximum benefit from the features available within the sysplex to achieve the stringent high availability and load balancing demands placed on your OS/390 servers.

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 an Advisory 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 Communications Server for OS/390, in RTP, NC. He holds a B.A. degree in Mathematics and a B.S. degree in Computer Science, both from Duke University, Durham, NC. He is currently pursuing the Ph.D. degree in Computer Science at Duke University, with a concentration on Networking Systems.

Claudia Possebon is a Services Specialist in IBM Brazil. She has seven years of experience in the networking field in IBM. She holds a degree in Computer Science from Pontifica Universidade Catolica in Sao Paulo. Her areas of expertise include TCP/IP and VTAM.

Sandra Freitag is a Services Specialist in IBM Brazil. She has eight years of experience in networking field. Her areas of expertise include SNA and TCP/IP in OS/390 and VM environments.

Fredy Pabst is an Advisory IT Specialist with CS/390 in IBM Switzerland. He has 11 years of experience in the networking field. He has worked with IBM for 10 years. His areas of expertise include APPN, HPR, APPC and TCP/IP.

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

Shawn Walsh, Gail Christensen, Tatsuhiko Kakimoto, Linda Robinson, Juan Rodriguez, George Baker, Byron Braswell
International Technical Support Organization, Raleigh Center

Tom Moore, Julie Krenzer, Dave Herr, Mike Law, Jay Aiken, Mac Devine, Alfred Christensen, Jeff Haggar, Mike Fitzpatrick, Bebe Israel
IBM CS for OS/390 Development

Comments welcome

Your comments are important to us!

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

- Fax the evaluation form found in “IBM Redbooks review” on page 325 to the fax number shown on the form.
- Use the online evaluation form found at **ibm.com**/redbooks
- Send your comments in an Internet note to redbook@us.ibm.com

Chapter 1. Introduction to TCP/IP in a sysplex

The increasing demands of network servers, and in particular S/390 servers, has led to 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, this cluster functionality is provided by the sysplex. That is, the sysplex provides the necessary capability to cluster together a number of S/390 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 functional descriptions of 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. We also make note of interesting issues regarding routing within the sysplex, including the role of dynamic routing.

The previous edition of this redbook 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 and Chapter 3, “Network Dispatcher” on page 63. The examples are still valid, but you should be aware that the details of the displays may have changed if you implement a later release than V2R8.

1.1 Sysplex objectives

Solutions utilizing the clustering approach to increase server availability and processing capability 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 OS/390 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, and Sysplex Distributor.

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 connections 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 for 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 techniques that provide load balancing: DNS/WLM, Network Dispatcher, and Sysplex Distributor. Each identifies the target S/390 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.

1.1.2.1 Single system-wide image

Clients connecting to a cluster should not be aware of the internal make-up 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 hostname to identify a service within the cluster. In this manner, clients making requests of the service use the hostname as the single system-wide identity. In Network Dispatcher (NDR) and Sysplex Distributor, however, the identity is that of some IP address associated with the cluster. In the case of Sysplex Distributor, this address is a distributed Virtual IP Address (VIPA).

1.1.2.2 Horizontal growth

As the clients' demands on the service increase, clusters must provide a way to expand the cluster of servers to accommodate for such growing demand. Put in 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 horizontally to accommodate for increased demand placed on the cluster by the clients.

1.1.2.3 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 S/390 servers. As such, many of the configurations can be shared among the different S/390 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 OS/390 (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 OS/390 system, yet retrieved more quickly than if it were on disk. Although the SNA component of CS for OS/390 makes use of the coupling facility to improve service to VTAM users, the IP component does not. 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 OS/390 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).

There is an excellent description of sysplexes in *OS/390 Parallel Sysplex Overview: Introducing Data Sharing and Parallelism in a Sysplex*, GC28-1860.

1.3 IBM Communications Server for OS/390

IBM Communications Server for OS/390 (formerly known as SecureWay Communications Server for OS/390 and OS/390 eNetwork Communications Server) is the communications engine of OS/390. It is comprised of IP (TCP/IP), SNA (VTAM), and AnyNet components. With CS for OS/390, TCP/IP and SNA are very closely integrated in the OS/390 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 OS/390 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 *OS/390 IBM Communications Server: IP Migration*, SC31-8512.

1.4 Network interfaces to the sysplex

One of the major ways in which the SNA and IP components of CS for OS/390 (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 OS/390. The DLC connection manager implements the following DLCs:

- Multipath Channel Plus (MPC+)
- Asynchronous Transfer Mode (ATM)
- XCF (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

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 S/390 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 OS/390 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 work load from the *dispatching agent*, the function that routes connections 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*, which is a recent improvement to the way WLM operates. For more information on how WLM functions, see *OS/390 MVS Planning: Workload Management*, GC28-1761. For more information specific to CS for OS/390 IP, see *OS/390 IBM Communications Server: IP Configuration Guide*, SC31-8725.

1.6 Cross-system coupling facility

The OS/390 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 OS/390 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 OS/390 IP can use XCF signalling for communication and coordination between the stacks themselves. The dynamic VIPA function (see Chapter 4, “Dynamic VIPA (for application instance)” on page 101), 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 OS/390 IP then makes it appear to the IP network that the VIPA address is on a separate subnetwork, and that CS for OS/390 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, or Sysplex Distributor 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 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, or Sysplex Distributor. 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 hostnames 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 in an IP network is also highly dependent, not so much on the transport network between servers and clients (because IP can reroute packets), 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. VIPA and IP routing are explained in detail later in this book.

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.

1.8.1 DNS/WLM solution

The DNS solution is based on the DNS name server and the OS/390 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 an OS/390 sysplex, the name server can utilize WLM to determine the best system to service a given client request.

In general, DNS/WLM relies on the hostname 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 hostname. Note that the system most suitable to receive an incoming client connection is determined only at the time of hostname resolution. Once the connection is made, the system being used cannot be changed without restarting the connection.

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 servers register with WLM under a particular *server group name* for the purpose of providing some service described by this name (for example, TN3270).
- At regular intervals, DNS asks WLM for its workload measurements.
- The client requests a server by IP *hostname* via a process called *hostname resolution*. This hostname is the server group name known to WLM.
- The DNS name server in the sysplex checks its WLM information for details of the servers 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 hostname resolution, and addresses obtained via WLM are no exception. 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 DNS. This means that dead applications will not be selected by 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 OS/390. Additionally, the technique of simply modifying hostname 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 an OS/390 system; it is just a matter of configuration. That is, the DNS server is included with the OS/390 operating system as part of CS for OS/390 IP.

- Performance

  Once the hostname 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 hostname and use the IP address returned on the hostname 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, this support is somewhat limited.

- Users can bypass load balancing

  Users that know the real hostnames 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 hostname-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 hostnames on every connection request.

1.8.2 Network Dispatcher

The Network Dispatcher is load balancing software that uses technology from IBM's Research Division to determine the most appropriate server to receive each new IP connection.

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. Read Chapter 3, “Network Dispatcher” on page 63 for a more detailed description of the Network Dispatcher.

The Network Dispatcher (NDR) technique does not depend on hostnames; 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. NDR 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. NDR uses all the information obtained from the advisors to select a server.

With NDR, 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 NDR. 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 NDR 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 NDR 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 NDR that remains ready to take over if the primary fails.

- Easy integration

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

- Flexibility

  The NDR 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

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

Possible disadvantages of NDR 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. NDR 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 NDR machine must maintain knowledge of the TCP connections to the servers in the cluster and thus requires more capacity than a simple router.

- Restricted design flexibility

  The Network Dispatcher must be located precisely one hop away from the target servers, although there is a way to cascade NDRs over a wide area. Also, adding new images, which is normally possible without any new network connections or definitions, will now require connections and definitions in the Network Dispatcher. The one hop restriction is expected to be lifted with the introduction of Generic Routing Encapsulation (GRE) into Network Dispatcher.

- Incompatibility with shared OSA cards

  The use of an OSA with EMIF requires the destination IP addresses and LPAR numbers to be associated in the OSA configuration. If the destination address is the same (the cluster address) in multiple LPARs, such an association is impossible. Multiple OSA cards may be required. In general, sharing OSA cards across multiple LPARs that are target servers causes challenging problems. This problem, however, is expected to be resolved with GRE.

- 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 limited

  Because FTP connection flow is somewhat different from the traditional client/server application flow, NDR 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 S/390 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 stacks of this server load so that the distributing stack 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 NDR, connection requests are directed to the distributed stack of Sysplex Distributor. The stack selects which target server is the best candidates 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 servers within the sysplex 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 *non-disruptively* 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 now 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

  The end users are not aware of the distribution being performed by Sysplex Distributor; they just connect to a new server IP address (or hostname).

- Independence from DNS

  Sysplex Distributor does not depend on hostname resolution for load balancing. Rather, the workload distribution occurs at TCP connection setup, as with NDR. As a result, Sysplex Distributor is not susceptible to hostname 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 OS/390 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.

Possible disadvantages of Sysplex Distributor may be:

- The cluster *is* the sysplex

  In Sysplex Distributor, all target servers must be S/390 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 NDR.

- Incompatibility with IPSec and VPN

  As with NDR, the end of an IPSec tunnel mode must not be a target server. This is hardly much of a limitation, but worth mentioning.

- 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.

1.8.4 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 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, 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, although 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 hostname 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. However, be aware that:
  - OSA with EMIF will not work (this should be corrected with GRE).
  - The Network Dispatcher must be just one hop from *every* server in the cluster (this should also be corrected with GRE).
- 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 NDR function.

Chapter 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 an OS/390 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 balancing, for example. 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 hostnames when accessing an application in the sysplex. The name server is the only place where address selection based upon resource availability can occur. If you use an IP address directly, no workload balancing can occur with the DNS/WLM solution.

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 hostnames to host numbers or addresses. The obvious advantage of this name-to-IP address mapping is that we can assign easily rememberable 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 hostnames and IP addresses. Essentially, the increased importance of hostnames and the size of the Internet hosts file led to the creation of the DNS. It is now the method of choice for resolving hostnames 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 (".").

2.1.2.1 Controlling the names

The uniqueness of hostnames 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.

2.1.2.2 Finding an address

Hosts on a network must be configured to look for a DNS server when a hostname 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 point to hosts running a DNS server. In the MVS 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.

2.1.2.3 Reverse mappings

Sometimes, we might already know an IP address, but we want to find the hostname 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:

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

2.1.2.4 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 OS/390 IP

The Domain Name System (DNS) server delivered with CS for OS/390 V2R5 and later has been completely rewritten. This implementation is now a BIND (Berkeley Internet Name Domain) 4.9.3-based name server. The requirement for DB2 tables has been completely removed, providing appeal to a wider range of customers. In addition, a cache file (root name server data file) can now be defined. The CS for OS/390 name server is also now able, in conjunction with WLM, to provide workload balancing across a group of equivalent server applications.

IBM Communications Server for OS/390 IP provides a Domain Name System implementation based on Berkeley Internet Name Domain (BIND) 4.9.3 protocols. DNS itself is not a new offering in the MVS TCP/IP environment; one that uses DNS files stored as DB2 data has been available in prior releases of TCP/IP. What is new with the BIND DNS is the ability to implement an OS/390-based DNS that is similar to de facto standard UNIX implementations of DNS in configuration and processing. The code for the BIND DNS has been ported from the BIND 4.9.3 level; for OS/390 systems the only additions required to the de facto standard code have been as follows:

- The provision of translate tables for ASCII-to-EBCDIC translation
- The high availability and load balancing functions provided by the DNS's cooperation with WLM in a sysplex environment

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 IBM Communications Server for OS/390.

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. This support is analogous to the support provided in SNA networks with VTAM Generic Resources, but there are many differences in the implementation of the two approaches. Nevertheless, the intent to provide connection (or session) optimization is an inherent part of both implementations.

The BIND DNS of CS for OS/390 IP includes full dynamic IP function, which allows the automatic registration of DNS clients. Dynamic IP involves cooperation between the Dynamic Host Configuration Protocol (DHCP) and the name server for the dynamic update of DNS tables, thus eliminating the labor-intensive administrative task of manually updating those tables. However, a domain cannot be configured as both a sysplex domain and a dynamic IP domain.

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 hostnames to IP addresses for specific domains. We refer to this zone file as the *forward domain* file, because 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 hostnames 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 hostname 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 1 on page 23):

1. When each application becomes active in the sysplex, it registers with WLM using the appropriate group, server, and host (stack) name.
2. The sysplex-enabled 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).

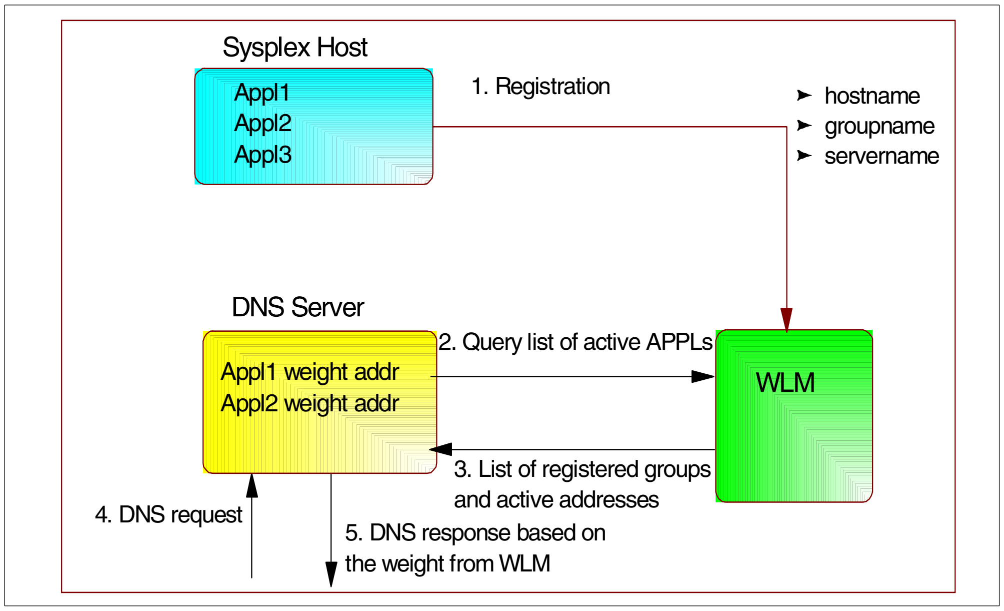


Figure 1. WLM and name server working together

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 hostname 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 hostname. 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.

2.2.1.1 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 see can see below in Figure 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. Partial output of debug trace with WLM weight for MVS images

2.2.1.2 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.

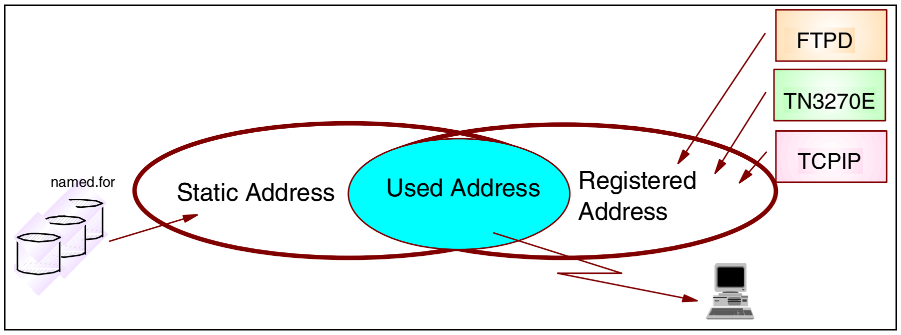


Figure 3. DNS addresses used in WLM distribution

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.

2.2.1.3 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 OS/390 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.

2.2.1.4 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 an OS/390 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 hostname 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 hostname. 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 hostname 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.

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.

- Hostname

  The TCP/IP hostname 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 hostname; that is, all the interfaces that have been successfully activated and have an address assigned via the HOME statement in the TCP/IP profile.

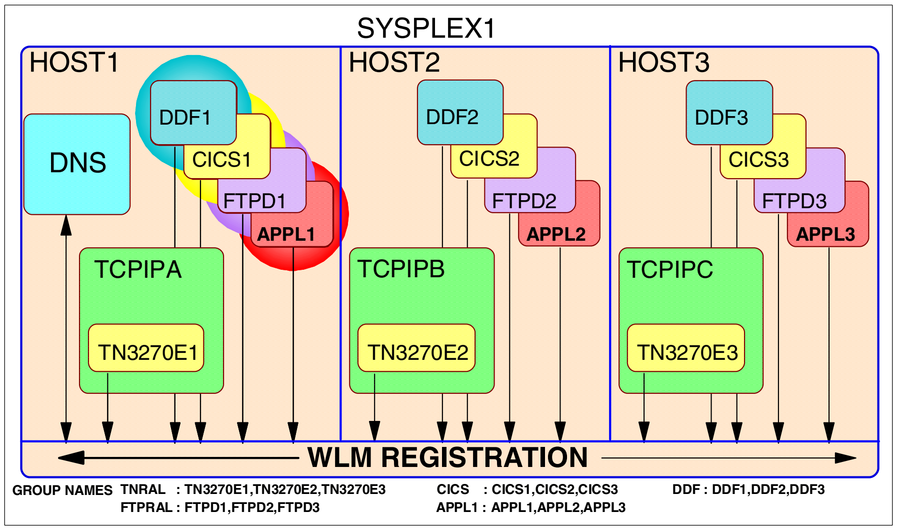


Figure 4. Application and stack registration to WLM

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 OS/390 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 CS for OS/390 V2R10 IP application support

Several applications shipped with CS for OS/390 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 OS/390 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. In DB2, registration supported is available with APAR PQ07417.

2.4.2.1 TN3270 configuration

```
TELNETPARMS
   WLMCLUSTERNAME     2
       TN03           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.

2.4.2.2 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.

2.4.2.3 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 A, “Sample applications and programs” on page 253 contains the description of a sample application that is capable of registering with WLM. The source code for this application is included in Appendix B, “Sample C program source code” on page 267.

For information on how to code the assembler macro, see *OS/390 MVS Programming: Workload Management Services*, GC28-1773.

2.4.3 DNS/WLM registration results

Unfortunately there is no generic query available to determine from OS/390 WLM the names of resources registered with it. Some applications issue messages about a successful registration like the FTP server:

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

Figure 5. FTP server registration with WLM

Other applications like Telnet have commands available to display the group names registered with WLM (see Figure 6 on page 29):

```
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 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 hostnames to verify the results. Another possibility to be certain if an application has registered is to dump the DNS database. Please review 2.5.6, “Dumping the DNS server cache” on page 36 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 7.

```
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 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 this: `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 28 on page 47.

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 OS/390-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 he connects to the same server he 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 OS/390 V2R10 IP in the OS/390 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 8.

```
//NAMED PROC B='/etc/named.boot',P='53'        2
//*************************************************************
//NAMED    EXEC PGM=EZANSNMD,REGION=0K,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 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 illustrative 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 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 9. Start DNS server in UNIX System Services shell environment

Table 1 lists additional startup parameter with a brief description:

Table 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 10 on page 34 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 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:

```
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 11. DNS messages in syslogd.log: file loading

You can see in Figure 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 will 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 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 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 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 13. Displaying process ID and DNS program

See Figure 14 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 only 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 51.

```
; 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 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 15 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 15. Name server statistics

2.5.8 Discovering signals available for process

Most of the signal numbers are documented in *OS/390 UNIX System Services Command Reference*, SC28-1892. If you want a list of the available signals in an environment, you may issue a `LIST` **1** command, as in Figure 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 16. kill -l: requesting a list of available process signals

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

Table 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 36.

The documentation for issuing signals with the name server is in *OS/390 IBM Communications Server: IP Configuration Guide*, SC31-8725. In Figure 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 17. Running NSSIG procedure from ITSO Raleigh

2.5.9 Tracing the name server

As you can see in the screen output in Figure 18, 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 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 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 52.

```
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 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:

1. Use `-d 11` on the named start command or in your JCL.
2. 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** and **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 43 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 OS/390, 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 S/390 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 S/390, but that does not necessarily force you to use the S/390 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 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.

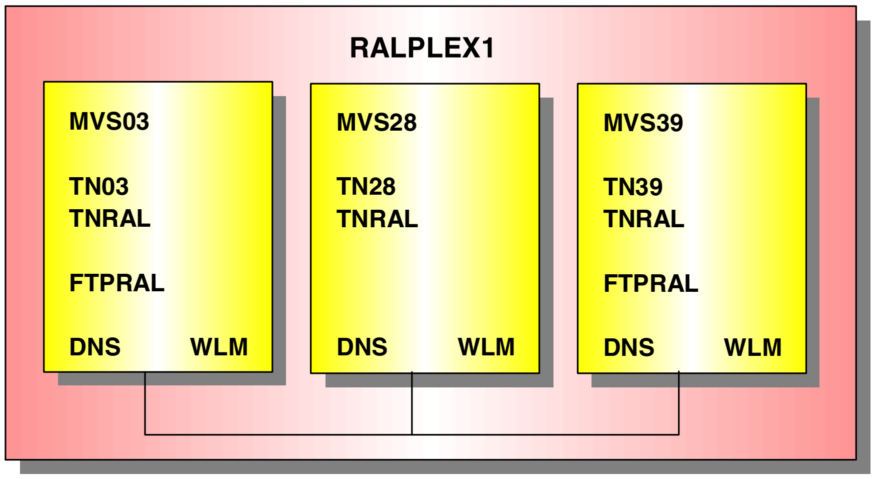


Figure 20. Telnet and FTP distribution in a sysplex

Figure 21 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. 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.

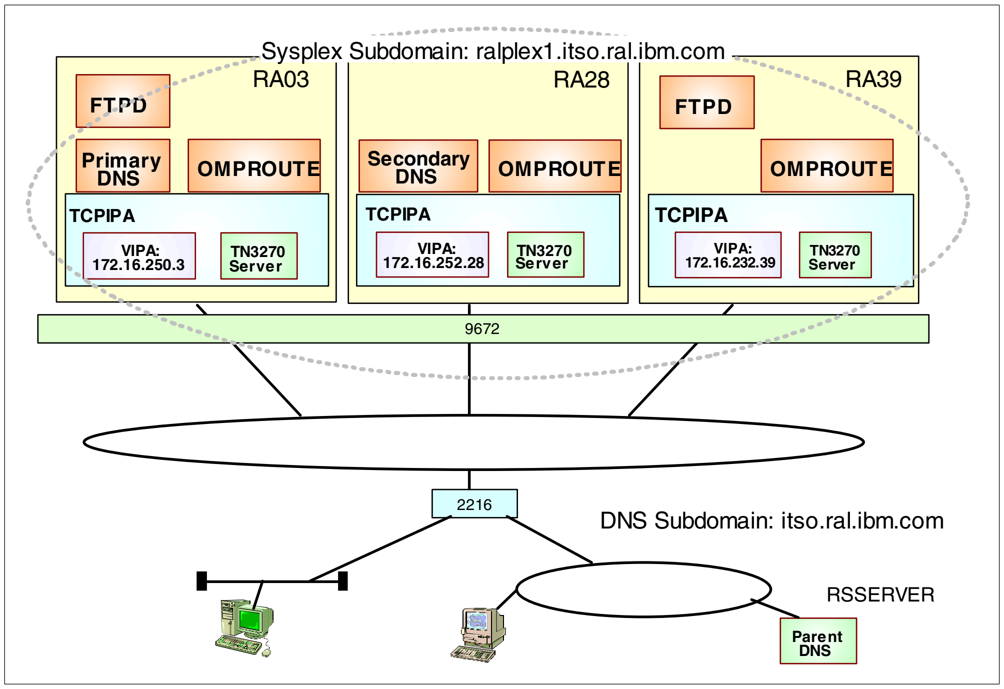


Figure 21. Environment at ITSO Raleigh

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 22 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 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 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 21 on page 43.

```
;  /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 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 24). 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 24 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 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 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 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 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 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 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 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 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 application 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, the following:

Table 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

2.6.4.1 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 21 on page 43. 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 will send 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 C, “REXX EXECs to gather connection statistics” on page 287 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
```

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 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 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 36 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 E, "Dump of T28ATCP name server - single-path network" on page 313.

```
; 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 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:

1. The sysplex domain
2. The in-addr.arpa
3. The loopback

These three sections refer to the three records defined in named.boot on MVS03. Figure 22 on page 44 shows the corresponding boot records.

In Figure 31, 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 23 on page 44.

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 31. More data from the primary DNS server

Figure 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 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 accordingly updated. 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 27.

```
$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 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 33 for this trace output. Please 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 33. WLM data from DNS trace part 1 (querying group names)

In Figure 34, 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 34. WLM data from DNS trace part2 (assigned WLM weight to OS/390 images)

In Figure 35 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 even load balance.

```
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 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 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 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 37 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 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 38 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 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 39 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 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 27 for how to register your own application.

On the client side we used the client test program described in C.2, “EXEC to connect to server using TCP” on page 290 to perform our tests. The EXEC will connect to a server on a given hostname/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 40 shows the results for the client with the external name server, and Figure 41 on page 60 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 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 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 42 shows how the DNS/WLM dealt with the failure in the external name server case; Figure 43 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 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.

Now for the sysplex name server client:

```
     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 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.

Chapter 3. Network Dispatcher

In this chapter, we first present an introduction to the Network Dispatcher feature. We then provide some test configurations that we set up to demonstrate the load balancing capabilities of the Network Dispatcher.

The Network Dispatcher (NDR) function was originally implemented in IBM networking hardware such as the IBM 2216. These various networking hardware machines have been discontinued from production. As a result, the Network Dispatcher function is currently exclusively distributed as part of the IBM WebSphere Edge Server. Currently, this is available on the AIX, Windows NT, Solaris, and Red Hat Linux platforms.

Throughout this chapter, we show configuration examples for the IBM 2216 and Windows NT running IBM WebSphere Edge Servers. Even though the choice of platform for running Network Dispatcher has changed somewhat, the primary functionality has remained the same for the most part. As a result, if you are familiar with the Network Dispatcher function on the IBM 2216, implementing the function on another platform will be trivial.

3.1 Network Dispatcher overview

Network Dispatcher is a feature that optimizes the performance of servers by forwarding TCP/IP connection requests and datagrams to different servers within a group or *cluster*. Thus, it attempts to balance the traffic across all the *target servers* according to the load on them. The forwarding is transparent to the users and to applications. Network Dispatcher may be used in conjunction with server applications such as HTTP (Web), FTP, and Telnet; it can also be used for load balancing UDP traffic across a server group.

Network Dispatcher can help maximize the potential of a site by providing a flexible and scalable solution to peak-demand problems. During peak demand periods, Network Dispatcher can automatically find the optimal server to handle incoming requests.

The Network Dispatcher function does not use a DNS name server for load balancing. As a result, the system-wide image of the cluster is based on the IP address as opposed to the hostname. It balances traffic among servers through a unique combination of load distribution and management software. Network Dispatcher also can detect a failed server and forward traffic only to the remaining available servers.

The clients send their packets to a special IP address that is defined as a *cluster address* to the Network Dispatcher, but as a loopback address to the servers. This address is externally advertised (via some dynamic routing protocol) by the Network Dispatcher alone, never by the servers. The Network Dispatcher inspects the destination address and the port number, and:

- If the destination is not the cluster address, the packet is treated in the normal IP manner.
- If the destination address is the cluster address but the packet is not UDP or TCP, it is treated in the normal IP manner.

- If the incoming datagram is a UDP packet or a TCP connection request, the Network Dispatcher selects one of the servers and forwards the packet to the appropriate port and cluster (now the loopback) address. For this to work, the connection between Network Dispatcher and server *must be direct*; an intermediate router would just reroute the packet back to the Network Dispatcher since this is the only instance of the cluster address known to the IP network.
- If the incoming datagram is a packet on an existing TCP connection, the Network Dispatcher forwards it in a similar manner to the server already chosen for this TCP connection. Thus, the NDR must maintain a table of TCP connections to the servers in its cluster(s).

The packet sent to the server from the Network Dispatcher has the original client address as the source address; thus, the server responds directly to the client instead of through the Network Dispatcher. In fact, the forwarded packet is identical to the original packet received by the Network Dispatcher.

Network Dispatcher has special logic to enable it to handle FTP connections, where multiple parallel TCP connections are needed between client and server. It recognizes an FTP control connection (because it uses the special well-known port of 21) and directs packets on the corresponding data connection to the same server. With passive mode FTP, a connection from a client to an unknown port on a cluster with an existing FTP control connection is directed to the same server as the existing connection. This means that all FTP connections from the same client are serviced by the same server, even if the client is an FTP client proxy.

All client requests sent to the Network Dispatcher machine are forwarded to the server that is selected by the Network Dispatcher as the optimal server according to certain dynamically set weights. You can use the default values for those weights or change the values during the configuration process.

No additional software is required on the servers to use Network Dispatcher, although (as we will see) it helps if the servers keep the Network Dispatcher aware of their internal workload.

With Network Dispatcher, it is possible to combine many individual servers into what appears to be a single generic server. The site thus appears as a single IP address to the world. Network Dispatcher functions independently of a domain name server; all requests are sent to the IP address of the Network Dispatcher machine.

Network Dispatcher allows a management application that is SNMP based to monitor Network Dispatcher status by receiving basic statistics and potential alert situations.

The way that Network Dispatcher distributes the load across servers is very efficient, but there are circumstances in which it does not work:

- If you are using IP Security with a virtual private network, the destination IP address is encrypted and cannot be read by Network Dispatcher. Thus the IPSec tunnel must end at (or before) the Network Dispatcher and not the server.
- If you are using an OSA (or a 3172) with ESCON multiple image facility (EMIF), the OSA must be able to distinguish to which LPAR each packet is to

be sent. Normally this is done by associating an IP address with an LPAR number in the OSA configuration. If the address in question is the NDR cluster address, it is associated with more than one LPAR and so the configuration becomes impossible.

3.1.1 Network Dispatcher components

Network Dispatcher can load balance requests to different servers based on the type of request, an analysis of the load on servers, or a configurable set of assigned weights. To manage all this, Network Dispatcher has the following components:

- The executor load balances connections based on the type of request received. Typical request types are HTTP, FTP, and Telnet. This component always runs in a Network Dispatcher machine.
- Advisors query the servers and analyze the results by protocol for each server. The advisor passes this information to the manager to set the appropriate weight. The advisor is an optional component. If you use an advisor, you must also use the manager.

  Network Dispatcher supports advisors for FTP, HTTP, SMTP, NNTP, POP3, and Telnet, plus a TN3270 advisor that works with TN3270 servers and an MVS advisor that works with WLM on OS/390 systems. WLM manages the amount of workload on an individual OS/390 (MVS) address space. Network Dispatcher can use WLM to help load balance requests to OS/390 servers.

  There are no protocol advisors specifically for UDP-based protocols. If the port is handling TCP and UDP traffic, the appropriate TCP protocol or MVS system advisor can be used to provide advisor input for the port. Network Dispatcher will use this input in load balancing both TCP and UDP traffic on that port.
- The manager sets weights for a server based on:
  - Internal counters in the executor
  - Feedback from the servers provided by the protocol advisors
  - Feedback from a system monitor, such as the MVS advisor

  The manager is an optional component. However, if you do not use the manager, the Network Dispatcher will balance the load using a round-robin scheduling method based on the current server weights.

When using Network Dispatcher to load balance stateless UDP traffic, you can only use servers that use the supplied source IP address on received requests to respond to the client.

3.1.2 High availability for Network Dispatcher

The base Network Dispatcher function has the following characteristics that make it a single point of failure unless it is backed up in some way:

- It examines all the traffic on the way in. If some of the packets for an existing connection use a different path through a different Network Dispatcher to reach a server, the server immediately resets the connection.
- It keeps track of all established connections and although it does not terminate them, entries lost from the Network Dispatcher connection table will result in the resetting of a connection.

- It appears to the previous router as the last hop in the route, and the connection's termination.

All these characteristics make the following failures critical for the whole cluster:

- If the Network Dispatcher fails for any reason, all the connection tables are lost. Therefore, all existing connections from the client to the server are also lost. Assuming there is a second Network Dispatcher that can direct a client to the servers, new connections will be able to go through only after the usual routing protocol delays, which could be several minutes.
- If the configured Network Dispatcher interface to the previous IP router fails, there must be another interface to get to the same Network Dispatcher (in which case recovery is performed by the IP router using the ARP aging mechanism with delays in the order of several minutes), or else all connections will be lost.
- If the Network Dispatcher interface to the servers fails, the previous hop router assumes that the Network Dispatcher is the last hop, and therefore will not reroute new connections. Existing connections will be lost and new connections will not be established.

In all these failure cases, which are not only Network Dispatcher failures but also Network Dispatcher neighborhood failures, all the existing connections are lost. Even with a backup Network Dispatcher running standard IP recovery mechanisms, recovery is, at best, slow and applies only to new connections. In the worst case, there is no recovery of the connections.

To improve Network Dispatcher availability, the Network Dispatcher high availability function uses the following mechanisms:

- Two Network Dispatchers with connectivity to the same clients, to the same cluster of servers, and between the Network Dispatchers themselves
- A heartbeat mechanism between the two Network Dispatchers to detect the failure of either of them
- Reachability criteria, to identify which IP hosts can and cannot be reached from each Network Dispatcher
- Synchronization of the Network Dispatcher databases (that is, the connection tables, reachability tables, and other databases)
- Logic to elect the active Network Dispatcher, which is in charge of a given cluster of servers, and the standby Network Dispatcher, which continuously gets synchronized for that cluster of servers
- A mechanism to perform fast IP takeover, when the logic or an operator decides to switch from active to standby

3.1.2.1 Failure detection

Besides the basic methods of failure detection (the loss of connectivity between active and standby Network Dispatchers, detected through the heartbeat messages) there is another failure detection mechanism, namely reachability criteria. When you configure the Network Dispatcher, you provide a list of hosts that each of the Network Dispatchers should be able to reach in order to work correctly. The hosts could be routers, servers or any other type of host. Host reachability is determined by pinging the hosts in question.

Takeover takes place if the heartbeat messages cannot go through, or if the reachability criteria are no longer met by the active Network Dispatcher but are met by the standby Network Dispatcher. To make the decision based on all available information, the active Network Dispatcher regularly sends the standby Network Dispatcher its reachability capabilities. The standby Network Dispatcher then compares the capabilities with its own and decides whether to initiate the switch.

3.1.2.2 Database synchronization

The primary and backup Network Dispatchers keep their databases synchronized through the heartbeat mechanism. The Network Dispatcher database includes connection tables, reachability tables and other information. The Network Dispatcher's high availability function uses a database synchronization protocol that ensures that both Network Dispatchers contain the same connection table entries. This synchronization takes into account a known error margin for transmission delays. The protocol performs an initial synchronization of databases and then maintains synchronization through periodic updates.

3.1.2.3 Recovery strategy

In the case of a Network Dispatcher failure, the IP takeover mechanism will direct all traffic toward the standby Network Dispatcher. The database synchronization mechanism ensures that the standby has the same entries as the active Network Dispatcher. When the failure occurs in the network (any intermediate piece of hardware or software between the client and the server), and there is an alternate path available through the standby Network Dispatcher, the takeover is performed across the alternate path.

3.1.2.4 IP takeover

Cluster (generic) IP addresses are assumed to be on the same logical subnet as the previous hop router (IP router). The IP router will resolve the cluster address through the ARP protocol. To perform the IP takeover, the Network Dispatcher (the standby becoming the active) will issue an ARP request to itself (also known as a gratuitous ARP), that is broadcast to all directly attached networks belonging to the logical subnet of the cluster. The previous hop IP router(s) will update their ARP tables (according to RFC 826) to send all traffic for that cluster to the new active (previously standby) Network Dispatcher.

3.2 Windows NT Network Dispatcher configuration

With the inclusion of Network Dispatcher in the WebSphere Edge Server, Network Dispatcher can be implemented on the Windows NT, AIX, Solaris, and Red Hat Linux platforms. Included with the Edge Server is the ability to use a GUI-based configuration tool. For general configuration, the GUI tool works quite well, allowing for quick configuration of the Network Dispatcher on these platforms.

However, for advanced functions, we recommend text-mode configuration as was supported in the IBM 2216. This allows for more complex configuration and the use of more detailed displays. In this section, we review highlights of using the GUI-based tool for configuration on the Windows NT platform. The configuration of NDR on the other Edge Server platforms is similar. For a description of text-mode commands, see 3.3, "2216 Network Dispatcher configuration" on page 76.

In this section, we show the necessary configuration for the scenario illustrated in Figure 44. This includes the executor, manager, advisor, and high availability portions. In this scenario, we have two Network Dispatchers servicing the cluster. The first, labeled Primary, is the primary owner of the cluster address. It has a backup Network Dispatcher, labeled Backup. Together they will provide the connection dispatching the target servers, RA03 and RA28 for the TN3270 service. Note that the primary NDR will send connections to both target servers, while the backup NDR, when active after a takeover, will only send requests to server RA28. This is simply for illustrative purposes, the point being that the group of target servers need not be the same for the primary and backup Network Dispatchers.

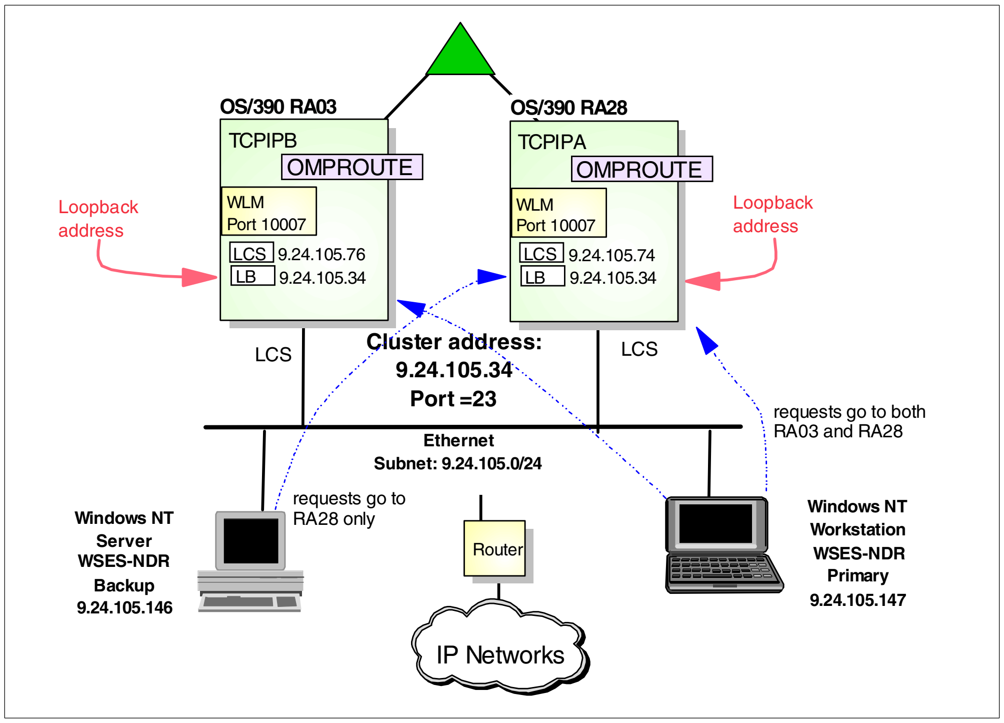
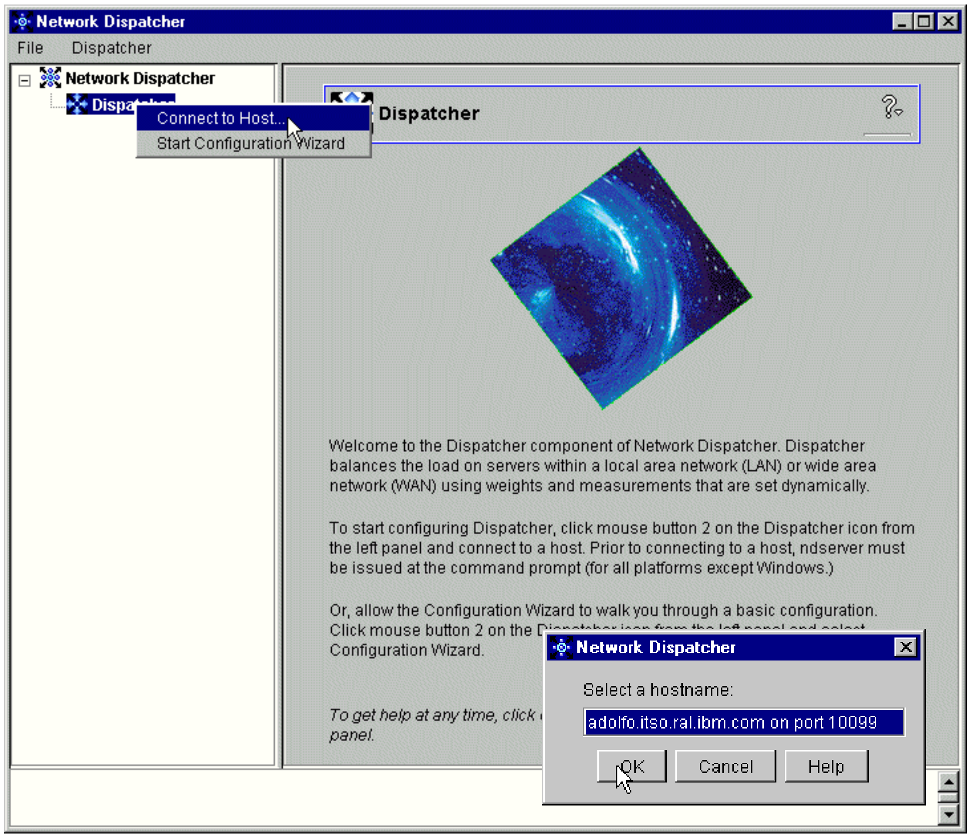


Figure 44. Scenario for Windows NT NDR GUI configuration

3.2.1 Windows NT executor configuration

The GUI-based configuration tool can be used to configure remote Network Dispatcher implementations. Because of this, we must first connect to the Network Dispatcher to be configured. In our case, this machine was called adolfo.itso.ral.ibm.com. We point the configuration tool to this machine as shown in Figure 45. We use the right mouse button to bring up the menu from which we choose to connect to the NDR host.



Welcome to the Dispatcher component of Network Dispatcher. Dispatcher balances the load on servers within a local area network (LAN) or wide area network (WAN) using weights and measurements that are set dynamically.

To start configuring Dispatcher, click mouse button 2 on the Dispatcher icon from the left panel and connect to a host. Prior to connecting to a host, ndserver must be issued at the command prompt (for all platforms except Windows.)

Or, allow the Configuration Wizard to walk you through a basic configuration. Click mouse button 2 on the Dispatcher icon from the left panel and select Configuration Wizard.

To get help at any time, click panel.

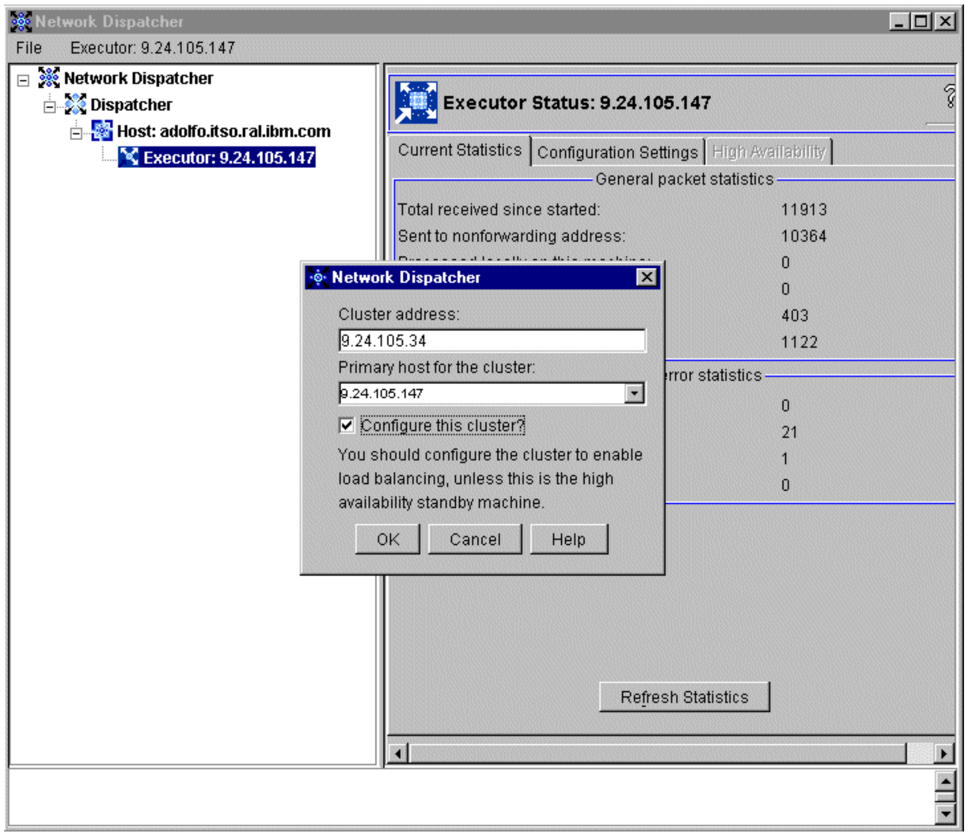
Select a hostname:

adolfo.itso.ral.ibm.com on port 10099

Figure 45. Connecting to the Network Dispatcher at adolfo.itso.ral.ibm.com

Once connected, we can begin the executor configuration. This is to involve the creation of the cluster that will represent the group of servers servicing TN3270 connections. Further, we will indicate to our Network Dispatcher the port being used by the service and the target servers that it should consider for connection distribution. Although here we only show the configuration primary NDR's executor, that of the backup NDR is almost identical.

Figure 46 shows the creation of the cluster within the primary Network Dispatcher. The most important aspect of this configuration is the indication of the cluster address, 9.24.105.34. Additionally, we must indicate the *primary owner* for this cluster. This is the Network Dispatcher that is considered the primary for this cluster. In this case, it is adolfo.itso.ral.ibm.com and we include its IP address here. This IP address is referred to as the *non-forwarding address* of this Network Dispatcher. That is, it is the real address of this NDR and packets sent to this address are not to be forwarded under any circumstances by the NDR. Of course, the cluster address must be different from this address.



Total received since started: 11913
Sent to nonforwarding address: 10364

Network Dispatcher

Cluster address: 9.24.105.34

Primary host for the cluster: 9.24.105.147

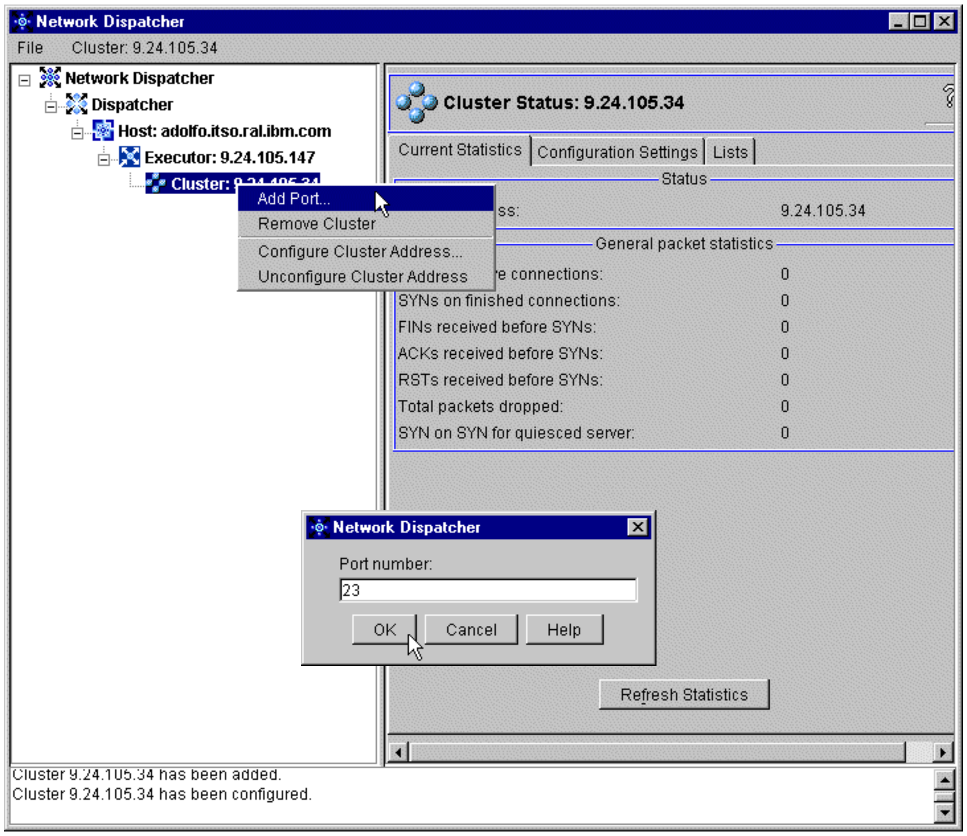
[✓] Configure this cluster?

You should configure the cluster to enable load balancing, unless this is the high availability standby machine.

Figure 46. Adding the cluster to the executor

Most services have some well-known port that is associated with them. For Telnet and TN3270, this port is 23 (although some installations have to use non-standards ports at times due to port contention). Our TN3270 servers on the RA03 and RA28 systems are configured to listen on the well-known port 23. As a result, the cluster that we have just configured must be associated with port 23.

This association is illustrated in Figure 47. We simply add the port to the cluster and indicate its value in the pop-up window.



ss:	9.24.105.34
e connections:	0
SYNs on finished connections:	0
FINs received before SYNs:	0
ACKs received before SYNs:	0
RSTs received before SYNs:	0
Total packets dropped:	0
SYN on SYN for quiesced server:	0

Port number: 23

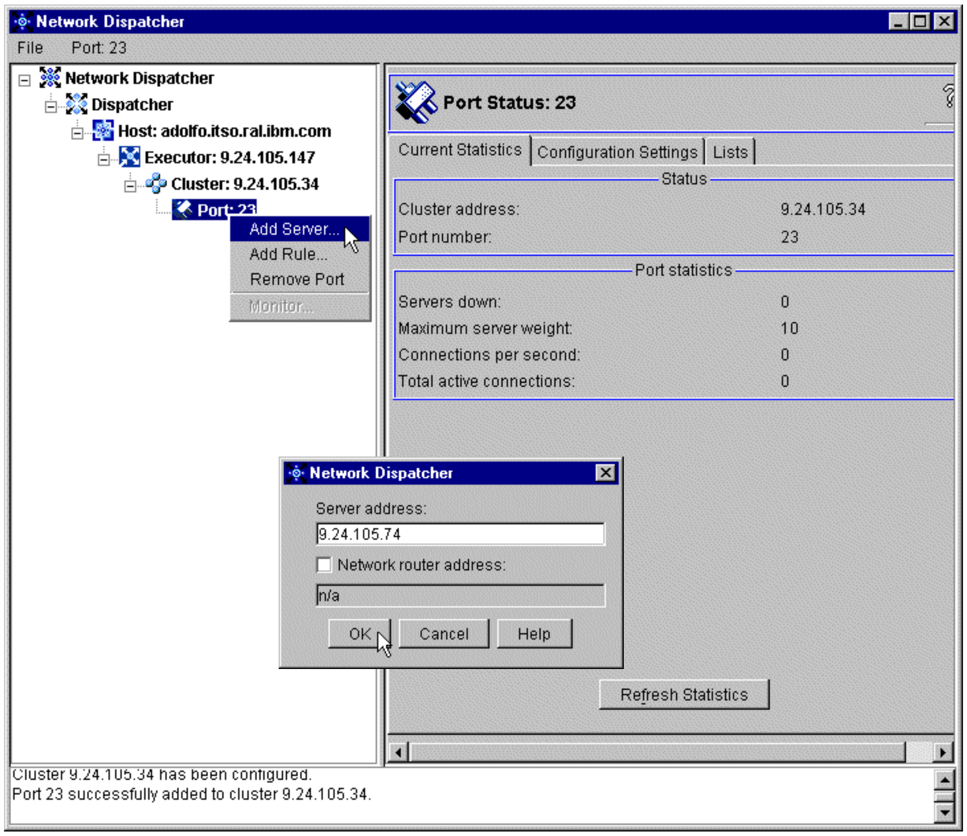
```
Cluster 9.24.105.34 has been added.
Cluster 9.24.105.34 has been configured.
```

Figure 47. Adding port 23 to the cluster

Once the port has been configured, we now must indicate to the Network Dispatcher which target servers will be candidates for receiving connections. This is done by adding servers to the port object in the NDR configuration.

Recall that the primary Network Dispatcher will be sending connections to both the RA03 and RA28 servers. Figure 48 shows how we added server RA28 with IP address 9.24.105.74. Although we don't show it here, we add the RA03 server in the same way.

At this point, we have done the minimal configuration necessary to have the cluster function. The primary Network Dispatcher will now begin dispatching connections to our target servers. Because we have not yet configured advisors or the manager and we have not statically defined weights to each of the target servers, the servers at this point will receive connections in a round-robin fashion.



Status

Cluster address: 9.24.105.34
Port number: 23

Port statistics

Servers down: 0
Maximum server weight: 10
Connections per second: 0
Total active connections: 0

Server address: 9.24.105.74
Network router address: n/a

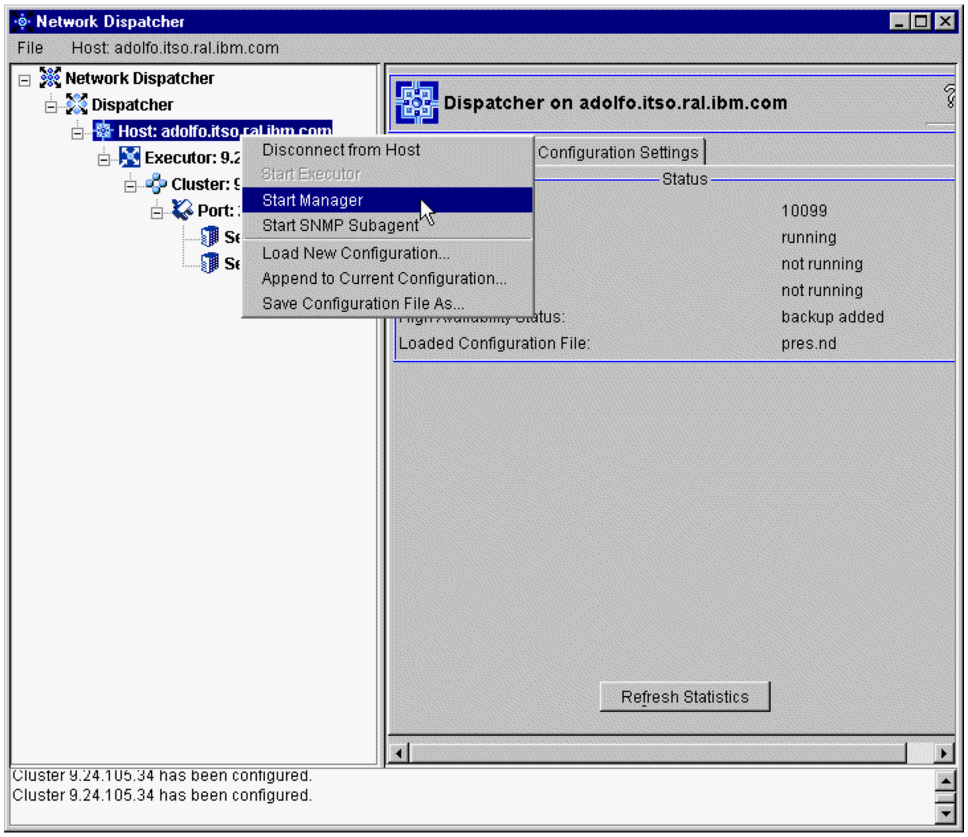
Cluster 9.24.105.34 has been configured.
Port 23 successfully added to cluster 9.24.105.34.

Figure 48. Adding server 9.24.104.74 to the port

3.2.2 Windows NT manager configuration

In order to take advantage of protocol and system load advisors, we must enable the manager. The manager is responsible for attaining feedback from advisors. Based on current policy, it dynamically updates the weights of target servers. The amount of importance given to the information provided by advisors is configurable. Therefore, it is the responsibility of the manager to determine what the correct weight values should be and update them accordingly. Once that is done, the executor that forwards requests will respect the new weight values in making its forwarding decisions.

Figure 49 shows the way we can start the manager with the configuration tool. We simply choose the **Start Manager** option from the pop-up menu (which comes up when we click the right mouse button on the NDR host).



```
                                   10099
                                   running
                                   not running
                                   not running
High Availability Status:          backup added
Loaded Configuration File:         pres.nd
```

```
Cluster 9.24.105.34 has been configured.
Cluster 9.24.105.34 has been configured.
```

Figure 49. Starting the manager

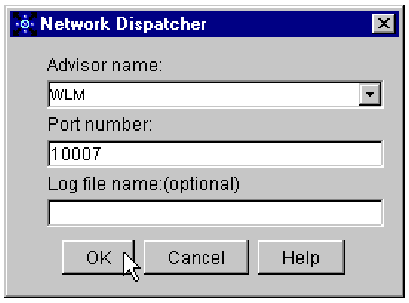
3.2.3 Windows NT MVS (WLM) advisor configuration

In order to have the manager dynamically update the weights associated with target servers, an advisor must be configured. The role of advisors is to attain some notion of target server performance and provide this information to the manager.

In general, there are two types of advisors, protocol advisors and system performance advisors. The protocol advisors interact with specific servers running on the target server machines. They understand the protocol being implemented by the servers and can gauge the servers' performance based on this. Examples of protocol advisors include HTTP and Telnet. These protocol advisors connect and send some protocol-recognized request to each of the servers. Based on the response time of the action (for example, the download of a Web object), the protocol advisor infers a notion of target server performability that is made available to the manager.

Another type of advisor is the system performance advisor. This advisor interacts with some general performance measurement tool on the target servers. That is, each platform has some software specific to that platform that can be used to gauge a server's performance. On the OS/390 system, this is done by Workload Manager (WLM). When using the WLM or MVS advisor, the advisor contacts the WLM software running on the target servers by connecting to the WLM well-known port. The advisor then queries the target server for its current performability metric and makes this information available to the manager. Figure

50 shows the configuration of the WLM advisor. The port of 10007 is the WLM well-known port.



Advisor name: WLM
Port number: 10007
Log file name:(optional)

Figure 50. Adding MVS (WLM) advisor

Because the two types of advisors may yield different notions of performance, one advisor from each type can function simultaneously with one of the other type. In this fashion, the manager can receive the most accurate measure of performance of the target servers.

3.2.4 Windows NT high availability configuration

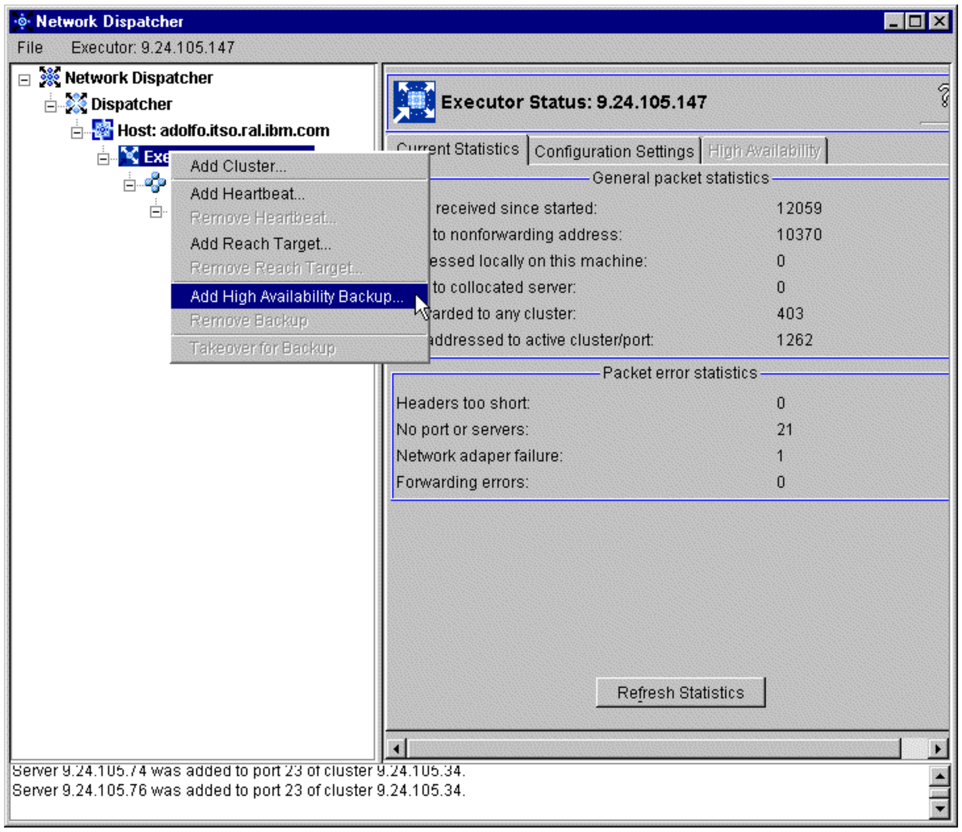
One of the nice features of Network Dispatcher is the ability to create a high availability configuration. That is, a second Network Dispatcher can be configured to act as a backup of an active Network Dispatcher. If a problem occurs with the active NDR, the backup NDR can take over the work of dispatching connections to target servers.

IP takeover occurs when the backup NDR determines that the active NDR has failed. The backup, or standby, Network Dispatcher marks the active NDR as failed if it stop receiving *heartbeat* messages from it. Heartbeat messages are sent periodically between Network Dispatchers that are working together in a high availability configuration to indicate that each is still functioning properly.

An active Network Dispatcher can also inform its standby of its failure due to some failed network or interface. An active NDR is configured with a list of IP hosts that it needs to be able to reach in order to function properly. Periodically, the active NDR attempts to ping these hosts. If the pings fail, the active NDR transfers the ownership of the cluster to the backup NDR in hopes that its reachability criteria is met.

Once a backup NDR takes over ownership of a cluster, it advertises ownership of the cluster address by sending a gratuitous ARP on the connected media. The first hop router must adjust its ARP cache entry so that it now sends data destined for the cluster to the new active (previously, the backup) NDR. This process is known as IP takeover.

Figure 51 shows how we add the backup NDR to the primary (active) NDR’s configuration.



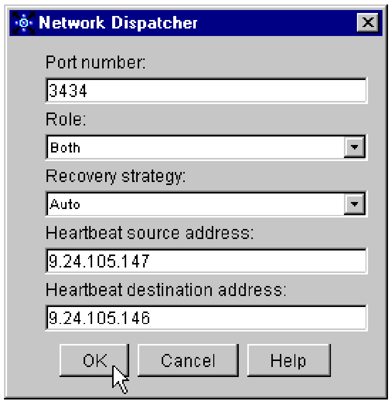
```
received since started:              12059
to nonforwarding address:            10370
essed locally on this machine:       0
to collocated server:                0
arded to any cluster:                403
ddressed to active cluster/port:     1262

Headers too short:                   0
No port or servers:                  21
Network adaper failure:              1
Forwarding errors:                   0
```

```
Server 9.24.105.74 was added to port 23 of cluster 9.24.105.34.
Server 9.24.105.76 was added to port 23 of cluster 9.24.105.34.
```

Figure 51. Adding backup (standby) NDR

Because our scenario shows the case of mutual availability in which each NDR will be the backup of the other, we configure the role of the NDR to be **Both**, meaning it can be both an active and a backup NDR. This is shown in Figure 52.

In addition to the role, we indicate the takeover is to occur automatically as opposed to manually and indicate the port that will be used for communication between the NDRs. The port of 3434 is not well-known, but simply one we chose for this purpose. Finally, the heartbeat is also configured here.



Port number:	3434
Role:	Both
Recovery strategy:	Auto
Heartbeat source address:	9.24.105.147
Heartbeat destination address:	9.24.105.146

Figure 52. Adding backup (standby) NDR, part 2

3.3 2216 Network Dispatcher configuration

In this section we describe an implementation of NDR using an IBM 2216 in our sysplex. To make it easier to understand the effect of enabling each of the following components, we performed the same tests against each configuration:

- Executor component configuration
- Manager component configuration
- Advisor component configuration

We also show some of the 2216 console log files we produced during the tests.

We used the same network configuration as described in 6.5, "Configuring OMPROUTE" on page 161 when we did our Network Dispatcher test. The network diagram is shown in Figure 53. The Network Dispatcher was configured on the channel-attached 2216.

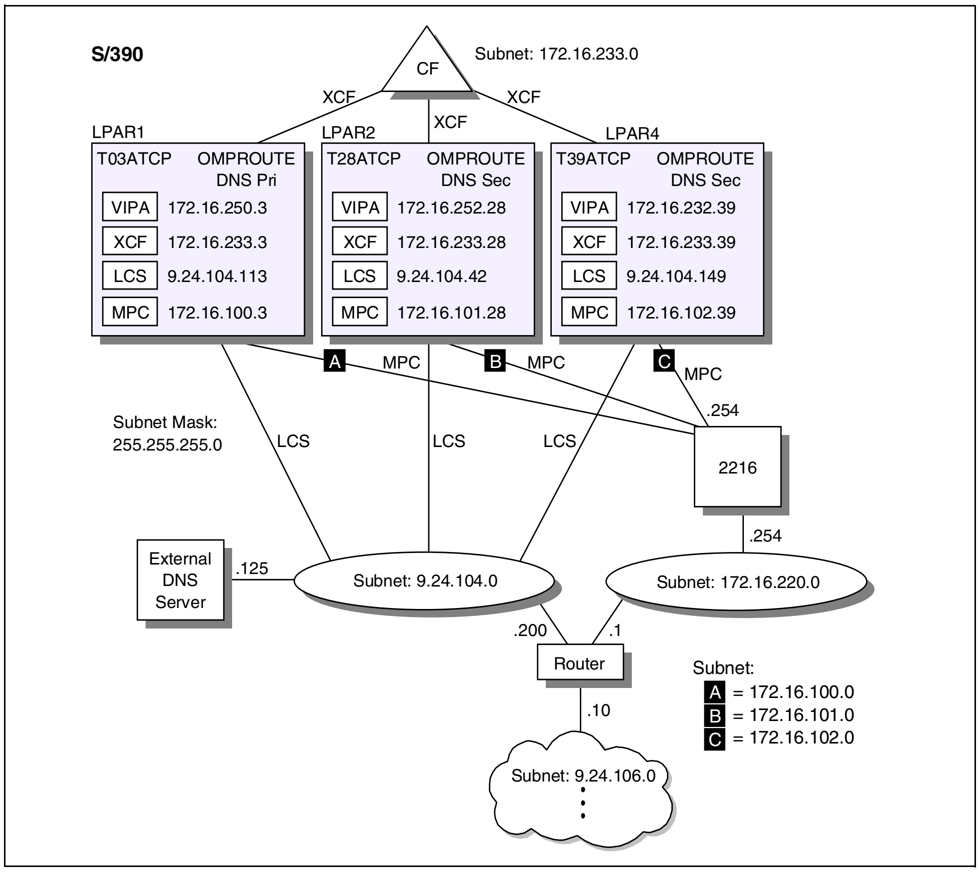


Figure 53. Network Dispatcher configuration

3.3.1 2216 NDR executor configuration

We first configured the cluster using a Telnet session to the 2216; see Figure 54.

In our test environment we set up one cluster, 172.16.220.50. The cluster address should be configured on the same subnet as the previous hop IP router. Next we configured the ports associated with our cluster. Finally, the last step before enabling the executor was to define the servers associated with the ports and cluster.

In general we kept the default values provided by the configuration process in the 2216 when we set the parameters for NDR.

```
*TALK 6

Config>FEATURE NDR                                                          1
NDR Config>ADD CLUSTER                                                      2
Cluster Address [0.0.0.0]? 172.16.220.50
FIN count [4000]?
FIN time out [30]?
Stale timer [1500]?
NDR Config>ADD PORT                                                         3
Cluster Address [0.0.0.0]? 172.16.220.50
Port number [80]? 23
Port type(tcp=1, udp=2, both=3) [3]? 1
Max. weight (0-100) [20]?
Only one pftp port per cluster allowed
Port mode (none=0, sticky=1 pftp=2 extcache=4) [0]?
NDR Config>ADD PORT 172.16.220.50 1234 1 20 0
Only one pftp port per cluster allowed
NDR Config>ADD PORT 172.16.220.50 2345 1 20 0
Only one pftp port per cluster allowed
NDR Config>ADD SERVER                                                       4
Cluster Address [0.0.0.0]? 172.16.220.50
Port number [80]? 23
Server Address [0.0.0.0]? 172.16.250.3
Server weight  [20]?
Server state (down=0, up=1) [1]?
NDR Config>ADD SERVER 172.16.220.50 23 172.16.252.28 20 1
NDR Config>ADD SERVER 172.16.220.50 23 172.16.232.39 20 1
NDR Config>ADD SERVER 172.16.220.50 1234 172.16.250.3 20 1
NDR Config>ADD SERVER 172.16.220.50 1234 172.16.252.28 20 1
NDR Config>ADD SERVER 172.16.220.50 1234 172.16.232.39 20 1
NDR Config>ADD SERVER 172.16.220.50 2345 172.16.250.3 20 1
NDR Config>ADD SERVER 172.16.220.50 2345 172.16.252.28 20 1
NDR Config>ADD SERVER 172.16.220.50 2345 172.16.232.39 20 1
NDR Config>ENABLE EXECUTOR                                                  5
```

Figure 54. 2216 NDR executor configuration console log

1 The `FEATURE NDR` command enables NDR configuration mode.

2 The `ADD CLUSTER` command defines a cluster address and its parameters.

3 The `ADD PORT` command defines a port number associated with a cluster. We configured ports 23, 1234 and 2345 to be used. The first port is the well-known Telnet port, and the last two were used by our test socket applications. Note that if you will be using passive FTP you will need to define a `pftp` mode port to invoke the NDR logic to handle this.

4 The `ADD SERVER` command defines the server address associated with a port and cluster. In this example, we used the VIPA addresses of our sysplex servers, but

this is not recommended. The VIPA address is logically *two* hops away from the NDR, so this breaks the rules although it worked in our example.

5 The `ENABLE EXECUTOR` command starts the executor immediately. It is possible to monitor the executor under `TALK 5`; see Figure 56 on page 79.

3.3.1.1 NDR base sample executor test 1

Before we started the test we added a name server entry c50 in the ralplex1 subdomain for the cluster IP address 172.16.220.50.

We also started our test server described in A.4, “SOCSRVR, a simple socket server program” on page 260 in all of our stacks. The servers were set to listen on port 1234.

Then we ran our client test program described in A.2, “Collecting statistics using REXX” on page 256. We issued `sysplex2 c50 1234 -c 99 -b 0.1` on the client.

```
+---------------------+-----------------+---------------+--------+-------
-+
| Hostname            | Resolved Addr   | Connected Addr | Resolv | Connec |
+---------------------+-----------------+---------------+--------+-------
-+
| c50                 | 172.16.220.50   | 172.16.252.28  | 0.0200 | 0.0200 |
| c50                 | 172.16.220.50   | 172.16.232.39  | 0.0100 | 0.0100 |
| c50                 | 172.16.220.50   | 172.16.250.3   | 0.0100 | 0.0200 |
| c50                 | 172.16.220.50   | 172.16.252.28  | 0.0100 | 0.0100 |
| c50                 | 172.16.220.50   | 172.16.232.39  | 0.0100 | 0.0100 |
  :

+---------------+---------------+
| Resolved Addr  | Resolved Count |
+---------------+---------------+
| 172.16.220.50  | 99             |

+---------------+---------------+
| Connected To   | Connected Count|
+---------------+---------------+
| 172.16.232.39  | 33             |
| 172.16.250.3   | 33             |
| 172.16.252.28  | 33             |
```

Figure 55. NDR base sample executor test 1

The result from this test shows that the connections were evenly distributed by the executor. The executor used the default weight 20 for each server, which explains the result. By modifying the weight for a server it is very easy to change the distribution of the connections. Figure 56 shows the status of the 2216 NDR as displayed from the operator prompt (use `t 5` to get it).

```
*TALK 5                                                                   1

+FEATURE NDR                                                              2
NDR >STATUS PORT                                                          3
Cluster Address [0.0.0.0]? 172.16.220.50
Port number  [0]? 1234

PORT 1234 INFORMATION:
 -------------------
Maximum weight................. 20
Port Mode ..................... none
Port Type ..................... TCP
All up nodes are weight zero.... FALSE
Total target nodes............. 3
Currently marked down.......... 0
Servers providing service to this port:
Address: 172.16.232.39 Weight: 20 Count: 335 TCP Count: 335 UDP Count: 0
Active:
 6 FIN 329 Complete 0 Status: up Saved Weight: -1
Address: 172.16.250.3 Weight: 20 Count: 336 TCP Count: 336 UDP Count: 0
Active:
3 FIN 333 Complete 0 Status: up Saved Weight: -1
Address: 172.16.252.28 Weight: 20 Count: 335 TCP Count: 335 UDP Count: 0
Active:
 2 FIN 333 Complete 0 Status: up Saved Weight: -1
```

Figure 56. NDR base sample test 1 - 2216 status port console log

1 The `TALK 5` command enables the operations console.

2 The `FEATURE NDR` command selects NDR monitor mode.

3 The `STATUS PORT` command displays the parameter values for the cluster and port you select.

The output shows the server addresses providing service on the port, their weights and connection counters.

3.3.1.2 NDR base sample executor test 2

In this test we stopped the server application in stack T03ATCP.

Then we executed: `sysplex2 c50 1234 -c 99 -b 0.1` on the client. See Figure 57 for the results.

```
+---------------------+-----------------+---------------+--------+-------
-+
| Hostname            | Resolved Addr   | Connected Addr | Resolv | Connec |
+---------------------+-----------------+---------------+--------+-------
-+
| c50                 | 172.16.220.50   | 172.16.232.39  | 0.0300 | 0.0100 |
Error on connecting socket to '172.16.220.50': ECONNREFUSED
| c50                 | 172.16.220.50   | 172.16.252.28  | 0.0100 | 0.0200 |
   :
| c50                 | 172.16.220.50   | 172.16.232.39  | 0.0100 | 0.0200 |
Error on connecting socket to '172.16.220.50': ECONNREFUSED
| c50                 | 172.16.220.50   | 172.16.252.28  | 0.0100 | 0.0100 |

+---------------+---------------+
| Resolved Addr  | Resolved Count |
+---------------+---------------+
| 172.16.220.50  | 99             |

+---------------+---------------+
| Connected To   | Connected Count|
+---------------+---------------+
| 172.16.232.39  | 33             |
| 172.16.252.28  | 33             |
| ECONNREFUSED   | 33             |
```

Figure 57. NDR base sample executor test 2

This time the executor continued to distribute the connections to the defined server even though the server application was stopped. There is no parameter provided to stop this from happening. The executor takes just the IP address and weight into consideration when it distributes the connections in our current implementation of NDR.

Even if the complete TCP/IP stack is stopped the executor will continue to distribute connections to the defined server on that stack.

To change this behavior, we enabled the manager and the MVS advisor. Please see 3.3.3.4, “NDR protocol advisors” on page 91 for more information about protocol advisors and how they affect the distribution of the connections.

3.3.2 2216 NDR manager configuration

The next step was to configure the manager. The manager calculates server weights and periodically sends the result to the executor. See Figure 58 for our definitions.

Remember that the manager will overwrite the weights that were set when we configured the ports and servers.

The manager uses the following external factors in its weighting decisions:

- The number of active connections on each TCP server
- The number of new connections on each TCP server
- Input from protocol advisors
- Input from the system monitor (MVS advisor)

The first two values are based on information that is generated and stored internally in the executor.

```
NDR Config>SET MANAGER                                                    1
Interval (seconds)  [2]?                                                  2
Proportion: Active, New, Advisor, System must add up to 100
Proportion: Active [50]?                                                  3
Proportion: New [50]?                                                     3
Proportion: Advisor [0]?                                                  3
Proportion: System  [0]?                                                  3
Refresh Cycle [2]?                                                        4
Sensitivity (0-100)  [5]?
Smoothing (> 1.00) [1.50]?
NDR Config>ENABLE MANAGER                                                 5
Manager interval was set to 2.
Manager proportions were set to [50] [50] [0] [0]
Manager refresh cycle was set to 2
Manager sensitivity was set to 5.
Manager smoothing factor was set to 1.50.
NDR Config>
```

Figure 58. 2216 NDR manager configuration console log

1 The `SET MANAGER` command, issued from the NDR configuration prompt, sets the manager parameters.

2 The manager interval specifies how often the manager will update the server weights that the executor uses in distributing connections.

3 The proportion parameters define how much weight is given to each of the four inputs by the manager. The defaults, taken here, ignore the advisor and system information when calculating weights.

4 The manager refresh cycle specifies how often the manager will ask the executor for status information. The refresh cycle is based on the interval time.

5 The `ENABLE MANAGER` command starts the manager at once. It is possible to monitor the manager and generate manager reports under `TALK 5` (the operator prompt).

3.3.2.1 NDR base sample manager test 1

By this time we had started the server application in stack T03ATCP again. We issued `sysplex2 c50 1234 -c 99 -b 0.1` on the client and saw the results shown in Figure 59.

```
+---------------------+------------------+----------------+--------+--------+
| Hostname            | Resolved Addr    | Connected Addr | Resolv | Connec |
+---------------------+------------------+----------------+--------+--------+
| c50                 | 172.16.220.50    | 172.16.252.28  | 0.0300 | 0.0200 |
| c50                 | 172.16.220.50    | 172.16.232.39  | 0.0100 | 0.0100 |
| c50                 | 172.16.220.50    | 172.16.250.3   | 0.0100 | 0.0200 |
| c50                 | 172.16.220.50    | 172.16.252.28  | 0.0100 | 0.0100 |
| c50                 | 172.16.220.50    | 172.16.232.39  | 0.0100 | 0.0100 |
| c50                 | 172.16.220.50    | 172.16.250.3   | 0.0200 | 0.0100 |
   :
| c50                 | 172.16.220.50    | 172.16.252.28  | 0.0100 | 0.0100 |
| c50                 | 172.16.220.50    | 172.16.232.39  | 0.0110 | 0.0100 |
| c50                 | 172.16.220.50    | 172.16.252.28  | 0.0100 | 0.0100 |
| c50                 | 172.16.220.50    | 172.16.232.39  | 0.0100 | 0.0100 |
| c50                 | 172.16.220.50    | 172.16.252.28  | 0.0100 | 0.0100 |
   :

+---------------+---------------+
| Resolved Addr | Resolved Count |
+---------------+---------------+
| 172.16.220.50 | 99            |

+---------------+---------------+
| Connected To  | Connected Count|
+---------------+---------------+
| 172.16.232.39 | 41            |
| 172.16.250.3  | 21            |
| 172.16.252.28 | 37            |
```

Figure 59. NDR base sample manager test 1

At this point we saw variations in the distribution of connections. The variations were caused by the modifications of weights that the manager sent to the executor.

This result is not really what we expected; we believed that the distribution would be more even. However, we found that the smoothing parameter can cause this type of behavior if it is set too low. We talk more about smoothing later in this chapter.

To find out more, we issued `REPORT MANAGER` from the NDR operator prompt (invoked by `t 5` followed by `feature ndr`). See Figure 60.

```
NDR >REPORT MANAGER

---------------------------------
|  HOST TABLE LIST  |   STATUS   |
---------------------------------
|  172.16.232.39    |    ACTIVE  |
|  172.16.250.3     |    ACTIVE  |
|  172.16.252.28    |    ACTIVE  |
---------------------------------

-----------------------------------------------------------------------------
|172.16.220.50  |WEIGHT | ACTIVE % 50  |  NEW % 50    | PORT %  0 |SYSTEM %  0|
|               -------------------------------------------------------------|
|PORT: 1234     |NOW|NEW| WT | CONNECT | WT | CONNECT | WT | LOAD | WT | LOAD |
|----------------------------------------------------------------------------|
|172.16.232.39  |  9|  9|   9|        0|   9|       19|   0|     0|   0|     0|
|172.16.250.3   |  9|  9|   9|        1|   9|       19|   0|     0|   0|     0|
|172.16.252.28  |  9|  9|   9|        0|   9|       10|   0|     0|   0|     0|
|----------------------------------------------------------------------------|
|PORT TOTALS:   | 27| 27|    |        0|    |        0|    |     0|    |     0|
-----------------------------------------------------------------------------

-----------------------------------------
| ADVISOR |  PORT   |  TIMEOUT   |  STATUS  |
-----------------------------------------
|     No advisors currently running.      |
-----------------------------------------
Manager report requested.

NDR >REPORT MANAGER
 :
-----------------------------------------------------------------------------
|172.16.220.50  |WEIGHT | ACTIVE % 50  |  NEW % 50    | PORT %  0 |SYSTEM %  0|
|               -------------------------------------------------------------|
|PORT: 1234     |NOW|NEW| WT | CONNECT | WT | CONNECT | WT | LOAD | WT | LOAD |
|----------------------------------------------------------------------------|
|172.16.232.39  | 13| 13|  18|        0|   9|       21|   0|     0|   0|     0|
|172.16.250.3   |  0|  0|  -8|        0|   9|        2|   0|     0|   0|     0|     1
|172.16.252.28  | 14| 14|  19|        0|   9|       25|   0|     0|   0|     0|
|----------------------------------------------------------------------------|
|PORT TOTALS:   | 27| 27|    |        1|    |        0|    |     0|    |     0|
-----------------------------------------------------------------------------
 :
```

Figure 60. 2216 report manager console log

1 The manager calculates the server weight dynamically. At this time the weight for server 172.16.250.3 was set to 0 and no distribution to that server would be performed during the current interval.

We found out that the smoothing factor (a configuration parameter in Figure 58) was essential to stop the manager from providing wildly oscillating weights to the executor. A higher smoothing factor will cause the server weights to change less dramatically. The default value of 1.5 does not prevent the oscillating effect, so we decided to set smoothing to 3 in our configuration.

There is also a way to prevent unnecessary updating of the weights when there is little change in server status. The sensitivity factor is used for this purpose. When the percentage weight change for all servers on a port is greater than the sensitivity threshold, the manager will update the weights used by the executor to distribute connections.

3.3.2.2 NDR base sample manager test 2

In this test we stopped the server application in stack T03ATCP. Then we issued `sysplex2 c50 1234 -c 99 -b 0.1` on the client. Figure 61 was the result:

```
+---------------------+-----------------+---------------+--------+--------+
| Hostname            | Resolved Addr   | Connected Addr | Resolv | Connec |
+---------------------+-----------------+---------------+--------+--------+
| c50                 | 172.16.220.50   | 172.16.232.39  | 0.0200 | 0.0200 |
Error on connecting socket to '172.16.220.50': ECONNREFUSED
| c50                 | 172.16.220.50   | 172.16.252.28  | 0.0200 | 0.0200 |
| c50                 | 172.16.220.50   | 172.16.232.39  | 0.0100 | 0.0100 |
  :

+---------------+---------------+
| Resolved Addr  | Resolved Count |
+---------------+---------------+
| 172.16.220.50  | 99             |

+---------------+---------------+
| Connected To   | Connected Count|
+---------------+---------------+
| 172.16.232.39  | 41             |
| 172.16.252.28  | 40             |
| ECONNREFUSED   | 18             |
```

Figure 61. NDR base sample manager test 2

In this scenario we also perceived variations in the distribution of connections. Again we used the `REPORT MANAGER` command to see what the counters looked like during the test. Please refer to Figure 62.

```
NDR >REPORT MANAGER
 :
--------------------------------------------------------------------------------
|172.16.220.50   |WEIGHT | ACTIVE % 50  |  NEW % 50    | PORT %  0 |SYSTEM %  0|
|                ---------------------------------------------------------------|
|PORT: 1234      |NOW|NEW| WT | CONNECT | WT | CONNECT | WT | LOAD | WT | LOAD |
|-------------------------------------------------------------------------------|
|172.16.232.39   |  9|  9|   9|        0|   9|        1|   0|     0|   0|     0|
|172.16.250.3    |  9|  9|   9|        1|   9|        1|   0|     0|   0|     0|
|172.16.252.28   |  9|  9|   9|        0|   9|        0|   0|     0|   0|     0|
|-------------------------------------------------------------------------------|
|PORT TOTALS:    | 27| 27|    |        0|    |        0|    |     0|    |     0|
--------------------------------------------------------------------------------
 :
--------------------------------------------------------------------------------
|172.16.220.50   |WEIGHT | ACTIVE % 50  |  NEW % 50    | PORT %  0 |SYSTEM %  0|
|                ---------------------------------------------------------------|
|PORT: 1234      |NOW|NEW| WT | CONNECT | WT | CONNECT | WT | LOAD | WT | LOAD |
|-------------------------------------------------------------------------------|
|172.16.232.39   | 11| 11|  13|        0|   9|        3|   0|     0|   0|     0|
|172.16.250.3    |  5|  5|   1|       10|   9|        3|   0|     0|   0|     0|
|172.16.252.28   | 11| 11|  13|        0|   9|        3|   0|     0|   0|     0|
|-------------------------------------------------------------------------------|
|PORT TOTALS:    | 27| 27|    |        7|    |        0|    |     0|    |     0|
--------------------------------------------------------------------------------
 :
--------------------------------------------------------------------------------
|172.16.220.50   |WEIGHT | ACTIVE % 50  |  NEW % 50    | PORT %  0 |SYSTEM %  0|
|                ---------------------------------------------------------------|
|PORT: 1234      |NOW|NEW| WT | CONNECT | WT | CONNECT | WT | LOAD | WT | LOAD |
|-------------------------------------------------------------------------------|
|172.16.232.39   | 11| 11|  14|        0|   9|        8|   0|     0|   0|     0|
|172.16.250.3    |  5|  5|   1|       18|   9|        3|   0|     0|   0|     0|
|172.16.252.28   | 11| 11|  13|        0|   9|        7|   0|     0|   0|     0|
|-------------------------------------------------------------------------------|
|PORT TOTALS:    | 27| 27|    |       15|    |        0|    |     0|    |     0|
--------------------------------------------------------------------------------
 :
```

Figure 62. 2216 report manager console log

The reports show that the executor registers active connections to the unavailable server; the counter gradually increases since the connections are not terminated at once.

The other two servers process and complete their connections. The client server transactions are short and the number of active connections will stay low for the two active servers. This results in a lower weight for the unavailable server and fewer connections distributed to that server from the executor. However, there should (eventually) be none at all.

Even if a complete TCP/IP stack is stopped, the manager will continue to send weights to the executor based on executor connection counters. Therefore, the executor will continue to distribute connections to the servers configured on the unavailable stack.

We found that only the protocol advisors could stop the manager from distributing connections to unavailable servers. For the Network Dispatcher to operate correctly, *all* the relevant advisors must be configured, as we show in our last example in 3.3.3.4, “NDR protocol advisors” on page 91.

3.3.3 2216 NDR advisor configuration

Now to the final configuration of the base example. We set up the NDR to use the advisor for MVS.

Before we enabled the advisor we had to prepare the manager to take advisor metrics into consideration when calculating weights. We changed the manager configuration to use proportions 25/25/25/25 via the `SET MANAGER` command, as used in Figure 58. After the configuration changes were made we disabled the manager using `DISABLE MANAGER`, and re-enabled it by issuing `ENABLE MANAGER`, from the `NDR Config>` prompt. The new manager parameters were now in use.

Figure 63 shows the configuration of the MVS advisor:

```
NDR Config>ADD ADVISOR                                                               1
Advisor name (0=ftp,1=http,2=MVS,3=TN3270,4=smtp,5=nntp,6=pop3,7=telnet) [1]? 2      2
Port number [10007]?                                                                 3
Interval (seconds) [5]?                                                              4
Timeout (0=unlimited) [0]?                                                           5
NDR Config>ENABLE ADVISOR                                                            6
Advisor name (0=ftp,1=http,2=MVS,3=TN3270,4=smtp,5=nntp,6=pop3,7=telnet) [0]? 2
Port number [0]? 10007
Advisor MVS on port 10007 interval was set to 5.
Advisor MVS on port 10007 timeout was set to unlimited
This advisor is now enabled.
NDR Config>
```

Figure 63. 2216 NDR MVS (WLM) advisor configuration console log

1 The `ADD ADVISOR` command adds an advisor and sets advisor parameters.

2 The advisor name prompt gives you the choice of available advisors.

3 WLM listens on port 10007.

4 The advisor interval specifies how often an advisor asks for status from the servers on the port it is monitoring and then reports the result to the manager.

5 To make sure that out-of-date information is not used by the manager in its load balancing decisions, the manager will not use records older than the time set in the advisor timeout. By default, advisor reports do not time out.

6 We enable the MVS (WLM) advisor.

The following command display shows our final settings for the NDR in the base sample:

```
NDR Config>LIST ALL

Executor: Enabled

Manager: Enabled

        Interval        Refresh-Cycle   Sensitivity     Smoothing
        2               2               5     %         3.00

        Proportions:    Active  New     Advisor         System
                        25 %    25 %    25 %            25 %

Advisor:
        Name    Port    Interval        TimeOut         State       CommPort
        MVS     10007   5               0               Enabled

Backup: Disabled

        Role            Strategy

        Reachability:   Address         Mask            Type

        HeartBeat Configuration:

Clusters:
        Cluster-Addr    FIN-count       FIN-timeout     Stale-timer
        172.16.220.50   4000            30              1500

Ports:
        Cluster-Addr    Port#  Weight  Port-Mode   Port-Type
        172.16.220.50   23     20   %  none        TCP
        172.16.220.50   1234   20   %  none        TCP
        172.16.220.50   2345   20   %  none        TCP

Servers:
        Cluster-Addr    Port#   Server-Addr     Weight  State
        172.16.220.50   23      172.16.232.39   20   %  up
        172.16.220.50   23      172.16.250.3    20   %  up
        172.16.220.50   23      172.16.252.28   20   %  up
        172.16.220.50   1234    172.16.232.39   20   %  up
        172.16.220.50   1234    172.16.250.3    20   %  up
        172.16.220.50   1234    172.16.252.28   20   %  up
        172.16.220.50   2345    172.16.232.39   20   %  up
        172.16.220.50   2345    172.16.250.3    20   %  up
        172.16.220.50   2345    172.16.252.28   20   %  up
```

Figure 64. 2216 NDR config list all command console log

3.3.3.1 NDR base sample advisor test 1

The server application on stack T03ATCP was started once more, and we issued `sysplex2 c50 1234 -c 99 -b 0.1` on the client. Figure 65 shows what we saw.

```
+---------------------+-----------------+---------------+--------+--------+
| Hostname            | Resolved Addr   | Connected Addr | Resolv | Connec |
+---------------------+-----------------+---------------+--------+--------+
| c50                 | 172.16.220.50   | 172.16.250.3   | 0.0300 | 0.0200 |
| c50                 | 172.16.220.50   | 172.16.232.39  | 0.0100 | 0.0200 |
| c50                 | 172.16.220.50   | 172.16.250.3   | 0.0100 | 0.0100 |
| c50                 | 172.16.220.50   | 172.16.252.28  | 0.0100 | 0.0100 |
 :

+---------------+---------------+
| Resolved Addr  | Resolved Count |
+---------------+---------------+
| 172.16.220.50  | 99             |

+---------------+---------------+
| Connected To   | Connected Count|
+---------------+---------------+
| 172.16.232.39  | 31             |
| 172.16.250.3   | 37             |
| 172.16.252.28  | 31             |
```

Figure 65. NDR base sample advisor test 1

This time we saw small differences in the distribution of connections. The output from the `REPORT MANAGER` command is shown in Figure 66:

```
NDR >REPORT MANAGER
 :
-----------------------------------------------------------------------------
|172.16.220.50   |WEIGHT | ACTIVE % 25   |  NEW % 25     | PORT % 25 |SYSTEM % 25|
|                -------------------------------------------------------------|
|PORT: 1234      |NOW|NEW| WT | CONNECT | WT | CONNECT | WT | LOAD | WT | LOAD |
|-----------------------------------------------------------------------------|
|172.16.232.39   |  6|  6|   9|        0|   9|        0|  0|      0|   9|  4309|
|172.16.250.3    |  7|  7|   9|        0|   9|        0|  0|      0|  10|  4435|
|172.16.252.28   |  6|  6|   9|        0|   9|        0|  0|      0|   9|  4391|
|-----------------------------------------------------------------------------|
|PORT TOTALS:    | 19| 19|    |        0|    |        0|   |      0|    | 13135|
-----------------------------------------------------------------------------

-----------------------------------------
| ADVISOR |  PORT  |  TIMEOUT  |  STATUS  |
-----------------------------------------
|  MVS    | 10007  | unlimited |   ACTIVE |
-----------------------------------------
```

Figure 66. 2216 report manager console log

We noticed that the WLM load metrics had been imported by the MVS advisor. The manager had calculated new weights, based on executor connections and WLM metrics that favored the T03ATCP stack during this interval.

We also used the `REPORT ADVISOR` command (see Figure 67).

```
NDR >REPORT ADVISOR
Advisor name (0=ftp,1=http,2=MVS,3=TN3270,4=smtp,5=nntp,6=pop3,7=telnet) [1]? 2
Port number [0]? 10007

------------------------------
|      ADVISOR:      MVS      |
|      PORT:       10007      |
------------------------------
| 172.16.232.39    |    4451  |
| 172.16.250.3     |    4336  |
| 172.16.252.28    |    4359  |
------------------------------
```

Figure 67. 2216 report advisor console log

This command displays the current load metrics that the MVS advisor has retrieved from WLM.

3.3.3.2 NDR base sample advisor test 2

In this test we stopped the server application in stack T03ATCP and issued `sysplex2 c50 1234 -c 99 -b 0.1` on the client. Figure 68 shows the test results:

```
+----------------+----------------+
| Resolved Addr  | Resolved Count |
+----------------+----------------+
| 172.16.220.50  | 99             |

+----------------+----------------+
| Connected To   | Connected Count|
+----------------+----------------+
| 172.16.232.39  | 43             |
| 172.16.252.28  | 30             |
| ECONNREFUSED   | 26             |
```

Figure 68. NDR base sample advisor test 2

The outcome of this test was similar to the one in 3.3.2.2, “NDR base sample manager test 2” on page 84.

3.3.3.3 NDR base sample advisor test 3

We performed an extra test with the advisor setup. In this test we stopped OMPROUTE in stack T39ATCP. This meant that the VIPA address of that stack was not advertised by OSPF.

The server application on stack T03ATCP was started again and the test program was run. See Figure 69.

```
+---------------------+-----------------+---------------+--------+--------+
| Hostname            | Resolved Addr   | Connected Addr | Resolv | Connec |
+---------------------+-----------------+---------------+--------+--------+
| c50                 | 172.16.220.50   | 172.16.250.3   | 0.0300 | 0.0200 |
| c50                 | 172.16.220.50   | 172.16.252.28  | 0.0100 | 0.0100 |
| c50                 | 172.16.220.50   | 172.16.250.3   | 0.0100 | 0.0100 |
| c50                 | 172.16.220.50   | 172.16.252.28  | 0.0100 | 0.0100 |
| c50                 | 172.16.220.50   | 172.16.250.3   | 0.0100 | 0.0100 |
 :

+---------------+---------------+
| Resolved Addr  | Resolved Count |
+---------------+---------------+
| 172.16.220.50  | 99             |

+---------------+---------------+
| Connected To   | Connected Count|
+---------------+---------------+
| 172.16.250.3   | 43             |
| 172.16.252.28  | 56             |
```

Figure 69. NDR base sample advisor test 3

This time all our connections were distributed to the two servers whose addresses were known to NDR. See Figure 70 for the output of the `REPORT MANAGER` command.

```
NDR >REPORT MANAGER
 :
-----------------------------------------------------------------------------
|172.16.220.50   |WEIGHT | ACTIVE % 25  |  NEW % 25    | PORT % 25 |SYSTEM % 25|
|                -------------------------------------------------------------|
|PORT: 1234      |NOW|NEW| WT | CONNECT | WT | CONNECT | WT | LOAD | WT | LOAD |
|-----------------------------------------------------------------------------|
|172.16.232.39   |  0|  0|  10|        0|  10|        0|   0|     0|-999|    -1|
|172.16.250.3    |  6|  6|   6|       25|  10|       15|   0|     0|   9|  4351|
|172.16.252.28   |  8|  8|  13|        1|  10|       20|   0|     0|  10|  4067|
|-----------------------------------------------------------------------------|
|PORT TOTALS:    | 14| 14|    |       26|    |        0|    |     0|    |  8417|
-----------------------------------------------------------------------------
```

Figure 70. 2216 report manager console log

This test showed that the MVS advisor put a -1 in the system load field (marked down). Then the manager calculated new weights and the result was weight 0 for the unavailable server.

If you run multiple TCP/IP stacks on OS/390 it is important to know that the WLM advisor will listen on only one of the stacks. The stack that is available first will be the one on which WLM will listen.

There are two other useful commands available with NDR: `quiesce`, which prevents the executor from distributing connections to a certain server, and `unquiesce`, to make the server available again. The current connections are not affected by the `quiesce` command; they continue until finished. See Figure 71 for an example.

```
NDR >QUIESCE MANAGER
Server Address [0.0.0.0]? 172.16.252.28
172.16.252.28 has been marked to be quiesced.
NDR > REPORT MANAGER

----------------------------------
|  HOST TABLE LIST  |   STATUS   |
----------------------------------
|  172.16.232.39    |    ACTIVE  |
|  172.16.250.3     |    ACTIVE  |
|  172.16.252.28    |  QUIESCED  |
----------------------------------
 :
NDR >UNQUIESCE 172.16.252.28
172.16.252.28 has been marked to be active.
```

Figure 71. 2216 console log quiesce/unquiesce

3.3.3.4 NDR protocol advisors

There are some protocol advisors available that can prevent the executor from distributing connections to unavailable servers. The protocol advisors also provide load metrics to the manager. We configured three different protocol advisors and tried out the Telnet advisor in an example where one of the Telnet servers became unavailable.

Note that we configured the Telnet advisor to manage our TN3270 servers in our sysplex TCP/IP stacks. The specific TN3270 advisor is just as suitable for use with TN3270 servers in the IBM routers. See Figure 72 for the configuration statements.

```
NDR Config> ADD ADVISOR
Advisor name (0=ftp,1=http,2=MVS,3=TN3270,4=smtp,5=nntp,6=pop3,7=telnet) [1]? 0
Port number [21]?
Interval (seconds) [5]?
Timeout (0=unlimited) [0]?
NDR Config>ADD ADVISOR 1 80 5 0
NDR Config>ADD ADVISOR 7 23 5 0
NDR Config>ENABLE ADVISOR
Advisor name (0=ftp,1=http,2=MVS,3=TN3270,4=smtp,5=nntp,6=pop3,7=telnet) [0]? 0
Port number [0]? 21
Advisor ftp on port 21 interval was set to 5.
Advisor ftp on port 21 timeout was set to unlimited
This advisor is now enabled.
NDR Config>ENABLE ADVISOR 1 80
Advisor http on port 80 interval was set to 5.
Advisor http on port 80 timeout was set to unlimited
This advisor is now enabled.
NDR Config>ENABLE ADVISOR 7 23
Advisor telnet on port 23 interval was set to 5.
Advisor telnet on port 23 timeout was set to unlimited
This advisor is now enabled.
NDR Config>LIST ADVISOR

Executor: Enabled


Advisor:
        Name    Port    Interval        TimeOut         State       CommPort
        ftp     21      5               0               Enabled
        http    80      5               0               Enabled
        MVS     10007   5               0               Enabled
        telnet  23      5               0               Enabled
```

Figure 72. 2216 protocol advisor console log

We configured and enabled three protocol advisors: FTP, HTTP and Telnet. Then we performed the following test. We stopped OMPROUTE in the RA28 stack and started three TN3270 sessions to the cluster address. Figure 73 shows the output of `REPORT MANAGER` on the 2216. So far, this should work the same way with or without the Telnet advisor.

```
NDR >REPORT MANAGER
 :
NDR >REPORT MANAGER
-------------------------------------------------------------------------------
|172.16.220.50   |WEIGHT | ACTIVE % 25   |  NEW % 25    | PORT % 25 |SYSTEM % 25|
|                ------------------------------------------------------------|
|PORT:   23      |NOW|NEW| WT | CONNECT | WT | CONNECT | WT | LOAD | WT | LOAD |
|----------------------------------------------------------------------------|
|172.16.232.39  | 10| 10|  11|        1|  10|        1|  11|    30|  10|  4220|
|172.16.250.3   |  8|  9|   8|        2|  10|        1|  10|    30|   9|  4400|
|172.16.252.28  |  0|  0|  10|        0|  10|        0|   8|    20|-999|    -1|
|----------------------------------------------------------------------------|
|PORT TOTALS:   | 18| 19|    |        3|    |        0|    |    80|    |  8619|
-------------------------------------------------------------------------------
```

Figure 73. 2216 report manager console log

In this report the MVS advisor has marked 172.16.252.28 as down because we stopped OMPROUTE in the T28ATCP stack. The TN3270 sessions have been distributed to the two available servers: two sessions to 172.16.250.3 and one session to 172.16.232.39.

Next, we quiesced the Telnet server in the T39ATCP stack using the command `v tcpip,,t,quiesce`. This command stops new connections to the server from being established while the active ones continue to work. Then we started three more TN3270 sessions to the cluster. Figure 74 shows the `REPORT MANAGER` output now:

```
NDR >REPORT MANAGER
-----------------------------------------------------------------------------
|172.16.220.50   |WEIGHT | ACTIVE % 25  |  NEW % 25    | PORT % 25 |SYSTEM % 25|
|                ------------------------------------------------------------|
|PORT:   23      |NOW|NEW| WT | CONNECT | WT | CONNECT | WT | LOAD | WT | LOAD |
|----------------------------------------------------------------------------|
|172.16.232.39   |  0|  0|   8|        2|  10|        0|-999|    -1|   8|  4244|
|172.16.250.3    | 10| 10|  11|        5|  10|        3|  11|    30|  11|  4421|
|172.16.252.28   |  0|  0|  10|        0|  10|        0|   8|    20|-999|    -1|
|----------------------------------------------------------------------------|
|PORT TOTALS:    | 10| 10|    |        5|    |        0|    |    49|    |  8664|
-----------------------------------------------------------------------------
```

Figure 74. 2216 report manager console log

All our sessions are now on the 172.16.250.3 server, which has five active connections. The Telnet protocol advisor has marked the 172.16.232.39 server as down, and the manager has calculated a 0 weight to that server.

3.3.4 2216 NDR high availability

In this chapter we set up an NDR high availability configuration. To achieve this we installed a second 2216 as a backup in our network. We placed the backup 2216 in parallel to the original one. We connected the backup machine to the same token-ring as the first 2216 and to the same LPARs via ESCON MPCs. Figure 75 shows the network diagram with the second 2216 added.

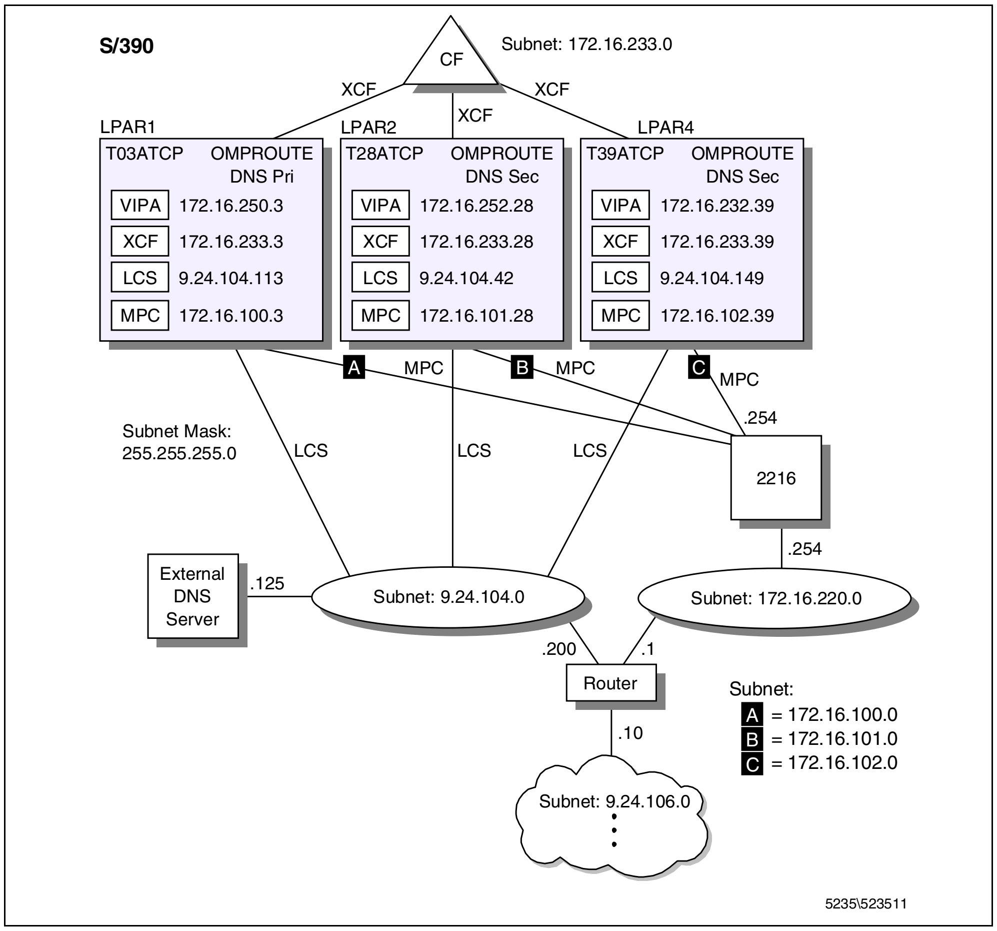


Figure 75. High availability Network Dispatcher configuration

The TCP/IP definitions we made were very similar to those in 6.5, “Configuring OMPROUTE” on page 161 as far as the following were concerned:

- Devices: token-ring and ESCON MPCs
- IP addresses
- OSPF definitions

For NDR we used the same definitions as in the NDR base example. We configured the same:

- Cluster
- Ports
- Servers
- Manager
- Advisor

in the backup machine as in the primary.

3.3.4.1 2216 NDR high availability configuration

The two Network Dispatcher machines are configured, one as primary and one as backup. At startup the primary machine sends all the connection data to the backup machine until that machine is synchronized. The primary machine becomes active, that is, it begins to route packets. The backup machine, meanwhile, monitors the status of the primary machine and is said to be in standby state.

If the backup machine at any point detects that the primary machine has failed, it performs takeover of the primary machine's routing functions and becomes the active machine. After the primary machine has once again become operational, the machines respond according to how the recovery takeback strategy has been configured:

- Automatic. The primary machine resumes routing packets as soon as it becomes operational again.
- Manual. The backup machine continues routing packets even after the primary becomes operational. Manual intervention is required to return the primary machine to active state and reset the backup machine to standby.

We configured backup for our 2216s as in Figure 76.

```
NDR Config>ADD BACKUP                                                    1
Role (0=PRIMARY, 1=BACKUP) [0]? 1                                        2
Switch back strategy(0=AUTO, 1=MANUAL [0]?                               3
NDR Config>ADD HEARTBEAT                                                 4
Source Heartbeat address [0.0.0.0]? 172.16.220.253
Target Heartbeat Address [0.0.0.0]? 172.16.220.254
NDR Config>ADD HEARTBEAT 172.16.103.253 172.16.100.254                   4
NDR Config>ADD HEARTBEAT 172.16.104.253 172.16.101.254                   4
NDR Config>ADD HEARTBEAT 172.16.105.253 172.16.102.254                   4
NDR Config>ENABLE BACKUP                                                 5
NDR Config>
```

Figure 76. 2216 NDR high availability configuration console log

1 The `ADD BACKUP` command configures the high availability option and sets the parameters.

2 We set the role to 0 in the primary 2216 and 1 in the backup 2216.

3 We configured the automatic backup strategy.

4 At least two heartbeat conversations should be configured (on different paths) to avoid unnecessary takeover. We configured four pairs; the same address pairs were defined on both machines but inverted in the one not shown.

5 The `ENABLE BACKUP` command starts the backup function at once.

In addition to the basic criteria of failure detection (heartbeat) there is another failure detection mechanism named reachability. When configuring reachability you provide a list of IP hosts that each of the Network Dispatchers must be able to reach in order to work correctly. The reachability configuration is invoked by the `add reach` command under the `NDR Config>` prompt. We did not configure any reachability criteria in our test environment.

3.3.4.2 NDR high availability test results

First we checked the status of our backup configurations in the 2216s. We used the `STATUS BACKUP` command to do this as in Figure 77 (primary) and Figure 78 (backup).

```
NDR >STATUS BACKUP
Dumping status ...
Role : PRIMARY   Strategy : AUTOMATIC  State : ND_ACTIVE Sub-State : ND_SYNCHRON
IZED
<<<Prefered target : 172.16.103.253>>>

Dumping HeartBeat Status ...

Dumping Reachability Status ...
```

Figure 77. Primary 2216 status backup console log

The primary 2216 was in active state; the heartbeats were running and the database synchronization was complete.

```
NDR >STATUS BACKUP
Dumping status ...
Role : BACKUP   Strategy : AUTOMATIC  State : ND_STANDBY Sub-State : ND_SYNCHRON
IZED
<<<Prefered target : 172.16.220.254>>>

Dumping HeartBeat Status ...

Dumping Reachability Status ...
```

Figure 78. Secondary 2216 status backup console log

The secondary 2216 was in standby state; the heartbeats were running and the database synchronization was complete. This machine was now fully qualified to take over in case of failure of the primary.

We started a couple of TN3270 sessions and we also used the multitasking version of our test socket application described in A.6, “Loading the system” on page 264. This application was configured to listen on port 2345.

The client program syntax is described in A.2, “Collecting statistics using REXX” on page 256. We issued the command `sysplex2 c50 2345 -c 3 -t 120 -b 3` from the client in 10 parallel sessions. The timer parameter (-t) was set so that the server would not reply before 120 seconds had passed. The purpose of this was to make sure that the active connections had been restored properly by the backup Network Dispatcher before ending the connection.

Before breaking anything, we issued `REPORT MANAGER` on the primary, as in Figure 79.

```
NDR >REPORT MANAGER
 :

 ------------------------------------------------------------------------------
 |172.16.220.50  |WEIGHT | ACTIVE % 25  |  NEW % 25    | PORT % 25 |SYSTEM % 25|
 |               ---------------------------------------------------------------|
 |PORT:   23     |NOW|NEW| WT | CONNECT | WT | CONNECT | WT | LOAD | WT | LOAD |
 |------------------------------------------------------------------------------|
 |172.16.232.39  |  8|  8|  13|        1|   9|        1|   0|     0|  10|  4310|
 |172.16.250.3   |  6|  6|   6|        2|   9|        0|   0|     0|   9|  4402|
 |172.16.252.28  |  6|  6|   9|        1|   9|        0|   0|     0|   9|  4427|
 |------------------------------------------------------------------------------|
 |PORT TOTALS:   | 20| 20|    |        3|    |        0|    |     0|    | 13139|
 ------------------------------------------------------------------------------


 :

 ------------------------------------------------------------------------------
 |172.16.220.50  |WEIGHT | ACTIVE % 25  |  NEW % 25    | PORT % 25 |SYSTEM % 25|
 |               ---------------------------------------------------------------|
 |PORT: 2345     |NOW|NEW| WT | CONNECT | WT | CONNECT | WT | LOAD | WT | LOAD |
 |------------------------------------------------------------------------------|
 |172.16.232.39  |  7|  7|   9|       17|   9|        0|   0|     0|  10|  4310|
 |172.16.250.3   |  6|  6|   9|       18|   9|        0|   0|     0|   9|  4402|
 |172.16.252.28  |  6|  6|   9|       15|   9|        0|   0|     0|   9|  4427|
 |------------------------------------------------------------------------------|
 |PORT TOTALS:   | 19| 19|    |       50|    |        0|    |     0|    | 13139|
 ------------------------------------------------------------------------------
```

Figure 79. Primary 2216 report manager console log

We noticed that the executor had registered several active connections on our test ports.

Now we were prepared to test if theory works in practice. We stopped the executor in the primary Network Dispatcher as you can see in Figure 80:

```
NDR Config>DISABLE EXECUTOR
Cluster 172.16.220.50 has been removed.
Executor is now disabled.
Advisor MVS on port 10007 exiting
Manager stopping....
Manager exiting
Manager stopped
```

Figure 80. Primary 2216 disable executor console log

Our TN3270 sessions remained active and the test applications continued to present valid results. We had a look at the secondary 2216, issuing `STATUS BACKUP` and `REPORT MANAGER` as seen in Figure 81.

```
NDR >STATUS BACKUP
Dumping status ...
Role : BACKUP   Strategy : AUTOMATIC  State : ND_ACTIVE Sub-State : ND_NOT_SYNCH
RONIZED

Dumping HeartBeat Status ...

Dumping Reachability Status ...
NDR >REPORT MANAGER

 :

--------------------------------------------------------------------------------
|172.16.220.50   |WEIGHT | ACTIVE % 25   |  NEW % 25    | PORT % 25 |SYSTEM % 25|
|                ---------------------------------------------------------------|
|PORT:   23      |NOW|NEW| WT | CONNECT | WT | CONNECT | WT | LOAD | WT | LOAD |
|------------------------------------------------------------------------------|
|172.16.232.39   |  7|  7|  11|        7|   9|        1|   0|     0|  10|  4324|
|172.16.250.3    |  6|  6|   8|       11|   9|        0|   0|     0|   9|  4430|
|172.16.252.28   |  7|  7|  10|        9|   9|        1|   0|     0|   9|  4408|
|------------------------------------------------------------------------------|
|PORT TOTALS:    | 20| 20|    |       25|    |        0|    |     0|    | 13162|
--------------------------------------------------------------------------------

 :

--------------------------------------------------------------------------------
|172.16.220.50   |WEIGHT | ACTIVE % 25   |  NEW % 25    | PORT % 25 |SYSTEM % 25|
|                ---------------------------------------------------------------|
|PORT: 2345      |NOW|NEW| WT | CONNECT | WT | CONNECT | WT | LOAD | WT | LOAD |
|------------------------------------------------------------------------------|
|172.16.232.39   |  6|  6|   9|       21|   9|        0|   0|     0|   9|  4324|
|172.16.250.3    |  6|  6|   9|       22|   9|        0|   0|     0|   9|  4430|
|172.16.252.28   |  7|  7|  10|       18|   9|        0|   0|     0|  10|  4408|
|------------------------------------------------------------------------------|
|PORT TOTALS:    | 19| 19|    |       61|    |        0|    |     0|    | 13162|
--------------------------------------------------------------------------------
```

Figure 81. Secondary 2216 status backup/report manager console log

The secondary 2216 had done its job and was now in active state; no heartbeats were running and the database was not synchronized.

The reports showed increased connection counters and the test application connections were evenly distributed.

Then we tried to start the executor in the primary 2216 again. Since the takeback strategy was set to automatic, we should be able to recover the dispatching to the primary 2216 again, without any manual intervention except to start the executor. See Figure 82.

```
NDR Config>ENABLE EXECUTOR
NDR >STATUS BACKUP
Dumping status ...
Role : PRIMARY   Strategy : AUTOMATIC  State : ND_ACTIVE Sub-State : ND_SYNCHRON
IZED
<<<Prefered target : 172.16.103.253>>>

Dumping HeartBeat Status ...

Dumping Reachability Status ...
NDR >REPORT MANAGER

 :

-------------------------------------------------------------------------------
|172.16.220.50   |WEIGHT | ACTIVE % 25  |  NEW % 25    | PORT % 25 |SYSTEM % 25|
|                ---------------------------------------------------------------|
|PORT:   23      |NOW|NEW| WT | CONNECT | WT | CONNECT | WT | LOAD | WT | LOAD |
|-----------------------------------------------------------------------------|
|172.16.232.39   |  7|  7|  10|        6|   9|        0|   0|     0|  10|  4323|
|172.16.250.3    |  6|  6|   8|       11|   9|        0|   0|     0|   9|  4431|
|172.16.252.28   |  7|  7|  10|        8|   9|        0|   0|     0|   9|  4407|
|-----------------------------------------------------------------------------|
|PORT TOTALS:    | 20| 20|    |       25|    |        0|    |     0|    | 13161|
-------------------------------------------------------------------------------

 :

-------------------------------------------------------------------------------
|172.16.220.50   |WEIGHT | ACTIVE % 25  |  NEW % 25    | PORT % 25 |SYSTEM % 25|
|                ---------------------------------------------------------------|
|PORT: 2345      |NOW|NEW| WT | CONNECT | WT | CONNECT | WT | LOAD | WT | LOAD |
|-----------------------------------------------------------------------------|
|172.16.232.39   |  7|  7|  10|       21|   9|        0|   0|     0|  10|  4324|
|172.16.250.3    |  7|  7|   9|       22|   9|        0|   0|     0|  10|  4430|
|172.16.252.28   |  5|  6|   9|       19|   9|        0|   0|     0|   9|  4407|
|-----------------------------------------------------------------------------|
|PORT TOTALS:    | 19| 20|    |       62|    |        0|    |     0|    | 13161|
-------------------------------------------------------------------------------
```

Figure 82. Primary 2216 status backup/report manager console log

The STATUS BACKUP command showed that the primary 2216 was in charge again, in active state. All our sessions continued to work and the reports showed that the connection counters continued to change.

Chapter 4. Dynamic VIPA (for application instance)

By providing IP addresses for the OS/390 applications that are not associated with a specific physical network attachment or gateway, VIPA enables fault tolerance against outages in the IP interfaces on the OS/390 host. However, in previous 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 OS/390 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 5, “Automatic VIPA takeover and takeback” on page 127.

4.1 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 OS/390) 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 OS/390 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 has alleviated this disruptiveness by allowing a smooth transition in VIPA ownership.

4.2 Implementing Dynamic VIPA

It is inherently 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 83.

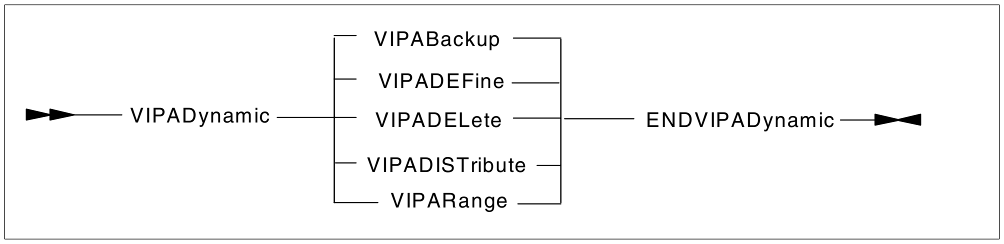


Figure 83. Definition format for Dynamic VIPA statement

VIPADEFine, VIPABackup, VIPADELete, and VIPADISTribute statements are used in conjunction with the automatic takeover flavor of VIPA as discussed in Chapter 5, “Automatic VIPA takeover and takeback” on page 127. 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 7, “Sysplex Distributor” on page 187.

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.

4.2.1 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, by 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 4.2.2, “Solutions for applications that bind() to INADDR_ANY” on page 103.
- 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 the application corresponds 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 84.

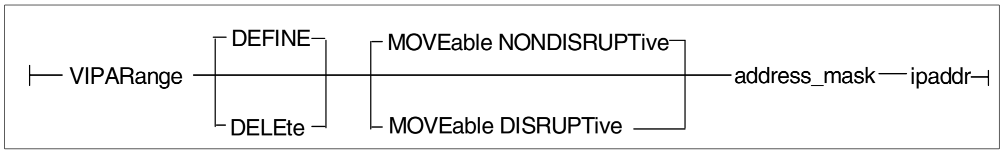


Figure 84. Definition format for VIPARange

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.

For our tests in 4.2.3, “Examples of Dynamic VIPA” on page 104, we use the configuration in Figure 85.

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

Figure 85. 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.

4.2.2 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, it is not possible to determine which VIPA IP address should be associated with it. There are three solutions for 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 advanced 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 A.4.1, “Modifying SOCSRVR for Dynamic VIPA” on page 261 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 OS/390 V2R10 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 new name of utility EZBXFDVP, already 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 documentation anymore. See 4.2.3.1, “Using utility MODDVIPA” on page 104 for an example.

4.2.3 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

4.2.3.1 Using utility MODDVIPA

Figure 86 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 86. 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
```

Figure 87. Message for authorization error

After authorizing the user, the job was executed again. Figure 88 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 88. Display NETSTAT,HOME

Figure 89 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 89. Display SYSPLEX,VIPADYN

4.2.3.2 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 90 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 90. Port definition

Refer to *OS/390 IBM Communications Server: IP Configuration Reference,* SC31-8726 for a complete description of the PORT statement.

We established a TN3270 connection to the DVIPA IP address and issued some display commands. Figure 91 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 91. Display NETSTAT,HOME

Figure 92 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 92. 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.

4.2.3.3 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 93 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 93. Application jcs

With this invocation, the application binds to port 4343 and the specific IP address 172.16.240.193. Figure 94 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 94. Display NETSTAT,HOME

Figure 95 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 95. Display SYSPLEX,VIPAD

Figure 96 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 96. Display NETSTAT,CONN

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

Figure 97 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 97. Display NETSTAT,HOME after the application ends

4.3 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.

4.3.1 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 98. In this regard, the movement of the DVIPA is *non-disruptive*. The DVIPA is eventually deleted on the old stack when all existing connections to the old stack terminate.

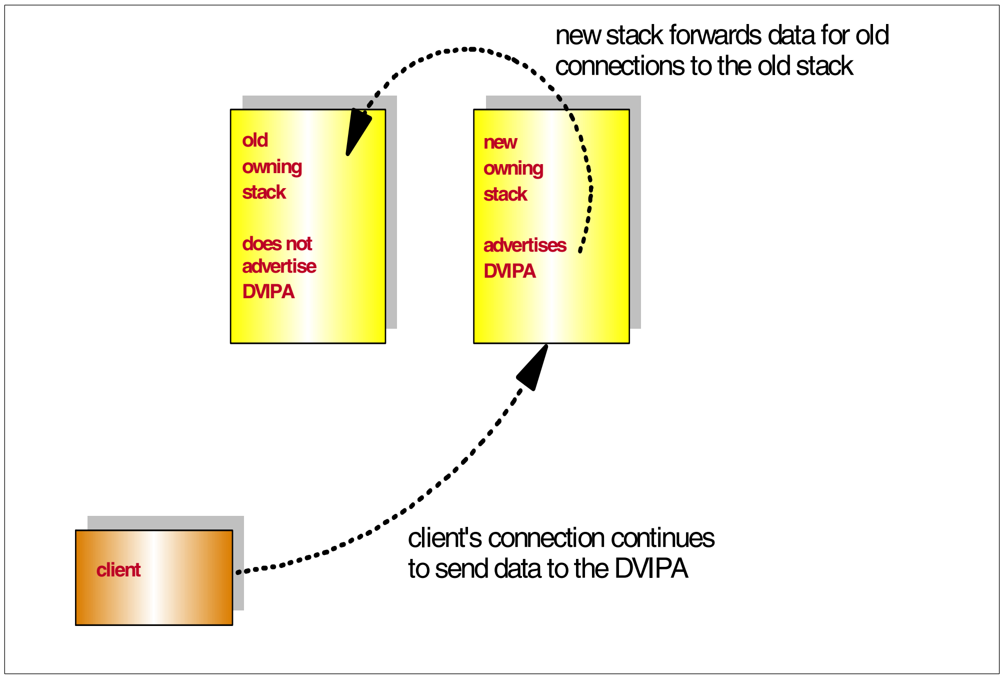

new stack forwards data for old connections to the old stack
old owning stack
does not advertise DVIPA
new owning stack
advertises DVIPA
client
client's connection continues to send data to the DVIPA

Figure 98. With non-disruptive 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 OS/390 V2R10 IP. Both stacks should be at a V2R10 (or later) code level or the behavior will be DISRUPTive.

4.3.2 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.

4.3.3 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 following environment was used:

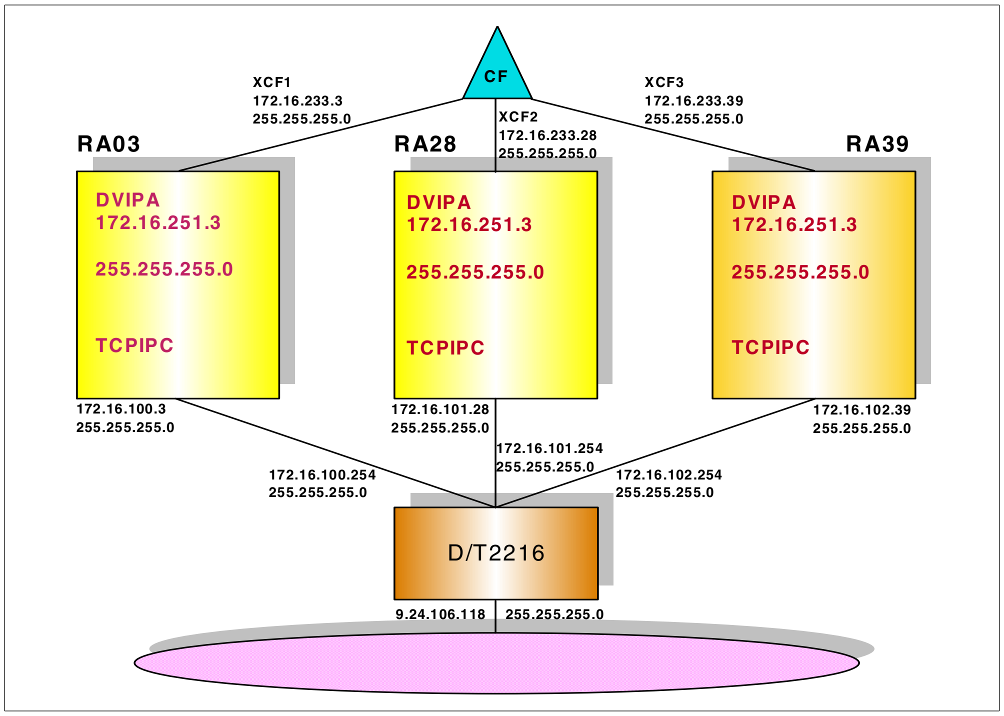


Figure 99. Test environment

For convenience, stack TCPIPC on RA39 was not used.

4.3.3.1 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 in Figure 85 on TCPIPC on RA03.
2. Define Dynamic VIPA IP address in VIPARange as in Figure 85 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 100 and Figure 101.

```
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 100. 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 101. 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 102.
7. Transfer a big file from a client to a 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 102. Display NETSTAT,CONN

8. Run MODDVIPA to activate Dynamic VIPA 172.16.240.193 on TCPIPC on RA28. Figure 103 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 103. 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 104 and Figure 105. 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 104. 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 105. Display SYSPLEX,VIPAD

10.Check if the FTP connections are still active on TCPIPC on RA03. Figure 106 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 106. Display NETSTAT,CONN

Note: 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 107 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 107. Display NETSTAT,CONN

4.3.3.2 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 108 on TCPIPC on RA03.
2. Define Dynamic VIPA in VIPARange as shown in Figure 108 on TCPIPC on RA28.

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

Figure 108. 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 109 and Figure 110.

```
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 109. 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 110. 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 111.
7. Transfer 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 111. Display NETSTAT,CONN

8. Run MODDVIPA to activate Dynamic VIPA 172.16.240.193 on TCPIPC on RA28. Figure 112 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 112. Console log

9. Verify the Dynamic VIPA 172.16.240.193 was deleted from stack TCPIPC on RA03 (see Figure 113) and activated in stack TCPIPC on RA28 (see Figure 114).

```
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 113. 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
```

Figure 114. Display NETSTAT,HOME on RA28

10. Verify the status of Dynamic VIPA IP address on the sysplex as shown in Figure 115.

```
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 115. 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 116.

```
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 116. Display NETSTAT,CONN

4.3.3.3 bind() and VIPARange defined as MOVEable NONDISRUPTive

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

1. Define Dynamic VIPA in VIPARange as in Figure 85 on TCPIPC on RA03.
2. Define Dynamic VIPA in VIPARange as in Figure 85 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 117 and Figure 118 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 117. 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
```

Figure 118. Display NETSTAT,HOME

5. Establish a connection to Dynamic VIPA 172.16.240.193 (TN3270). This connection can be seen in Figure 119.

```
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 119. Display NETSTAT,CONN

6. Start the same application to bind to Dynamic VIPA 172.16.240.193 on RA28. Figure 120 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 120. Console log

7. Verify the Dynamic VIPA 172.16.240.193 was activated on TCPIPC on RA28 as shown in Figure 121.

```
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 121. 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 connection on this stack. Figure 122 and Figure 123 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
```

Figure 122. 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 123. Display SYSPLEX,VIPAD

9. Verify the connections status on TCPIPC on RA03 as illustrated in Figure 124.

```
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 124. 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 125.

```
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 125. Display NETSTAT,CONN

4.3.3.4 bind() and VIPARange defined as MOVEable DISRUPTive

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

1. Define Dynamic VIPA in VIPARange as in Figure 108 on TCPIPC on RA03.
2. Define Dynamic VIPA in VIPARange as in Figure 108 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 126.

```
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 126. Display NETSTAT,CONN

4. Verify the Dynamic VIPA 172.16.240.193 is activated on TCPIPC on RA03 as illustrated in Figure 127 and Figure 128.

```
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
```

Figure 127. 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 128. 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 129.

```
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 129. Messages for application jcs

6. Verify the Dynamic VIPA 172.16.240.193 is still active on TCPIPC on RA03. Figure 130 shows the home list display and Figure 131 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
```

Figure 130. 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 131. 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 131.

```
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 132. Display NETSTAT,CONN

For 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 *OS/390 IBM Communications Server: IP Configuration Guide*, SC31-8725.

Chapter 5. Automatic VIPA takeover and takeback

By providing IP addresses for the OS/390 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 S/390. 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 has solved this problem and has 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 will be covered in detail in Chapter 7, “Sysplex Distributor” on page 187.

5.1 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 OS/390 IP.

5.1.1 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 OS/390, it is extremely common to have more than one link into an OS/390 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 OS/390 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 OS/390 IP stack, and the TCP stack looks like another router that happens to have reachability to that IP address. On the OS/390 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 OS/390.

5.1.2 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, 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.

A new type of DVIPA is provided in CS for OS/390 V2R10 IP called a *distributed DVIPA* that is used for implementing the Sysplex Distributor function. This 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 7, “Sysplex Distributor” on page 187. This function is new in CS for OS/390 IP V2R10.

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 4, “Dynamic VIPA (for application instance)” on page 101. This function is available since CS for OS/390 IP V2R8.

5.1.3 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 OS/390 XCF messaging (the same mechanism as dynamic XCF and sysplex sockets), so that all stacks know, for each dynamic VIPA:

- 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 insures that the DVIPA is still reachable.

Figure 133 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.

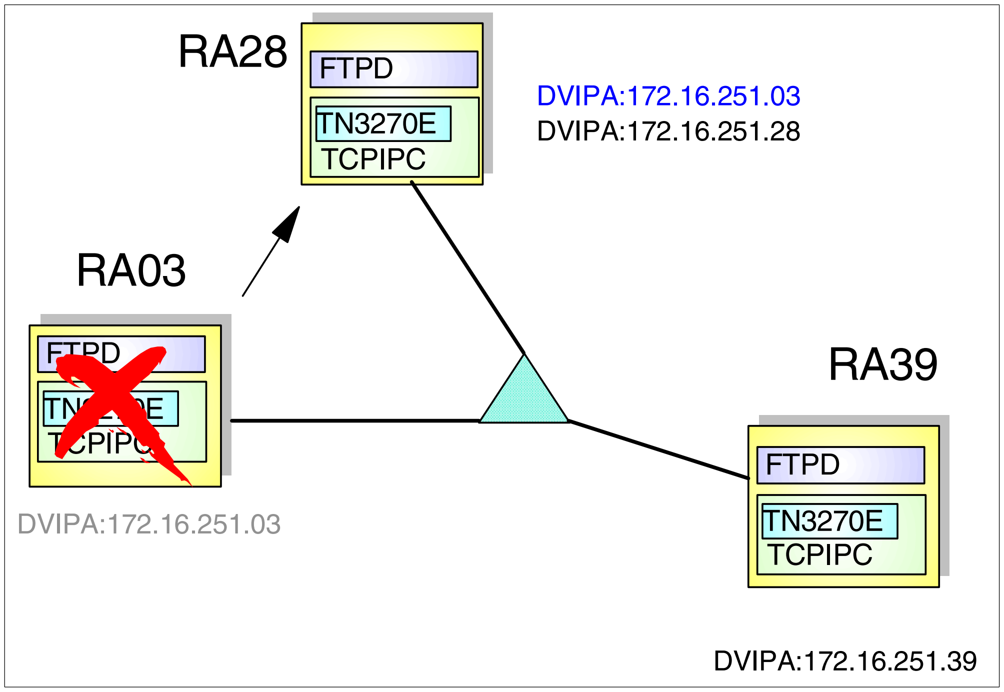


Figure 133. Automatic VIPA takeover of 172.16.251.03 by stack RA28

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 134. 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 OS/390 V2R10 IP. If one of the stacks is running CS for OS/390 IP V2R8, the dynamic VIPA remains on the current backup stack as long as there are active connections to that dynamic VIPA on that stack. In CS for OS/390 IP V2R10, this older behavior can be enabled by setting different parameters on the VIPADEFINE statement.

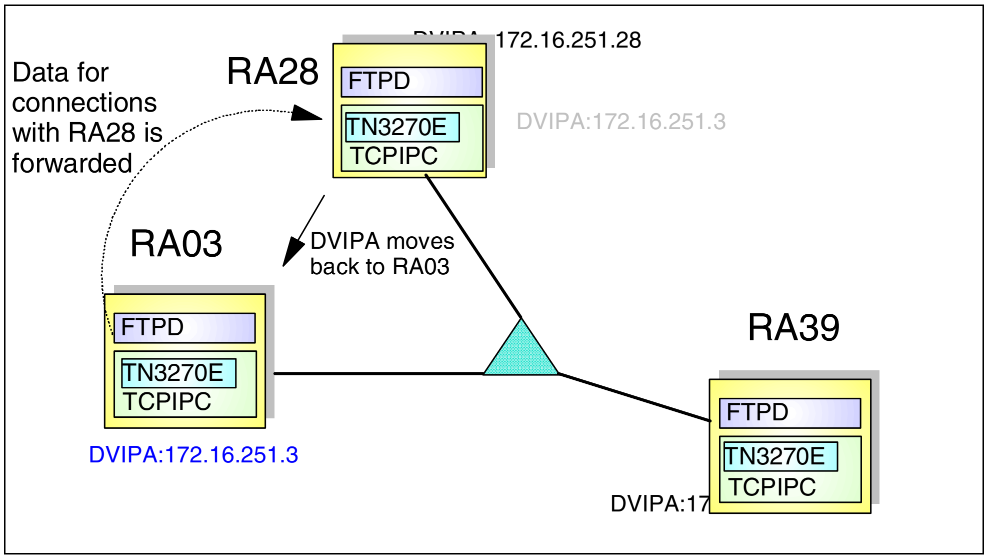


Figure 134. 172.16.251.03 is taken back by the restarted RA03 stack

5.1.4 Benefits of sysplex-wide VIPA takeover

When a stack or its underlying OS/390 fails, it is not necessary to restart the stack with the same configuration profile on a different OS/390 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 receive 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.

5.1.5 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 has been improved in CS for OS/390 V2R10 IP.

If the original stack has defined a VIPA address supporting the Sysplex Distributor function (see Chapter 7, "Sysplex Distributor" on page 187), 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.

5.2 Implementing VIPA takeover and takeback

In this section we give an overview of how VIPA takeover and takeback can be defined; 5.4, "Examples of VIPA takeover and takeback" on page 136 shows practical examples.

5.2.1 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 OS/390 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 135.

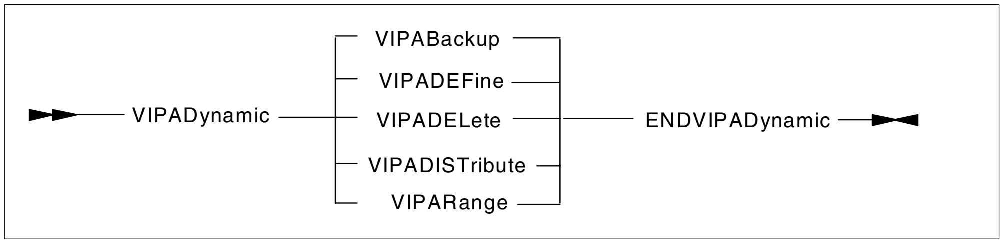


Figure 135. Definition format for dynamic VIPA block

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 136.

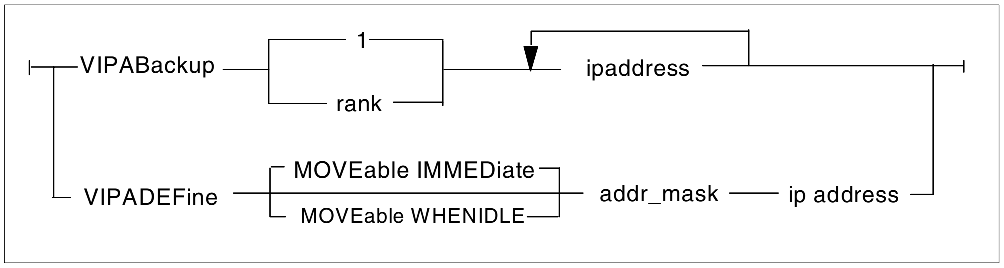


Figure 136. Definition format for VIPA backup

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), all bits should be 0.
- If every subnet mask is logically ANDed with all IP address in the list, the result should be the same.

CS for OS/390 IP V2R10 introduces the parameters MOVEable IMMEDiate and MOVEable 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 TCP/IP profile.

The VIPABackup statement designates one or more VIPAs for which this stack will provide automatic backup when the owing 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.

5.3 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 137 shows the base network configuration used.

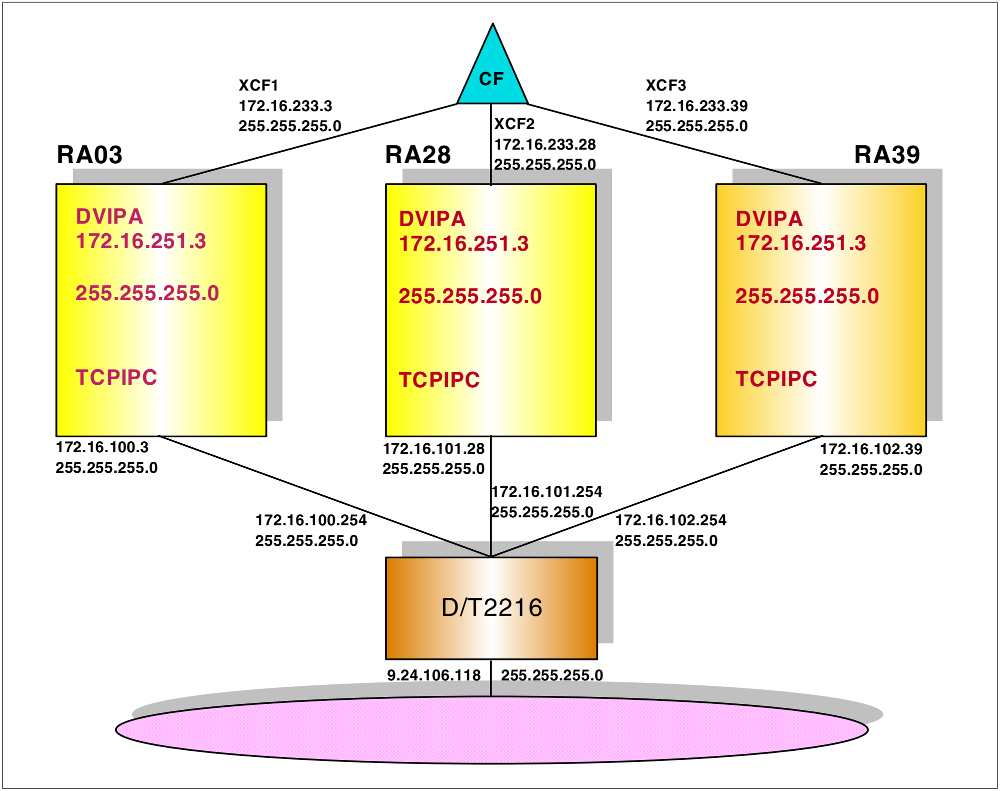


Figure 137. Base network configuration

5.3.1 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 138 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 138. 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.

5.3.2 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 for all 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 new 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 139).

```
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 139. 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 140.

```
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 140. Display NETSTAT,VIPADFCG command

Figure 141 shows the results of a `DISPLAY N,VIPADYN` command. The same display can be achieved using `onetstat -v` from a USS 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 141. 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 142 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 142. Display NETSTAT,VCRT command

5.4 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.

5.4.1 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 143 shows our VIPA definitions on TCPIPC on RA03, and Figure 144 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 143. 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 144. Dynamic VIPA definition for TCPIPC on RA28

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 145 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 145. Display VIPA definition for TCPIPC on RA28 before TCPIPC on RA03 stack failure

We also displayed the OSPF routing table on TCPIPC on RA28, as shown in Figure 146. 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 146. Display 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 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 147.

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

Figure 147. Console message on RA28 at VIPA takeover

We then issued some TCP/IP display commands on RA28 to check the new status, as in Figure 148.

```
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
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
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 148. 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 149 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 149. FTP connection using the VIPA 172.16.251.3 after stopping TCPIPC on RA03

Figure 150 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 150. 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 151 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 151. 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 152 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 152. FTP connection using the VIPA 172.16.251.3 after restarting TCPIPC on RA03

Figure 153 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 153. Display VCRT on RA03 and RA28 after takeback

Figure 154 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 154. Display VIPAD after takeback

5.4.2 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 155 shows our VIPA definitions on TCPIPC on RA28 and Figure 156 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 155. 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 156. Dynamic VIPA definition for TCPIPC on RA28

Before stopping the TCPIPC on RA03, we used the `DISPLAY SYSplex` command to see our definitions. Figure 157 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 157. 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 158.

```
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 158. 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 159.

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

Figure 159. Console message on RA28 at VIPA takeover

Figure 160 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 160. 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 161 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 161. Display NETSTAT,VIPADYN after stopping TCPIPC on RA03

Figure 162 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 162. 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 163 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 163. FTP connection after restarting TCPIPC on RA03

Figure 164 shows active connections to VIPA address 172.16.251.3 on TCPIPC on RA28.

```
RO 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 164. Display NETSTAT,SOCKETS after restarting TCPIPC on RA03

Figure 165 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 165. Display SYSPLEX,VIPAD after restarting TCPIPC on RA03

Figure 166 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 166. 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 167.

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

Figure 167. 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 168.

```
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 168. Display NETSTAT,HOME after takeback

Figure 169 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 169. Display SYSPLEX,VIPAD after takeback

Figure 170 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 170. FTP connection after takeback

Chapter 6. Routing in a sysplex environment

In this chapter we discuss some of the interesting issues related to routing in a sysplex environment. Although this chapter concentrates primarily on the use of dynamic IP routing, we also make mention of the *pre-routing* done by an OSA in a shared configuration. The OSA family of adapters (including OSA-2 and OSA-Express) allows the sharing of one single adapter among multiple sysplex LPARs. As a result, a shared OSA must determine which sysplex LPAR a particular incoming packet will be forwarded to based on the destination IP address of the packet. When using Dynamic VIPA or Network Dispatcher, problems may arise in making this pre-routing decision. Please see 6.1, “Shared OSA pre-routing” on page 152 for additional information.

The bulk of this chapter deals with dynamic routing in the sysplex. Although one can use static routing in a sysplex environment, exclusively using static routes is limiting. In order to take full advantage of new functions in CS for OS/390 IP such as Dynamic VIPA and Sysplex Distributor, static routes are not sufficient.

Dynamic routing protocols provide for the reactive behavior of hosts in an environment in which networks, network adapters, and hosts come and go. A dynamic routing protocol is implemented by a routing daemon, which is simply an application running on a host that understands current routing and receives information from neighboring routers regarding the state of the network. It updates routes in the local routing table and informs other neighbors of any changes that it has been made aware of. The routing daemon running in each of the network hosts work together to collectively react to changes in the network topology.

We elaborate on the differences between static and dynamic routing in 6.2, “IP routing overview” on page 153. We discuss the routing issues involved with VIPA in a sysplex. Additionally, we consider two main routing protocols in this chapter. The first of these is the Routing Information Protocol (RIP), which is a distance-vector routing algorithm. A more sophisticated routing algorithm is used in the Open Shortest Path First (OSPF) routing protocol. We detail the implementation of these protocols with the routing daemons provided by CS for OS/390 IP, ORouteD and OMPROUTE.

For the remainder of this chapter, we will refer to the sample topology shown in Figure 171. This topology includes two LPARs in a sysplex, each containing one stack. Additionally, each LPAR has two links to the 2216 router in our network, one MPC+ and one via a shared OSA adapter (LCS). In some scenarios in this chapter, however, both links are not used. Within the sysplex, XCF links connect the stacks.

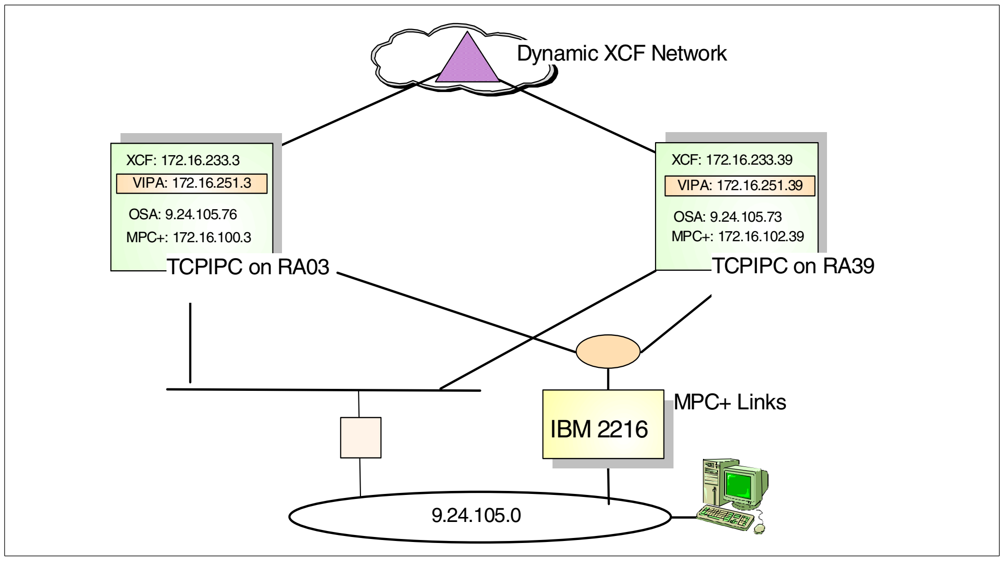


Figure 171. Sample network topology

6.1 Shared OSA pre-routing

Because multiple sysplex LPARs (and stacks) can simultaneously make use of an OSA adapter, the card itself must make certain pre-routing decisions regarding incoming packets. That is, when a packet arrives at the OSA adapter, the only indication as to which LPAR (and stack) it should be forwarded is the IP information contained within it. In particular, the ultimate destination of the IP packet is used for this purpose. Each of the LPARs (stacks) owns a set of IP addresses and will receive all IP packets destined for them. Additionally, exactly one LPAR can be identified as the default LPAR, thereby receiving packets with IP destination addresses not owned by any LPAR.

When making use of VIPA in a shared OSA environment, the OSA must be made aware of the VIPA in the owning stack. The OSA simply treats the VIPA as another IP address owned by the particular LPAR (stack). If the VIPA is active on multiple LPARs (stacks) within the sysplex, however, the OSA might be confused as to which LPAR (stack) it should direct packets. With CS for OS/390 V2R10 IP, this is not yet a problem since all of this data must be forwarded via the distributing stack anyway (please refer to Chapter 7, “Sysplex Distributor” on page 187). With Network Dispatcher, however, the cluster address is active multiple LPARs within the sysplex, meaning that a shared OSA configuration would not function properly with Network Dispatcher. This restriction is corrected with the expected introduction of Generic Routing Encapsulation (GRE) within CS for OS/390 IP and Network Dispatcher.

The method in which the OSA adapter is informed of a particular LPAR’s (stack’s) IP addresses is different depending on the type of adapter it is. With OSA-Express, this is done dynamically via communication between CS for OS/390 IP and the OSA adapter. In OSA-2 (which we are using in our topology), this is done via manual configuration. The configuration is in the form of the OSA Address Table (OAT) and we can simply include the VIPA address in this configuration. We issued the `IOACMD get OAT` command to show you the RA03 entry in Figure 172. If the address is not in an OAT entry, the packet is forwarded to the default image (stack). If there is no default, the OSA adapter discards the packet.

```
***************************************************************************
*** OSA/SF Get OAT output created 09:17:39 on 09/21/2000             ***
*** IOACMD APAR level - OW39984      ** sandra.sample(OATTABD8)      ***
***************************************************************************
***             Start of OSA address table for CHPID D8             ***
***************************************************************************
* UA(Dev) Mode     Port     Entry specific information     Entry  Valid
***************************************************************************
                              LP 1 (A1      )
00(2060) passthru   00  PRI 009.024.104.113                 SIU    ALL
                            172.016.250.003
01(2061) passthru   00  PRI 009.024.104.113                 SIU    ALL
                            172.016.250.003
02(2062) passthru   00  no  009.024.104.033                 S      ALL
                            172.016.251.004
03(2063) passthru   00  no  009.024.104.033                 S      ALL
                            172.016.251.004
04(2064) passthru   01  no  009.024.105.076                 S      ALL 1
                            172.016.251.003                            2
05(2065) passthru   01  PRI 009.024.105.076                 S      ALL
                            172.016.250.003
06(2066) N/A                                                N/A    CSS
07(2067) N/A                                                N/A    CSS
08(2068) N/A                                                N/A    CSS
09(2069) N/A                                                N/A    CSS
0A(206A) N/A                                                N/A    CSS
0B(206B) SNA        00                                      S      ALL
0C(206C) N/A                                                N/A    CSS
0D(206D) N/A                                                N/A    CSS
0E(206E) N/A                                                N/A    CSS
```

Figure 172. OSA address table

1,**2** We are using device 2064 for the RA03 host and defined both IP addresses in the OAT: 9.24.105.76 (OSA card IP address) and 172.16.251.3 (VIPA address).

6.2 IP routing overview

One of the major functions of a network protocol such as TCP/IP is to connect together a number of disparate networks efficiently. These networks may include LANs and WANs, fast and slow, reliable and unreliable, inexpensive and expensive connections. The simplest way to connect them all together is to bridge them. However, this results in every part of the network receiving all traffic and leads to slower links being overloaded and perhaps to network failure altogether. What is needed, particularly when all these networks are joined together in the worldwide Internet, is some form of intelligence at the boundaries of all these networks, which can look at the packets flowing and make rational decisions as to where and how they should be forwarded. This function is known as IP routing. Routing allows you to create networks that can be managed separately but which are still linked and can communicate with one another.

IP is the de facto standard for most large routed networks. It is also the protocol used on the global Internet. In order to route packets, each network interface on a device (PC, UNIX workstation, router, mainframe, and so on) on the network has a unique IP address. Whenever a packet is sent, the destination and source addresses are included in the packet's header information. Routers examine the destination address to see if there is a matching address in their routing tables. These tables are either created by the system administrator, or built dynamically using information received from other routers, or (often) a combination of both.

Along with the IP address of each interface, a *subnet mask* is also defined, which indicates to the network the distinction between the part of the address that represents the subnetwork (a LAN or point-to-point connection, for example) and the part that represents the host (the network interface on a device). The use of a subnet mask allows flexibility in network design (LANs may be small or large), but the proper use of routing tables requires that the subnetwork address and the host portion of an IP address can be discerned.

Figure 173 on page 154 shows that the routing function in a TCP/IP network is performed at the internetwork layer (layer 3) of the architectural model, whereas the IP subnetworks are represented by the data link layers. Each node on the connection path from client to server must:

- Inspect the destination address of the packet (192.168.200.1)
- Divide it into subnetwork and host addresses
- Determine whether the subnetwork (represented by the DLC layer) is directly attached to it
- If not, forward the packet to the next router as defined by the routing tables
- If so, forward the packet directly to the destination over the appropriate DLC

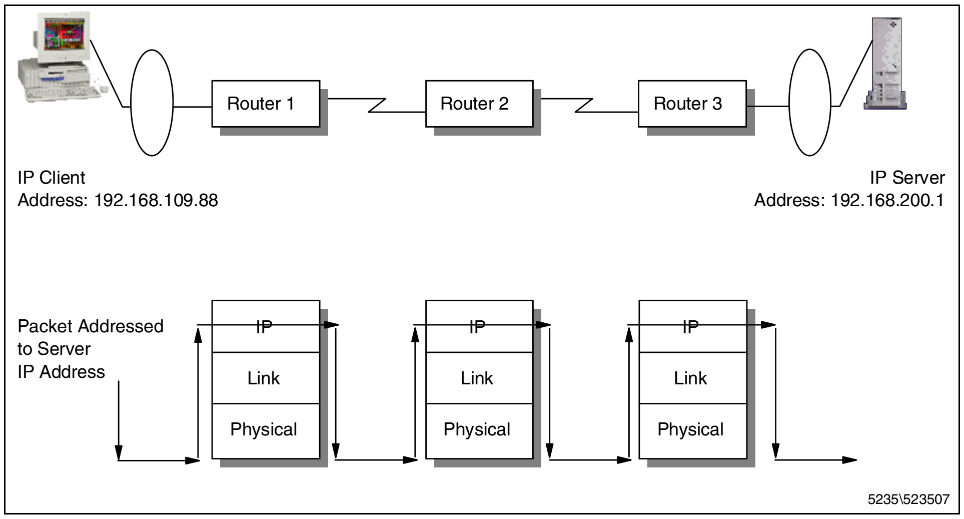


Figure 173. Network routing flow

The Internet is logically divided into *autonomous systems*, which are essentially individual customers' networks. There are two classes of routing protocols used in IP: exterior gateway protocols (EGPs) are used between autonomous systems and interior gateway protocols (IGPs) are used within the autonomous systems.

The two most common standard IGPs are the Routing Information Protocol (RIP) and the Open Shortest Path First (OSPF) protocol.

Routing in the CS for OS/390 IP system can be set up to use dynamic or static routing:

- Static routes

  With static routing, the paths to reach networks and hosts are hard coded in a routing file, accessible to a TCP/IP host. Each host has its own set of definitions. If something changes (a route, host or network is added or deleted), then the static routes of some or all hosts in a network may need to be updated. Static routing is suitable for small, stable networks, but quite inadequate for large or changing scenarios. With static routing, however, one has better administrative control over address allocation and resource access.

  In CS for OS/390 IP, static routes are configured in the TCPIP profile by coding either a BEGINROUTES/ENDROUTES block statement or by making use of the GATEWAY statement. The BEGINROUTES block was created in CS for OS/390 V2R10 IP to overcome inconsistencies and sometimes awkward syntax of the GATEWAY statement. It defines static IP routing table entries in standard BSD format. For more information, please consult *OS/390 IBM Communications Server: IP Configuration Reference*, SC31-8726.

- Dynamic routes

  Dynamic routing removes the need for coding static routing tables. All router addressing and path information is built dynamically. These tables are automatically exchanged between all routers in a network. This information sharing enables routers to calculate the best path through the network to any given destination.

Whether static or dynamic routes are implemented, the basic routing mechanisms are the same. Every IP host can route IP datagrams and maintains an IP routing table. The table indicates the IP address of the next hop in order to route an IP datagram. Each host or router along the way needs to know only the next hop IP address in the path. Routing tables, of course, need to be maintained in both directions.

When taking advantage of dynamic routing, an IP host employs the use of a *routing daemon*. The routing daemon adds, deletes, or changes route entries within the host's routing table. Additionally, a routing daemon executing on a particular host communicates with routing daemons on neighbor hosts to exchange topological routing information. Ultimately, this information exchange leads to the update of routing table entries.

CS for OS/390 IP implements both RIP and OSPF dynamic routing protocols. Before CS for OS/390 V2R6, only RIP was available and was implemented by the ORouteD server (in V2R5) and by RouteD (in previous releases). In V2R6 a new server called OMPROUTE was introduced, which runs under UNIX System Services and provides both RIP and OSPF functionality. ORouteD may be withdrawn from CS for OS/390 in due course, leaving OMPROUTE as the only routing daemon. Both ORouteD and OMPROUTE are UNIX System Services applications and require the HFS

Often, you will come across the term *RouteD* meaning routing daemon. This is a common term for a RIP server.

ORouteD and its predecessor RouteD are so named because they are indeed RIP servers. The expression *GateD* is also used, usually for a router with more function such as OSPF or EGP capability.

6.2.1 RIP

RIP is an internal gateway protocol (IGP) designed to manage relatively small networks. RIP uses a hop count (distance vector) to determine the best possible route to a network or host. The hop count is also known as the routing *metric*. A router is defined as being zero hops away from its directly connected networks, one hop from networks that can be reached through one gateway (router), and so on. In RIP, a hop count of 16 means infinity, or the destination cannot be reached. Thus, very large networks with more than 15 hops between potential partners cannot make use of RIP.

The RIP server broadcasts routing information (in other words, its own distance vector tables) to the gateways of directly connected networks every 30 seconds. The server receives updates from neighboring gateways periodically and updates its routing tables. If an update is not received for three minutes, the gateway is assumed down and all the routes through that gateway are set to a metric of 16 (infinity). The server can, for example, determine if a new route has been created, if a route is temporarily unavailable, or if a more efficient route exists. A complete definition of RIP Version 1 is documented in RFC 1058.

RIP Version 2 is compatible with existing RIP Version 1 implementations. It also supports variable subnet masks. Variable subnet mask support means that an IP device can use different subnet masks on each of its network interfaces. It requires, in general, that both the TCP/IP stack and the dynamic update protocol that is used by that stack support variable length subnet masks. The TCP/IP stack from OS/390 V2R5 IP onward can be enabled for variable subnetting via the VARSUBNETTING keyword on IPCONFIG, and the ORouteD server supports both RIP Version 2 and Version 1 protocols.

Techniques such as immediate next hop for shorter paths and multicast addressing are used in RIP Version 2 to reduce the load on hosts. A full description of this protocol is detailed in RFCs 1721, 1722, 1723 and 1724.

In CS for OS/390, RIP servers (whether ORouteD or OMPROUTE) can run within a multiple-stack IP environment. A copy of the server has to be started for each stack. The server determines which TCP/IP stack it should connect to based on the TCPIPJOBNAME keyword in the file pointed to by the environmental variable RESOLVER_CONFIG.

The routing implementation we used is illustrated as part of the testing scenarios later in this book.

6.2.2 OSPF

Where RIP is based on distance vectors (hop counts), OSPF is based on link states. In other words, OSPF routing tables contain details of the connections between routers, their status (active or inactive), their cost (desirability for routing) and so on. Updates are broadcast whenever a link changes status, and consist merely of a description of the changed status. This is in contrast with RIP where broadcasts occur every 30 seconds and contain the complete distance vector tables. Because of this difference, and for other reasons such as the lack of the 16-hop limit, OSPF is more suitable for large networks than RIP.

In fact, OSPF is similar in concept to APPN, where the routers (network nodes) maintain the network topology and broadcast any changes whenever they occur. OSPF, like APPN, can divide its network into topology subnets (known as *areas*) within which broadcasts are confined. The current version (V2) of OSPF is described fully in RFC 2328.

The OMPROUTE server in CS for OS/390 makes use of advanced OSPF V2 techniques for improving availability and performance. For example, it can balance the connection load among up to four alternative routes if the costs of the routes are equal (equal cost multipath). It can also interact with VIPA (see “High availability with Virtual IP Addressing (VIPA)” on page 5), whether using RIP or OSPF, to ensure that an IP route is always available to host applications as long as at least one interface to the network is active.

6.3 VIPA considerations

Because VIPAs are different from normal physical interface IP addresses, there are some special considerations that must be taken when using them. When a host owns a VIPA, it advertises reachability to it as if it were one hop away from it. Dynamic VIPAs can come and go; the dynamic routing protocol must therefore distribute this information to neighboring routers.

6.3.1 VIPA address assignment

It is strongly recommended that you define a subnet for VIPA separate from the physical interfaces. We defined the same subnet for all VIPA IP addresses in our sysplex environment. The routing daemon must therefore advertise the host and subnet routes of each VIPA to downstream routers. ORouteD and OMPROUTE have this capability.

In our network, shown in Figure 171, our 2216’s route table will include a host route for each of the VIPAs (172.16.251.3 and172.16.251.39) through the local token-ring and MPC+ interfaces. It will also have a subnet route for the 172.16.251.0 subnet through one of the interfaces. This subnet route is not a concern because host routes take precedence over subnet routes; hence the subnet route will not be used.

Note: You should make sure that your routers accept host routes. Otherwise, each VIPA address must be in a separate subnetwork from each other.

6.3.2 Fault tolerance with VIPA

To improve fault tolerance beyond the use of VIPA, you can make use of redundant hardware. In our environment, we have an OSA card and an MPC-attached 2216 router connected to both hosts RA03 and RA39. The RA03 host is using the MPC+ link for outbound packets. If the interface goes down:

- The routing daemon in the RA03 host will switch the outbound traffic from the MPC+ interface to the OSA interface as soon as it detects the interface failure
- The RA03 adjacent routers will update their route table in light of the new information being provided by RA03

In our sysplex environment, we also have the XCF link that can be used as a backup interface in case of failure. We can use the RA39 host as a path from RA03 to network 9.24.105.0 after an MPC+ interface failure instead of using the OSA interface.

6.3.3 Beware of ICMP redirection

In general, we found that the alternate routing available when VIPA was implemented allowed us to maintain sessions whenever one path failed and alternates were available. However, there are a couple of issues:

1. Some workstations can dynamically add routes even when dynamic routing is disabled on the workstation. With a Windows NT server on the same network as the OSA card and the 2216 router, we found a route created dynamically on the Windows NT server. We had set the 2216 router as the default router for the Windows NT server. The 2216 router was aware of the existence of the MP redirect message so that it would send data directly to a host owning the VIPA via the OSA card.

   We set up a TN3270 session from Windows NT to the VIPA and then killed the OSA card link. Our session died. When we looked at the routes on the Windows NT server, we found a host route for the VIPA via the OSA card IP address. It took some time for this route to disappear.

   This scenario would be uncommon for a user's workstation, but many installations may well have Windows NT servers on the same network as their OS/390 servers.

2. Routing protocols require time to converge whenever the status of a link changes. If the session (TN3270, for example) times out before the routers can converge, then the session will be lost. In the case of host links into the OS/390 stack, then probably only one or two levels of routers need to change routing tables to react to a network change. That will normally occur quickly, but you should keep this in mind.

   If the routing protocol were OSPF, the convergence time can be improved because the OSPF protocol is based on the link state and not a distance vector.

6.3.4 Using SOURCEVIPA

Coding SOURCEVIPA in the profile will make the TCP/IP stack insert a static VIPA IP address as a source address on certain outbound packets. If SOURCEVIPA is coded, the VIPA address will be used for UDP (and RAW) data originating from this host and for new outbound connections in which the application does not specify which local IP address should be used.

The VIPA address used is always the closest static VIPA configured above the address of the outgoing interface in the home list. That is, once a packet's outgoing interface is determined, the local IP address corresponding to that interface is located in the home list. The stack then chooses the lowest VIPA in the home list above the local IP address found. Since this selection depends on VIPAs higher in the home list, Dynamic VIPAs are never candidates for selection. Only static VIPAs are used with SOURCEVIPA.

We usually code the VIPA IP address as the first in the home list and the physical interface next will be used to transport outbound datagrams. If you do not want to use the VIPA address as a source in packets through a particular link, you can specify its IP address before the VIPA IP address in the home list.

6.4 Configuring ORouteD

ORouteD is a routing daemon that implements the RIP protocols described in RFC 1058 (RIP Version 1) and RFC 1753 (RIP Version 2). In our environment, we introduced RIP routing into the two TCPIPC stacks shown in Figure 171.

We added the following procedure and parameter files to enable RIP for our testing. Figure 174 shows the JCL that runs the ORouteD server under UNIX System Services.

```
//OROUTEDC  PROC MODULE='BPXBATCH'
//OROUTED   EXEC PGM=&MODULE,REGION=4096K,TIME=NOLIMIT,
//    PARM='PGM /usr/lpp/tcpip/sbin/orouted -h'
//STDOUT    DD PATH='/tmp/orouted.&SYSCLONE.c.stdout',
//          PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//STDERR    DD PATH='/tmp/orouted.&SYSCLONE.c.stderr',
//          PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//          PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
//STDENV    DD DSN=TCPIP.TCPPARMS(RD&SYSCLONE.CENV),DISP=SHR
//CEEDUMP   DD SYSOUT=*,DCB=(RECFM=FB,LRECL=132,BLKSIZE=132)
//SYSERR    DD PATH='/tmp/orouted.&SYSCLONE.c.syserr',
//          PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//          PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
```

Figure 174. Procedure for OROUTDC

We can use this start procedure by both TCPIPC stacks, in the RA03 host and RA39 host, since we are using system variables (&SYSCLONE.). The `-h` parameter allows ORouteD to handle host routes, which is necessary for point-to-point connections to be broadcast. The `-hv` parameter enables host virtual routing, which enables adjacent routers in the network to receive VIPA host routes. We highly recommend that you assign a separate subnetwork for the VIPA. The same subnetwork can be used for all VIPA addresses in the sysplex environment.

Figure 175 shows the TCP/IP stack profile statements relevant to our ORouteD discussion.

```
IPCONFIG
 VARSUBNETTING                                              1
 IGNOREREDIRECT                                             2
 DATAGRAMFWD                                                3
 SOURCEVIPA                                                 4
 SYSPLEXROUTING                                             5
 DYNAMICXCF 172.16.233.3 255.255.255.0 1                    6
;
PORT
 520 UDP OROUTEDC                                           7
;
AUTOLOG
 OROUTEDC                                                   8
ENDAUTOLOG

VIPADYNAMIC                                                 9
  VIPADEFINE MOVE IMMED 255.255.255.0 172.16.251.3
  VIPABACKUP 100 172.16.251.39
ENDVIPADYNAMIC

;
BSDROUTINGPARMS FALSE
 VIPA03C   DEFAULTSIZE 0  255.255.255.0 0
 EN103        1492     1  255.255.255.0 0
 M032216B  DEFAULTSIZE 0  255.255.255.0 0                   10
ENDBSDROUTINGPARMS
```

Figure 175. Statements in profile for OROUTEDC (stack RA03)

1 VARSUBNETTING makes this stack support variable subnet since ORouteD is going to work with RIP-2.

2 IGNOREREDIRECT parameter makes the stack ignore ICMP redirect packets.

3 DATAGRAMFWD parameter enables the forwarding of datagrams through this stack.

4 The SOURCEVIPA parameter forces the TCP/IP stack to insert the VIPA address as the source field in some outbound packets (see 6.3, “VIPA considerations” on page 157). In this example, coding SOURCEVIPA has no effect since we have no static VIPAs defined.

5 SYSPLEXROUTING parameter specifies that this host is a part of a sysplex environment and will communicate the interface changes to WLM. The following message will confirm that it is enabled: `SysplexRouting support is enabled`.

6 Dynamic XCF support is enabled by this parameter. 172.16.233.3 is the XCF IP address used for home list. The subnet mask and the metric will be used for BSDROUTINGPARMS.

7 The PORT statement is to reserve port 520 for ORouteD.

8 AUTOLOG statement starts the ORouteD server when the TCPIPC stack starts.

9 The VIPADYNAMIC block defines the Dynamic VIPAs used by this stack.

10 BSDROUTINGPARMS statements are to define the interface information, MTU value, and subnet mask. BEGINROUTES or GATEWAYS statements are not used with ORouteD. If you need to define any static routes, you must code an ORouteD gateway file.

```
RESOLVER_CONFIG=//'TCPIP.TCPPARMS(TCPD03C)'
ROUTED_PROFILE=//'TCP.TCPPARMS(RD03CCFG)'
```

Figure 176. STDENV parameter file RD03CENV

The contents of our standard environment file are listed in Figure 176. This file is pointed to by the STDENV DD statement in the start procedure. This DD JCL card is part of the BPXBATCH function, and is a mechanism for setting USS environment variables. For details on how to define an environment, see the *OS/390 UNIX System Services User's Guide*, SC28-1891.

The first variable controls where to find the TCPIP.DATA file, or the equivalent of `/etc/resolv.conf`. You can also set TCPIP.DATA in the `ENVAR` parameter in your EXEC JCL card.

```
RIP_SUPPLY_CONTROL: RIP2
RIP_RECEIVE_CONTROL: ANY
```

Figure 177. RouteD profile

In Figure 177 we show the defined profile used by the RouteD daemon. We enabled RIP Version 2 to supply control and RIP Version 1 and Version 2 receiving packets for our interfaces.

We did not define a static network and subnet route or static host route. If this is needed, you can define it in the ORouteD GATEWAY file. In the GATEWAY file you can also alter the default values for the RIP timers and define filters to the interfaces.

Our procedures and data files were identical for the TCPIPC stacks on RA03 and RA39 hosts. The only changes were the necessary (obvious) stack, file, and address differences.

6.5 Configuring OMPROUTE

OMPROUTE implements the OSPF protocol, described in RFC 1583 (OSPF Version 2) and RFC 1850 (OSPF subagent protocol), as well as the RIP protocol, described in RFC 1058 (RIP Version 1) and RFC 1723 (RIP Version 2). The IBM Communications Server for OS/390 IP running the OMPROUTE server becomes an OSPF and RIP router. If both OSPF and RIP protocols are used simultaneously, OSPF routes will be preferred over RIP routes to the same destination.

6.5.1 Common OMPROUTE configuration

Some of the configuration of OMPROUTE is the same regardless of whether you are implementing RIP, OSPF, or both in your network. This includes the creation of the OMPROUTE started procedure and other global configuration steps. For this section, we again use the same environment depicted in Figure 171 on page 152.

OMPROUTE can be started as a start procedure. OMPROUTE deletes all routes when it is started and rebuilds your route table based on your configuration files.

```
//OMPROUTE PROC
//OMPROUTE EXEC PGM=OMPROUTE,REGION=4096K,TIME=NOLIMIT,
// PARM=('POSIX(ON)','CTRACE(CYIORA00)',
//       'ENVAR("_CEE_ENVFILE=DD:STDENV")/-t1')
//STDOUT   DD PATH='/tmp/omproute.stdout',
//         PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//         PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
//STDERR   DD PATH='/tmp/omproute.stderr',
//         PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//         PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
//STDENV   DD DSN=TCPIP.TCPPARMS.R2617(OM&SYSCLONE.CENV),DISP=SHR
//CEEDUMP  DD DUMMY
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//CEEDUMP  DD SYSOUT=*,DCB=(RECFM=FB,LRECL=132,BLKSIZE=132)
```

Figure 178. OMPROUTED startup procedure

Unlike ORouteD, the OMPROUTE does not use the BSDROUTINGPARMS statement to define its route table. It has its own configuration file pointed to by the STDENV JCL card as shown in Figure 179.

```
RESOLVER_CONFIG=//'TCPIP.TCPPARMS(TCPD03C)'
OMPROUTE_FILE=//'TCPIP.TCPPARMS.R2617(OM03CFG)'
```

Figure 179. STDENV file

RESOLVER_CONFIG points to the TCPIP.DATA data set or file. The resolver uses the following search order to allocate the actual resolver configuration data set or file to use:

1. The environment value of RESOLVER_CONFIG
   - The syntax for an MVS data set name is //'mvs.dataset.name'.
   - The syntax for an HFS file name is /dir/subdir/file.name.
2. /etc/resolv.conf
3. userid.TCPIP.DATA for TSO/E or jobname.TCPIP.DATA for a batch request
4. SYS1.TCPPARMS(TCPDATA)

OMPROUTE_FILE points to the OMPROUTE configuration file. A single configuration file is used to define the OSPF and RIP environments. The following search order is used to locate the OMPROUTE server configuration data set or file:

1. The value of the OMPROUTE_FILE environment variable
   - The syntax rule is the same as for RESOLVER_CONFIG

2. /etc/omproute.conf
3. hlq.ETC.OMPROUTE.CONF

We used CEE_ENVFILE with MVS data set pointing to STDENV. In this case the data set must be allocated with RECFM=V, because RECFM=F enables padding with blanks in environment variables.

The contents of TCPIP.DATA are shown in Figure 180.

```
TCPIPJOBNAME TCPIPC                    1
HOSTNAME  MVS03C
DOMAINORIGIN  itso.ral.ibm.com
NSINTERADDR  9.24.106.15
NSPORTADDR 53
RESOLVEVIA UDP
RESOLVERTIMEOUT 10
RESOLVERUDPRETRIES 2
DATASETPREFIX TCPIP                    2
MESSAGECASE MIXED
```

Figure 180. TCPIP.DATA

1 The TCPIPJOBNAME statement is used by OMPROUTE to establish connection with TCPIPC stack.

2 The value specified in this statement is used as a high-level qualifier in the search order for the OMPROUTE configuration file.

Figure 181 on page 164 shows a part of the TCPIPC profile we used on the RA03 host.

```
;******************************* TOP OF DATA *******************
;*TCPIP.TCPPARMS.R2617(PROF03C) SYSPLEX DISTRIBUTOR RA03 - SANDRAEF
;*****************************************************************
IPCONFIG
 DATAGRAMFWD                          1
 SYSPLEXROUTING                       2
 ARPTO  1200
 IGNOREREDIRECT                     3
 DYNAMICXCF 172.16.233.3 255.255.255.0 1      4
 STOPONCLAWERROR
 TTL    60
 VARSUBNETTING                       5

PORT
 520 UDP OMPROUTC                     6
;520 UDP OMVS

AUTOLOG 5
 OMPROUTC                           7
ENDAUTOLOG

 DEVICE M032216B MPCPTP AUTORESTART
 LINK   M032216B MPCPTP M032216B

 DEVICE EN103  LCS           2064
 LINK   EN103     ETHEROR802.3  1     EN103

VIPADYNAMIC
  VIPADEFINE MOVE IMMED 255.255.255.0 172.16.251.3
  VIPABACKUP 100 172.16.251.39
ENDVIPADYNAMIC

HOME
 172.16.100.3 M032216B
 9.24.105.76  EN103

START M032216B
START EN103
```

Figure 181. TCPIPC profile

1 DATAGRAMFWD parameter enables the forwarding of datagrams through this stack.

2 SYSPLEXROUTING parameter specifies that this host makes a part of a sysplex environment and will communicate the interface changes to WLM. The following message will confirm that it is enabled: `SysplexRouting support is enabled.`

3 IGNOREREDIRECT parameter makes the stack ignore ICMP redirect packets.

4 Dynamic XCF support is enabled by this parameter. 172.16.233.3 is the XCF IP address used for the home list. The subnet mask and metric will be defined in the OMPROUTE configuration file.

5 VARSUBNETTING makes this stack support a variable subnet since this server is going to work with RIP-2.

6 The PORT statement is to reserve port 520 for the OMPROUTC daemon. This is only needed if RIP is being implemented. If OMPROUTE is started from the UNIX System Services shell, the PORT statement should be used to reserve the RIP as shown here:

```
PORT

    520 UDP OMVS
```

7 AUTOLOG statement starts the OMPROUTC server when the TCPIPC stack starts.

6.5.2 OMPROUTE with RIP

The OMPROUTE daemon can be used to implement RIP. Since it can also implement OSPF, OMPROUTE can make it easier to migrate your network's routing protocol from RIP to OSPF.

The contents of the OMPROUTE configuration file are found in Figure 182 on page 166. The OMPROUTE routing application reads the file to determine which routing protocol it is going to use: RIP and/or OSPF protocols. If you want to implement your CS for OS/390 IP as a RIP router, you must configure your interfaces with the RIP_Interface statement. To implement OSPF protocol over an interface, define this interface with the OSPF_Interface statement. OMPROUTE can handle both routing protocols at the same time.

Instead of the BSD routing parameters, parameters such as maximum transmission unit (MTU), subnet mask, and interface name are configured via the OSPF_interface, RIP_interface, and Interface statements in the OMPROUTE configuration file.

```
ROUTESA_CONFIG ENABLED=YES                                              1
              COMMUNITY="MVSsubagent";
RIP_Interface IP_Address=172.16.100.3                                   2
              Name=M032216B
              Subnet_mask=255.255.255.0
              RIPV2=Yes
              In_Metric=0
              Out_Metric=0
              MTU=32768;
RIP_Interface IP_Address=9.24.105.76                                    3
              Name=EN103
              Subnet_mask=255.255.255.0
              RIPV2=Yes
              In_Metric=1
              Out_Metric=1
              MTU=32768;
Interface IP_Address=172.16.251.*                                       4
              Subnet_mask=255.255.255.0
              In_Metric=2
              Out_Metric=2
              MTU=32768;
RIP_Interface IP_Address=172.16.233.*                                   5
              Subnet_mask=255.255.255.0
              RIPV2=Yes
              In_Metric=2
              Out_Metric=2
              MTU=32768;
```

Figure 182. OMPROUTC configuration file

1 This statement is coded to configure the OMPROUTE SNMP subagent to communicate with the CS for OS/390 IP agent. The COMMUNITY parameter must match the one defined in SNMP.

2 This statement sets the RIP parameters for the 2216 ESCON MPC+ interface. The IP address and the name of the interface should match the ones on the HOME statement in the TCP/IP profile. We configured a lower metric for the 2216 ESCON MPC+ than the other interfaces to ensure that this interface will take precedence over others.

3 This statement sets the RIP parameters for the OSA card interface.

4 We configured DVIPA using the Interface statement. Because the stacks have DVIPA and DVIPA ranges defined, this statement is coded with a wild card value. By using a wild card, we ensure that all configured interfaces whose IP address matches the wild card will be configured as an interface. A point-to-point interface that is neither an OSPF nor a RIP interface should be configured to OMPROUTE via the Interface statement.

5 This statement sets the OSPF parameters for the XCF links between the stacks in the sysplex. This time we specified the IP address with a wild card (*): 172.16.233.* instead of 172.16.233.3.

6.5.3 OMPROUTE with OSPF

Our next configuration was set up to perform some tests with the OSPF protocol. Again we make use of the network configuration described in Figure 171 on page 152 which includes our two systems in separate LPARs, RA03 and RA39.

Dynamic routing was enabled by starting the OMPROUTE daemon on each stack. We configured OMPROUTE to use the OSPF routing protocol. The two external routers were also configured to run OSPF. Since we were connected to a production network (the 9.0.0.0 IBM network), we also received some OSPF information from other production routers.

The contents of the OMPROUTE configuration file are found in Figure 183 on page 168. The OMPROUTE routing application reads the file to determine which routing protocol to implement: OSPF and/or RIP protocols. If you want to implement your CS for OS/390 IP as an OSPF router, you must configure the interfaces with the OSPF_Interface statement. If there is an interface that implements the RIP protocol, you define this interface with the RIP_Interface statement. OMPROUTE can handle both routing protocols at the same time.

Instead of the BSDROUTINGPARMS block, parameters such as maximum transmission unit (MTU), subnet mask, and interface name parameters are configured in the OSPF_interface, RIP_interface, and Interface statements in the OMPROUTE configuration file.

```
Area    Area_Number=0.0.0.0                                  1
Stub_Area=NO
              Authentication_type=None;
              RouterID=172.16.100.3;                         2
              ROUTESA_CONFIG ENABLED=YES                     3
              COMMUNITY="MVSsubagent";
OSPF_Interface IP_Address=172.16.100.3                       4
              Name=M032216B
              Cost0=3
              Subnet_mask=255.255.255.0
              MTU=32768;
OSPF_Interface IP_Address=9.24.105.73                        5
              Name=EN103
              Cost0=6
              Subnet_mask=255.255.255.0
              MTU=32768;
OSPF_Interface IP_Address=172.16.251.*                       6
              Subnet_mask=255.255.255.0
              Cost0=8
              Non_Broadcast=Yes
              MTU=32768;
OSPF_Interface IP_Address=172.16.233.*                       7
              Subnet_mask=255.255.255.0
              Cost0=8
              Non_Broadcast=Yes
              MTU=32768;
OSPF_Interface IP_Address=172.16.252.*
              Subnet_mask=255.255.255.0
              Cost0=8
              Non_Broadcast=Yes
              MTU=32768;
AS_Boundary_routing                                          8
              Import_Direct_Routes=YES;
```

Figure 183. OMPROUTE configuration file OM03CCFG

1 These parameters are set for an OSPF area. In our network the only area is defined as the backbone area (Area_Number=0.0.0.0) and as a non-stub area (Stub_Area=NO). If you specify Stub_Area=YES, the area will not receive any inter-autonomous system (AS) routes. You cannot configure virtual links through a stub area or a router within the stub area as an AS boundary router.

2 ROUTERID statement should be configured to assign every router a unique router ID. It is highly recommended that the router ID be an IP address of a physical interface or a static VIPA, not a Dynamic VIPA.

3 This statement is coded to configure the OMPROUTE SNMP subagent to communicate with the CS for OS/390 IP agent. The COMMUNITY parameter must match the one defined in SNMP.

4 This statement sets the OSPF parameters for the 2216 ESCON MPC+ interface. The IP address and the name of the interface should match the ones on the HOME statement in the TCP/IP profile.

5 This statement sets the OSPF parameters for the OSA token-ring interface. We configured a higher cost for this interface than the 2216 router interface.

6 We configured DVIPA using the Interface statement. Because the stacks have DVIPA and DVIPA ranges defined, this statement is coded with a wild card value. By using a wild card, we ensure that all configured interfaces whose IP address matches will be configured as interfaces. A point-to-point interface that is neither an OSPF nor a RIP interface should be configured to OMPROUTE via the Interface statement.

7 This statement sets the OSPF parameters for the XCF links between the stacks in the sysplex. This time we specified the IP address with a wild card (*): 172.16.233.* instead of 172.16.233.3.

8 This statement enables AS boundary routing capability. It allows OMPROUTE to import routes learned from other methods such as the RIP protocol, static routes, and direct routes from other ASs into this OSPF domain. Because we have defined VIPA on the Interface statement, Import_Direct_Routes=YES is necessary. This definition will import the direct routes to VIPA into the OSPF routing domain and the routes will be advertised to the adjacent routers.

4, 5, 6, 7 The MTU size defined on each OSPF interface must not exceed the MTU size defined on the IP interface in the partner OSPF router. The OSPF MTU size is exchanged between routers, and some products (such as OS/390) check that the largest possible OSPF packet can be received on the appropriate interface. If not, OSPF is disabled on that interface.

Note: According to the Information APAR II11555, the OSPF_INTERFACE statement is recommended for use when configuring VIPA interfaces to OMPROUTE in an OSPF environment. There is no need to define Import_Direct_Routes=YES in the AS_Boundary_routing statement for the VIPA interface. If the OSPF protocol is *not* being used on *any* interfaces, then the Interface statement is used to configure the VIPA to OMPROUTE. Please refer to the text of APAR II11555 for details. We did not test this alternative form of definition.

6.5.3.1 OMPROUTE commands

OMPROUTE is controlled from the MVS operator console using the MVS system commands. We introduce some commands to display OSPF and RIP configuration and state information, and `MODIFY` commands to control OMPROUTE. You will find more details in *OS/390 IBM Communications Server: IP Configuration Guide*, SC31-8725.

To display all OSPF configuration information, you can issue `Display TCPIP,<tcpipjobname>,OMPROUTE,OSPF,LIST,ALL`. Sample results are shown in Figure 184 on page 170:

```
RO RA39,D TCPIP,TCPIPC,OMP,OSPF,LIST,ALL
D TCPIP,TCPIPC,OMP,OSPF,LIST,ALL
EZZ7831I GLOBAL CONFIGURATION 610
    TRACE: 1, DEBUG: 0, SADEBUG LEVEL: 0
    STACK AFFINITY:          TCPIPC                 1
    OSPF PROTOCOL:           ENABLED                2
    EXTERNAL COMPARISON:     TYPE 2                 3
    AS BOUNDARY CAPABILITY:  ENABLED                4
    IMPORT EXTERNAL ROUTES:  DIR SUB                5
    ORIG. DEFAULT ROUTE:     NO                     6
    DEFAULT ROUTE COST:      (1, TYPE 2)            7
    DEFAULT FORWARD. ADDR.:  0.0.0.0                8
    DEMAND CIRCUITS:         ENABLED                9
EZZ7832I AREA CONFIGURATION                         10
AREA ID          AUTYPE          STUB? DEFAULT-COST IMPORT-SUMMARIES?
0.0.0.0          0=NONE           NO          N/A           N/A

EZZ7833I INTERFACE CONFIGURATION       11
IP ADDRESS      AREA             COST  RTRNS  TRNSDLY PRI  HELLO
DEAD
172.16.233.39   0.0.0.0             8      5      1     1     10       12
40
172.16.233.39   0.0.0.0             8      5      1     1     10
40
172.16.233.39   0.0.0.0             8      5      1     1     10
40
172.16.251.39   0.0.0.0             8      5      1     1     10
40
9.24.105.73     0.0.0.0             6      5      1     1     10
40
172.16.102.39   0.0.0.0             3      5      1     1     10
40
```

Figure 184. Display command to list OSPF definitions

1 The stack on which OMPROUTE is running.

2 OSPF protocol enabled.

3 Comparison value defined at this server. OSPF supports two types of external metrics (see 6.5.5, “Interface cost considerations” on page 176): Type 1 external metrics are equivalent to the link state metric and Type 2 external metrics are greater than the cost of any path internal to the AS.

4 Indicates whether the router will import external routes into the OSPF domain.

5 Displays the type of external routes that this OSPF domain will import. This is displayed only when AS Boundary Capability is enabled.

6 Indicates whether the router will originate a default route into the OSPF domain.

7 Displays the default route cost.

8 Displays the forwarding address, in our example the default forwarding address is the backbone address. The forwarding address is displayed only when AS Boundary is enabled.

9 Indicates the demand circuit support availability.

10 Displays information about configuration area(s); in our example only the backbone area exists.

11 Displays information about the interface IP address and configured parameters. OSPF dynamically obtains this address from the HOME statement in the TCP/IP profile data set.

12 This is the DYNAMICXCF IP address that OSPF obtains dynamically from the TCP/IP profile data set.

We issued the command from the RA03 host to the RA39 host through the `ROUTE` command. You can issue the following `modify` command to OMPROUTE `F omproutejobname,OSPF,LIST,ALL` to get the same display.

To display current run-time statistics related to the OSPF interface use `Display TCPIP,<tcpipjobname>,OMPROUTE,OSPF,InterFaces, NAME=<if_name>`. Figure 185 shows a typical result:

```
D TCPIP,TCPIPC,OMP,OSPF,IF,NAME=M032216B
EZZ7850I INTERFACE DETAILS 574
                INTERFACE ADDRESS:      172.16.100.3
                ATTACHED AREA:          0.0.0.0
                PHYSICAL INTERFACE:     M032216B
                INTERFACE MASK:         255.255.255.0
                INTERFACE TYPE:         P-2-MP       1
                STATE:                  16           2
                DESIGNATED ROUTER:      0.0.0.0      3
                BACKUP DR:              0.0.0.0

  DR PRIORITY:       1  HELLO INTERVAL:   104 RXMT INTERVAL:     5
  DEAD INTERVAL:    405 TX DELAY:          1  POLL INTERVAL:     0 6
  DEMAND CIRCUIT:  OFF  HELLO SUPPRESS:  OFF  SUPPRESS REQ:    OFF
  MAX PKT SIZE:  32768  TOS 0 COST:        3

  # NEIGHBORS:       17 # ADJACENCIES:     1  # FULL ADJS.:      1
  # MCAST FLOODS:  3258 # MCAST ACKS:    6319 DL UNICAST:      OFF
  MC FORWARDING:   OFF

  NETWORK CAPABILITIES: 10
   POINT-TO-POINT
   EMULATED-BROADCAST
   DEMAND-CIRCUITS
```

Figure 185. Display command to display statistics for the OSPF interface

1 The interface type can be Brdcst (broadcast), P-P (a point-to-point), Multi (non-broadcast), VLink (an OSPF virtual link), or P-2-MP (point-to-multipoint).

2 State values can be the following: 1 (down), 2 (backup), 4 (looped back), 8 (waiting), 16 (point-to-point), 32 (DR other), 64 (backup DR), or 128 (designated router).

3 This specifies the designated router IP address.

4 Shows the current hello interval value used.

5 Displays the current dead interval value.

6 Displays the current poll interval value.

7 Specifies the number of routers whose hellos have been received, plus those that have been configured.

8 Number of link state updates that were flooded out the interface (not counting retransmissions).

9 Number of link state acknowledgments that were flooded out the interface (not counting retransmissions).

10 Displays the capabilities of the interface

To display all of the routes in the OMPROUTE routing table, you can issue `Display TCPIP,<tcpipjobname>,OMPROUTE,RTTABLE` as shown in Figure 186:

```
D TCPIP,TCPIPC,OMP,RTTABLE
EZZ7847I ROUTING TABLE 653
TYPE   DEST NET         MASK      COST    AGE     NEXT HOP(S)

SPE2   0.0.0.0                 0  1       65919   172.16.100.254
SPE2   9.0.0.0          FF000000  1       65916   172.16.100.254
 SPF   9.1.150.0        FFFFFE00  1816    65922   172.16.100.254
 SPF   9.3.1.0          FFFFFF00  1816    65920   172.16.100.254
 SPF   9.3.240.0        FFFFFF00  1816    65920   172.16.100.254
SPE2   9.12.0.0         FFFFFF00  1       65919   172.16.100.254
 SPF   9.12.2.0         FFFFFF00  16      65922   172.16.100.254
 SPF   9.12.3.0         FFFFFFF0  1810    65922   172.16.100.254
 SPF   9.12.3.16        FFFFFFF0  1810    65922   172.16.100.254
 SPF   9.12.3.48        FFFFFFF0  1810    65922   172.16.100.254
SPE2   9.12.6.0         FFFFFF00  1       65919   172.16.100.254
SPE2   9.12.8.0         FFFFFF00  1       65919   172.16.100.254
SPE2   9.12.9.0         FFFFFF00  1       65919   172.16.100.254
SPE2   9.12.13.0        FFFFFF00  1       65919   172.16.100.254
 SPF   9.12.14.0        FFFFFF00  16      65922   172.16.100.254
SPE2   9.12.15.0        FFFFFF00  1       65919   172.16.100.254
 SPF   9.24.104.0       FFFFFF00  9       65967   172.16.100.254
 SPF   9.24.104.1       FFFFFFFF  9       65967   172.16.100.254
 SPF   9.24.104.18      FFFFFFFF  3       65967   172.16.100.254
 SPF   9.24.105.0       FFFFFF00  19      65967   172.16.100.254
 SPF   9.24.106.0       FFFFFF00  9       65967   172.16.100.254
SPE2   9.32.41.40       FFFFFFFC  1       65919   172.16.100.254
SBNT   172.16.0.0       FFFF0000  1       65976   NONE
 DIR*  172.16.100.0     FFFFFF00  1       65978   172.16.100.3
 SPF   172.16.100.3     FFFFFFFF  0       65977   M032216B
 SPF*  172.16.100.254   FFFFFFFF  3       65967   172.16.100.254
SPE2   172.16.101.0     FFFFFF00  1       65919   172.16.100.254   (2)
 SPF   172.16.101.28    FFFFFFFF  8       65967   172.16.100.254   (2)
 SPF   172.16.101.254   FFFFFFFF  3       65967   172.16.100.254
SPE2   172.16.102.0     FFFFFF00  1       65919   172.16.100.254   (2)
 SPF   172.16.102.39    FFFFFFFF  8       65967   172.16.100.254   (2)
 SPF   172.16.102.254   FFFFFFFF  3       65967   172.16.100.254
 SPF   172.16.232.0     FFFFFF00  10      65967   172.16.100.254
 SPF   172.16.232.39    FFFFFFFF  10      65967   172.16.100.254
 DIR*  172.16.233.0     FFFFFF00  1       66036   172.16.233.3     (3)
 SPF   172.16.233.3     FFFFFFFF  0       66036   EZASAMEMVS
STAT*  172.16.233.28    FFFFFFFF  0       66037   172.16.233.3
STAT*  172.16.233.39    FFFFFFFF  0       66037   172.16.233.3
 SPF*  172.16.251.0     FFFFFF00  16      65967   172.16.100.254   (2)
 DIR*  172.16.251.3     FFFFFFFF  1       66036   VIPLAC10FB03
 SPF   172.16.251.28    FFFFFFFF  16      65967   172.16.100.254   (2)
 SPF   172.16.251.39    FFFFFFFF  16      65967   172.16.100.254   (2)
 SPF   172.16.252.0     FFFFFF00  10      65967   172.16.100.254
 SPF   172.16.252.28    FFFFFFFF  10      65967   172.16.100.254
SBNT   192.168.10.0     FFFFFF00  1       65916   NONE
 SPF   192.168.10.1     FFFFFFFF  10      65922   172.16.100.254
SPE2   192.168.21.0     FFFFFF00  1       65916   172.16.100.254
 SPF   192.168.21.1     FFFFFFFF  1811    65922   172.16.100.254
 SPF   192.168.21.2     FFFFFFFF  1810    65922   172.16.100.254
 SPF   192.168.21.12    FFFFFFFF  1810    65922   172.16.100.254
SPE2   192.168.31.0     FFFFFF00  1       65916   172.16.100.254
 SPF   192.168.31.1     FFFFFFFF  1811    65920   172.16.100.254
 SPF   192.168.31.2     FFFFFFFF  1810    65922   172.16.100.254
 SPF   192.168.31.12    FFFFFFFF  1810    65920   172.16.100.254
SPE2   192.168.41.0     FFFFFF00  1       65916   172.16.100.254
 SPF   192.168.41.1     FFFFFFFF  10      65922   172.16.100.254
 SPF   192.168.41.12    FFFFFFFF  9       65967   172.16.100.254
 SPF   192.168.64.0     FFFFFF00  69      65967   172.16.100.254
STAT*  192.168.64.2     FFFFFFFF  0       66037   172.16.233.3
 SPF   192.168.192.0    FFFFFF00  69      65967   172.16.100.254

DEFAULT GATEWAY IN USE.

TYPE COST     AGE       NEXT HOP
SPE2 1        65919     172.16.100.254
                       0 NETS DELETED, 0 NETS INACTIVE
```

Figure 186. Display command to list OMPROUTE routes

If you want to see a specific route in the OMPROUTE, you can issue `D TCPIP,<tcpipjobname>,OMPROUTE,RTTABLE,DEST=<destipaddress>` as shown in Figure 187.

```
D TCPIP,TCPIPC,OMP,RTTABLE,DES=9.24.105.0
EZZ7857I SYNTAX ERROR (DES) ON OMPROUTE CONSOLE COMMAND
D TCPIP,TCPIPC,OMP,RTTABLE,DEST=9.24.105.0
EZZ7874I ROUTE EXPANSION 650
DESTINATION:    9.24.105.0
MASK:           255.255.255.0
ROUTE TYPE:     SPF
DISTANCE:       19
AGE:            76351
NEXT HOP(S):    172.16.100.254    (M032216B)
```

Figure 187. Display command to list OMPROUTE route for a specific destination

To stop OMPROUTE, use the `d <procname>` or `F <procname>,KILL`. command.

To reread the configuration file, issue the `F <procname>,RECONFIG` command.

6.5.4 OMPROUTE and automatic VIPA takeover

One of the functions supported by Dynamic VIPA is automatic VIPA takeover. This function is discussed in detail in Chapter 5, “Automatic VIPA takeover and takeback” on page 127. It provides VIPA IP address backup capability in another stack in the sysplex environment in the event the stack fails.

We implemented automatic VIPA takeover in our environment. The DVIPA IP address in the RA03 host is 172.16.251.3. RA03 provides VIPA backup for 172.16.251.39 owned by RA39. Figure 188 shows a portion of the profile.

```
VIPADYNAMIC
  VIPADEFINE MOVE IMMED 255.255.255.0 172.16.251.3
  VIPABACKUP 100 172.16.251.39
  VIPARANGE  DEFINE MOVEABLE NONDISRUPT 255.255.255.0 172.16.240.193
ENDVIPADYNAMIC
```

Figure 188. VIPADYNAMIC block on RA03 host

In case of failure of the TCPIPC stack in RA03, the VIPA IP address 172.16.251.3 will continue to be reachable and will move to the RA39 host (which is configured to provide backup capability).

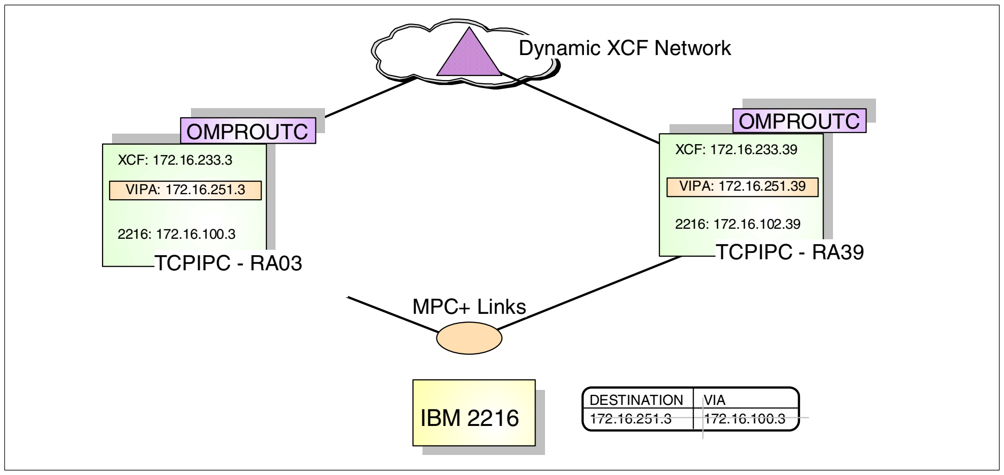


Figure 189. Dynamic VIPA

Figure 189 shows the host route to the DVIPA IP address of RA03 in the 2216 during normal operation.

```
D TCPIP,TCPIPC,N,ROUTE
EZZ2500I NETSTAT CS V2R10 TCPIPC 189
DESTINATION      GATEWAY          FLAGS  REFCNT  INTERFACE
172.16.251.0     172.16.102.254   UG     000000  M392216B
172.16.251.3     172.16.233.3     UGH    000000  EZAXCF03
172.16.251.39    0.0.0.0          UH     000000  VIPLAC10FB1C
```

Figure 190. Display route command on RA39 host

The result of the command `D TCPIP,TPCIPC,NETSTAT,ROUTE` issued to the TCPIPC stack in the RA39 host is illustrated in Figure 190. It shows the host route to DVIPA 172.16.251.3. If the TCPIPC stack on the RA03 host fails, its DVIPA IP address moves to the TCPIPC stack on the RA39 host as shown in Figure 191.

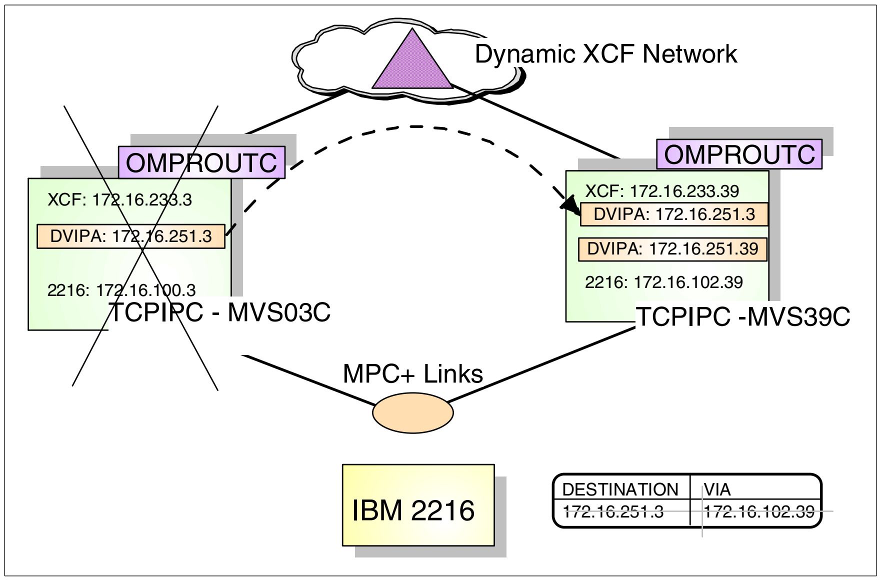


Figure 191. Backup dynamic VIPA

In Figure 191 we can see the host route to DVIPA 172.16.251.3 in the 2216 router after a TCPIPC stack failure on RA03. Figure 192 shows the result of the command `D TCPIP,TPCIPC,NETSTAT,ROUTE` issued to the TCPIPC stack in the RA39 host. It shows the host route to of DVIPA 172.16.251.3 after the TCPIPC stack on RA03 host failure.

```
D TCPIP,TCPIPC,N,ROUTE
EZZ2500I NETSTAT CS V2R10 TCPIPC 168
DESTINATION      GATEWAY          FLAGS  REFCNT  INTERFACE
DEFAULT          172.16.102.254   UG     000000  M392216B
172.16.251.0     172.16.102.254   UG     000000  M392216B
172.16.251.3     0.0.0.0          UH     000000  VIPLAC10FB03
172.16.251.39    0.0.0.0          UH     000000  VIPLAC10FB1C
```

Figure 192. Display route in RA39 host after RA03 host failure

6.5.5 Interface cost considerations

All interfaces have a cost value associated with them in both the OSPF and RIP routing protocols. The value of a route to reach a destination is calculated by the sum of the costs of each link that will be traversed on the way to the destination.

The method for defining cost values for each interface differs between the OSPF and RIP protocols. In the OSPF_Interface statement we use the Cost0 parameter, which can be from 0 to 65535. In the RIP_Interface statement, we use the IN_Metric parameter to define a value from 1 to 15 to be added to every route received over the interface, and the Out_Metric parameter to define a value from 0 to 15 to be added to every route advertised over the interface. Please see *OS/390 IBM Communications Server: IP Configuration Reference,* SC31-8726 for more detailed information.

As you can see, it is quite complicated to set route precedence for both protocols. If you use only one of the routing protocols, route precedence is not a concern, but if you use a multiple protocol or AS Boundary Routers in your network, you should take care with this topic.

The COMPARISON statement is set to make OMPROUTE know if the cost value received from another AS or from another routing protocol can be comparable and useful. You can configure the COMPARISON value as Type1 or Type2.

When COMPARISON=Type1 is set, the cost value received from other routing protocols or exchanged between AS Boundary Routers is to be used because they are comparable. Note that if you use the OSPF and RIP protocols and set COMPARISON=Type1, your cost values in the OSPF interfaces must be low because the OSPF AS must be reached from the RIP AS with a cost value less than 16.

A COMPARISON value configured as Type2 indicates that cost values are non-comparable. In this case there are some rules to set cost values in an RIP AS and/or OSPF AS.

Table 4 shows the order of precedence used by routers that must learn routing information in a network with multiple routing protocols or when there is an OSPF Boundary Router.

Table 4. Route precedence.

Source Comparison	Route A - Type	Route B -Type	Route chosen
Type 1	OSPF Internal	RIP	OSPF Internal
Type 1	OSPF Internal	OSPF Type 1 External	OSPF Internal
Type 1	OSPF Internal	OSPF Type 2 External	OSPF Internal
Type 1	RIP	OSPF Type 1 External	Lowest Cost Route
Type 1	RIP	OSPF Type 2 External	RIP Route
Type 1	OSPF Type 1 External	OSPF Type 2 External	OSPF Type 1 External
Type 2	OSPF Internal	RIP	OSPF Internal
Type 2	OSPF Internal	OSPF Type 1 External	OSPF Internal
Type 2	OSPF Internal	OSPF Type 2 External	OSPF Internal
Type 2	RIP	OSPF Type 1 External	OSPF Type 1 External
Type 2	RIP	OSPF Type 2 External	Lowest Cost Route
Type 2	OSPF Type1 External	OSPF Type 2 External	OSPF Type 1 External

Source Comparison is the value to which COMPARISON is set. *Route A* and *Route B* are two possible routes from one source to the same destinations that can be chosen.

The following terms are used in the table:

- *RIP Route* - a route learned from the RIP protocol.
- *OSPF Internal Route* - a route learned from the OSPF protocol where the entire path lies within the same OSPF AS.

- *OSPF External Route* - a route learned from the OSPF protocol whose path traverses another AS. There are two categories of OSPF external routes: *OSPF external route type 1*, when COMPARISON Type 1 is set and OSPF External routes are imported from another AS, and *OSPF External routes type 2*, when COMPARISON Type 2 is set and OSPF External routes are imported from another AS.

6.5.6 Multipath considerations

The OMPROUTE daemon enables multipath equal-cost routes for each source and destination pair. To enable the multipath function, code the MULTIPATH parameter in the IPCONFIG statement in the TCP/IP profile. Using the multipath function enables outbound connections to be spread over existing routes. Up to four equal-cost routes can be used for multipath in OSPF routes. For RIP, multiple equal-cost routes will be added only to directly connected destinations over redundant interfaces. The traffic spread over these routes depends on the parameter coded; it can be done on a connection basis or packet-basis.

By specifying the PERCONNECTION parameter, TCP/IP will select a route on a round-robin basis from a multipath routing list to the destination and will use this route for all IP packets that use this connection. This is independent of whether the data is connection or connectionless oriented. All other associations will be spread over the multiple equal-cost routes.

If the PERPACKET parameter is specified and there are IP packets to be delivered over a multipath route, a route will be selected on a round-robin basis to send this IP packet to its destination, and connection or connectionless-oriented IP packets using the same source and destination pair will not always use the same route.

You should pay attention if you choose the PERPACKET option, because it may consume additional CPU cycles on the final destination to reassemble packets if they arrive out of order. Fragmented IP datagrams are restricted from using the PERPACKET option.

The subparameter FWDMULTIPATH PERPACKET in the DATAGRAMFWD parameter is used for transferring data between networks if there are multipath equal-cost routes. It is coded in the IPCONFIG statement and indicates that the connection or connectionless-oriented IP packets using the same destination address do not always use the same route.

When there is more than one route to the same subnet you can select one as a primary and another as backup for the OSPF protocol traffic by coding Parallel_OSPF values in the OSPF_Interface statement.

6.5.7 VTAM and I/O definitions

We set up the 2216 MPC+ connection for our host stacks using a unique control unit address (CUADD) for each LPAR. In the latest release of MAS, this is not necessary and the same address can be used on each LPAR.

Figure 192 contains the I/O definitions we used to define the ESCON connections for our 2216:

```
                            --- SWITCH ----                                    UNIT ADDR            UNIT
                             ID  PR  CU  DYN  --- CONTROL UNIT ---  CU-           RANGE    -- DEVICE --
ADDR   DEVICE
  CHPID  TYPE    SIDE  MODE       PN  PN  ID   NUMBER  TYPE-MODEL    ADD  PROTOCOL  FROM  TO   NUMBER,RANGE
START  TYPE-MODEL
   A8    CNC           SHR   E1  D0  C2  E1    0380   3172          1             00   1F     0380,32
00    3172
   A8    CNC           SHR   E1  D0  C2  E1    03A0   3172          2             00   1F     03A0,32
00    3172
   A8    CNC           SHR   E1  D0  C2  E1    03C0   3172          4             00   1F     03C0,32
00    3172
```

Figure 193. I/O definitions for 2216 ESCON channel

Since this is an MPC+ connection we needed to configure it using VTAM definitions. MPC+ is one of the three DLCs that the SNA and IP portions of CS for OS/390 share, and it is the SNA component that owns the connection manager.

Configuring VTAM for MPC+ requires an entry in the VTAM transport resource list (TRL). A TRL entry (TRLE) corresponds to an MPC+ group. Figure 194 contains the VTAM TRL definition on RA03, which was to be used by the TCPIPC stack:

```
         VBUILD TYPE=TRL
M032216B TRLE   LNCTL=MPC,                                  1
               MAXBFRU=9,
               READ=381,                                    2
               WRITE=380,                                   2
               MPCLEVEL=HPDT,
               REPLYTO=3.0
```

Figure 194. VTAM TRL major node for 2216 MPC+ on T03ATCP

Notes to Figure 194:

1 The TRLE name (M032216B) in the TRL definition must match the name on the corresponding DEVICE statement in the TCP/IP profile.

2 MPC+ allows multiple subchannels in each direction for maximum availability; you need to define at least one read and one write subchannel for each connection. Note that the read subchannel defined here corresponds to the write subchannel on the 2216, and vice versa.

The TRLE definitions for the rest of the MPC links can be modeled on Figure 194. The only differences are the TRLE name and the READ and WRITE addresses.

You need to activate the associated TRL major node prior to activating the MPC+ device. Then you can display the status of the TRLE as follows:

```
D NET,TRL,TRLE=M032216B
IST097I DISPLAY ACCEPTED
IST075I NAME = M032216B, TYPE = TRLE
IST486I STATUS= ACTIV, DESIRED STATE= ACTIV
IST087I TYPE = LEASED             , CONTROL = MPC , HPDT = YES
IST1715I MPCLEVEL = HPDT       MPCUSAGE = SHARE
IST1577I HEADER SIZE = 4096 DATA SIZE = 32 STORAGE = ***NA***
IST1221I WRITE DEV = 0380 STATUS = ACTIVE     STATE = ONLINE
IST1577I HEADER SIZE = 4092 DATA SIZE = 32 STORAGE = DATASPACE
IST1221I READ  DEV = 0381 STATUS = ACTIVE     STATE = ONLINE
IST314I END
```

Figure 195. Displaying VTAM TRL definition for 2216 MPC+ on T03ATCP

Note that the status of the TRLE will not be ACTIVE until either VTAM or TCP/IP activates the underlying connection (TCP/IP starts the device or VTAM activates the link station).

6.5.8 2216 configuration

The 2216 is an APPN and IP router that provides mainframe access using either ESCON or a parallel channel adapter. It supports a wide range of LAN, WAN and ATM adapters.

Multiprotocol Access Services (MAS) is the operational code for the 2216. MAS provides a comprehensive set of multiprotocol routing protocols, transport code, and features.

6.5.8.1 Hardware configuration

In our network we used a 2216 Model 400 that was equipped with one two-port token-ring adapter in slot 1 and one ESCON adapter in slot 8.

First we connected an ASCII terminal to the 2216's service port. We used the console interface to configure the hardware adapter. Alternatively, we could have configured the 2216 by connecting via Telnet, or by using a GUI on a PC and uploading the resulting file.

When configuring the 2216, most commands can be abbreviated to one or two characters.

It is also important to notice that the prompt may change after you have entered a command. Not all commands are valid under every prompt and the syntax/parameters may change for the same command under different prompts.

```
MOS Operator Console

*talk 6                                                                 1

Config>add device token-ring
Device Slot #(1-8) [1]?                                                 2
Device Port #(1-2) [1]?
Adding Token Ring device in slot 1  port 1 as interface #0
Use "net 0" to configure Token Ring parameters
Config>add device escon
Device Slot #(1-8) [1]? 8
Adding ESCON Channel device in slot 8  port 1 as interface #1
Use "net 1" to configure ESCON Channel parameters
Config>network 0                                                        3
Token-Ring interface configuration
TKR config>speed 16
TKR config>packet-size 4399
TKR config>exit
Config>network 1
ESCON Config>add mpc                                                    4
ESCON Add Virtual>sub addr
ESCON Add MPC+ Read Subchannel>link d0
ESCON Add MPC+ Read Subchannel>lpar 1
ESCON Add MPC+ Read Subchannel>cu 1
ESCON Add MPC+ Read Subchannel>dev 0
ESCON Add MPC+ Read Subchannel>ex
ESCON Add Virtual>sub addw
ESCON Add MPC+ Write Subchannel>dev 1
ESCON Add MPC+ Write Subchannel>ex
ESCON Add Virtual>exit
ESCON Config>add mpc
ESCON Add Virtual>sub addr
ESCON Add MPC+ Read Subchannel>link d0
ESCON Add MPC+ Read Subchannel>lpar 2
ESCON Add MPC+ Read Subchannel>cu 2
ESCON Add MPC+ Read Subchannel>dev 0
ESCON Add MPC+ Read Subchannel>ex
ESCON Add Virtual>sub addw
ESCON Add MPC+ Write Subchannel>dev 1
ESCON Add MPC+ Write Subchannel>ex
ESCON Add Virtual>ex
ESCON Config>add mpc
ESCON Add Virtual>sub addr
ESCON Add MPC+ Read Subchannel>link d0
ESCON Add MPC+ Read Subchannel>lpar 4
ESCON Add MPC+ Read Subchannel>cu 4
ESCON Add MPC+ Read Subchannel>dev 0
ESCON Add MPC+ Read Subchannel>ex
ESCON Add Virtual>sub addw
ESCON Add MPC+ Write Subchannel>dev 1
ESCON Add MPC+ Write Subchannel>ex
ESCON Add Virtual>ex
```

Figure 196. 2216 device configuration console log, part 1

```
ESCON Config>modify 2                                                        5
ESCON Config Virtual>max 32768
ESCON Config Virtual>ex
ESCON Config>mod 3
ESCON Config Virtual>max 32768
ESCON Config Virtual>ex
ESCON Config>mod 4
ESCON Config Virtual>max 32768
ESCON Config Virtual>ex
ESCON Config>list all                                                        6
Net:  2   Protocol: MPC+   LAN type: MPC+           LAN number:  0
          Maxdata: 32768
          Reply TO: 45000   Sequencing Interval Timer: 3000
          Outbound protocol data blocking is enabled
          Block Timer:    5 ms   ACK length:   10 bytes
          Read Subchannels:
          Sub  0   Dev addr:  0  LPAR: 1  Link addr: D0  CU addr: 1
          Write Subchannels:
          Sub  1   Dev addr:  1  LPAR: 1  Link addr: D0  CU addr: 1

Net:  3   Protocol: MPC+   LAN type: MPC+           LAN number:  1
          Maxdata: 32768
          Reply TO: 45000   Sequencing Interval Timer: 3000
          Outbound protocol data blocking is enabled
          Block Timer:    5 ms   ACK length:   10 bytes
          Read Subchannels:
          Sub  0   Dev addr:  0  LPAR: 2  Link addr: D0  CU addr: 2
          Write Subchannels:
          Sub  1   Dev addr:  1  LPAR: 2  Link addr: D0  CU addr: 2

Net:  4   Protocol: MPC+   LAN type: MPC+           LAN number:  2
          Maxdata: 32768
          Reply TO: 45000   Sequencing Interval Timer: 3000
          Outbound protocol data blocking is enabled
          Block Timer:    5 ms   ACK length:   10 bytes
          Read Subchannels:
          Sub  0   Dev addr:  0  LPAR: 4  Link addr: D0  CU addr: 4
          Write Subchannels:
          Sub  1   Dev addr:  1  LPAR: 4  Link addr: D0  CU addr: 4

ESCON Config>ex
ESCON configuration has been changed.
Do you wish to keep the changes? [Yes]: y
Config>
Config>list devices                                                          7
Ifc 0     Token Ring                        Slot: 1   Port: 1
Ifc 1     ESCON Channel                     Slot: 8   Port: 1
Ifc 2     MPC - ESCON Channel               Base Net: 1
Ifc 3     MPC - ESCON Channel               Base Net: 1
Ifc 4     MPC - ESCON Channel               Base Net: 1
```

Figure 197. 2216 device configuration console log, part 2

In the figures:

1 The `talk 6` command enables configuration mode.

2 The values within [ ] are the defaults. You can confirm the default by pressing Enter, or change it.

3 The `network` command gives you the opportunity to change the parameters for the specified interface.

4 The `add mpc` command adds and configures MPC interfaces for the ESCON adapter.

5 The `modify` command changes parameters on the MPC interface. After the MPC interfaces were defined, the 2216 assigned the three host connections the numbers 2, 3 and 4, so `modify 2` refers to the first one defined (LPAR 1, or RA03).

6 The `list` command used under the ESCON configuration prompt displays the ESCON configuration being defined.

7 The `list devices` command used under the base configuration prompt displays the defined devices. Note the confusing use of the terms *port*, *interface* and *network*. Here interfaces 2, 3 and 4 are actually connections within the physical ESCON port called interface 1.

6.5.8.2 2216 IP configuration

After the device configuration we configured the IP protocol via the 2216 console, as shown in Figure 198:

```
Config>protocol ip
Internet protocol user configuration
IP config>add address 0
1
New address []? 172.16.220.254
Address mask [255.255.0.0]? 255.255.255.0
IP config>ad ad 2
New address []? 172.16.100.254
Address mask [255.255.0.0]? 255.255.255.0
IP config>ad ad 3
New address []? 172.16.101.254
Address mask [255.255.0.0]? 255.255.255.0
IP config>ad ad 4
New address []? 172.16.102.254
Address mask [255.255.0.0]? 255.255.255.0
IP config>set internal 172.16.220.254
2
IP config>list addresses
IP addresses for each interface:
   intf    0   172.16.220.254   255.255.255.0    Local wire broadcast, fill 1
   intf    1                                    IP disabled on this interface
   intf    2   172.16.100.254   255.255.255.0    Local wire broadcast, fill 1
   intf    3   172.16.101.254   255.255.255.0    Local wire broadcast, fill 1
   intf    4   172.16.102.254   255.255.255.0    Local wire broadcast, fill 1
Internal IP address: 172.16.220.254
IP config>ex
```

Figure 198. 2216 IP configuration console log

1 The `add address` command adds an IP address to the specified interface (0, the token-ring in this case).

2 The `set internal-ip-address` command adds the internal IP address for the router. This address acts rather like a VIPA address in that it can be reached independently of the active real interfaces. In fact, it is usual (as with a VIPA address) to have the internal address on a unique subnetwork.

6.5.8.3 2216 OSPF configuration

Next, we configured OSPF on the router as shown in Figure 199:

```
Config>protocol ospf                                                           1
Open SPF-Based Routing Protocol configuration console
OSPF Config>enable ospf
Estimated # external routes [100]?
Estimated # OSPF routers [50]?
Maximum Size LSA [2048]?
OSPF Config>set interface 172.16.220.254                                       2
Attaches to area [0.0.0.0]?
Retransmission Interval (in seconds) [5]?
Transmission Delay (in seconds) [1]?
Router Priority [1]?
Hello Interval (in seconds) [10]?
Dead Router Interval (in seconds) [40]?
TOS 0 cost [1]?
Demand Circuit (Yes or No)? [No]:
Authentication Type (0 - None, 1 - Simple) [0]?
OSPF Config>se in 172.16.100.254
*** 9 rows deleted ***
OSPF Config>se in 172.16.101.254
*** 9 rows deleted ***
OSPF Config>se in 172.16.102.254
*** 9 rows deleted ***
OSPF Config>list all

                        --Global configuration--
                OSPF Protocol:          Enabled
                # AS ext. routes:       100
                Estimated # routers:    50
                Maximum LSA size:  :    2048
                External comparison:    Type 2
                RFC 1583 compatibility: Enabled
                AS boundary capability: Disabled
                Multicast forwarding:   Disabled
                Demand Circuits:        Enabled
                Least Cost Area Ranges: Disabled
                Maximum Random LSA Age: 0

                --Area configuration--
Area ID           Stub? Default-cost Import-summaries?
0.0.0.0(Implicit)  No         N/A           N/A

                        --Interface configuration--
IP address        Area            Auth   Cost  Rtrns  Delay  Pri  Hello   Dead
172.16.220.254    0.0.0.0           0       1      5      1    1     10     40
172.16.100.254    0.0.0.0           0       1      5      1    1     10     40
172.16.101.254    0.0.0.0           0       1      5      1    1     10     40
172.16.102.254    0.0.0.0           0       1      5      1    1     10     40
OSPF Config>ex
Config>write                                                                   3
Config Save: Using bank A and config number 1
Config>
4
*reload
Are you sure you want to reload the gateway? (Yes or [No]): y
```

Figure 199. 2216 OSPF configuration console log

1 The `protocol` command enters configuration mode for the selected protocol.

2 The `set interface` command sets the OSPF parameters for the desired interface.

After defining the OSPF parameters for all the interface, we exited the OSPF configuration prompt and used the `write` command **3** to save the configuration data in the 2216's bank.

4 Ctrl-P exits configuration mode.

Chapter 7. Sysplex Distributor

Sysplex Distributor is a new function for 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. IBM Communications Server for OS/390 V2R10 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 OS/390 is lost.

This chapter includes:

7.1 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 OS/390 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 an OS/390 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 OS/390 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 OS/390 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 OS/390 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 forward 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 OS/390 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 an OS/390 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 OS/390 image after an OS/390 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.

Non-disruptive, 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 non-disruptive 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.

7.2 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 OS/390 IP’s Service 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 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 total CS for OS/390 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 non-disruptive 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

   Non-disruptive application server instance movement

In summary, Sysplex Distributor provides:

1. 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.
2. It raises the limit of 64 DVIPAs on a stack to 256.
3. Backup capability is enhanced. In case of failure of the distributing IP stack, the connections distributed to other IP stacks in the sysplex will not be disrupted.
4. Dynamic VIPA takeback without any disruption in the connections already established.
5. New commands are added to display both the connection routing table and destination port table information, showing information of configuration and current connection distribution.

The specific profile statements that should be used to configure Sysplex Distributor will be detailed later in 7.5, “Sysplex Distributor implementation” on page 207.

7.2.1 Sysplex Distributor functionality

Let us consider the scenario depicted in Figure 200 on page 191. 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 200 on page 191) 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.

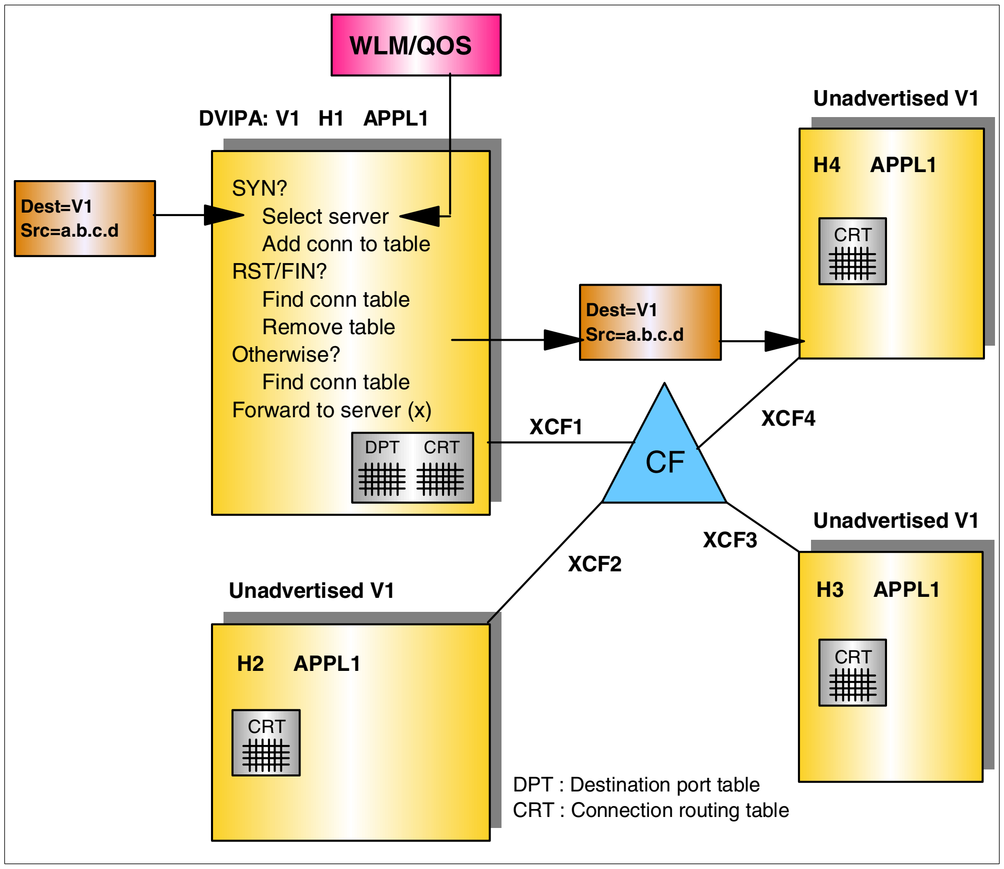


Figure 200. Sysplex Distributor functionality

7.2.2 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 has been enhanced to support Sysplex Distributor. Consider the scenario described in Figure 201.

H1 is the distributing IP stack and H2 is the primary VIPABACKUP IP stack. When H1 fails (Figure 201):

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

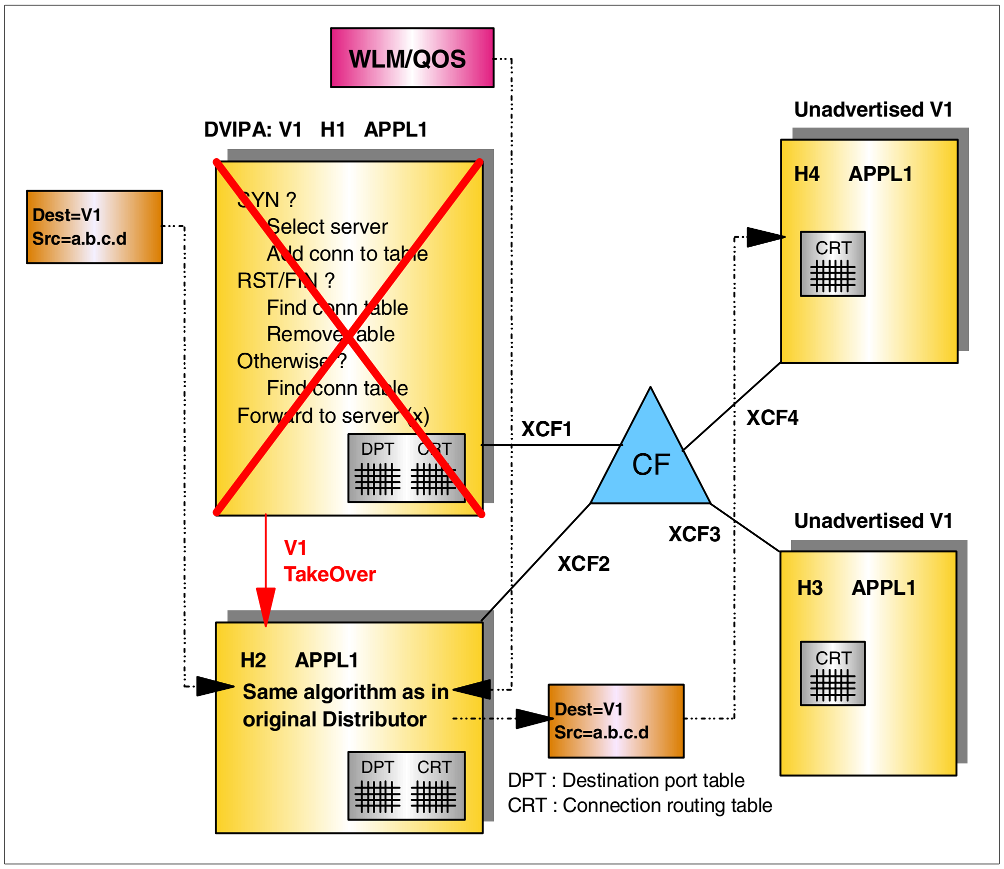


Figure 201. Sysplex Distributor and VIPA takeover

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.

7.2.3 Recovery

Once the H1 IP stack is activated again the process of taking back V1 to H1 is started. This process is non-disruptive 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 202).

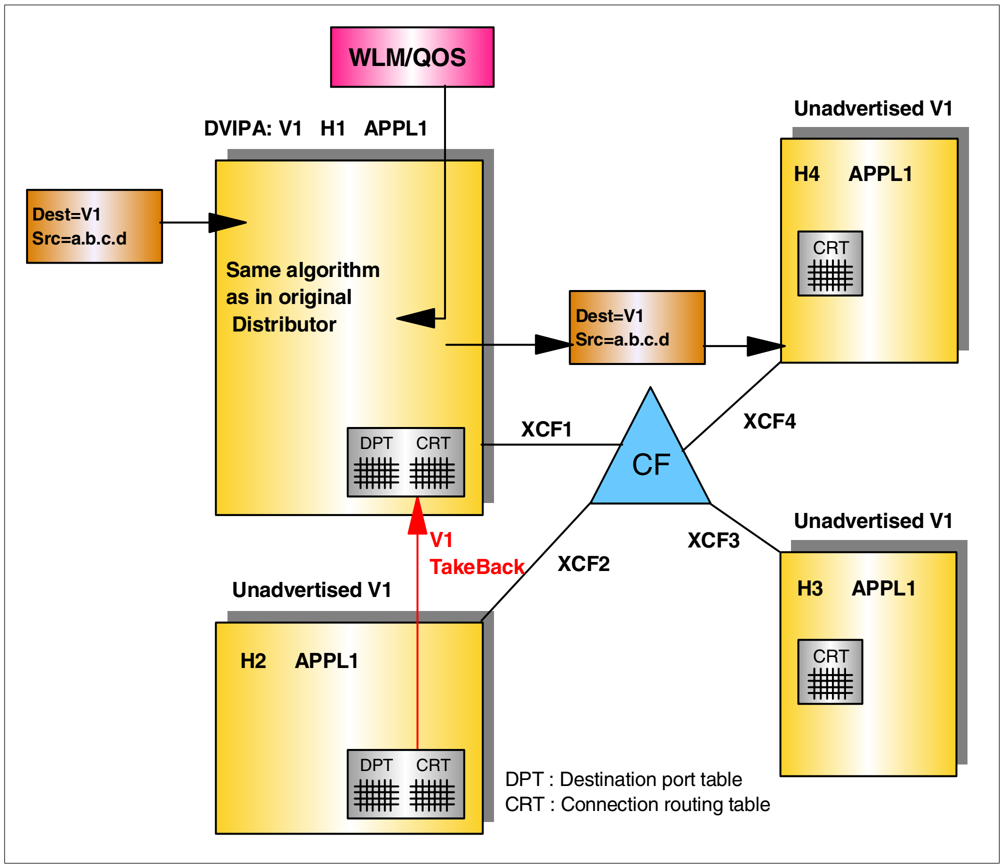


Figure 202. Sysplex Distributor and VIPA takeback

7.3 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 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 203:

- 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 203. 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.

7.4 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 IBM Communications Server for OS/390 V2R10 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.

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 204 on page 196).
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 exceeds 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 204 on page 196).

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 OS/390 provides all policy information in the network.

The sysplex consists of three OS/390 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 204 illustrates the system environment at ITSO Raleigh.

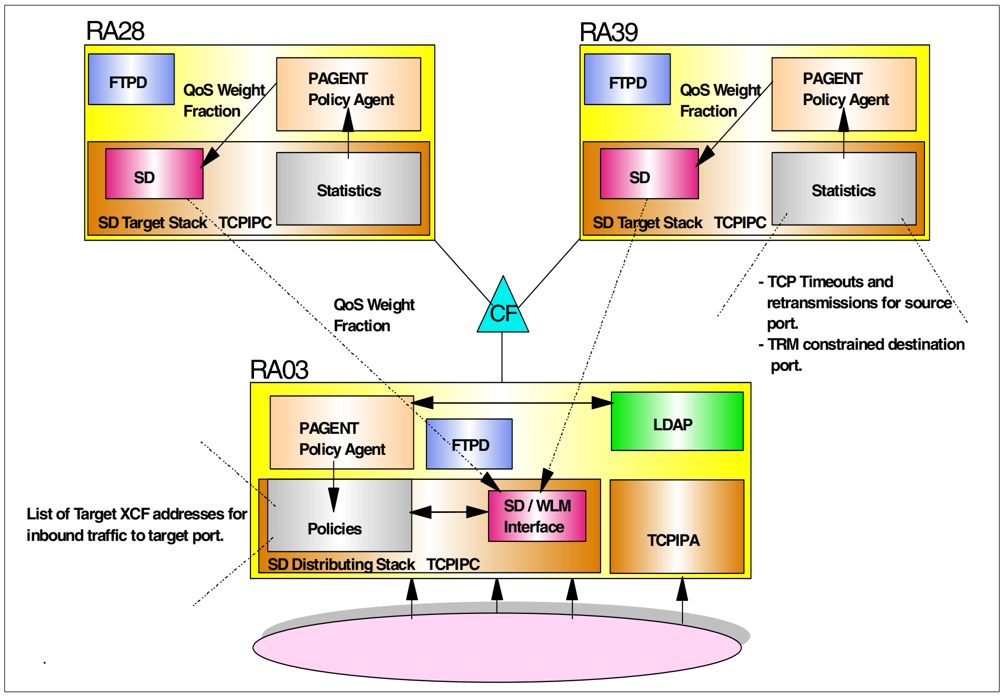


Figure 204. Sysplex Distributor policy implementation at ITSO Raleigh

7.4.1 Sysplex Distributor QoS policy in the PAGENT file

Please review the definition of the TCPIP.PROFILE for the stacks RA03, RA28 and RA39 in 7.5, “Sysplex Distributor implementation” on page 207. 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 *IBM Communications Server for OS/390 IP Configuration Guide*, SC31-8725 and *IBM Communications Server for OS/390 IP Configuration Reference*, SC31-8726. More detailed information is also found in the redbook *IBM Communications Server for OS/390 V2R10 TCP/IP Implementation Guide Volume 2: UNIX Applications*, SG24-5228.

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 7.5, “Sysplex Distributor implementation” on page 207 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 3control 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.

7.4.2 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 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

Note: 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=0K,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 which contains:

```
 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 205. 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 shall 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
```

7.4.3 Monitoring the Sysplex Distributor QoS

You can use the `NETSTAT` command to display Sysplex Distributor information as shown in Figure 206.

```
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 206. 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: Work Load 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 require a superuser authority. Figure 207 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 207. 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.

7.4.4 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:11111111111111111111111111111111
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 OS/390 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` OS/390 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.

You will find further information about the SD policy and SD performance monitor policy in conjunction with an LDAP server in *OS/390 IBM Communications Server: IP Configuration Guide*, SC31-8725 and *OS/390 IBM Communications Server: IP Configuration Reference*, SC31-8726. More detailed information is also found in *IBM Communications Server for OS/390 V2R10 TCP/IP Implementation Guide Volume 2: UNIX Applications*, SG24-5228.

7.5 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.

7.5.1 Requirements

The IPCONFIG DATAGRAMFWD and DYNAMICXCF statement 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 OS/390 V2R10 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 OS/390 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).

7.5.2 Incompatibilities

As specified in the *OS/390 IBM Communications Server: IP Migration,* SC31-8512, IBM recommends that all the IP stacks that may be participants in a Sysplex Distributor environment be at V2R10 because of the following restrictions:

- Workload distribution through Sysplex Distributor is available only when both the distributing and target stacks are at V2R10.
- IP stacks at V2R7 and previous releases do not support automatic VIPA backup.

- IP stacks at V2R8 may back up a distributing Dynamic VIPA but would be unable to distribute workload if the distributing stack were brought down. Moreover, VIPA takeback is disruptive in IP V2R8.

If any IP stack (different from the distributing one) is on the path from the client to the distributing IP stack and if this intermediate IP stack is a target stack within the sysplex cluster, new connection requests passing through this intermediate IP stack might be routed to the local application and the service is not subject to distribution.

7.5.3 Limitations

FTP passive mode should not be used with Sysplex Distributor, as documented in *OS/390 IBM Communications Server: IP User's Guide,* GC31-8514. 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.

Figure 208 on page 209 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.

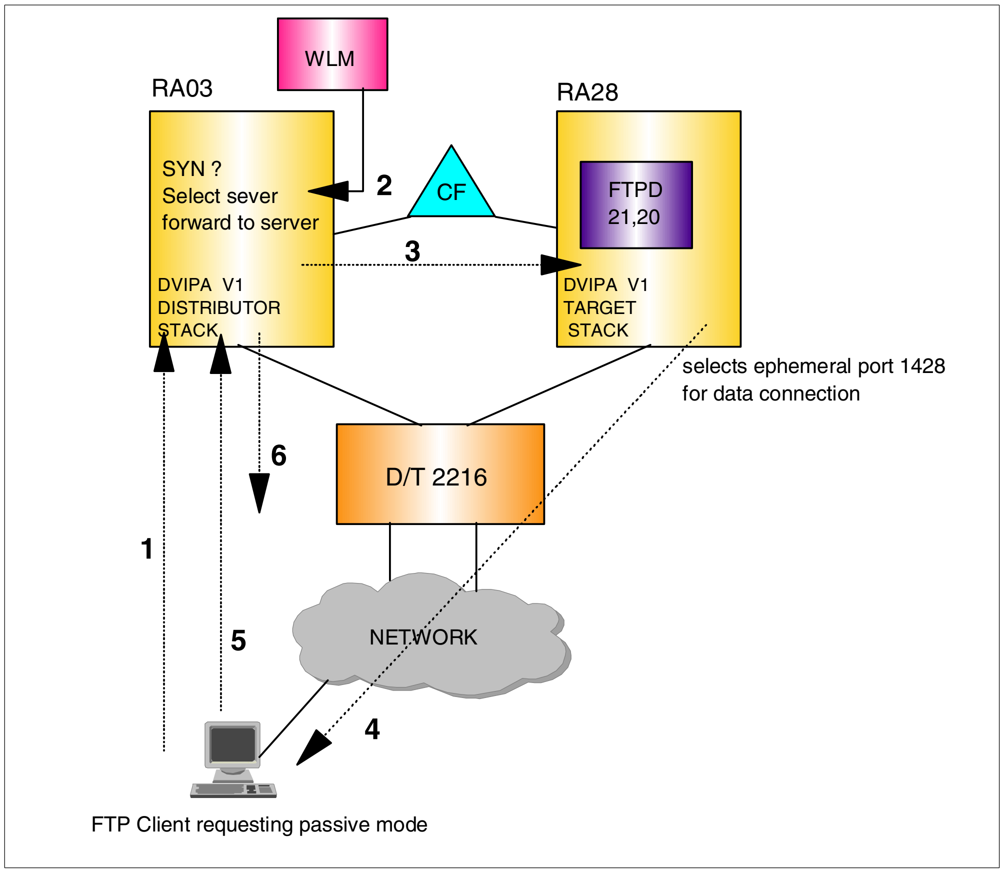


Figure 208. FTP passive mode limitations for Sysplex Distributor

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 209 on page 210:

- 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.

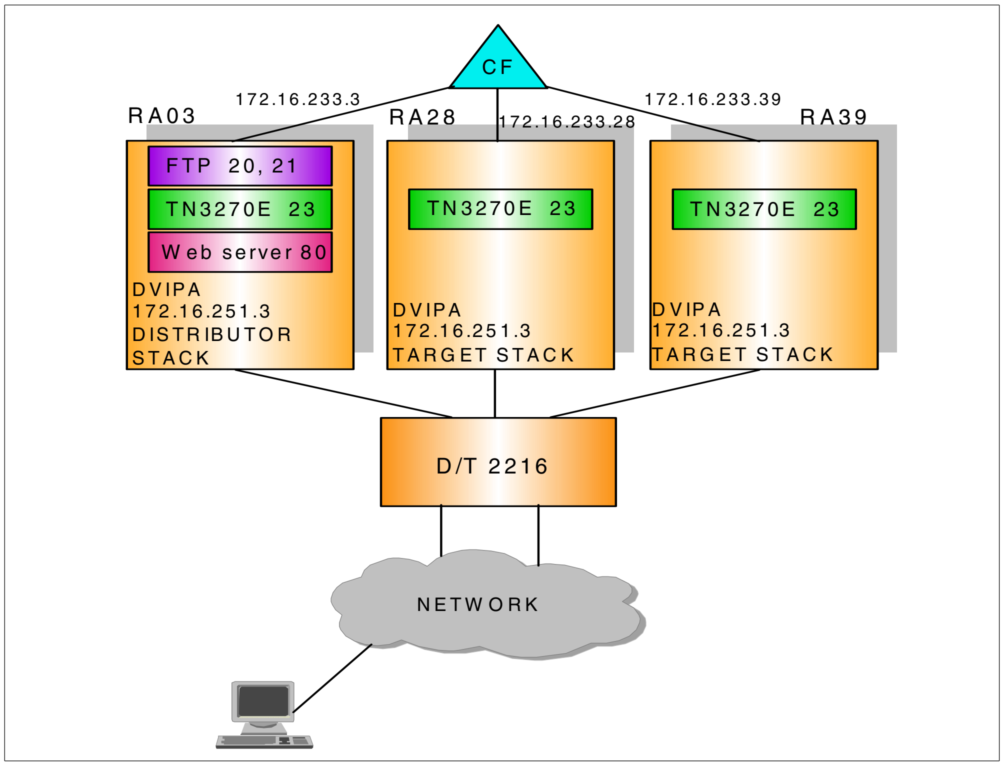


Figure 209. Sysplex Distributor limitation

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 hostnames/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.

7.5.4 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 stack 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 have to be defined through IPCONFIG DYNAMICXCF.
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 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.

The way that Sysplex Distributor distributes the workload can be modified using the Policy Agent. Please refer to 7.4, "Sysplex Distributor and policy" on page 194. For a detailed description about Policy-based network management refer to *IBM Communications Server for OS/390 V2R10 TCP/IP Implementation Guide Volume 2: UNIX Applications*, SG24-5228.

7.6 Monitoring Sysplex Distributor

Three new parameters have been added to the `NETSTAT` command (or the `onetstat` command) and output for three already existing parameters have been modified in order to monitor the SYSPLEX DISTRIBUTOR activity. Table 5 briefly describes those new and changed parameters:

Table 5. New and modified NETSTAT/onetstat parameter to monitor Sysplex Distributor

Type of Command	Command	Parameter	Description
NEW	NETSTAT	VIPADCFG	Display Dynamic VIPA configuration data.
NEW	onetstat	-F	Display Dynamic VIPA configuration data.
NEW	NETSTAT	VDPT	Display DVIPA port distribution table.

Type of Command	Command	Parameter	Description
NEW	onetstat	-O	Display DVIPA port distribution table.
NEW	NETSTAT	VCRT	Display DVIPA connection routing table.
NEW	onetstat	-V	Display DVIPA connection routing table
CHANGED	NETSTAT	CONFIG	Dynamic VIPA information removed.
CHANGED	onetstat	-f	Dynamic VIPA information removed.
CHANGED	NETSTAT	VIPADYN	Expanded OUTPUT
CHANGED	onetstat	-v	Expanded OUTPUT
CHANGED	SYSPLEX	VIPADyn	Expanded OUTPUT

Refer to *OS/390 IBM Communications Server: IP User's Guide,* GC31-8514 for more information on those commands.

7.7 Implementation examples

In this section, we cover a number of scenarios to illustrate the way in which Sysplex Distributor is implemented. We make use of the VIPA configuration statements shown in Figure 210. 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 *OS/390 IBM Communications Server: IP Configuration Reference,* SC31-8726.

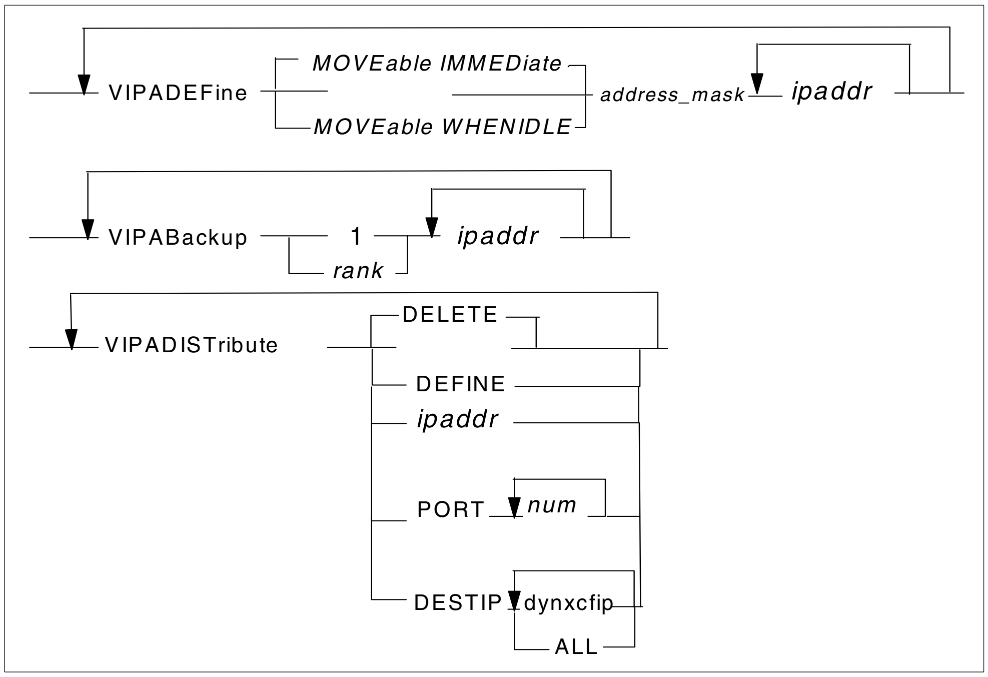


Figure 210. VIPADEFINE, VIPABACKUP, and VIPADISTRIBUTE syntax

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.

7.7.1 Scenario 1: Three IP stacks distributing FTP services

Let us consider the scenario depicted in Figure 211. We have three LPARs, each running IBM Communications Server for OS/390 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 7.5.4, “Implementation” on page 210 to set up our Sysplex Distributor configuration properly.

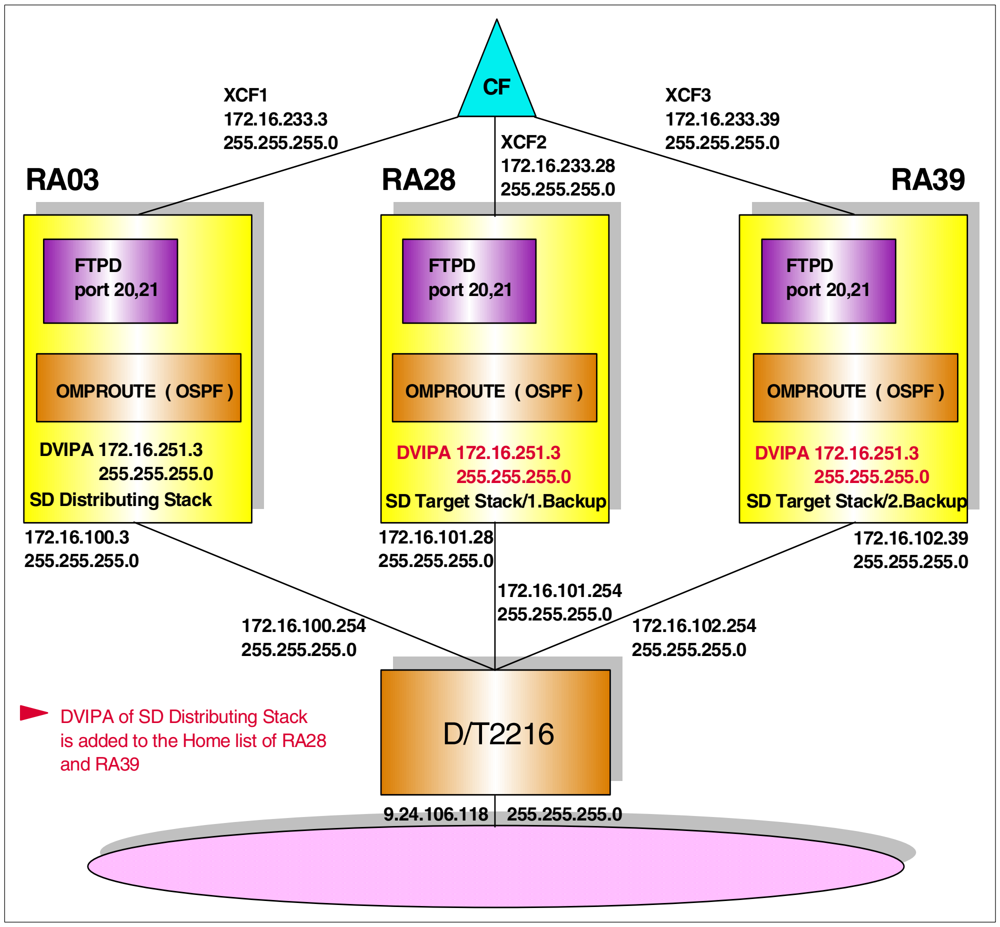


Figure 211. Scenario 1: Three IP stack in the Sysplex Distributor for FTP services

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 212, Figure 213, and Figure 214.

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), DYNAMIXCF (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 212. 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 213. 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 214. 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 215, Figure 216, and Figure 217 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 215. 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 216. 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 217. OMPROUTE configuration file for RA39

We now describe how this scenario works once all the IP stacks are active. We use the commands described in 7.6, “Monitoring Sysplex Distributor” on page 211 to illustrate the state of the system during operation.

Figure 218 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 218. NETSTAT VIPADCFG for RA03, RA28, and RA39 IP stacks

Figure 219 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 219. NETSTAT HOME command for RA03, RA28, and RA39 IP stacks

Figure 220 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 220. NETSTAT VDPT command for RA03, RA28, and RA39 IP stacks

Figure 221 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 221. NETSTAT VDPT,DET for RA03, RA28 and RA39 IP stacks

Figure 222 and Figure 223 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 222. SYSPLEX VIPADYN for all the stacks participating in the sysplex

Figure 223 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 223. NETSTAT VIPADYN for RA03, RA28 and RA39 IP stacks

Figure 224 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 224 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 224. NETSTAT VCRT for RA03, RA28 and RA39 IP stacks

Figure 225 on page 226 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 one 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 226 and Figure 227 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 225. 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 226. 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 227. NETSTAT VCRT from RA39 IP stack when there are several concurrent FTP sessions

Figure 228 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 228. NETSTAT VDPT DET from RA03 IP stack with multiple concurrent FTP sessions

Figure 229 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 229. NETSTAT VDPT from RA03 IP stack after 100 FTP sessions

Figure 230 shows how these connections are distributed taking the WLM weights into account (as was done in Figure 228). 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.

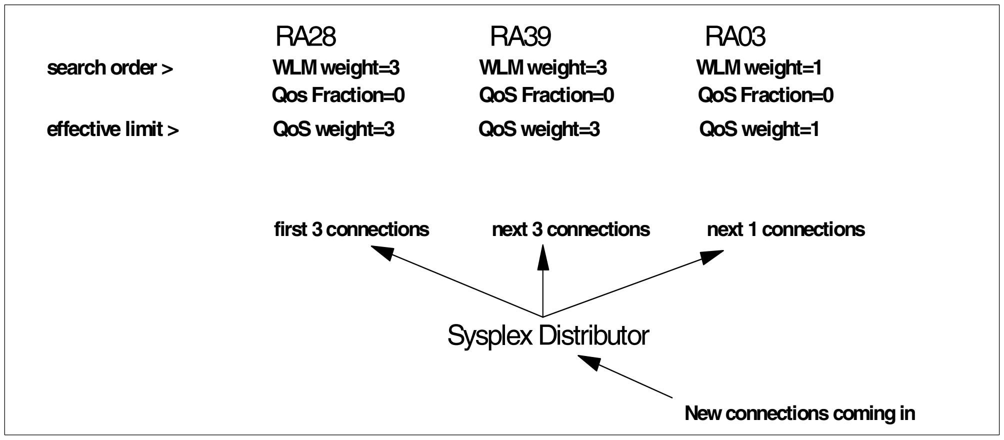


Figure 230. Using WLM/QoS data to select a target stack

7.7.2 Scenario 2: VIPA takeover and takeback with Sysplex Distributor

This second example, as shown in Figure 231, 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.

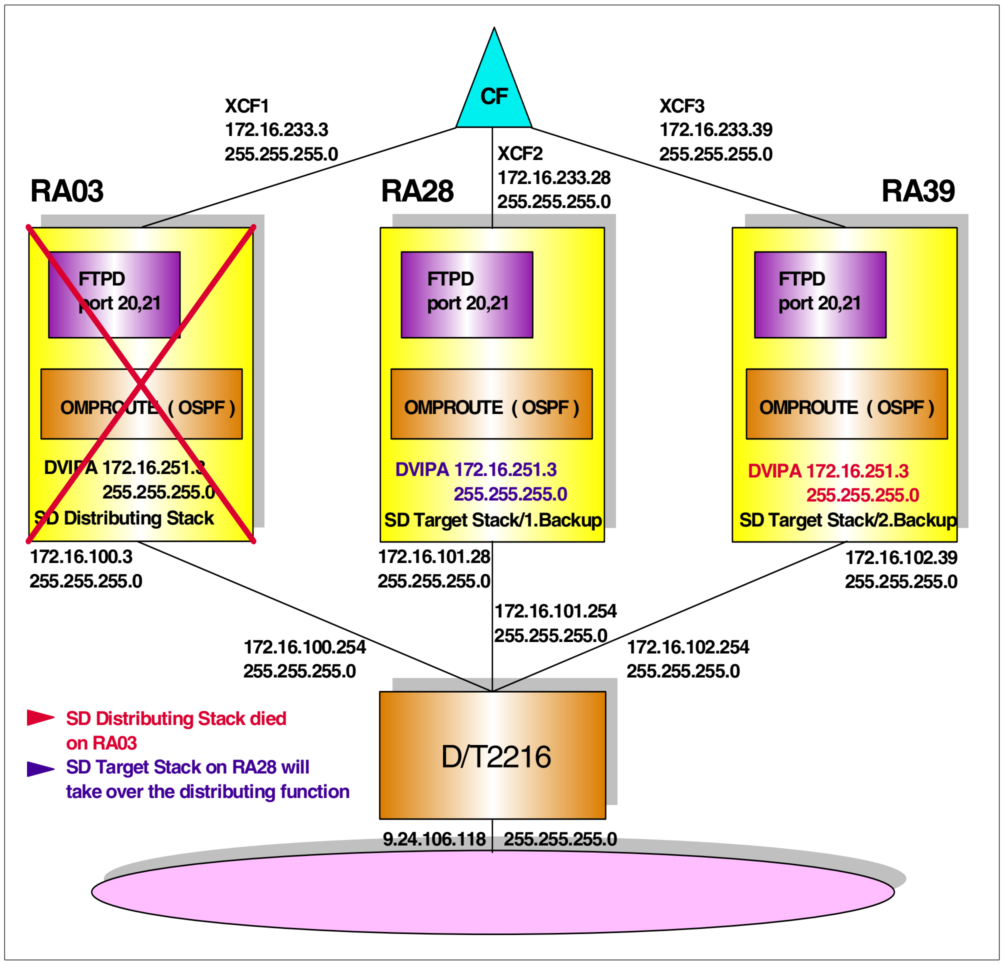


Figure 231. RA28 IP stack will take over the distributing function of RA03

In this example, we will 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 232 on page 230 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 232. D TCPIP,TCPIPC,N,VDPT on RA03 IP stack after establishing one FTP session

Figure 233 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 233. D TCPIP,TCPIPC,N,VCRT on RA03 IP stack after establishing one FTP session

Figure 234 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 234. 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 235 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 235. D TCPIP,TCIPC,N,VDPT on RA28 IP stack after DVIPA takeover

Figure 236 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 233 on page 230.

```
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 236. D TCPIP,TCPIPC,N,VCRT on RA28 IP stack after DVIPA takeover

Figure 237 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 237. D TCPIP,TCPIP,SYSPLEX VIPADYN on RA28 IP stack after DVIPA takeover

Figure 238 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 238. D TCPIP,TCPIPC,N,HOME on RA28 IP stack after DVIPA takeover

Figure 239 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 239. Starting RA03 IP stack

Because the VIPADEFINE is coded with MOVE IMMED in the PROFILE data set, as shown in Figure 212 on page 215, the RA03 IP stack takes ownership of the DVIPA without waiting for existing connections to finish. This process is non-disruptive 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 240 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 240. D TCPIP,TCPIPC,N,VDPT on RA03 IP stack after DVIPA takeback

Figure 241 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 241. 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 242.

```
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 242. D TCPIP,TCPIPC,SYSPLEX,VIPADYN on RA03 IP stack after DVIPA takeback

7.7.3 Scenario 3: Distributing multiple IP services

Figure 243 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 the 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 number 1492. Please refer to Figure 243.

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.

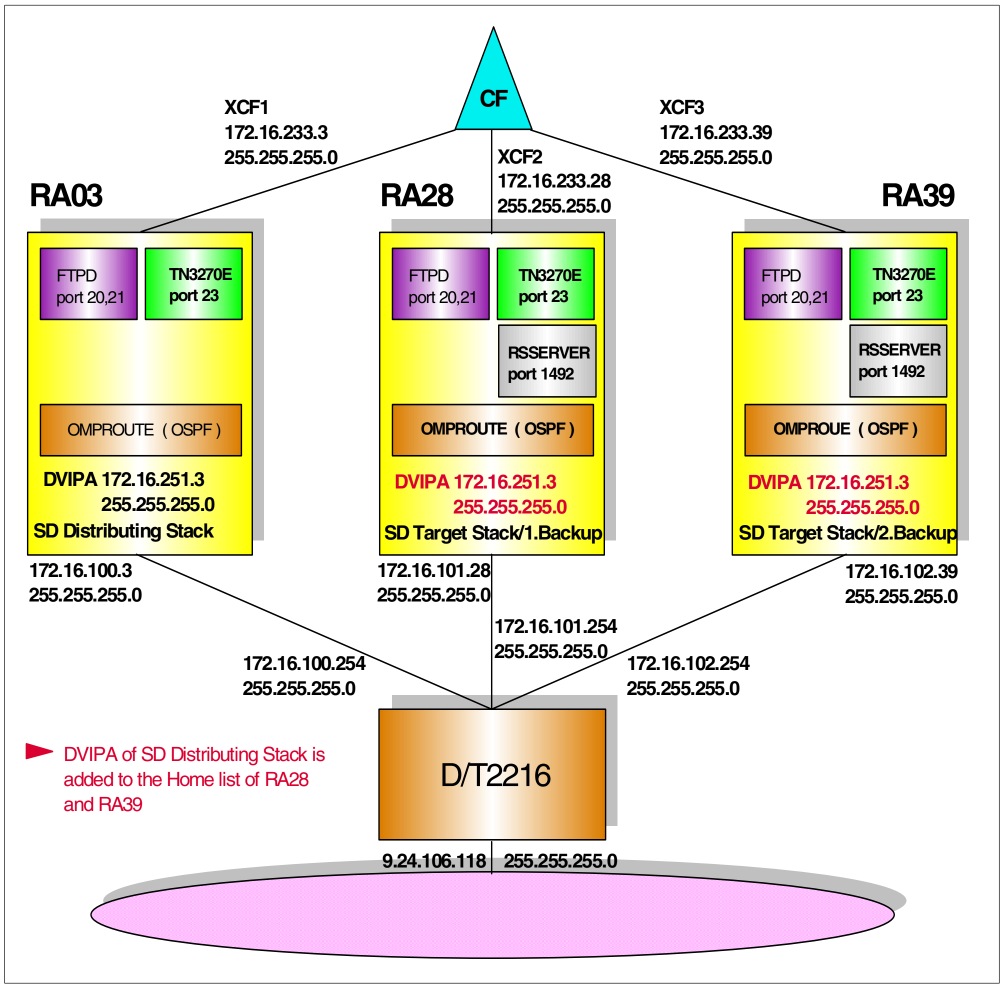


Figure 243. Distributing multiple IP services

Figure 244, Figure 245, and Figure 246 show the PROFILE data set being used for this scenario. There are some differences from those shown in Figure 212 on page 215, Figure 213 on page 216, and Figure 214 on page 217*:*

- 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 244. 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 245. 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 246. PROFILE data set for RA39 IP stack

Figure 247 shows the `NESTAT VIPADCFG` output for all the IP stacks participating in the sysplex cluster. Comparing this output with the one shown in Figure 218 on page 220, 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 247. The three stacks RA03, RA28, and RA39 will be considered for FTP **1** and TN3270E **2** connection request. 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 247. D TCPIP,TCPIPC,N,VIPACDFG for RA03, RA28 and RA39 IP stack

Figure 248 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 248. 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 249 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 249. 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 250 and Figure 251 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 250. 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 251. D TCPIP,TCPIPC,N,VDPT,DET on RA03 IP stack after running multiple IP services for some time

7.7.4 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 252), 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> keyword. You must use the VIPADISTRIBUTE DELETE with the keyword DISTIP ALL and then you can use VIPADISTRIBUTE DEFINE with keyword DESTIP <dynxcfip> to specify the new specific stacks for distribution consideration.

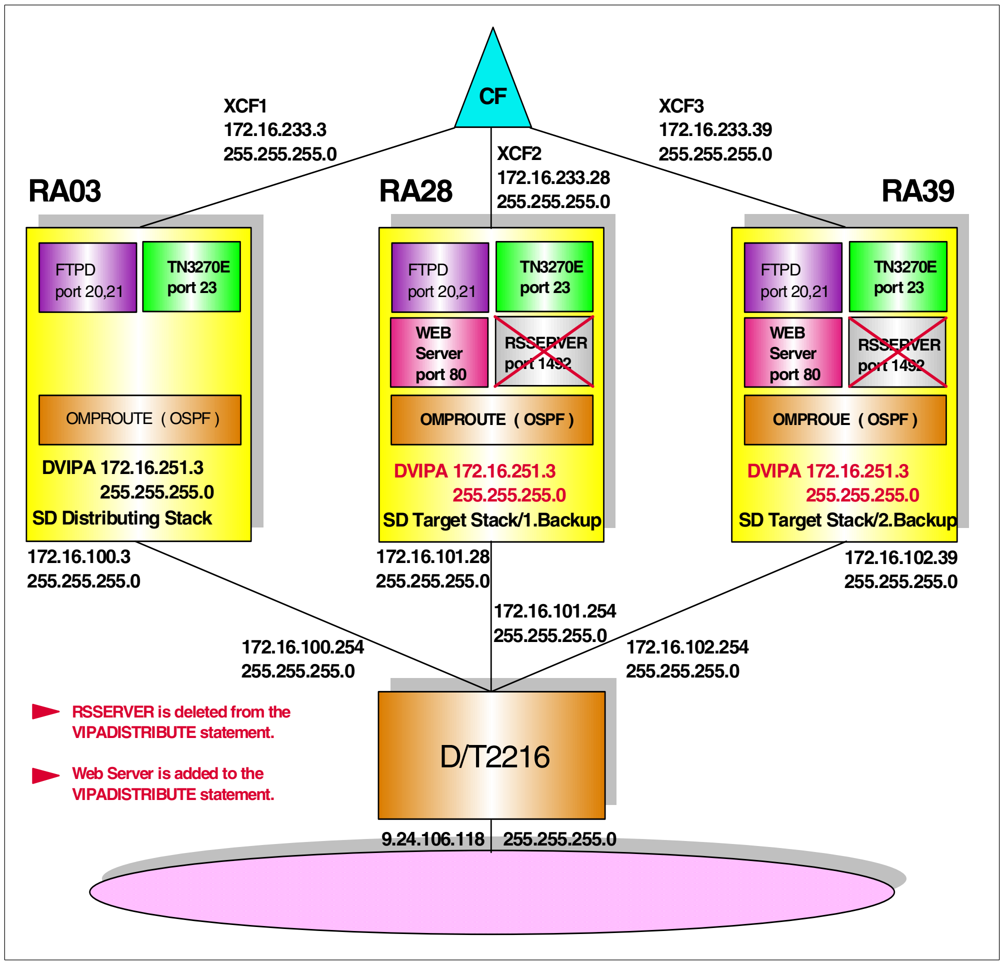


Figure 252. Deleting RSSERVER distribution and adding Web Server distribution dynamically

We used the VARY OBEY file shown in Figure 253 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.

```
VIPADYNAMIC
 VIPADISTRIBUTE DELETE 172.16.251.3 PORT 1492   1
     DESTIP 172.16.233.28 172.16.233.39         2
ENDVIPADYNAMIC
```

Figure 253. OBEY file for VIPADISTRIBUTE DELETE of port 1492

Figure 254 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 254. 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 255.

```
VIPADYNAMIC
 VIPADISTRIBUTE 172.16.251.3 PORT 80
     DESTIP 172.16.233.28 172.16.233.39
ENDVIPADYNAMIC
```

Figure 255. VARY OBEY file to distribute the Web server on port 80

Figure 256 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 256. D TCPIP,TCPIPC,N,VDPT on RA03 after VIPADISTRIBUTE DEFINE for port 80

Figure 257 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 257. 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 258.

```
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 258. D TCPIP,TCPIPC,N,VDPT on RA03 after some time running the distribution

7.7.5 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.

Assume we still have the same environment as we had after 7.7.4, “Scenario 4: Deleting and adding a VIPADISTRIBUTE statement” on page 243. 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 259 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 259. 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 260. We also automatically delete the VIPBACKUP definition on RA39 by deleting the RA03 DVIPA on RA39.

```
VIPADYNAMIC
 VIPADELETE 172.16.251.3
ENDVIPADYNAMIC
```

Figure 260. 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 261 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 261. D TCPIP,TCPIPC,SYSPLEX,VIPADYN after deleting VIPABACKUP definition

Figure 262 and Figure 263 shows 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 262. 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 263. 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 7.7.4, “Scenario 4: Deleting and adding a VIPADISTRIBUTE statement” on page 243). 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 264.

```
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
ENDVIPADYNAMIC
VIPADYNAMIC
 VIPADISTRIBUTE 172.16.251.3 PORT 20 21 23
     DESTIP 172.16.233.3  172.16.233.28  2
ENDVIPADYNAMIC
```

Figure 264. OBEY file to remove only the distribution to target IP stack RA39

Figure 265 and Figure 266 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 265. 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 266. 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 is new 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 267.

```
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 267. 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 268 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 269 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 268. 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 269. D TCPIP,TCPIPC,N,HOME after disconnecting all connections to RA39

7.8 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 7.5.4, “Implementation” on page 210.
2. Use the `NETSTAT` commands described in 7.7.1, “Scenario 1: Three IP stacks distributing FTP services” on page 213 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 C, “REXX EXECs to gather connection statistics” on page 287. The REXX program connects to a server on a given hostname/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 *OS/390 IBM Communications Server: IP Configuration Reference,* SC31-8726 for further information.

Appendix A. 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.

A.1 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 B.1, "WLMREG registration sample" on page 267.

We will now show how to set up and use a simple user TCP/IP server application written in C.

A.1.1 The registration call

The IWMDNREG C function is documented in *IBM Communications Server for OS/390 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. See *OS/390 MVS Programming: Workload Management Services*, GC28-1773, for more information.

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.

A.1.2 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.

A.1.3 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 for 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.

A.2 Collecting statistics using REXX

In 2.6.5, “Observing the effects of WLM and DNS” on page 48 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 (NDR). However, the NDR 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 of 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 270:

```
rc = SockGetHostByName(connectToName, "resolvedHost.!")
if(rc = 0) then
do
  say "Error resolving hostname: " errno
  return
end
```

Figure 270. Resolving hostname in REXX sockets API

It then allocates a standard stream socket and connects to it as shown in the code excerpt in Figure 271:

```
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 271. 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 272:

```
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 272. 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 C.2, “EXEC to connect to server using TCP” on page 290. The subroutine that is called at the end of the REXX program to sort and output the statistics, `sysstats`, is in C.3, “REXX statistics subroutine” on page 293. 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.

A.3 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. 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 273 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 273. 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.

A.4 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 274:

```
    if(gethostname(hostName, 64) !=0)
    {
        tcperror("Gethostname()");
        exit(2);
    }

    if((hostId = gethostid()) == 0)
    {
        tcperror("Gethostid()");
        exit(2);
    }
```

Figure 274. 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 275:

```
    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 275. 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 276:

```
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 276. Accept conversation, send data and close

The complete source for the sample is available in B.3, “SOCSRVR single threading server” on page 274.

See Figure 277 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 277. Sample JCL to run the WLMREG then the SOCSRVR programs

A.4.1 Modifying SOCSRVR for Dynamic VIPA

To modify the SOCSRVR (see A.4, “SOCSRVR, a simple socket server program” on page 260) sample to exploit dynamic VIPA is very simple. First we change the parameter checking so that two arguments are required as in Figure 278:

```
if (argc != 3)
{
  fprintf(stderr, "Usage: %s port VIPA\n", argv[0]);
  exit(1);
}
```

Figure 278. 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 279:

```
    server.sin_addr.s_addr = inet_addr( argv[2] );

    if (bind(s, (struct sockaddr *)&server, sizeof(server)) < 0)
    {
        tcperror("Bind()");
        exit(4);
    }
```

Figure 279. Binding to a specific VIPA

A.5 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 OS/390 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 OS/390 sysplex where applicable.

A.5.1 Discovering partner information

This is done by way of a new option on the socket call `getsockopt()`, which is described in more detail in *IBM Communications Server for OS/390 IP Application Programming Interface Guide*, SC31-8516. This new option is `SO_CLUSTERCONNTYPE` and is coded as shown in Figure 280:

```
    if (getsockopt(s, SOL_SOCKET, SO_CLUSTERCONNTYPE, (char *)&type,
&typelen) < 0)
    {
        tcperror("GetSockOpt()");
        exit(5);
```

Figure 280. getsockopt() call

The call can return any of the values shown in Table 6. In this context a *cluster* is a sysplex:

Table 6. 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 in the code excerpt in Figure 281:

```
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 281. getsockopt() call

A.5.2 SSOCCLNT, a sample sysplex sockets program

The sample sysplex sockets program very simply connects to the SOCSRVR program introduced in A.4, “SOCSRVR, a simple socket server program” on page 260 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 282:

```
Connection Type :
 Same Image
 Same Cluster
Server's IP address : 9.24.104.113
```

Figure 282. Output from SSOCCLNT

The complete source for the sample is available in B.4, “SSOCCLNT sysplex sockets sample” on page 276.

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.

A.6 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.

A.6.1 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 *IBM Communications Server for OS/390 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 B.5, "MTCSRVR multitasking sockets program" on page 278, 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 283.

```
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 283. 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 284.

```
      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 284. 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 285.

```
      if((hostId = gethostid()) == 0)
      {
        tcperror("Csub: Gethostid()");
      }
      sendbytes = send(socket, (char*) &hostId, sizeof(hostId), 0);
      if (sendbytes < 0)
      {
        tcperror("Csub: Send()");
      }
```

Figure 285. Querying the host ID and sending the value to the client

One thing to note when getting the subtask to work is that 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 B.6, “MTCSUBT subtask for the multitasking sockets program” on page 285.

A.6.2 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 A.2, “Collecting statistics using REXX” on page 256. 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 C.2, “EXEC to connect to server using TCP” on page 290.

Appendix B. 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.

B.1 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);
}
```

B.2 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" );
}
```

B.3 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);
    }
}
```

B.4 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);
}
```

B.5 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;
}
```

B.6 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);
}
```

Appendix C. REXX EXECs to gather connection statistics

In this section we show the three REXX EXECs used to invoke server functions in the sysplex for the purpose of gathering statistics on load balancing.

C.1 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 <> '' À 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' ÀÀ mmv ÀÀ ssv ÀÀ Substr(therest,1,2) ÀÀ '.dat'
pingfile = '\pf' ÀÀ mmv ÀÀ ssv ÀÀ Substr(therest,1,2) ÀÀ '.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 ÀÃ ' -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
```

C.2 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"
```

C.3 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
```

Appendix D. Profiles, data files and parameter files

This appendix shows the profiles and configuration files used in our tests described in Chapters 2 through 7.

D.1 Configuration files for RIP examples

This section shows the profiles and configuration files used in ORouteD with RIPF protocol tests described in 6.5.2, “OMPROUTE with RIP” on page 165.

D.1.1 Profile for TCPIPC stack at RA03 image - PROF03C

```
;-----------------------------------------------------------------
;TCPIP.TCPPARMS.R2617(PROF03C) SYSPLEX DISTRIBUTOR RA03 - SANDRAEF
;-----------------------------------------------------------------
IPCONFIG
 DATAGRAMFWD
 DYNAMICXCF 172.16.233.3 255.255.255.0 1
 SYSPLEXROUTING
 ARPTO  1200
 IGNOREREDIRECT
 SOURCEVIPA
 STOPONCLAWERROR
 TTL    60
 VARSUBNETTING

TCPCONFIG
 UNRESTRICTLOWPORTS
 TCPSENDBFRSIZE  16384
 TCPRCVBUFRSIZE  16384
 SENDGARBAGE FALSE

UDPCONFIG
 UNRESTRICTLOWPORTS
 UDPCHKSUM
 UDPSENDBFRSIZE  16384
 UDPRCVBUFRSIZE  16384

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
 25  TCP OMVS
 53  TCP OMVS
 53  UDP OMVS
 80  TCP WEBSRV SHAREPORT
 80  TCP WEBSRVC
 111 TCP OMVS
 111 UDP OMVS
 135 UDP LLBD
 161 UDP OSNMPD
 162 UDP OMVS
 443 TCP OMVS
 512 TCP RSHDA
 514 TCP RSHDA
 514 UDP OMVS
 515 TCP T03ALPD
 520 UDP OROUTEDC
;520 UDP OMVS

TELNETPARMS
 PORT 23
 INACTIVE 0
ENDTELNETPARMS

AUTOLOG 5
 FTPDC JOBNAME FTPDC1
 OROUTEDC
ENDAUTOLOG

 DEVICE VIPA03C  VIRTUAL  0
 LINK   VIPA03C  VIRTUAL  0  VIPA03C

 DEVICE M032216B MPCPTP AUTORESTART
 LINK   M032216B MPCPTP M032216B

 DEVICE EN103  LCS           2064
 LINK   EN103  ETHEROR802.3  1     EN103

HOME
 172.16.251.3 VIPA03C
 172.16.100.3 M032216B
 9.24.105.75  EN103

BSDROUTINGPARMS FALSE
 VIPA03C   DEFAULTSIZE 0  255.255.255.0 0
 EN103        1492     1  255.255.255.0 0
 M032216B    32768     0  255.255.255.0 0
ENDBSDROUTINGPARMS

BEGINVTAM
 PORT 23
 DEFAULTLUS
    RA03TN11..RA03TN19
 ENDDEFAULTLUS
 ALLOWAPPL *
 USSTCP TELNUST
ENDVTAM

INCLUDE TCPIP.TCPPARMS.R2617(TELN03C)

START M032216B
START EN103
```

D.1.2 TCPIP.DATA File for TCPIPC stack at RA03 image - TCPD03C

```
;-----------------------------------------------------------------
;TCPIP.TCPPARMS(TCPD03C) SYSPLEX DISTRIBUTOR RA03 - SANDRAEF
```

```
;---------------------------------------------------------------
TCPIPJOBNAME TCPIPC
HOSTNAME  MVS03C
DOMAINORIGIN  itso.ral.ibm.com
NSINTERADDR  9.24.106.15
NSPORTADDR 53
RESOLVEVIA UDP
RESOLVERTIMEOUT 10
RESOLVERUDPRETRIES 2
DATASETPREFIX TCPIP
MESSAGECASE MIXED
```

D.1.3 Telnet parameters for TCPIPC stack at image RA03 - TELN03A

```
;---------------------------------------------------------------
;TCPIP.TCPPPARMS.R2617(TELN03C) SYSPLEX DISTRIBUTOR RA03 - SANDRAEF
;---------------------------------------------------------------
TELNETGLOBALS
   KEYRING HFS /u/iwan/security/ssl/hostsec.kdb
ENDTELNETGLOBALS
TELNETPARMS
   SECUREPORT 23
   CONNTYPE SECURE
   DEBUG DETAIL
ENDTELNETPARMS
TELNETPARMS
   TKOSPECLU 0
   PORT 23
   WLMCLUSTERNAME  TN03 TNRAL TNTSO   ENDWLMCLUSTERNAME
ENDTELNETPARMS

TELNETPARMS
   SECUREPORT 6623 KEYRING HFS /u/sandra/ssl/server/r2612.kdb
   CLIENTAUTH SAFCERT
   TKOSPECLU 3
ENDTELNETPARMS

TELNETPARMS
   SECUREPORT 7723 KEYRING HFS /u/sandra/ssl/server/r2612.kdb
   CLIENTAUTH NONE
   TKOSPECLU 3
ENDTELNETPARMS

TELNETPARMS
   SECUREPORT 8823 KEYRING HFS /u/sandra/ssl/server/r2612.kdb
   CLIENTAUTH SSLCERT
   TKOSPECLU 3
ENDTELNETPARMS

TELNETPARMS
   SECUREPORT 9923 KEYRING HFS /u/sandra/ssl/server/r2612.kdb
   CLIENTAUTH SAFCERT
   TKOSPECLU 3
ENDTELNETPARMS

BEGINVTAM
   PORT 23 6623 7723 8823 9923
   ALLOWAPPL *
   MSG07
   IPGROUP IP1 0.0.0.0:0.0.0.0 ENDIPGROUP
   LUGROUP LU1 RA03TN01..RA03TN20 ENDLUGROUP
   PRTGROUP PR1 RA03TP01..RA03TP20 ENDPRTGROUP
   LUMAP LU1 IP1 GENERIC PR1
   PRTMAP PR1 IP1 GENERIC

  IPGROUP IP2
      9.24.105.220 9.24.106.165 9.24.106.31
  ENDIPGROUP
   LUGROUP LU3 RA03TN50 RA03TN51 ENDLUGROUP
   PRTGROUP PR3 RA03TP50 RA03TP51 ENDPRTGROUP
   PRTMAP PR3 IP2 SPECIFIC
   LUMAP LU3 IP2 SPECIFIC PR3

; LUTSO group includes lus from RA03TN70 to RA03TN75
  LUGROUP LUTSO
      RA03TN70..RA03TN75
  ENDLUGROUP

; LUNETV group includes RA03TN78 lu
  LUGROUP LUNETV
      RA03TN78
  ENDLUGROUP

; LUNVAS group includes RA03TN79 lu
  LUGROUP LUNVAS
      RA03TN79
  ENDLUGROUP

; TSOPRT group includes PRINTERS from RA03TP70 to RA03TP75
  PRTGROUP TSOPRT
      RA03TP70..RA03TP75
  ENDPRTGROUP

  USSTCP TELNUST IP2
  USSTCP TELNUST
   TELNETDEVICE 3287-1     ,SCS
   TELNETDEVICE 3277     DSILGMOD          ; 24 x 80   old model 2
   TELNETDEVICE 3278-2   D4B32782,SNX32702 ; 24 x 80
   TELNETDEVICE 3278-2-e NSX32702,SNX32702 ; 24 x 80
   TELNETDEVICE 3278-3   D4B32783,SNX32703 ; 32 x 80,  primary 24 x 80
   TELNETDEVICE 3278-3-e NSX32703,SNX32703 ; 32 x 80,  primary 24 x 80
   TELNETDEVICE 3278-4   D4B32784,SNX32704 ; 43 x 80,  primary 24 x 80
   TELNETDEVICE 3278-4-e NSX32704,SNX32704 ; 43 x 80,  primary 24 x 80
   TELNETDEVICE 3278-5   D4B32785,SNX32705 ; 27 x 132, primary 24 x 80
   TELNETDEVICE 3278-5-e NSX32705,SNX32705 ; 27 x 132, primary 24 x 80
   TELNETDEVICE 3279-2   D4B32782          ; 24 x 80
   TELNETDEVICE 3279-2-e NSX32702          ; 24 x 80
   TELNETDEVICE 3279-3   D4B32783          ; 32 x 80,  primary 24 x 80
   TELNETDEVICE 3279-3-e NSX32703          ; 32 x 80,  primary 24 x 80
   TELNETDEVICE 3279-4   D4B32784          ; 43 x 80,  primary 24 x 80
   TELNETDEVICE 3279-4-e NSX32704          ; 43 x 80,  primary 24 x 80
   TELNETDEVICE 3279-5   D4B32785          ; 27 x 132, primary 24 x 80
   TELNETDEVICE 3279-5-e NSX32705          ; 27 x 132, primary 24 x 80
   TELNETDEVICE LINEMODE INTERACT          ; linemode terminals
ENDVTAM
```

D.1.4 FTP data parameters for FTP server - FDATA03C and 39C

```
;-----------------------------------------------------------------
;TCPIP.TCPPARMS(FTPD03C) SYSPLEX DISTRIBUTOR RA03 - SANDRAEF
;-----------------------------------------------------------------
AUTOMOUNT     TRUE             ; automatic mount of unmounted volume
AUTORECALL    TRUE             ; automatic recall of migrated data sets
BLOCKSIZE     6233             ; new data set allocation blocksize
BUFNO         5                ; number of access method buffers
CHKPTINT      0                ; checkpoint interval
CONDDISP      CATLG            ; data sets catalogued if transfer fails
CTRLCONN      IBM-850          ; ascii code set for control connection
DIRECTORY     27               ; new data set allocation directory blocks
DIRECTORYMODE FALSE            ; directorymode vs. data set mode
FILETYPE      SEQ              ; file transfer mode
INACTIVE      300              ; inactive time out
JESLRECL      80               ; lrecl of jes jobs
JESPUTGETTO   600              ; timeout for remote job submission put/ge
JESRECFM      F                ; recfm of jes jobs
LRECL         256              ; new data set allocation lrecl
PRIMARY       1                ; new data set allocation primary space
RDW           false            ; if RDWs are treated as a part of record
RECFM         VB               ; new data set allocation record format
SBDATACONN   (IBM-1047,IBM-850) ; ebcdic/ascii code sets for data conn.
SECONDARY     1                ; new data set allocation secondary space
SPACETYPE     TRACK            ; new data set allocation space type
SPREAD        FALSE            ; sql output format
SQLCOL        NAMES            ; sql output uses column names as headings
STARTDIR      HFS              ; use MVS directory at connect time
UCSHOSTCS     IBM-939          ; the EDCDIC code page from/to UCS-2
UCSSUB        FALSE            ; whether substitution is permitted.
UCSTRUNC      FALSE            ; whether truncation is permitted.
WLMCLUSTERNAME FTPOERAL        ; group name registered in DNS/WLM sysplex
WRAPRECORD    FALSE            ; data is NOT wrapped to next record
```

D.1.5 Profile for TCPIPC stack at RA39 image - PROF39C

```
;-----------------------------------------------------------------
;TCPIP.TCPPARMS.R2617(PROF39C) SYSPLEX DISTRIBUTOR RA39 - SANDRAEF
;-----------------------------------------------------------------
IPCONFIG
 DATAGRAMFWD
 DYNAMICXCF 172.16.233.39 255.255.255.0 1
 SYSPLEXROUTING
 ARPTO  1200
 IGNOREREDIRECT
 SOURCEVIPA
 STOPONCLAWERROR
 TTL    60
 VARSUBNETTING

TCPCONFIG
 UNRESTRICTLOWPORTS
 TCPSENDBFRSIZE  16384
 TCPRCVBUFRSIZE  16384
 SENDGARBAGE FALSE

UDPCONFIG
 UNRESTRICTLOWPORTS
 UDPCHKSUM
 UDPSENDBFRSIZE  16384
 UDPRCVBUFRSIZE  16384

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
 25  TCP OMVS
 53  TCP OMVS
 53  UDP OMVS
 80  TCP WEBSRV SHAREPORT
 80  TCP WEBSRVC
 111 TCP OMVS
 111 UDP OMVS
 135 UDP LLBD
 161 UDP OSNMPD
 162 UDP OMVS
 443 TCP OMVS
 512 TCP RSHDA
 514 TCP RSHDA
 514 UDP OMVS
 515 TCP T39ALPD
 520 UDP OROUTEDC
;520 UDP OMVS

TELNETPARMS
 PORT 23
 INACTIVE 0
ENDTELNETPARMS

AUTOLOG 5
 FTPDC JOBNAME FTPDC1
 OROUTEDC
ENDAUTOLOG

 DEVICE VIPA39C  VIRTUAL  0
 LINK   VIPA39C  VIRTUAL  0  VIPA39C

 DEVICE M392216B MPCPTP AUTORESTART
 LINK   M392216B MPCPTP M392216B

 DEVICE EN139  LCS           2064
 LINK   EN139  ETHEROR802.3  1     EN139

HOME
 172.16.251.39 VIPA39C
 172.16.102.39 M392216B
 9.24.105.73   EN139

BSDROUTINGPARMS FALSE
 VIPA39C   DEFAULTSIZE 0  255.255.255.0 0
 EN139        1492     1  255.255.255.0 0
 M392216B    32768     0  255.255.255.0 0
ENDBSDROUTINGPARMS

BEGINVTAM
 PORT 23
 DEFAULTLUS
    RA39TN11..RA39TN19
 ENDDEFAULTLUS
 ALLOWAPPL *
 USSTCP TELNUST
ENDVTAM

INCLUDE TCPIP.TCPPARMS.R2617(TELN39C)

START M392216B
START EN139
```

D.1.6 TCPIP.DATA file for TCPIPC stack at RA39 image - TCPD39C

```
;-------------------------------------------------------------------
; TCPIP.TCPPARMS(TCPD39C) SYSPLEX DISTRIBUTOR RA39 - SANDRAEF
;-------------------------------------------------------------------
TCPIPJOBNAME TCPIPC
HOSTNAME  MVS39C
DOMAINORIGIN  itso.ral.ibm.com
NSINTERADDR  9.24.106.15
NSPORTADDR 53
RESOLVEVIA UDP
RESOLVERTIMEOUT 10
RESOLVERUDPRETRIES 1
DATASETPREFIX TCPIP
MESSAGECASE MIXED
```

D.1.7 Telnet parameters for TCPIPC stack on RA39 image - TELN39C

```
;-------------------------------------------------------------------
;TCPIP.TCPPARMS.R2617(TELN39C) SYSPLEX DISTRIBUTOR RA39 - SANDRAEF
;-------------------------------------------------------------------
TELNETPARMS
    PORT 23
    INACTIVE 0
    TIMEMARK 7200
    SCANINTERVAL 300
    SMFINIT STD
    SMFTERM STD
    WLMCLUSTERNAME TNRAL ENDWLMCLUSTERNAME
ENDTELNETPARMS
;
BEGINVTAM
  TELNETDEVICE 3277     DSILGMOD          ; 24 x 80   old model 2
  TELNETDEVICE 3278-2   D4B32782,SNX32702 ; 24 x 80
  TELNETDEVICE 3278-2-e NSX32702,SNX32702 ; 24 x 80
  TELNETDEVICE 3278-3   D4B32783,SNX32703 ; 32 x 80,  primary 24 x 80
  TELNETDEVICE 3278-3-e NSX32703,SNX32703 ; 32 x 80,  primary 24 x 80
  TELNETDEVICE 3278-4   D4B32784,SNX32704 ; 43 x 80,  primary 24 x 80
  TELNETDEVICE 3278-4-e NSX32704,SNX32704 ; 43 x 80,  primary 24 x 80
  TELNETDEVICE 3278-5   D4B32785,SNX32705 ; 27 x 132, primary 24 x 80
  TELNETDEVICE 3278-5-e NSX32705,SNX32705 ; 27 x 132, primary 24 x 80
  TELNETDEVICE 3279-2   D4B32782          ; 24 x 80
  TELNETDEVICE 3279-2-e NSX32702          ; 24 x 80
  TELNETDEVICE 3279-3   D4B32783          ; 32 x 80,  primary 24 x 80
  TELNETDEVICE 3279-3-e NSX32703          ; 32 x 80,  primary 24 x 80
  TELNETDEVICE 3279-4   D4B32784          ; 43 x 80,  primary 24 x 80
  TELNETDEVICE 3279-4-e NSX32704          ; 43 x 80,  primary 24 x 80
  TELNETDEVICE 3279-5   D4B32785          ; 27 x 132, primary 24 x 80
  TELNETDEVICE 3279-5-e NSX32705          ; 27 x 132, primary 24 x 80
  TELNETDEVICE LINEMODE INTERACT          ; linemode terminals
  TELNETDEVICE DYNAMIC          ,D4C32XX3 ; tbd by application (QUERY)
  TELNETDEVICE 3287-1           ,SCS      ; printer             LU1
  DEFAULTLUS
      RA39TN31..RA39TN50
      RA39TP31..RA39TP50  ; printers
  ENDDEFAULTLUS
;
  LUGROUP SPECLU
      RA39TN90..RA39TN99  ; specials
  ENDLUGROUP
;
  PRTGROUP PRINTERS
      RA39TPR8..RA39TPR9  ; printers
  ENDPRTGROUP
;
  LUMAP SPECLU   9.170.3.123
;
  MSG07            ; Error messages will be issued
  LUSESSIONPEND    ; On termination of a Telnet server connection,
;                  ; the user will revert to the DEFAULTAPPL
  USSTCP TELNUST
  LINEMODEAPPL RA03T ; Send all line-mode terminals directly to TSO.
  ALLOWAPPL RA*  ;* DISCONNECTABLE  Allow all users access to TSO
  ALLOWAPPL AD*
  ALLOWAPPL A2*
  ALLOWAPPL FD*
  ALLOWAPPL X6*
  ALLOWAPPL X7*
;
ENDVTAM
```

D.2 Configuration files for OSPF examples

This section shows the profiles and configuration files used in OMPROUTED with OSPF protocol tests described in 6.5.3, “OMPROUTE with OSPF” on page 167.

D.2.1 Profile for TCPIPC stack at RA03 image - PROF03C

```
;*****************************************************************
;TCPIP.TCPPARMS.R2617(OM39CCFG) SYSPLEX DISTRIBUTOR RA03 - SANDRAEF
;*****************************************************************
IPCONFIG
 DATAGRAMFWD
 DYNAMICXCF 172.16.233.3 255.255.255.0 1
 SYSPLEXROUTING
 ARPTO  1200
 IGNOREREDIRECT
 SOURCEVIPA
 STOPONCLAWERROR
 TTL    60
 VARSUBNETTING

TCPCONFIG
 UNRESTRICTLOWPORTS
 TCPSENDBFRSIZE  16384
 TCPRCVBUFRSIZE  16384
 SENDGARBAGE FALSE

UDPCONFIG
 UNRESTRICTLOWPORTS
 UDPCHKSUM
 UDPSENDBFRSIZE  16384
 UDPRCVBUFRSIZE  16384

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
 25  TCP OMVS
 53  TCP OMVS
 53  UDP OMVS
 80  TCP WEBSRV SHAREPORT
 80  TCP WEBSRVC
 111 TCP OMVS
 111 UDP OMVS
 135 UDP LLBD
 161 UDP OSNMPD
 162 UDP OMVS
 443 TCP OMVS
 512 TCP RSHDA
 514 TCP RSHDA
 514 UDP OMVS
 515 TCP T03ALPD
 520 UDP OMPROUTC
;520 UDP OMVS

TELNETPARMS
 PORT 23
 INACTIVE 0
ENDTELNETPARMS

AUTOLOG 5
 FTPDC JOBNAME FTPDC1
 OMPROUTC
ENDAUTOLOG

 DEVICE M032216B MPCPTP AUTORESTART
 LINK   M032216B MPCPTP M032216B
 DEVICE EN103  LCS           2064
 LINK   EN103     ETHEROR802.3  1     EN103

HOME
 172.16.100.3 M032216B
 9.24.105.76  EN103

VIPADYNAMIC
  VIPADEFINE MOVE IMMED 255.255.255.0 172.16.251.3
  VIPABACKUP 100 172.16.251.39
  VIPARANGE  DEFINE MOVEABLE DISRUPT 255.255.255.0 172.16.240.193
ENDVIPADYNAMIC

BEGINVTAM
 PORT 23
 DEFAULTLUS
    RA03TN11..RA03TN19
 ENDDEFAULTLUS
 ALLOWAPPL *
 USSTCP TELNUST
ENDVTAM

INCLUDE TCPIP.TCPPARMS.R2617(TELN03C)

START M032216B
START EN103
```

D.2.2 TCPIP.DATA file for TCPIPC stack at RA03 image - TCPD03C

```
;*****************************************************************
;TCPIP.TCPPARMS(TCPD03C) SYSPLEX DISTRIBUTOR RA03 - SANDRAEF
;*****************************************************************
TCPIPJOBNAME TCPIPC
HOSTNAME  MVS03C
DOMAINORIGIN  itso.ral.ibm.com
NSINTERADDR  9.24.106.15
NSPORTADDR 53
RESOLVEVIA UDP
RESOLVERTIMEOUT 10
RESOLVERUDPRETRIES 2
DATASETPREFIX TCPIP
MESSAGECASE MIXED
```

D.2.3 Telnet parameters for TCPIPC stack at image RA03 - TELN03A

```
;******************************************************************
;TCPIP.TCPPPARMS.R2617(TELN03C) SYSPLEX DISTRIBUTOR RA03 - SANDRAEF
;******************************************************************

TELNETGLOBALS
   KEYRING HFS /u/iwan/security/ssl/hostsec.kdb
ENDTELNETGLOBALS
TELNETPARMS
   SECUREPORT 23
   CONNTYPE SECURE
   DEBUG DETAIL
ENDTELNETPARMS
```

```
TELNETPARMS
   TKOSPECLU 0
   PORT 23
   WLMCLUSTERNAME  TN03 TNRAL TNTSO   ENDWLMCLUSTERNAME
ENDTELNETPARMS

TELNETPARMS
   SECUREPORT 6623 KEYRING HFS /u/sandra/ssl/server/r2612.kdb
   CLIENTAUTH SAFCERT
   TKOSPECLU 3
ENDTELNETPARMS

TELNETPARMS
   SECUREPORT 7723 KEYRING HFS /u/sandra/ssl/server/r2612.kdb
   CLIENTAUTH NONE
   TKOSPECLU 3
ENDTELNETPARMS

TELNETPARMS
   SECUREPORT 8823 KEYRING HFS /u/sandra/ssl/server/r2612.kdb
   CLIENTAUTH SSLCERT
   TKOSPECLU 3
ENDTELNETPARMS

TELNETPARMS
   SECUREPORT 9923 KEYRING HFS /u/sandra/ssl/server/r2612.kdb
   CLIENTAUTH SAFCERT
   TKOSPECLU 3
ENDTELNETPARMS

BEGINVTAM
   PORT 23 6623 7723 8823 9923
   ALLOWAPPL *
   MSG07
   IPGROUP IP1 0.0.0.0:0.0.0.0 ENDIPGROUP
   LUGROUP LU1 RA03TN01..RA03TN20 ENDLUGROUP
   PRTGROUP PR1 RA03TP01..RA03TP20 ENDPRTGROUP
   LUMAP LU1 IP1 GENERIC PR1
   PRTMAP PR1 IP1 GENERIC

  IPGROUP IP2
     9.24.105.220 9.24.106.165 9.24.106.31
  ENDIPGROUP
   LUGROUP LU3 RA03TN50 RA03TN51 ENDLUGROUP
   PRTGROUP PR3 RA03TP50 RA03TP51 ENDPRTGROUP
   PRTMAP PR3 IP2 SPECIFIC
   LUMAP LU3 IP2 SPECIFIC PR3

; LUTSO group includes lus from RA03TN70 to RA03TN75
  LUGROUP LUTSO
     RA03TN70..RA03TN75
  ENDLUGROUP

; LUNETV group includes RA03TN78 lu
  LUGROUP LUNETV
     RA03TN78
  ENDLUGROUP
```

```
; LUNVAS group includes RA03TN79 lu
  LUGROUP LUNVAS
      RA03TN79
  ENDLUGROUP

; TSOPRT group includes PRINTERS from RA03TP70 to RA03TP75
  PRTGROUP TSOPRT
      RA03TP70..RA03TP75
  ENDPRTGROUP

  USSTCP TELNUST IP2
  USSTCP TELNUST
   TELNETDEVICE 3287-1     ,SCS
   TELNETDEVICE 3277     DSILGMOD          ; 24 x 80   old model 2
   TELNETDEVICE 3278-2   D4B32782,SNX32702 ; 24 x 80
   TELNETDEVICE 3278-2-e NSX32702,SNX32702 ; 24 x 80
   TELNETDEVICE 3278-3   D4B32783,SNX32703 ; 32 x 80,  primary 24 x 80
   TELNETDEVICE 3278-3-e NSX32703,SNX32703 ; 32 x 80,  primary 24 x 80
   TELNETDEVICE 3278-4   D4B32784,SNX32704 ; 43 x 80,  primary 24 x 80
   TELNETDEVICE 3278-4-e NSX32704,SNX32704 ; 43 x 80,  primary 24 x 80
   TELNETDEVICE 3278-5   D4B32785,SNX32705 ; 27 x 132, primary 24 x 80
   TELNETDEVICE 3278-5-e NSX32705,SNX32705 ; 27 x 132, primary 24 x 80
   TELNETDEVICE 3279-2   D4B32782          ; 24 x 80
   TELNETDEVICE 3279-2-e NSX32702          ; 24 x 80
   TELNETDEVICE 3279-3   D4B32783          ; 32 x 80,  primary 24 x 80
   TELNETDEVICE 3279-3-e NSX32703          ; 32 x 80,  primary 24 x 80
   TELNETDEVICE 3279-4   D4B32784          ; 43 x 80,  primary 24 x 80
   TELNETDEVICE 3279-4-e NSX32704          ; 43 x 80,  primary 24 x 80
   TELNETDEVICE 3279-5   D4B32785          ; 27 x 132, primary 24 x 80
   TELNETDEVICE 3279-5-e NSX32705          ; 27 x 132, primary 24 x 80
   TELNETDEVICE LINEMODE INTERACT          ; linemode terminals
ENDVTAM
```

D.2.4 FTP data parameters for FTP server - FDATA03C and 39C

```
;TCPIP.TCPPARMS(FTPD03C) SYSPLEX DISTRIBUTOR RA03 - SANDRAEF
;*******************************************************************
AUTOMOUNT     TRUE              ; automatic mount of unmounted volume
AUTORECALL    TRUE              ; automatic recall of migrated data sets
BLOCKSIZE     6233              ; new data set allocation blocksize
BUFNO         5                 ; number of access method buffers
CHKPTINT      0                 ; checkpoint interval
CONDDISP      CATLG             ; data sets catalogued if transfer fails
CTRLCONN      IBM-850           ; ascii code set for control connection
DIRECTORY     27                ; new data set allocation directory blocks
DIRECTORYMODE FALSE             ; directorymode vs. data set mode
FILETYPE      SEQ               ; file transfer mode
INACTIVE      300               ; inactive time out
JESLRECL      80                ; lrecl of jes jobs
JESPUTGETTO   600               ; timeout for remote job submission put/ge
JESRECFM      F                 ; recfm of jes jobs
LRECL         256               ; new data set allocation lrecl
PRIMARY       1                 ; new data set allocation primary space
RDW           false             ; if RDWs are treated as a part of record
RECFM         VB                ; new data set allocation record format
SBDATACONN   (IBM-1047,IBM-850) ; ebcdic/ascii code sets for data conn.
SECONDARY     1                 ; new data set allocation secondary space
SPACETYPE     TRACK             ; new data set allocation space type
```

```
SPREAD         FALSE          ; sql output format
SQLCOL         NAMES          ; sql output uses column names as headings
STARTDIR       HFS            ; use MVS directory at connect time
UCSHOSTCS      IBM-939        ; the EDCDIC code page from/to UCS-2
UCSSUB         FALSE          ; whether substitution is permitted.
UCSTRUNC       FALSE          ; whether truncation is permitted.
WLMCLUSTERNAME FTPOERAL       ; group name registered in DNS/WLM sysplex
WRAPRECORD     FALSE          ; data is NOT wrapped to next record
```

D.2.5 OMPROUTE configuration file for TCPIPC stack at RA03 image - OM03CCFG

```
;*****************************************************************
;TCPIP.TCPPARMS.R2617(OM39CCFG) SYSPLEX DISTRIBUTOR RA39 - SANDRAEF
;*****************************************************************
Area    Area_Number=0.0.0.0
              Stub_Area=NO
              Authentication_type=None;
RouterID=172.16.100.3;
ROUTESA_CONFIG ENABLED=YES
              COMMUNITY="MVSsubagent";
OSPF_Interface IP_Address=172.16.100.3
              Name=M032216B
              Cost0=3
              Subnet_mask=255.255.255.0
              MTU=32768;
OSPF_Interface IP_Address=9.24.105.76
              Name=EN103
              Cost0=6
              Subnet_mask=255.255.255.0
              MTU=32768;
OSPF_Interface IP_Address=172.16.251.*
              Subnet_mask=255.255.255.0
              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;
AS_Boundary_routing
              Import_Direct_Routes=YES; ;
```

D.2.6 Profile for TCPIPC stack at RA39 image - PRO39C

```
;-----------------------------------------------------------------
;TCPIP.TCPPARMS.R2617(PROF39C) SYSPLEX DISTRIBUTOR RA39 - SANDRAEF
;-----------------------------------------------------------------
IPCONFIG
 DATAGRAMFWD
 DYNAMICXCF 172.16.233.39 255.255.255.0 1
 SYSPLEXROUTING
 ARPTO  1200
 IGNOREREDIRECT
 SOURCEVIPA
 STOPONCLAWERROR
 TTL    60
 VARSUBNETTING
```

```
TCPCONFIG
 UNRESTRICTLOWPORTS
 TCPSENDBFRSIZE  16384
 TCPRCVBUFRSIZE  16384
 SENDGARBAGE FALSE

UDPCONFIG
 UNRESTRICTLOWPORTS
 UDPCHKSUM
 UDPSENDBFRSIZE  16384
 UDPRCVBUFRSIZE  16384

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
 25  TCP OMVS
 53  TCP OMVS
 53  UDP OMVS
 80  TCP WEBSRV
 111 TCP OMVS
 111 UDP OMVS
 135 UDP LLBD
 161 UDP OSNMPD
 162 UDP OMVS
 443 TCP OMVS
 512 TCP RSHDA
 514 TCP RSHDA
 514 UDP OMVS
 515 TCP T03ALPD
 520 UDP OMPROUTC
;520 UDP OMVS

TELNETPARMS
 PORT 23
 INACTIVE 0
ENDTELNETPARMS

AUTOLOG 5
 FTPDC JOBNAME FTPDC1
 OMPROUTC
ENDAUTOLOG

 DEVICE M392216B MPCPTP AUTORESTART
 LINK   M392216B MPCPTP M392216B

 DEVICE EN139  LCS           2064
 LINK   EN139     ETHEROR802.3  1     EN139

HOME
 172.16.102.39 M392216B
 9.24.105.73   EN139

VIPADYNAMIC
  VIPADEFINE MOVE IMMED 255.255.255.0 172.16.251.39
  VIPABACKUP 200 172.16.251.3
  VIPARANGE  DEFINE MOVEABLE NONDISRUPT 255.255.255.0 172.16.240.193
ENDVIPADYNAMIC

BEGINVTAM
 PORT 23
 DEFAULTLUS
    RA39TN31..RA39TN39
 ENDDEFAULTLUS
 ALLOWAPPL *
 USSTCP TELNUST
ENDVTAM

START M392216B
START EN139
```

D.2.7 TCPIP.DATA file for TCPIPC stack at RA39 image - TCPD39C

```
;-------------------------------------------------------------------
; TCPIP.TCPPARMS(TCPD39C) SYSPLEX DISTRIBUTOR RA39 - SANDRAEF
;-------------------------------------------------------------------
TCPIPJOBNAME TCPIPC
HOSTNAME  MVS39C
DOMAINORIGIN  itso.ral.ibm.com
NSINTERADDR  9.24.106.15
NSPORTADDR 53
RESOLVEVIA UDP
RESOLVERTIMEOUT 10
RESOLVERUDPRETRIES 1
DATASETPREFIX TCPIP
MESSAGECASE MIXED
```

D.2.8 Telnet parameters for TCPIPC stack on RA39 image - TELN39C

```
;-------------------------------------------------------------------
;TCPIP.TCPPARMS.R2617(TELN39C) SYSPLEX DISTRIBUTOR RA39 - SANDRAEF
;-------------------------------------------------------------------
TELNETPARMS
    PORT 23
    INACTIVE 0
    TIMEMARK 7200
    SCANINTERVAL 300
    SMFINIT STD
    SMFTERM STD
    WLMCLUSTERNAME TNRAL ENDWLMCLUSTERNAME
ENDTELNETPARMS
;
BEGINVTAM
  TELNETDEVICE 3277     DSILGMOD          ; 24 x 80   old model 2
  TELNETDEVICE 3278-2   D4B32782,SNX32702 ; 24 x 80
  TELNETDEVICE 3278-2-e NSX32702,SNX32702 ; 24 x 80
  TELNETDEVICE 3278-3   D4B32783,SNX32703 ; 32 x 80,  primary 24 x 80
  TELNETDEVICE 3278-3-e NSX32703,SNX32703 ; 32 x 80,  primary 24 x 80
  TELNETDEVICE 3278-4   D4B32784,SNX32704 ; 43 x 80,  primary 24 x 80
  TELNETDEVICE 3278-4-e NSX32704,SNX32704 ; 43 x 80,  primary 24 x 80
  TELNETDEVICE 3278-5   D4B32785,SNX32705 ; 27 x 132, primary 24 x 80
  TELNETDEVICE 3278-5-e NSX32705,SNX32705 ; 27 x 132, primary 24 x 80
  TELNETDEVICE 3279-2   D4B32782          ; 24 x 80
  TELNETDEVICE 3279-2-e NSX32702          ; 24 x 80
  TELNETDEVICE 3279-3   D4B32783          ; 32 x 80,  primary 24 x 80
  TELNETDEVICE 3279-3-e NSX32703          ; 32 x 80,  primary 24 x 80
  TELNETDEVICE 3279-4   D4B32784          ; 43 x 80,  primary 24 x 80
  TELNETDEVICE 3279-4-e NSX32704          ; 43 x 80,  primary 24 x 80
  TELNETDEVICE 3279-5   D4B32785          ; 27 x 132, primary 24 x 80
  TELNETDEVICE 3279-5-e NSX32705          ; 27 x 132, primary 24 x 80
  TELNETDEVICE LINEMODE INTERACT          ; linemode terminals
  TELNETDEVICE DYNAMIC          ,D4C32XX3 ; tbd by application (QUERY)
  TELNETDEVICE 3287-1           ,SCS      ; printer            LU1
  DEFAULTLUS
      RA39TN31..RA39TN50
      RA39TP31..RA39TP50  ; printers
  ENDDEFAULTLUS
;
  LUGROUP SPECLU
      RA39TN90..RA39TN99  ; specials
  ENDLUGROUP
;
  PRTGROUP PRINTERS
      RA39TPR8..RA39TPR9  ; printers
  ENDPRTGROUP
;
  LUMAP SPECLU   9.170.3.123
;
  MSG07            ; Error messages will be issued
  LUSESSIONPEND    ; On termination of a Telnet server connection,
;                  ; the user will revert to the DEFAULTAPPL
  USSTCP TELNUST
  LINEMODEAPPL RA03T ; Send all line-mode terminals directly to TSO.
  ALLOWAPPL RA*  ;* DISCONNECTABLE  Allow all users access to TSO
  ALLOWAPPL AD*
  ALLOWAPPL A2*
  ALLOWAPPL FD*
  ALLOWAPPL X6*
  ALLOWAPPL X7*
;
ENDVTAM
```

D.2.9 OMPROUTE configuration file for TCPIPC stack at RA39 image - OM39CCFG

```
;-------------------------------------------------------------------
;TCPIP.TCPPARMS.R2617(OM39CCFG) SYSPLEX DISTRIBUTOR RA39 - SANDRAEF
;-------------------------------------------------------------------
Area    Area_Number=0.0.0.0
              Stub_Area=NO
              Authentication_type=None;
RouterID=172.16.102.39;
ROUTESA_CONFIG ENABLED=YES
              COMMUNITY="MVSsubagent";
OSPF_Interface IP_Address=172.16.102.39
              Name=M392216B
              Cost0=3
              Subnet_mask=255.255.255.0
              MTU=32768;
OSPF_Interface IP_Address=9.24.105.73
              Name=EN139
              Cost0=6
              Subnet_mask=255.255.255.0
              MTU=32768;
OSPF_Interface IP_Address=172.16.251.*
              Subnet_mask=255.255.255.0
              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;
AS_Boundary_routing
              Import_Direct_Routes=YES;
```

Appendix E. Dump of T28ATCP name server - single-path network

```
$ORIGIN itso.ral.ibm.com.
ralplex1       IN     SOA    mvs03a.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
ralplex1       IN     CNAME  ralplex1.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     SOA    mvs03a.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     PTR    mvs03a.ralplex1.itso.ral.ibm.com.;Cl=4
$ORIGIN 101.16.172.in-addr.arpa.
28             IN     PTR    mvs28a.ralplex1.itso.ral.ibm.com.;Cl=4
$ORIGIN 232.16.172.in-addr.arpa.
39             IN     PTR    mvs39a.ralplex1.itso.ral.ibm.com.;Cl=4
$ORIGIN 102.16.172.in-addr.arpa.
39             IN     PTR    mvs39a.ralplex1.itso.ral.ibm.com.;Cl=4
$ORIGIN 233.16.172.in-addr.arpa.
28             IN     PTR    mvs28a.ralplex1.itso.ral.ibm.com.;Cl=4
39             IN     PTR    mvs39a.ralplex1.itso.ral.ibm.com.;Cl=4
3              IN     PTR    mvs03a.ralplex1.itso.ral.ibm.com.;Cl=4
```

```
$ORIGIN 250.16.172.in-addr.arpa.
3              IN      PTR     mvs03a.ralplex1.itso.ral.ibm.com.;Cl=4
$ORIGIN 251.16.172.in-addr.arpa.
5              IN      PTR     mvs03c.ralplex1.itso.ral.ibm.com.;Cl=4
$ORIGIN 252.16.172.in-addr.arpa.
28             IN      PTR     mvs28a.ralplex1.itso.ral.ibm.com.;Cl=4
$ORIGIN 0.127.in-addr.arpa.
0              IN      SOA     mvs03a.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      PTR     loopback.;Cl=5
$ORIGIN 24.9.in-addr.arpa.
104            IN      SOA     mvs03a.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      PTR     mvs28a.ralplex1.itso.ral.ibm.com.;Cl=5
149            IN      PTR     mvs39a.ralplex1.itso.ral.ibm.com.;Cl=5
113            IN      PTR     mvs03a.ralplex1.itso.ral.ibm.com.;Cl=5
; --- Hints ---
$ORIGIN .
.      3600    IN      NS      mvs25o.buddha.ral.ibm.com.;Cl=0
$ORIGIN buddha.ral.ibm.com.
mvs25o  3600   IN      A       9.24.104.125;Cl=0
```

Appendix F. Special notices

This publication is intended to help system administrators to understand and implement TCP/IP solutions within a sysplex. The information in this publication is not intended as the specification of any programming interfaces that are provided by IBM Communications Server for OS/390 IP. See the PUBLICATIONS section of the IBM Programming Announcement for IBM Communications Server for OS/390 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

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

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

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

Appendix G. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

G.1 IBM Redbooks

For information on ordering these publications see “How to get IBM Redbooks” on page 319.

- *IBM Communications Server for OS/390 V2R10 TCP/IP Implementation Guide Volume 1: Configuration and Routing*, SG24-5227
- *IBM Communications Server for OS/390 V2R10 TCP/IP Implementation Guide Volume 2: UNIX Applications*, SG24-5228
- *OS/390 eNetwork Communications Server V2R7 TCP/IP Implementation Guide Volume 3: MVS Applications*, SG24-5229
- *Accessing OpenEdition from the Internet*, SG24-4721
- *TCP/IP Tutorial and Technical Overview,* GG24-3376
- *IP Network Design Guide,* SG24-2580
- *Stay Cool on OS/390: Installing Firewall Technology,* SG24-2046
- *Secureway Communications Server for OS/390 V2R8 TCP/IP: Guide to Enhancements,* SG24-5631
- *Security in OS/390-based TCP/IP Networks,* SG24-5383

G.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at **ibm.com**/redbooks for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	**Collection Kit Number**
IBM System/390 Redbooks Collection	SK2T-2177
IBM Networking Redbooks Collection	SK2T-6022
IBM Transaction Processing and Data Management Redbooks Collection	SK2T-8038
IBM Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
IBM AS/400 Redbooks Collection	SK2T-2849
IBM Netfinity Hardware and Software Redbooks Collection	SK2T-8046
IBM RS/6000 Redbooks Collection	SK2T-8043
IBM Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

G.3 Other resources

These publications are also relevant as further information sources:

- *OS/390 UNIX System Services Command Reference*, SC28-1892
- *OS/390 C/C++ Programming Guide*, SC09-2362
- *OS/390 MVS Programming: Workload Management Services*, GC28-1773

- *OS/390 Parallel Sysplex Overview: Introducing Data Sharing and Parallelism in a Sysplex*, GC28-1860
- *OS/390 MVS Planning: Workload Management*, GC28-1761
- *OS/390 UNIX System Services Planning*, SC28-1890
- *OS/390 IBM Communications Server: IP User's Guide,* GC31-8514
- *OS/390 IBM Communications Server: IP Migration*, SC31-8512
- *OS/390 IBM Communications Server: IP Configuration Guide*, SC31-8725
- *OS/390 IBM Communications Server: IP Configuration Reference*, SC31-8726
- *OS/390 IBM Communications Server: IP Messages Volume 1 (EZA)*, SC31-8517
- *OS/390 IBM Communications Server: IP Messages Volume 2 (EZB)*, SC31-8570
- *OS/390 IBM Communications Server: IP Messages Volume 3 (EZY-EZZ-SNM)*, SC31-8674
- *OS/390 IBM Communications Server: Application Programming Interface Guide*, SC31-8516
- *OS/390 UNIX System Services Planning*, SC28-1890
- *OS/390 IBM Communications Server: IP and SNA Codes*, SC31-8571
- *OS/390 UNIX System Services Planning*, SC28-1890
- *OS/390 UNIX System Services User's Guide*, SC28-1891
- *OS/390 UNIX System Services Messages and Codes*, SC28-1908
- *OS/390 MVS System Messages, Vol. 1 (ABA-ASA)*, GC28-1784
- *OS/390 MVS System Messages, Vol. 2 (ASB-ERB)*, GC28-1785
- *OS/390 MVS System Messages, Vol. 3 (EWX-IEB)*, GC28-1786
- *OS/390 MVS System Messages, Vol. 4 (IEC-IFD)*, GC28-1787
- *OS/390 MVS System Messages, Vol. 5 (IGD-IZP)*, GC28-1788
- *X/Open Portability Guides* (XPG), `http://www.xopen.org/`
- *UNIX for Dummies, 3rd Edition*, SR23-8083
- *S/390 ESCON Channel PCI Adapter User's Guide and Service Information,* SC23-4232
- *DNS and BIND* by Paul Albitz and Cricket Liu, O'Reilly & Associates, Inc., 1997, SR23-8771

G.4 Referenced Web sites

These Web sites are also relevant as further information sources:

- `http://www.s390.ibm.com`
- `http://www.ietf.org`

How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** **ibm.com**/redbooks

  Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

  Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders by e-mail including information from the IBM Redbooks fax order form to:

	e-mail address
In United States or Canada	pubscan@us.ibm.com
Outside North America	Contact information is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at http://w3.ibm.com/ for redbook, residency, and workshop announcements.

IBM Redbooks fax order form

Please send me the following:

Title	**Order Number**	**Quantity**

First name	Last name	
Company		
Address		
City	Postal code	Country
Telephone number	Telefax number	VAT number

☐ Invoice to customer number

☐ Credit card number

Credit card expiration date	Card issued to	Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Index

IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, **please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.**

- Use the online **Contact us** review redbook form found at **ibm.com**/redbooks
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Document Number **Redbook Title**	SG24-5235-02 TCP/IP in a Sysplex
Review	
What other subjects would you like to see IBM Redbooks address?	
Please rate your overall satisfaction:	O Very Good O Good O Average O Poor
Please identify yourself as belonging to one of the following groups:	O Customer O Business Partner O Solution Developer O IBM, Lotus or Tivoli Employee O None of the above
Your email address: The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.	 O Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.
Questions about IBM's privacy policy?	The following link explains how we protect your personal information. **ibm.com**/privacy/yourprivacy/

IBM

Redbooks

TCP/IP in a Sysplex

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

TCP/IP in a Sysplex

Implement load balancing and high availability in your sysplex

Contains the latest technology, including Sysplex Distributor

Includes useful configuration details and scenarios

The sysplex environment provides a unique setting for the creation of high performing application servers. Quite simply, the main goals of a sysplex are high availability and load balancing. That is, the sysplex provides the ability to maintain a service as highly available and the ability to add resources so that service performance can scale with a growing number of client requests.

In this redbook, we include a discussion of three sysplex-specific solutions that help to meet these demands, Sysplex Distributor, Domain Name Service/Workload Manager, and Network Dispatcher. All of these solutions make use of, to some extent, the MVS Workload Manager (WLM). Because of this, we describe the benefits of WLM-aware system solutions and include mechanisms by which we can work with WLM. Additionally, because the sysplex notion of high availability is so closely tied together with the Virtual IP Addressing (VIPA) concept, we discuss the advantages of VIPA and the necessary steps that need to take place when using VIPA. This includes a detailed routing discussion as we deal with VIPAs in the sysplex.

The main focus of this book is Sysplex Distributor, a new function available on sysplex systems as of IBM Communications Server for OS/390 V2R10 IP. Sysplex Distributor is the state of the art technology of achieving efficient load balancing and high availability within the sysplex.

In summary, this redbook will help you design your OS/390-based IP network to gain the maximum benefit from the features available with the sysplex to achieve the high availability and load balancing demands placed on your OS/390 servers.

SG24-5235-02 ISBN 0738421219