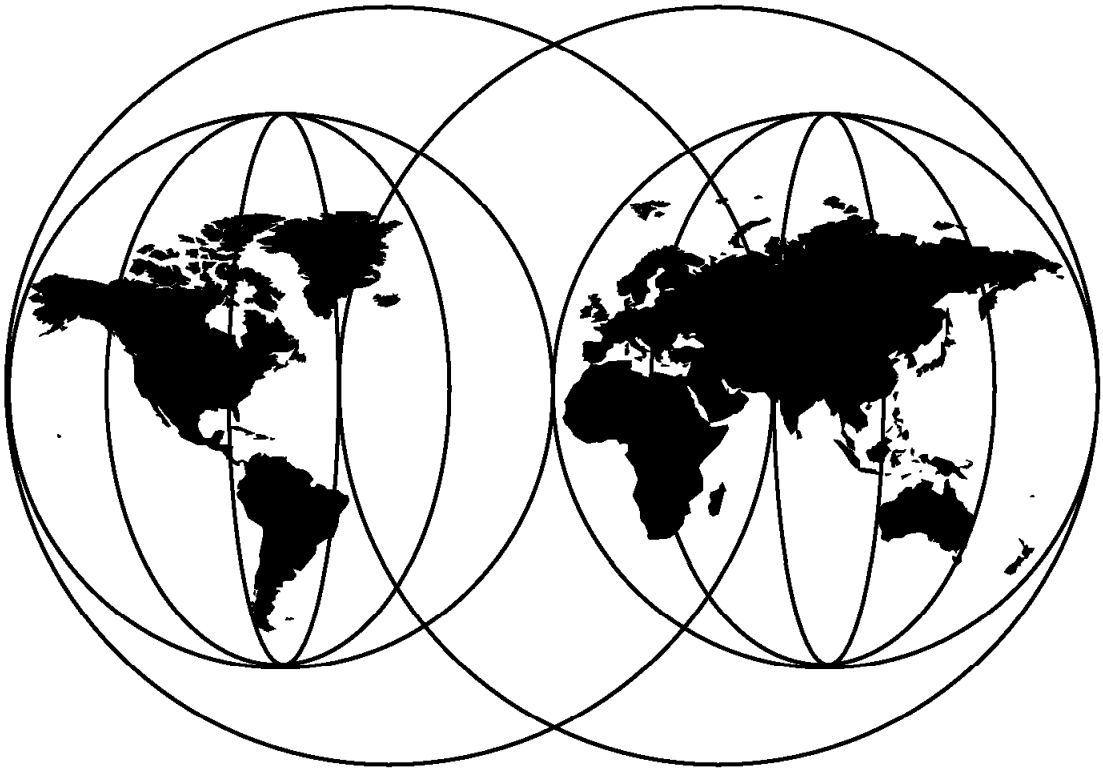
TCP/IP in a Sysplex

Karl Wozabal, Jerzy Buczak, Russell Finn, Morag Hughson, Ulf Jakobsson, Neil Johnston, Shizuka Kato



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-5235-01

International Technical Support Organization

SG24-5235-01

TCP/IP in a Sysplex

June 1999

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix H, “Special Notices” on page 263.

Second Edition (June 1999)

This edition applies to OS/390 SecureWay Communications Server, Version 2 Release 8, program number 5647-A01.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 678
P.O. Box 12195
Research Triangle Park, NC 27709-2195

Contents

Figures

Tables

Preface

The main goals of a sysplex are high availability and high performance. This redbook demonstrates how these goals can be achieved in the particular environment of OS/390 SecureWay Communications Server and the TCP/IP applications that run under it.

We study the way the MVS workload manager interacts with the Domain Name System server and virtual IP addressing to improve load balancing and availability. We investigate how DNS placement and selection of routing methods can influence the way WLM and DNS operate. By contrast, we also look at how the network dispatcher achieves similar objectives to DNS, and compare the two methods of operation to help you choose the best solution. In addition, we describe the many sysplex-related functions introduced to MVS TCP/IP up to CS for OS/390 Release 8.

This redbook will help you design your OS/390-based IP network to gain the maximum benefit from the features available with the sysplex.

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.

Karl Wozabal is a Senior Networking 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, he worked in IBM Austria as a networking support specialist.

Jerzy Buczak is an IT Consultant at the International Technical Support Organization, Raleigh Center. He writes extensively and teaches IBM classes worldwide on SNA and CS for OS/390. Before joining the ITSO in 1996, Jerzy worked for Networking Systems in the UK. He has 18 years′ experience in SNA networking, network management, and a wide variety of product implementations. Jerzy holds an M.A. degree in mathematics from Cambridge University, England.

Russell Finn, Morag Hughson and Neil Johnston are members of the MQSeries development team in Hursley, UK.

Ulf Jakobsson is an Advisory IT Networking Specialist from IBM Sweden. He has 15 years′ experience in mainframe and LAN networking.

Shizuka Kato is an IT Networking Specialist from IBM Japan. She has seven years′ experience in the System/390 networking field, especially in the area of advanced solution verification.

The first edition of this redbook was written by **Karl Wozabal** and the following people:

Garth Madella is an information technology specialist with IBM South Africa. He has 14 years′ experience in the System/390 networking software field.

Matt Nuttall is a technical support specialist with IBM Canada. He has nine years' experience in a System/390 software environment. He holds a degree in Computer Science from the University of British Columbia.

Bill Thomas is a network specialist from IBM Australia. He has 19 years' experience in mainframe and LAN networking, and holds a degree in mathematics and physics from MacQuarie University.

Our thanks are due to the following people who made invaluable contributions to this book:

Jeff Brodd
2216 Development, Research Triangle Park

Alfred Christensen
CS for OS/390 Development, Research Triangle Park

Jonathan Follows
International Technical Support Organization, Raleigh Center

Tatsuhiko Kakimoto
International Technical Support Organization, Raleigh Center

John Parker
IBM United Kingdom

Julie Spitzer
CS for OS/390 Development, Research Triangle Park

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 "ITSO Redbook Evaluation" on page 273 to the fax number shown on the form.
- Use the online evaluation form found at `http://www.redbooks.ibm.com/`
- Send your comments in an Internet note to `redbook@us.ibm.com`

Chapter 1. Introduction to TCP/IP in a Sysplex

This chapter provides a brief overview of the main terms and system components involved in the operation and configuration of CS for OS/390 IP in a sysplex environment. If this chapter is of interest to you, we strongly recommend that you refer to the indicated references when suggested.

This book contains a series of examples detailing the configuration of CS for OS/390 IP in a variety of situations. We have tried to provide a simple introductory network example, followed by examples intended to mimic production environment implementations. The first example, found in Chapter 4, “Basic Sysplex with DNS” on page 21, provides the most detail on how sysplex, workload manager and TCP/IP work together.

The earlier edition of this redbook was based on a CS for OS/390 V2R5 environment. In this updated redbook we have not made any changes to the examples provided in the earlier edition. This applies to the examples in Chapter 4, “Basic Sysplex with DNS” on page 21, Chapter 6, “Sysplex with RIP” on page 69 and Chapter 7, “Sysplex with OROUTED and VIPA” on page 79. 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 V2R5. Later examples used V2R7 or V2R8 of CS for OS/390.

1.1 Sysplex

In the context of this document, 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.

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. In fact, most of the subsystems that run under OS/390 can exploit the coupling facility, so a Parallel Sysplex is always recommended.

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 DASD, provide backup capabilities and distribute workload (load balancing). Unfortunately, it is not possible to share a Hierarchical File System (HFS) across multiple hosts in a network. In a sysplex environment, this can be quite limiting because your HFS is unique on each host in the sysplex.

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

1.2 OS/390 SecureWay Communications Server

OS/390 SecureWay Communications Server (formerly known as OS/390 eNetwork Communications Server) is the communications engine of OS/390. It comprises the 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 it is still possible. We decided to test multiple stacks of TCP/IP, for the reasons given in Chapter 4, “Basic Sysplex with DNS” on page 21.

For a complete list of changes in recent releases of the TCP/IP for MVS product, see *OS/390 eNetwork Communications Server: IP Planning and Migration Guide*, SC31-8512.

1.3 Shared Connections and HPDT

One of the major ways in which the SNA and IP components of CS for OS/390 (formerly VTAM and TCP/IP 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 looks after the following DLCs:

- Multipath Channel Plus (MPC+)
- ATM
- XCF (see 1.6, “Cross-System Coupling Facility” on page 3)
- LAN Channel Station (LCS)
- Channel Data Link Control (CDLC)
- Channel to Channel (CTC)
- HYPERchannel
- Common Link Access to Workstations (CLAW)
- SNALINK
- Same-host, which is also used for X.25 communication

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.

For TCP/IP to use MPC+ or ATM, the appropriate VTAM definitions must be active; these definitions are transport resource list elements (TRLEs). XCF and the TCP/IP exclusive DLCs do not require predefined TRLEs as they are created dynamically when the connection is activated (the device is started).

MPC+ and ATM use a new technique called high-performance data transfer (HPDT) to read and write data. HPDT makes use of multiple seldom-ending channel programs to drive the I/O; it also uses the communications storage manager (CSM) address space to hold data buffers in order to minimize the number of data moves in storage.

1.4 The New DNS Name Server

The Domain Name System (DNS) server delivered with CS for OS/390 V2R5 and above 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.

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. CS for OS/390 IP uses its own interface to WLM that provides the DNS with information allowing it to balance the load across the appropriate TCP/IP servers. Periodic updates received from WLM tell the server which hosts in the sysplex have the most processing resources available.

To take advantage of sysplex load balancing, 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 Chapter 4, “Basic Sysplex with DNS” on page 21.

Note that when we use the term *load balancing* throughout this book, we mean the intelligent distribution of *connections* or *datagrams* among servers, based on the perceived load on those servers. A more accurate term might be *connection optimization*.

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; 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 XCF connectivity 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 up to handle higher workloads without impacting existing systems and their users. XCF dynamics automatically creates device and link definitions and you need define only one new IP address per system.

Aside from actually transferring IP traffic, CS for OS/390 IP uses XCF for communication between the stacks themselves. The dynamic VIPA function (see

3.3, “Virtual IP Addressing (VIPA)” on page 18), for example, uses XCF to coordinate the placement of VIPA addresses within a sysplex.

1.7 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 individual 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 a set of hosts in a network 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 packet 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 9, “Network Dispatcher” on page 111 for a more detailed description of the network dispatcher.

The new DNS name server and the network dispatcher obviously have a common goal, that of load balancing across a group of IP servers. Each is more suited to particular types of traffic, as we describe in the next section.

1.8 Load Balancing and Availability

The major objective of a sysplex is high availability without compromising performance. High availability requires a number of servers providing similar service to their clients, whereas load balancing ensures that such a group of servers can maintain optimum performance. An evenly-spread workload minimizes the number of users affected by the failure of a single server, whereas there is no point in load balancing to an unavailable 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% availability and at the same time provide top performance to the end users when they request server functions.

In a sysplex TCP/IP environment there are various ways of addressing these goals, but first we review some load balancing techniques:

- One technique is to let the users themselves choose a server at random from a number of host names or addresses. When users try to connect to a server they are not aware if the server is available; nor do the users know how well the server will perform when they connect to it. This approach is quite common but very inefficient.

- 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, even the number of connections) on different servers to some extent, before selecting a server.
- 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.
- Finally, you can try to avoid the situation by oversizing, which is very costly.

Of course, you could combine these techniques to some degree.

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 like 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 both of the main approaches available to TCP/IP users in a sysplex to perform load balancing and to accomplish high availability: the DNS name server and the network dispatcher. Both techniques can make use of the MVS workload manager to distribute IP traffic across a number of servers, but they are very different in the approach they take.

1.8.1 DNS Name Server Implementation

The DNS solution is based on the new 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.

Note that, for TCP connections, the best system is determined only at connection setup time. Once the connection is made, the system being used cannot be changed without reinitiating 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 4, “Basic Sysplex with DNS” on page 21, but in essence this is the way it works:

- The servers register with WLM under a group name.
- At regular intervals, DNS asks WLM for its workload measurements.
- The client requests a server by IP *host name*, never the address. This host name is the 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 (group) name.
- DNS responds to the client with the IP address of the most suitable server instance.

All BIND-based name servers use a simple sorting algorithm (by default) and addresses obtained via WLM are no exception. This is more or less the basic round-robin technique. But since WLM also is a what you could call a performance agent, and because the addresses it returns are ordered according to server workload, we get rather more than just 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. The main concern in a sysplex environment is understanding the name server-to-WLM connection.

- No new software needed.

  The prerequisites are there when you have an OS/390 system, it is just a matter of configuration.

- Performance.

  Once the host name is resolved to a server address there is no more involvement from the DNS/WLM load balancing function.

- High availability.

  Secondary name servers can be implemented in a sysplex to fulfill the functions of the primary one should it fail.

- Ease of use.

  Users (clients) are not aware of changes in the location or the IP address of an application server.

Possible drawbacks may be:

- May affect end users. The users may need to learn new names for the generic applications, although the use of aliases can reduce or eliminate this work.
- Time gaps. The name server queries WLM periodically for updated information, by default every 60 seconds. This means that the name server's information can be incorrect in between intervals.
- Users can bypass.

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

- Name servers must behave.

  The algorithm can also be subverted if external name servers do not honor the time-to-live value of zero returned by the sysplex name server. They

may cache the results of a query for a long time, so that their clients keep receiving the same IP address for an extended period.

1.8.2 Network Dispatcher Implementation

The network dispatcher (NDR) technique does not depend on host names, rather on IP addresses. NDR will respond to a special IP address that is known internally (but not propagated externally) by the servers. Clients contact this generic address, and reach the network dispatcher which forwards packets to its chosen server. This time, 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. NDR uses all the information gleaned 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 and that address belongs to NDR. From the server 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 processor complexes 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.

The NDR implementation in our test environment was based on the 2216 router with its MAS software together with WLM on OS/390.

Advantages of the network dispatcher approach include:

- Scalability.

  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 optimum 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 host name).

- Functionality.

  The NDR-supporting hardware (2216, 3746 MAE, Network Utility) has many more functions than just the network dispatcher, as these machines are all full-function multiprotocol routers.

- 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.

Possible disadvantages of NDR may be:

- Extra costs.

  Unless you are using a 2216 or equivalent for multiprotocol routing already there is new hardware to be purchased, procedures to be devised and education to be undergone.

- Performance and capacity.

  Advisors typically poll servers for information every five seconds. This, together with the fact that each packet may (depending on the configuration) require one extra hop on the IP network, 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.

- Design Flexibility.

  The network dispatcher must be located precisely one hop away from the servers. The IP routing mechanism prevents any other configuration from being used with this technique. This means that, in a sysplex, all images must be directly connected to the network dispatcher box(es). Adding new images, which is normally possible without any new network connections or definitions, will now require connections and definitions in the network dispatcher.

  Because of the NDR design, if both NDR and the servers have interfaces on the same subnetwork, no client may be connected to that subnetwork as well. The use of cluster addresses that are known to both NDR and servers prevents this scenario from working.

- 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.

- Incompatibility with IPSec and VPN.

  IPSec 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 environment we are describing here that is probably not an issue since the network dispatcher will usually be in a secure environment.

The network dispatcher, like the sysplex DNS name server, can learn the performance and availability characteristics of the servers to which it directs traffic. Like the DNS, it can be configured (by means of redundancy) to provide very high availability. In very general terms, the circumstances under which each approach should be used are:

- If you have lots of short connections (UDP or Web access), use network dispatcher because it does not require name resolution every time. However, be aware that:
  - OSA with EMIF will not work.
  - IPSec will not work all the way to the server.
  - The network dispatcher must be just one hop from *every* server in the cluster.

- If your 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. Domain Name System Overview

This chapter provides a very brief overview of the Domain Name System (DNS, which shares its acronym with the DNS name server). 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 (O'Reilly & Associates, Inc., 1997, SR23-8771). If you are going to implement a DNS, we strongly recommend you to refer to such texts on the subject for more complete descriptions.

2.1 Why DNS?

The TCP/IP applications refer to host computers by their IP addresses. IP addresses are numeric, in the format `nnn.nnn.nnn.nnn` where nnn can range from 0 to 255 (with a few exceptions). The major drawback of this system is that, for most people, numbers are difficult to remember. As a result, today's IP-based networks use a mapping of host names to host numbers or addresses. The obvious advantage of this name-to-IP address mapping is that we can assign easy 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.

2.2 What Is the Domain Name System?

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*.

The DNS is similar in concept to a hierarchical file system. All levels of directories in a file system begin with a root directory. All levels of domains in the DNS also begin with a root domain. Instead of separating each level of domain with a slash (/), the DNS uses a dot (.).

2.2.1 Controlling the Names

The uniqueness of host names within a domain is managed in a similar fashion to the way file names are used within a directory. For example, the files `/bin/matt` and `/sbin/matt` are obviously different. The DNS uses the same principle, but the root directory is listed on the far right, and successive subdomains (equivalent to successive subdirectories) are listed from right to left. For example, if we have an address such as `buddha.ral.ibm.com`, our highest (or closest to the root) domain is `com`. Note that the root domain is represented by a dot, just as on a file system the root directory is a slash. The same address could correctly be written as `buddha.ral.ibm.com.`. Often we leave the final dot out of the address, but you will see situations later in this text where it becomes very important. The next subdomain is `ibm`, and we continue down to the lowest subdomain of `buddha`.

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

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

```
mvs03a     IN    A   172.16.250.3
```

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

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

2.2.2 Finding an Address

Hosts on a network must be configured to look for a DNS server when a host name is used instead of an address. The request for name resolution is handed to a resolver routine. The resolver routine will have an address or list of addresses that 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 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.

2.2.3 Reverse Mappings

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

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

2.2.4 Primary and Secondary

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 server for a particular zone. To do this, we provide the name of the zone and the address of the primary 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 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.

2.3 DNS Implementation

OS/390 SecureWay Communications Server 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 has been available in prior releases of TCP/IP that uses DNS files stored as DB2 data. 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 OS/390 SecureWay Communications Server.

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.

2.4 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 *etcnamed.boot*.
2. A data file or zone file that maps host names to IP addresses for specific domains. We refer to this zone file as the *forward domain* file, 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 host names for IP networks. We refer to this zone file as the *reverse domain* file, or the in-addr.arpa file; it conventionally uses a suffix of *rev*, as in named.rev. Again, the file name is usually the zone name, but it can be anything.

Optionally a DNS server may have additional configuration files:

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

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

Chapter 3. Routing Overview

One of the major functions of a serious 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, cheap and expensive connections. The simplest way to connect them all together is to bridge them, but this results in every part of the network receiving all the traffic, and soon the slower links are overloaded and the network grinds to a halt. What is needed, particularly when all these networks are joined together in the world wide 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 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 each other.

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 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 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 a subnetwork address and a host address can be told apart.

Figure 1 on page 16 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

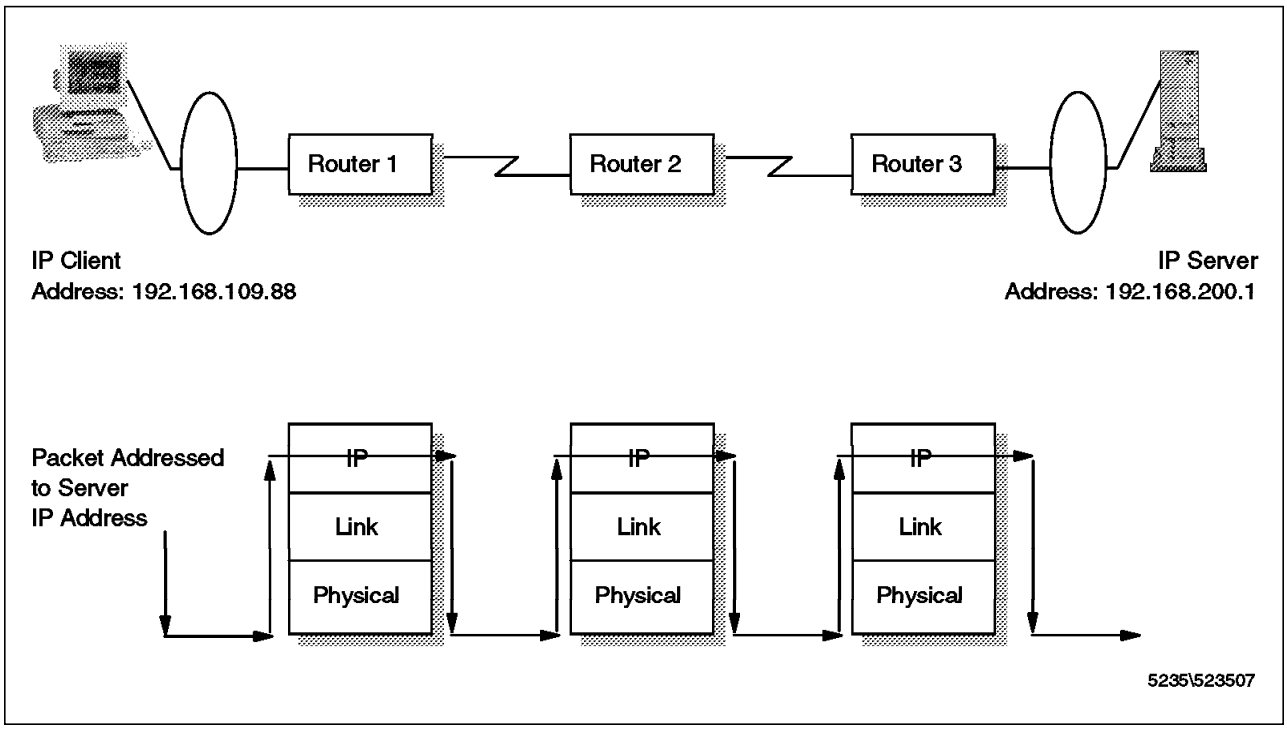


Figure 1. Network Routing Flow

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 changing scenarios. With static routing, however, one has better administrative control over address allocation and resource access.

- 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.

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. OROUTED may be withdrawn from CS for OS/390 in due course, leaving OMPROUTE as the only routing server. 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.

3.1 RIP

Dynamic routing on CS for OS/390 IP supports both RIP Version 1 and Version 2. RIP is an 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 *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.

3.2 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 below), 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.

3.3 Virtual IP Addressing (VIPA)

The purpose of VIPA is 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.

With VIPA, you define a virtual IP address that does not correspond to any physical attachment. 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.

OS/390 V2R8 IP introduced two enhancements to VIPA:

- VIPA backup allows you to define the same VIPA address on multiple TCP/IP stacks in a sysplex. One is defined as primary and the others are defined as secondary. Only the primary one is made known to the IP network. If it fails, then one of the secondary stacks takes its place and the network simply sees a change in the routing tables.
- Dynamic VIPA 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 may move.

These new VIPA enhancements are enabled by the use of XCF to communicate between the TCP/IP stacks.

Chapter 4. Basic Sysplex with DNS

On a sysplex, WLM and TCP/IP can interact to allow an entirely new way of implementing a network on the OS/390 platform. The first section of this chapter is designed to give you a detailed introduction to how these products work together. Later, we provide a detailed description of a simple example network configuration and the applications to exercise it created here at the ITSO.

4.1 Dancing with WLM, DNS and Sysplex

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. As mentioned in the introduction, from a TCP/IP perspective our only requirement is to have WLM configured in goal mode (see Chapter 1, “Introduction to TCP/IP in a Sysplex” on page 1).

A DNS server running on a host in the sysplex can take advantage of WLM′s knowledge and use it to control how often an address for a particular host in the sysplex is returned on a DNS query. When a sysplex name server queries the WLM for information, it is provided with a weight corresponding to the relative resource availability of each participating host in the sysplex. These weights are used by the name server to control the frequency with which an address will be returned for a given host. If a host in the sysplex is relatively busy, its address will not be returned by the server as often as a less busy host′s address. As you might have guessed, this means that you *must* use host names when accessing an application on 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.

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.

- 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.

- Host Name

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

When the name server requests a list of registered applications from WLM, the above information is returned for each one, along with a list of active addresses associated with the host (stack) name; that is, all the interfaces that have been

successfully activated and have an address assigned via the HOME statement in the TCP/IP profile.

For information on how to code the assembler macro, see *OS/390 MVS Programming: Workload Management Services*, GC28-1773. For information on how to code the C function, see the *OS/390 eNetwork Communications Server IP Configuration*, SC31-8513.

Several applications shipped with CS for OS/390 IP are able to register with WLM. For example, the CICS listener, the TN3270 server and the FTP server can all be configured to register with WLM. We tested the Telnet and FTP applications, and a server application we wrote ourselves.

4.1.1 How It Works

Before any application can take part in workload balancing (connection optimization), the TCP/IP stack *should* register itself to WLM. It, and only it, can correctly maintain the IP addresses that DNS/WLM will need to resolve a host name to an address. Once the stack has 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, this is what happens (see Figure 2 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 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 weightings. This can be controlled by the -t parameter at startup of the name server task (daemon).

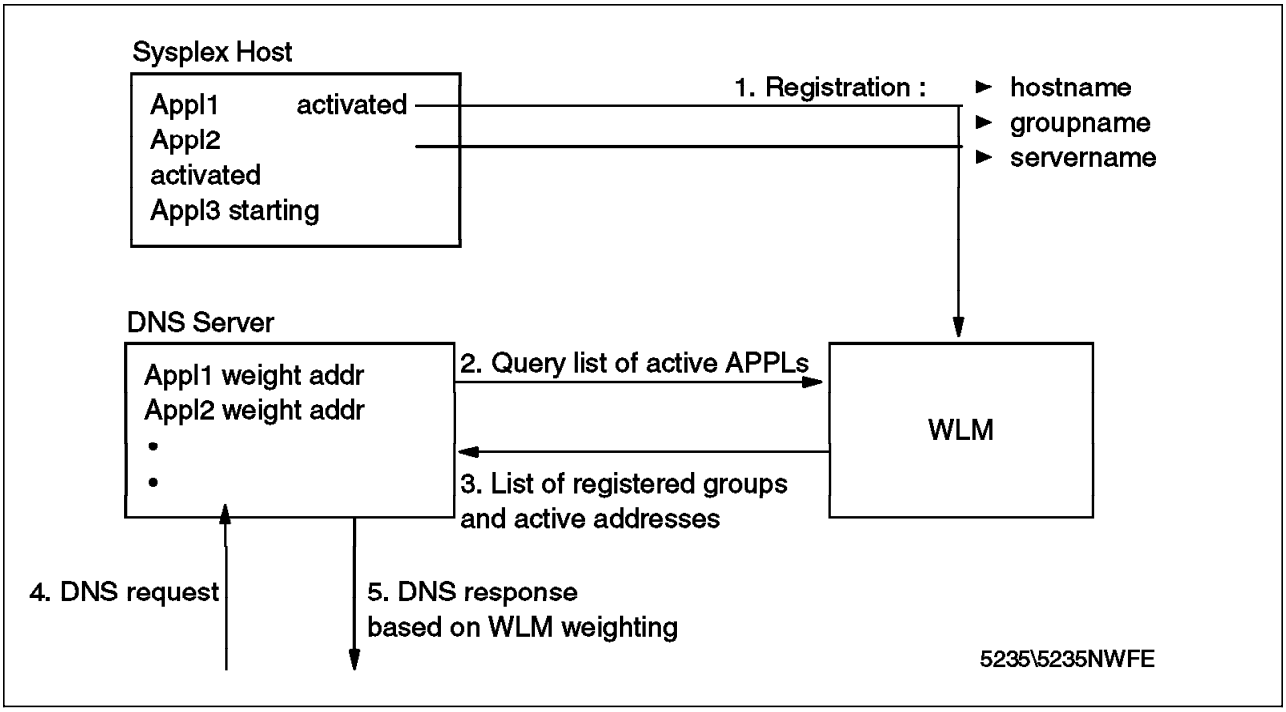


Figure 2. WLM and Name Server Working Together

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 host address or addresses, and then the name server picks up this information and creates a host name entry. The name server dynamically maps an application (actually the group name representing the application) within the sysplex to a host address. While this might be confusing at first, it makes great sense. If an application goes down, either WLM will be notified by a deregistration command, or else it will detect automatically that the address space has terminated. Then WLM will remove the entry from its tables. When the name server next queries WLM, it will no longer be given that application, group and host name. Since we can have multiple applications within a single group, another request for the same group will still succeed if another application registered with the same group is active within the sysplex. The following detailed example will make this all a little clearer.

4.2 Sysplexing in a Basic Network

The key configuration files of our network are outlined in the following sections. The network layout consists of a 3172 running the Interconnect Controller Program (ICP) with LAN Channel Station (LCS) connections to all the hosts in the sysplex and also to MVS25O.

We opted to configure two stacks on each host, even though this does not necessarily provide any processing advantages. The reason we did this was to configure an FTP server that defaulted to the HFS and another FTP server that defaulted to the MVS file system on each host. By associating each FTP server with a different stack, they were both able to listen on port 21. Each FTP server

registers with a different group name (either FTPOERAL or FTPRAL). You could implement a similar capability for Telnet by running INETD configured for Telnet on one stack, and MVS Telnet on the other. Both INETD and Telnet would be able to listen on port 23 if they were associated with different stacks. We did not implement this scenario, primarily because INETD is not yet able to register an application with WLM.

The following diagram (Figure 3) outlines the topology of the network:

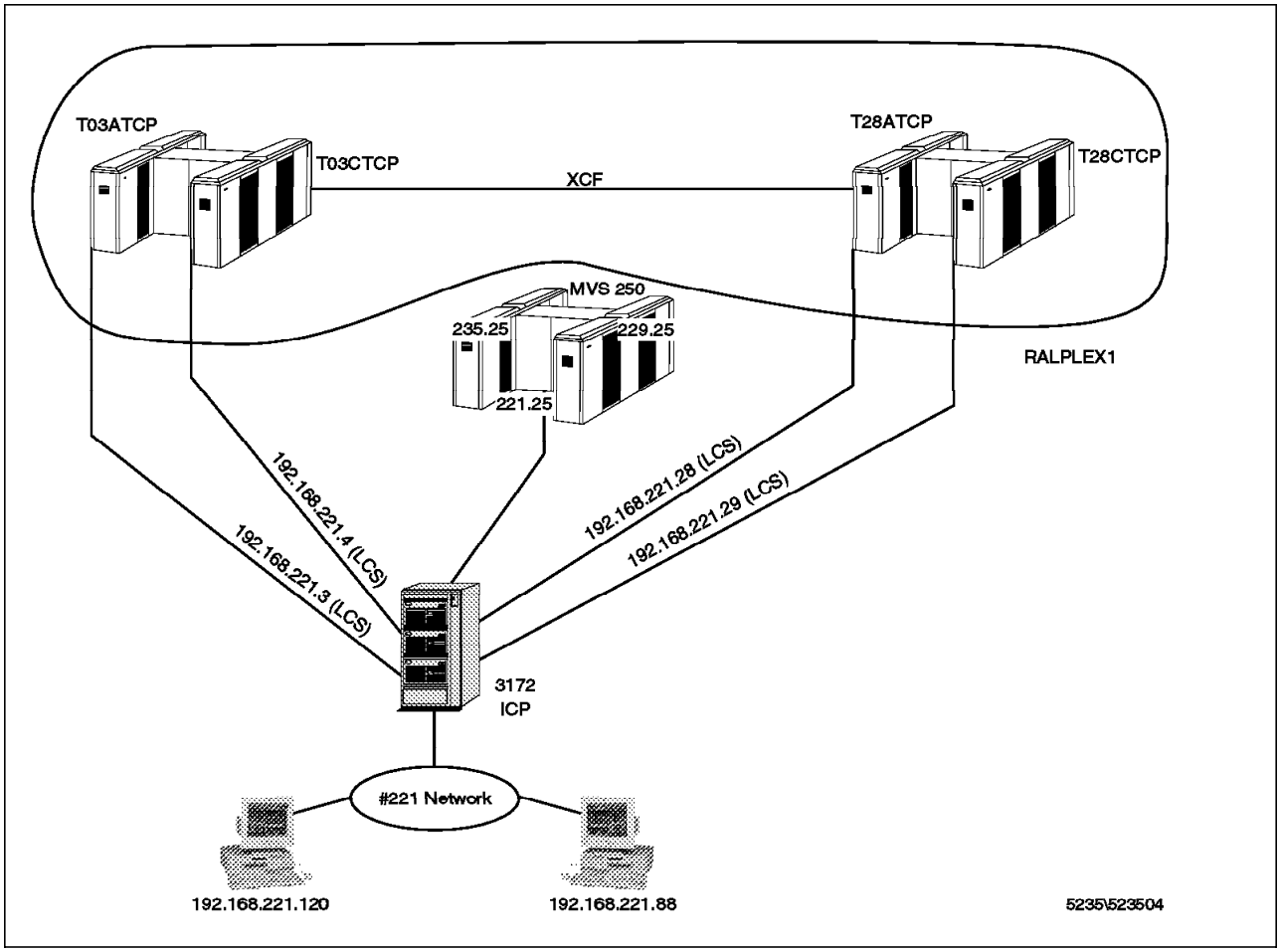


Figure 3. The Single-Path Network

4.2.1 Search Order and Configuration Files for the TCP/IP Stack

Two configuration files are used by the TCP/IP stack: PROFILE.TCPIP and TCPIP.DATA. PROFILE.TCPIP is used for configuration of the TCP/IP stack only. TCPIP.DATA is used during configuration of the TCP/IP stack as well as other applications.

PROFILE.TCPIP Search Order: The documented search order used by the TCP/IP stack to find PROFILE.TCPIP is as follows:

- The Profile DD statement "//PROFILE DD DSN=datasetname"
- Jobname.nodename.TCPIP
- TCPIP.nodename.TCPIP
- Jobname.PROFILE.TCPIP

- TCPIP.PROFILE.TCPIP

TCPIP.DATA Search Order: The search order used to find TCPIP.DATA by the TCP/IP stack and applications is as follows:

- The MVS data set or HFS file that is identified in an environmental variable called RESOLVER_CONFIG; this variable is passed as a parameter RESOLVER_CONFIG to the TCP/IP or FTP procedure
- /etc/resolv.conf, which resides in the HFS
- The //SYSTCPD DD statement, which can be included in the stack procedure
- Jobname.TCPIP.DATA
- SYS1.TCPPARMS(TCPDATA)
- TCPIP.TCPIP.DATA

For examples of our data files and profiles, see Appendix C, “Configuration Files for Base and RIP Examples” on page 193.

4.2.2 Procedures Used for the Network

This section shows examples of the procedures used for our network. We included parameters (PARM keywords on the JCL) for the setting of the environmental variable called RESOLVER_CONFIG. Figure 4, which is one of the TCP/IP stack procedures on RA03, includes the correct syntax for this parameter:

```
//TO3ATCP  PROC PARMS='CTRACE(CTIEZB01)'
//TCPIP    EXEC PGM=EZBTCPIP,REGION=7500K,TIME=1440,
//  PARM=(&PARMS,'ENVAR("RESOLVER_CONFIG=//''TCP.DATA03A''")',
//      '/')
//STEPLIB   DD DSN=TCPIP.SEZATCP,DISP=SHR
//SYSPRINT DD SYSOUT=A,DCB=(RECFM=FB,LRECL=137,BLKSIZE=137)
//ALGPRINT DD SYSOUT=A,DCB=(RECFM=FB,LRECL=137,BLKSIZE=137)
//SYSOUT   DD SYSOUT=A,DCB=(RECFM=FB,LRECL=137,BLKSIZE=137)
//CEEDUMP  DD SYSOUT=*,DCB=(RECFM=FB,LRECL=137,BLKSIZE=137)
//SYSERROR DD SYSOUT=A
//PROFILE  DD DISP=SHR,DSN=TCP.TCPPARMS(PROF03A)
```

Figure 4. Procedure for T03ATCP

Why the need for a RESOLVER_CONFIG variable at all? This variable controls the search order for the resolver routine that runs in the OS/390 UNIX System Services environment (the LE resolver). The TCP/IP stack is now a USS application, so setting this variable is all that should be required to direct the resolver to find the correct resolver configuration data. In TCP/IP V3R2, the stack was still an MVS application, and the SYSTCPD DD would have worked fine. But, if you glance again at our search order above, if RESOLVER_CONFIG has not been set, the next data set we would check is our /etc/resolv.conf. Since we are running two separate stacks on a single host, it would be impossible to allow this file to be used. A SYSTCPD DD card comes next in the search order, and is hence of no advantage. We could have renamed /etc/resolv.conf, but then any USS client applications (for example, OPING or ONETSTAT) would not be able to find /etc/resolv.conf.

If you are familiar with UNIX or AIX, you will have realized that we could have set RESOLVER_CONFIG in the profile. For example, a simple export

RESOLVER_CONFIG=/etc/resolv.conf.mvs03a would have allowed us to remove or rename /etc/resolv.conf. TCP/IP started tasks would never find this file, and SYSTCPD would be used correctly. This is an alternative solution to setting RESOLVER_CONFIG.

Finally, you have probably noted that only one resolver configuration can be accessible at a time from inside the native USS environment. You could use the export RESOLVER_CONFIG command to change what stack you would like a TCP/IP USS client to access. We coded our /etc/resolv.conf files to point always to our "A" stacks, T03ATCP and T28ATCP.

The procedure in Figure 4 on page 25 was used also for our T03CTCP, T28ATCP and T28CTCP stacks. The only differences are the job name and RESOLVER_CONFIG values.

Our FTP procedures proved to be a little more complex. We not only wanted to pass our RESOLVER_CONFIG variable, but we also wanted to add the _BPXK_SETIBMOPT_TRANSPORT variable. This variable is used to assign affinity to a certain TCP/IP stack. We are running two FTP servers on each host: one that defaults to the HFS and one that defaults to the MVS file system.

The procedure we had working for all our FTP applications is as follows:

```
//T03AFTP    PROC MODULE='FTPD'
//FTPD   EXEC PGM=&MODULE,REGION=4096K,TIME=NOLIMIT,
//       PARM=('POSIX(ON) ALL31(ON)',
//             'ENVAR("_BPXK_SETIBMOPT_TRANSPORT=T03ATCP","RESOLVER_CONX
//             FIG=//''TCP.DATA03A''")',',')
//CEEDUMP  DD SYSOUT=*
//SYSFTPD DD DISP=SHR,DSN=TCP.TCPPARMS(FDATA03A)
```

Figure 5. Procedure for T03AFTP

Setting the environment variable _BPXK_SETIBMOPT_TRANSPORT creates an affinity with a particular TCP/IP stack when running multiple stacks. In the above example affinity is set for the T03ATCP stack. Why affinity? Without this variable, the FTP server would accept requests from either TCP/IP stack. Since one of our FTP servers defaults to the HFS, we need to associate one FTP to a specific group and address combination (in other words, a stack). As a result, the group FTPOERAL will register only with addresses from the "C" stacks (T03CTCP and T28CTCP), while the FTPRAL group will register only with addresses from the "A" stacks (T03ATCP and T28ATCP).

The procedure in Figure 5 was used also for our T03CTCP, T28ATCP and T28CTCP stacks. Of course, the ENVAR variables, the job name and SYSFTPD values would all be different. For information on the data files used for FTP, see Appendix C, "Configuration Files for Base and RIP Examples" on page 193.

The procedure we used for the T03ATCP name server is shown in Figure 6 on page 27.

```
//TO3DNS PROC  PARMS='POSIX(ON),ALL31(ON)'
//NAMED    EXEC PGM=EZANSNMD,REGION=OK,TIME=NOLIMIT,
//  PARM=(&PARMS,'ENVAR("RESOLVER_CONFIG=//''TCP.DATAO3A''")',
//      '/')
//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=*
```

Figure 6. Procedure for T03DNS

Again, notice that we have coded RESOLVER_CONFIG in this procedure. When we were testing scenarios, we also found it quite useful to add the -d 11 parameter after the slash (/). You will see more on this in 4.7, “Observation of TCP/IP Sysplex Operation” on page 37. This procedure, when started, will by default search for /etc/named.boot to get its zone configuration information. We discuss the boot file more in 4.4, “The DNS Layout” on page 28.

4.2.3 Data File Definitions for DNS/WLM

There are basically three steps necessary to take full advantage of DNS/WLM connection optimization for this scenario. We need to ensure that the TCP/IP stack itself registers, that TN3270 registers and that FTP registers.

The TCP/IP stack is configured to register with WLM by coding the SYSPLEXROUTING parameter on the IPCONFIG statement in the TCP/IP profile for each stack that is expected to register with WLM. Note that the stack uses the group name TCPIP for its registration, and this name cannot be used by any other registering application. Why register the stack? If you do not register the stack, then TCP/IP will not be able to communicate interface changes to WLM. See C.1, “Profile for T03ATCP Stack - PROF03A” on page 193 for an example. All the other profiles can be found also in Appendix C, “Configuration Files for Base and RIP Examples” on page 193.

To register TN3270, we coded a WLMCLUSTERNAME statement, which must be included within the TELNETPARMS statement. In our profile, we coded the following:

```
WLMCLUSTERNAME TNRAL ENDWLMCLUSTERNAME
```

The name TNRAL is used as the group name when the TN3270 server registers with WLM. We coded this statement for both T03ATCP and T28ATCP. On this statement, it is possible to code more than one group name. This may or may not be of value, since a similar effect can be had by coding multiple CNAME records in the appropriate name server′s data file. We coded a WLMCLUSTERNAME statement for T03ATCP and T28ATCP. See C.3, “Telnet Parameters for T03ATCP Stack - TELN03A” on page 196 and C.12, “Telnet Parameters for T28ATCP Stack - TELN28A” on page 201 for details.

A similar statement had to be coded to enable our FTP server to register. For FTP on T03ATCP and T28ATCP, we coded the following in our FTP.DATA data set:

```
WLMCLUSTERNAME FTPRAL
```

This enables the FTP server to register with the group name FTPRAL.

For T03CTCP and T28CTCP, our FTP.DATA data set had:

```
WLMCLUSTERNAME FTPOERAL
```

For FTP, only one group name can be supplied. For an example of the FTP.DATA file for T03ATCP, see C.6, “FTP Data Parameters for T03ATCP FTP Server” on page 197.

4.3 The Workstation

We used two workstations on our LAN, one running Windows NT and one running OS/2. In each case, we configured the TCP/IP stack to point to the name server on MVS25O, at address 192.168.221.25.

4.4 The DNS Layout

A critical step toward sysplex/WLM implementation is the configuration of the Domain Name System. DNS is certainly not a new concept to IP networks, but it has not had widespread use on the MVS platform. From a DNS perspective, we wanted to mimic a typical customer′s network, so we chose the following configuration for our name servers (see Figure 7):

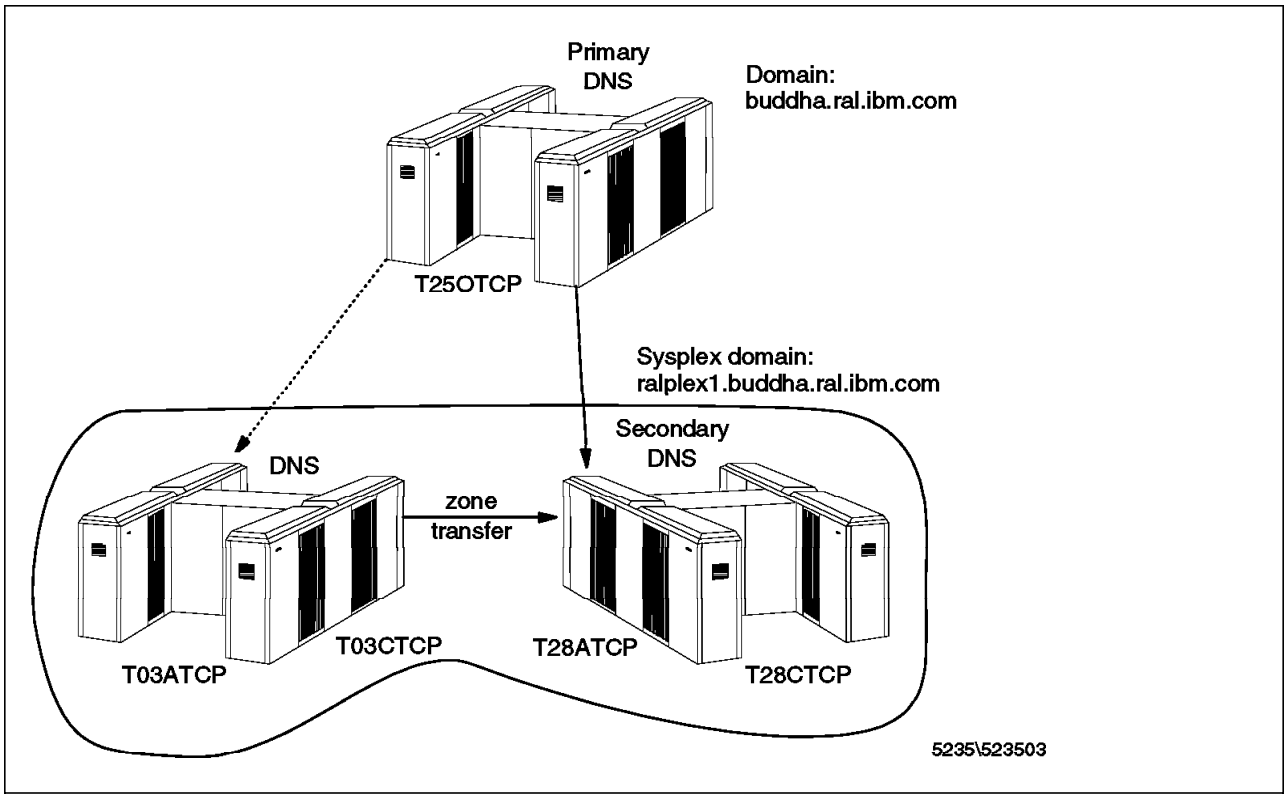


Figure 7. The DNS Layout, Single-Path Network

Our primary name server runs on the T25OTCP stack. It will resolve queries for addresses within the sysplex by acting recursively. We chose not to point directly to one of our sysplex enabled name servers. We expect that most customers will already have an existing name server somewhere on their IP network. By pointing the existing server to the sysplex enabled name server, a customer can minimize the impact of the changes required for sysplex. Of

course, there is no reason why the entire DNS could not be configured using the sysplex-based name servers (we do this in a later example). In this example our name server is outside the sysplex and simply points to our two sysplex name servers.

Although our examples show us configuring a CS for OS/390 IP server as the non-sysplex server, we also configured and tested this same topology using a Windows NT name server. In terms of handling workload balancing, the NT server functioned the same as when we pointed to the name server running on the T25OTCP system.

The sysplex itself is represented by subdomain ralplex1. We have only two separate hosts within the sysplex. Each host runs two stacks, identified as ″A″ and ″C″. The name servers for the ralplex1 domain run on the ″A″ stacks: T03ATCP and T28ATCP. The server on T03ATCP is configured as a primary, while the T28ATCP one is a secondary. This configuration allows us to have a name server on both of our MVS images. The sysplex can continue to function should T03ATCP or T28ATCP, or one of the hosts in the sysplex, become unavailable.

Note that the zone transfers from the name server on T03ATCP to the server on T28ATCP do not include any of the resource records representing our registered groups. A name server running on the sysplex is able to query the WLM running on its own host. Since WLM shares information about registered applications, each copy of WLM will provide complete information about all registered applications within the entire sysplex.

4.4.1 DNS Configuration, T25OTCP

In order to have a solid understanding of how sysplex/WLM worked in a network environment, our initial testing was done using hard coded (static) network routes.

For the MVS25O server, the name server′s boot file was as coded in Figure 8. The three primary statements correspond to the forward mapping file, the reverse mapping file, and the loopback reverse mapping file, respectively. There is no cache statement in this boot file. Therefore, we will never send iterative queries to a root name server. We have no intention of enabling this DNS to work outside our own test domain, so there is no need for us to ever attempt to resolve unexpected domain name resolution requests.

```
;
;    /etc/named.boot for T25OTCP
;
;  TYPE        DOMAIN                          FILE OR HOST
;
directory      /etc/dnsdata
;
primary      buddha.ral.ibm.com                 buddha.for
primary      168.192.in-addr.arpa               buddha.rev
primary      0.0.127.in-addr.arpa               mvs25o.lbk
```

Figure 8. T25OTCP DNS Boot File, Single-Path Network

The contents of the forward mapping data file can be found in Figure 9 on page 30. Our intention here is to provide a reasonably simple example for users

new to sysplex/WLM with TCP/IP. There are no outside networks accessible: everything is on a single ICP-attached LAN (the 192.168.221 network). The only hosts we have defined are the previously mentioned MVS TCP/IP stacks, and a test PC attached to the LAN called thatpc. This is the only name entry we entered for a workstation on the LAN. If you wanted to provide mappings for all of your workstations on the LAN, this is the place where you would most likely want to list them.

```
;
;    /etc/dnsdata/buddha.for for T25OTCP
;
$ORIGIN buddha.ral.ibm.com.
@ IN      SOA     mvs25o.buddha.ral.ibm.com. garthm@mvs03a  (
              1998051901  ; 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      mvs25o
mvs25o          IN      A       192.168.221.25
localhost       IN      A       127.0.0.1
thatpc          IN      A       192.168.221.88
mvs03a          IN      A       192.168.221.3
mvs03c          IN      A       192.168.221.4
mvs28a          IN      A       192.168.221.28
mvs28c          IN      A       192.168.221.29
TNRAL           IN      CNAME   tnral.ralplex1.buddha.ral.ibm.com.
FTPRAL          IN      CNAME   ftpral.ralplex1.buddha.ral.ibm.com.
FTPOERAL        IN      CNAME   ftpoeral.ralplex1.buddha.ral.ibm.com.
mvs03a.ralplex1 IN      A       192.168.221.3
mvs28a.ralplex1 IN      A       192.168.221.28
ralplex1        IN      NS      mvs28a.ralplex1.buddha.ral.ibm.com.
ralplex1        IN      NS      mvs03a.ralplex1.buddha.ral.ibm.com.
```

Figure 9. T25OTCP Forward File, Single-Path Network

The CNAME (alias) records in Figure 9 are very important. Because a sysplex is implemented as its own subdomain (ralplex1.buddha.ral.ibm.com), we need an easier method of finding these hosts than typing in the canonical name (complete host and domain name). These alias records allow a user to reference a sysplex host via the group names tnral, ftpral, or ftpoeral. This means, of course, that users are going to have to use new names for accessing their usual hosts on the network. There are no restrictions on the name you choose for the alias. The names we have chosen above include information about the application the alias represents. Remember, over on the sysplex, the server is creating resource records that represent a group name, which represents registered application(s) on the host. To access one of these applications, the CNAME record allows an end user simply to type: `telnet tnral` or `ftp ftpral` or `ftp ftpoeral` to reach the desired host application somewhere in the sysplex.

Note that CNAME records are part of BIND/DNS functionality. We take advantage of this record type only to simplify our access to sysplex hosts. These records are not actually required (the canonical name can always be used). The CNAME records make life easier for end users. When most resolver routines send a query to a name server, they append the local domain to the

query. The only exception is if a dot has been included on the end of the host name, in which case nothing will be appended. For example, on our network, if a user types `ping tnral` from a host in the `buddha.ral.ibm.com` domain (which would be any host on our 192.168.221 LAN), the resolver will send the canonical name `tnral.buddha.ral.ibm.com.` to the name server. But, our `tnral` group is actually in the `ralplex1` subdomain. The CNAME records tell the name server to substitute `ralplex1.buddha.ral.ibm.com.` for the actual local domain name `buddha.ral.ibm.com.`. After the name server performs this substitution, it will recognize that this is a different domain, and we will send the request to the first host for the `ralplex1` subdomain: `mvs28a.ralplex1.buddha.ral.ibm.com.`

This is not the only way to solve this domain name problem. Many TCP/IP implementations (OS/2, Windows 95 and NT, for example) allow a list of alternate domains to be tried in event of the local domain failing. By adding the `ralplex1.buddha.ral.ibm.com` subdomain to your local domain list (after, of course, the `buddha.ral.ibm.com` domain), you could eliminate the need for the CNAME records. But, if you code the aliases in the name server, you don't need to add the sysplex domain to all of the hosts in your network.

Some of our results in this area did not match exactly what we expected to see after reading the configuration guide. We never saw any "application instances" registered, even in a dump of the name server itself (taken via the `kill -int PID` command). Nor were we ever able to ping an "application instance". But, what we did see was even more interesting: we were always able to ping the combination of `hostname.groupname`. In our scenario, this meant that `ping mvs03a.tnral.ralplex1.buddha.ral.ibm.com` was a valid ping when we had `tnral` registered! The answer to this is that we actually did ping the application instance. The Telnet and FTP applications use the host name as the instance for registration.

Finally, in the last four entries of Figure 9 on page 30, we have the A records to allow the T25OTCP stack to resolve the names for the name servers. The NS records, of course, point to our two sysplex name servers. When a request for a name within the `ralplex1` domain comes to this name server, we will try the first server listed. If there is no response from the first name server, the server will then try the next one. This provides us with our backup scenario. We pointed to the secondary name server, T28ATCP, first rather than the primary T03ATCP server in order to try to balance the overall load on the servers.

```
;
;    /etc/dnsdata/buddha.rev for T25OTCP
;
@            IN    SOA     mvs25o.buddha.ral.ibm.com. garthm. (
              1998051901  ; 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      mvs25o.buddha.ral.ibm.com.
25.221       IN  PTR     mvs25o.buddha.ral.ibm.com.
3.221        IN  PTR     mvs03a.buddha.ral.ibm.com.
4.221        IN  PTR     mvs03c.buddha.ral.ibm.com.
28.221       IN  PTR     mvs28a.buddha.ral.ibm.com.
29.221       IN  PTR     mvs28c.buddha.ral.ibm.com.
```

Figure 10. T25OTCP Reverse File, Single-Path Network

Figure 10 shows our reverse (sometimes referred to as in-addr.arpa data file). It is used only for mapping an address to a host name, and so it is not actually used for DNS sysplex/WLM functionality. But it is an important part of DNS configuration, and the contents for our scenario are listed in Figure 10. Some resolver routines (for example, NSLOOKUP's resolver routines) will request a reverse lookup on a name server's address before they will send a request to that name server. As a result, you should make sure that you have coded this reverse mapping file.

Finally, Figure 11 is the loopback reverse mapping data file. This mapping is not necessary for any sysplex/WLM functionality, but it is shown to complete the picture of our DNS data files.

```
;
;    /etc/dnsdata/mvs25o.lbk for T25OTCP
;
@   IN      SOA     mvs25o.buddha.ral.ibm.com. garthm@mvs03a  (
              1998051901  ; 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      mvs25o.buddha.ral.ibm.com.
1               IN      PTR     loopback.
```

Figure 11. T25OTCP Loopback Reverse Mapping File, Single-Path Network

4.4.2 DNS Configuration, T03ATCP

The T03ATCP name server functions as a primary server for the ralplex1 subdomain (that is, the sysplex subdomain). The boot file is detailed in Figure 12.

```
;
;    /etc/named.boot for T03ATCP
;
;  TYPE      DOMAIN                    HOST        FILE
;
directory    /etc/dnsdata
;
primary    ralplex1.buddha.ral.ibm.com            ralplex1.for cluster
primary    168.192.in-addr.arpa                   ralplex1.rev
primary    0.0.127.in-addr.arpa                   mvs03a.lbk
```

Figure 12. T03ATCP DNS Boot File, Single-Path Network

The primary statement points to /etc/dnsdata/ralplex1.for as the location for the forward mapping data file for this domain. The most important statement in this file is our primary statement for the ralplex1.buddha.ral.ibm.com. In particular, the cluster keyword signifies that this is a sysplex enabled server. With this keyword on a primary record, our name server will query WLM every minute (the default for -t on startup of the server). The next two statements simply point us to our data files for reverse and loopback information. The reverse and loopback files are really business as usual for DNS configuration.

```
;
;  /etc/dnsdata/ralplex1.for for T03ATCP
;
$ORIGIN ralplex1.buddha.ral.ibm.com.
@ IN      SOA     mvs03a.ralplex1.buddha.ral.ibm.com. garthm@mvs03a  (
              1998051901  ; 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
mvs03a          IN      A       192.168.221.3
mvs03c          IN      A       192.168.221.4
mvs28a          IN      A       192.168.221.28
mvs28c          IN      A       192.168.221.29
localhost       IN      A       127.0.0.1
```

Figure 13. T03ATCP Forward File, Single-Path Network

The forward mapping file for T03ATCP is shown in Figure 13. It must contain the address of every interface on every host that you want to be made available as a WLM managed address in the sysplex. If an address is not listed in here, it will never be used on a group name resource record in the name server. Even if we supply the address with our registration of the application to WLM, the server will ignore it if it does not match one of its hard coded host addresses. Note that the canonical names pointed to by the three CNAME records in the forward data file for the name server on T25OTCP are never actually coded into this or any DNS data files. The resource records for `tnral.ralplex1.buddha.ibm.com`, `ftpral.ralplex1.buddha.ibm.com` and `ftpoeral.ralplex1.buddha.ibm.com` are all retrieved into the name server's memory from WLM dynamically. Note as well that the group name, `tnral`, is the name we code in our TCP/IP profile Telnet parameters using the WLMCLUSTERNAME keyword. The same applies to the FTP group names, `ftpral` and `ftpoeral`. We discuss this further when we look at our configuration files later in this chapter; see 4.2.3, "Data File Definitions for DNS/WLM" on page 27.

What happens, though, if a user tries a `ping tnral`? If a TCP/IP stack has registered an active Telnet server with WLM, and the name server has retrieved this information, you will get a PING response from the next available (according to WLM) host in the SYSPLEX. By contrast, if there are no registered Telnet applications the PING will fail with host unknown. This is a subtle concept: instead of pinging a host, we are in some respects pinging an application. In our observations sections, we used the ping command to verify that WLM and the sysplex DNS were actually doing load balancing.

```
;
;    /etc/dnsdata/ralplex1.rev for T03ATCP
;
@     IN    SOA     mvs03a.ralplex1.buddha.ral.ibm.com. garthm@mvs03a (
               1998051901  ; 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.buddha.ral.ibm.com.
3.221         IN  PTR     mvs03a.ralplex1.buddha.ral.ibm.com.
4.221         IN  PTR     mvs03c.ralplex1.buddha.ral.ibm.com.
28.221        IN  PTR     mvs28a.ralplex1.buddha.ral.ibm.com.
29.221        IN  PTR     mvs28c.ralplex1.buddha.ral.ibm.com.
```

Figure 14. T03ATCP Reverse Mapping File, Single-Path Network

The reverse mapping file for T03ATCP is shown in Figure 14. It simply provides the reverse mappings for the hosts within our sysplex subdomain. Again, if you ever pointed a resolver directly to one of these sysplex name servers, the resolver routine might do a reverse lookup before sending any queries to the name server. If these mappings are not available, the resolver might not go through with the name query.

```
;
;    /etc/dnsdata/mvs03a.lbk for T03ATCP
;
@   IN      SOA     mvs03a.ralplex1.buddha.ral.ibm.com. garthm@mvs03a  (
               1998051901  ; 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.buddha.ral.ibm.com.
1                IN      PTR     loopback.
```

Figure 15. T03ATCP Loopback Reverse Mapping File, Single-Path Network

The loopback file in Figure 15 is again business as usual; nothing specific to DNS/WLM.

Note: T03ATCP is acting as a primary server to T28ATCP. That means the forward and reverse mapping files will be transferred to T28ATCP, and they do not need to be coded for T28ATCP's name server. When we start the DNS on T28ATCP, a zone transfer request is sent to T03ATCP and the files in Figure 13 on page 33 and Figure 14 are sent to T28ATCP.

4.5 DNS Configuration, T28ATCP

Because T28ATCP has a name server configured as secondary, the files are simplified quite a bit. The boot file is shown in Figure 16 on page 35. For the first time, we see a secondary statement in a boot file. This tells the name server that it is a secondary name name server for the sysplex subdomain, ralplex1.buddha.ral.ibm.com. This DNS uses the T03ATCP DNS server at 192.168.221.3 to get its zone information for the forward and reverse mapping

files. At startup, it will send a request to T03ATCP, and receive the mapping files at this time.

Finally, we see the cluster keyword again. Even though this server is functioning as a secondary server to T03ATCP's name server, this server will go directly to the WLM running on this MVS host to get information about registered applications and load balancing. This information will never be sent in a zone transfer.

```
;
;    /etc/named.boot for T28ATCP
;
;  TYPE      DOMAIN                        FILE OR HOST
directory      /etc/dnsdata
;
secondary   ralplex1.buddha.ral.ibm.com   192.168.221.3  fback  cluster
secondary   168.192.in-addr.arpa          192.168.221.3  rback
primary     0.0.127.in-addr.arpa          mvs28a.lbk
```

Figure 16. T28ATCP Boot File, Single-Path Network

In Figure 16, the fback and rback keywords are actually the file names to be used for writing backup copies of transferred zone data. The server will write backup copies of transferred zone data to the directory /etc/dnsdata/. Zone data transfers will occur only when the DNS data files for T03ATCP are updated. See 4.6, "Do Not Forget Your SOA Records" for more details on controlling zone transfers.

The only other file used is the loopback file, shown in Figure 17:

```
;
;  /etc/dnsdata/named.lbk for T28ATCP
;
@   IN      SOA     mvs28a.ralplex1.buddha.ral.ibm.com. garthm@mvs03a  (
              1998051901  ; 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.buddha.ral.ibm.com.
1               IN      PTR     loopback.
```

Figure 17. T28ATCP Loopback Reverse Mapping File, Single-Path Network

4.6 Do Not Forget Your SOA Records

Up to now we not described the series of numbers you see on the Start of Authority (SOA) records in any of our forward, reverse and loopback mapping data files. These numbers can make the difference between a working DNS-based network and a failing one.

The first number you see, 1998051901, is the serial number. This number is entirely controlled by you, the network systems programmer, so who really cares? Well, any secondary name server cares very much. When a zone

transfer occurs, this serial number is passed also to the secondary server (actually, the entire SOA record is passed). The secondary server immediately makes a backup copy of this zone data. In our examples for T28ATCP (see Figure 16), the forward data file will be written to /etc/dnsdata/fback and the reverse data file will be written to /etc/dnsdata/rback. More about how serial numbers and these backup files work in a moment.

The next number is the refresh interval, which is set to 7200 seconds in our examples. Every two hours, our secondary server on T28ATCP will poll the primary server, at T03ATCP, and ask for its SOA record. If the serial number on this SOA record is greater than the serial number on the existing copy of zone data held by T28ATCP, then the name server will request a zone transfer. If it is not, then no zone transfer occurs. What does this mean? Well, if you do some updates to your data files on T03ATCP, and you do not update your serial number, T28ATCP will never receive your changes.

Ah, you think you could simply recycle your server on T28ATCP, and the old zone data would be lost. Not so! Remember the backup files fback and rback? Well, when the server starts up, it asks for the serial number from its primary server. It will compare this serial number to the number in its backup files. If the serial number from the primary is not greater, it will simply load its zone data from its local backup files. No zone transfer will occur.

Why function this way? Because if both your name servers were to go down, the secondary could still function after a restart, even if the primary never came back online. The moral of the story here is: it is a good idea to increase your serial number each time you update your data files. The alternative is to make sure you delete your backup files before you reboot your secondary name server if you want updates to be zone transferred. In other words, if no backup files exist a zone transfer must take place when we start the secondary server.

When we started a name server with secondary statements and we had no backup file for (as an example) the ralplex1.buddha.ral.ibm.com domain, we saw the following message:

```
BPXF024I (TCPIP3) Jun  2 21:23:19 named 150994959 : EZZ6628I Error 911
getting serial number or serial number = 0, for 'ralplex1.buddha.ral.ibm.com'
```

This message indicates that the name server has queried its primary server for the serial number on the SOA record, compared it to the serial number in the local backup file (/etc/dnsdata/rback, in this case) and found them to be different. In fact, this server failed to find a backup file at all (because rback did not exist), so the serial number defaulted to 0. Thus, this message implies that a zone transfer is going to be necessary, and shortly afterward we found confirmation of this in the message:

```
BPXF024I (TCPIP3) Jun  2 21:23:32 named 150994959 : EZZ6540I Static 921
secondary zone 'ralplex1.buddha.ral.ibm.com' loaded (serial 1998051901)
```

So what about all these other numbers that you see in our forward, reverse and loopback files? The third number is the retry interval. If our refresh request fails for some reason (the primary server is down, for example), we will ask again for the SOA record information at this interval. In our example, this was set to 3600 seconds, or one hour. But, what if the primary server is still down? The fourth number, 604800, tells the name server how long it should remain with unrefreshed zone data. If we go seven days without getting a response from our refresh and subsequent retry attempts, we will cease to be a name server

altogether. This is not a bad idea if we have spent a week as an orphaned secondary name server.

The final number, 3600, tells us that if we send any of these data file entries to another name server, we want this record to be cached at that name server for a maximum time to live of one hour. This number applies to all records in the data file, but it can be overridden by supplying a time to live (TTL) value on the individual resource record entry (second column, between the name or address and the IN keyword).

When the name server in the sysplex adds a dynamic entry it has pulled from WLM, the TTL will be set to 0. Why a value of 0? What happens, for example, when the name server on T25OTCP receives an answer from the name server on T28ATCP, for the name TNRAL? If the TTL was not 0, then this entry would remain in the name server on T25OTCP until the length of time specified elapsed (and then it would be deleted). During that period, the name server on T25OTCP would not query T28ATCP for TNRAL. Instead, it would answer with the address in its own cache. This would reduce the load balancing capabilities of T28ATCP′s name server, since it would not have as many opportunities to provide the most appropriate address.

Most name servers would not use a TTL of 0, since it is far more efficient to allow the entries to be cached. On a busy system, name queries could add quite a lot of traffic. The WLM managed entries are a special case where a TTL of 0 is desired, but is it necessary? Perhaps not. In fact, when we used T25OTCP as our name server location, every name query to T25OTCP caused an iterative query to be sent to T28ATCP. If a value of, say 10 seconds was used, we do not believe that connection optimization would be adversely affected. Certainly, it might not load balance as accurately as when TTL is 0. The advantage of using 10 seconds could be a great reduction in network traffic, depending on the frequency of name server queries.

4.7 Observation of TCP/IP Sysplex Operation

Where are all these definitions leading? We now need to test whether things go as expected. We are using the same basic configuration as documented earlier in this chapter. The configuration diagram is shown in Figure 3 on page 24. Note that this is as simple a configuration as you can get. There are four TCP/IP stacks on two MVS systems and each stack has only one active interface to the LAN. Testing is from one of two workstations on the 192.169.221.0 network (the same subnetwork as the TCP/IP stacks).

Table 1 shows which TCP/IP applications are registered in which generic groups on our sysplex.

Table 1. RALPLEX1 Stack Functions

MVS Name	Stack Name	ICP Address	Telnet TN3270	TNRAL Appl	FTPRAL Appl	FTPOERAL Appl
MVS03	T03ATCP	192.168.221.3	TN3270E	Y	Y	N
MVS03	T03CTCP	192.168.221.4	Telnet	N	N	Y
MVS28	T28ATCP	192.168.221.28	TN3270E	Y	Y	N
MVS28	T28CTCP	192.168.221.29	Telnet	N	N	Y

The theory of sysplex operation indicates that we should be able to balance the load between applications registered with the same 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. Both the TO3ATCP and T28ATCP stacks are configured for TN3270E so each stack has registered TNRAL with its respective Workload Manager using the home address of its 3172 ICP connection.

Earlier in this chapter we described a configuration with three name servers. The primary name server for the sysplex runs on TO3ATCP and a secondary sysplex name server runs on T28ATCP. In many organizations the TCP/IP network was established long before the MVS sysplex so workstations have been configured to use an existing name server for name resolution. In many cases this happened before TCP/IP was installed on the mainframe. In our implementation, this ″existing″ name server resides on a third system, T25ATCP.

We ran our tests using both the sysplex and the MVS25O name servers, and observed some differences in the way they worked. The next sections describe the use of the sysplex name server T28ATCP, while 4.7.3, “Using an External DNS” on page 52 shows what happened when we used an existing external name server.

4.7.1 Information Flow

Figure 18 on page 39 shows a logical overview of the data exchanges between MVS, WLM, a name server and a workstation. In this example, only one name server is shown, although there are two name servers in the complex. Since each of the name servers gets workload information independently from the WLM running on its MVS image, each name server will be aware of all instances of an application in the entire sysplex.

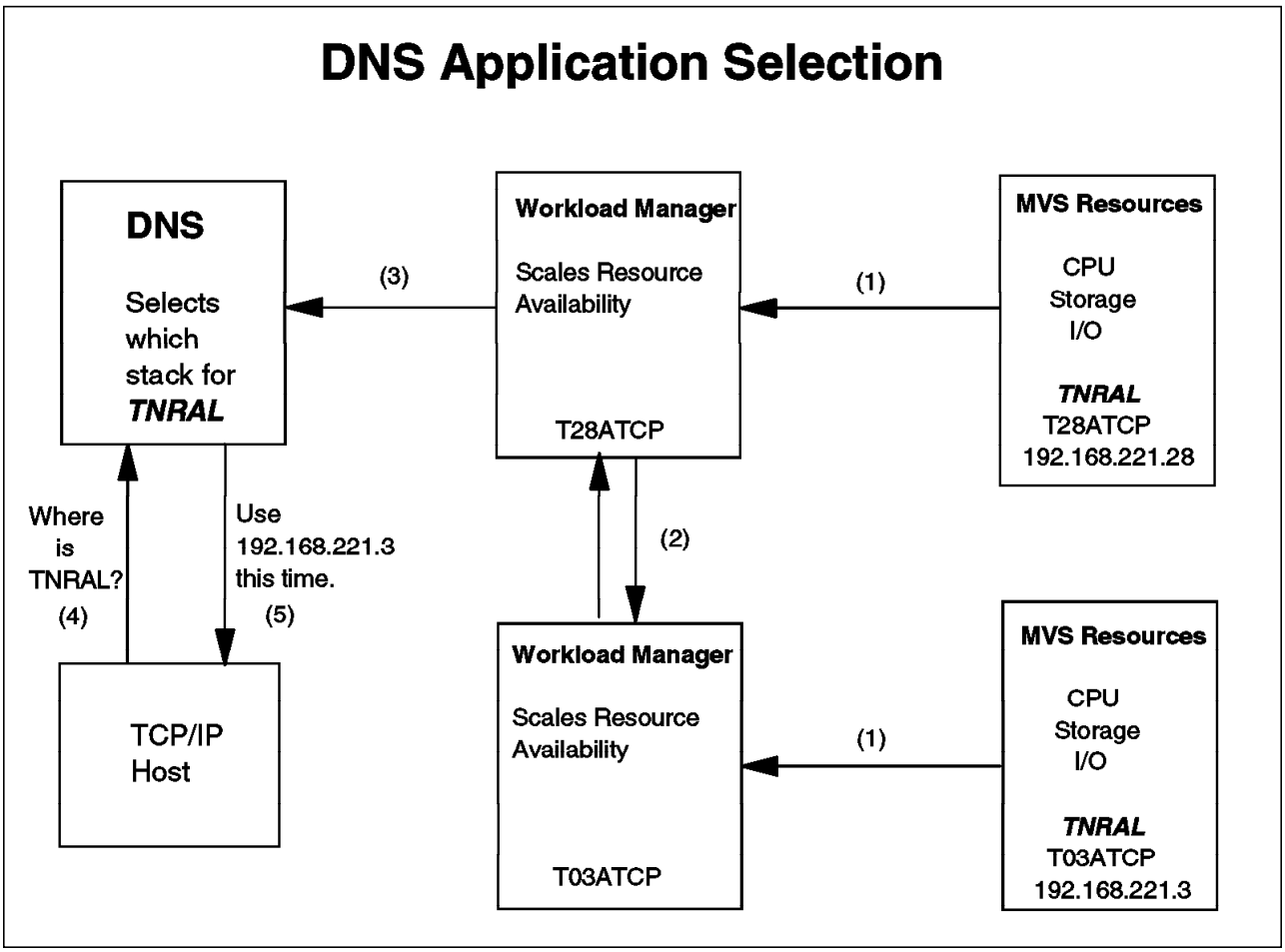


Figure 18. Information Flow in Sysplex Components

4.7.1.1 MVS Resources and WLM

The MVS resources monitored by WLM include system resource availability (CPU, storage and I/O), the TCP/IP stack with its interface address(es) and the application(s) registered with WLM. This information is provided by the various components of the MVS system and is consolidated in WLM. This is shown by the arrow (1) in the figure.

WLM is a component of MVS (as is DNS), but to better illustrate the areas we will be observing, it has been shown as a separate logical entity.

The WLM on each system collects information concerning *its* MVS environment, but note that the two WLMs shown in Figure 18 are connected over an XCF link (2). This connection permits each instance of WLM to be aware of the resources and resource availability for *all* of the MVS images in the sysplex.

4.7.1.2 WLM and DNS

The name server obtains information from WLM (3) concerning sysplex-wide resource availability. This information is provided on a periodic basis (usually one to five minutes) to reflect the current state of resources in the sysplex. The information passed by WLM includes a relative numeric weighting for each MVS image in the sysplex based upon workload and the number of instances of each of the registered applications available on each image. The name server uses the data passed from WLM to build a weighted table for session distribution.

This table contains the registered application name (TNRAL in this example) and the IP addresses pointing to each of the IP interfaces available for the application. The load balancing is provided by the position and frequency of each IP address in the table. Figure 19 shows a logical DNS selection table for an evenly balanced environment:

```
TNRAL
     192.168.221.3
     192.168.221.28
          .
          .
          .
     192.168.221.3
     192.168.221.28
```

Figure 19. An Evenly Balanced Sysplex Name Server Table

4.7.1.3 View from the Workstation

When a workstation requests a session with TNRAL the workstation resolver sends a query to its name server (T28ATCP) requesting the IP address for the application (TNRAL). This is shown as arrow (4) in Figure 18 on page 39. The name server takes the next entry in the TNRAL table and returns the IP address to the workstation. In the sample shown in Figure 19, 192.168.221.3 is the first entry in the table. The workstation would then send a connection request to port 23 on that IP address.

Note: The name server is actually running on the T28ATCP stack (192.168.221.28) and has received its workload information from the WLM on the same image. The end result, however, has been to pass the work on to a *different* MVS image and IP stack in the sysplex.

4.7.2 Observing the Effects of WLM and DNS

So much for what should happen. How do we determine that all of the activities mentioned in 4.7.1, “Information Flow” on page 38 actually occur? Or more importantly, how can we simply test a sysplex environment to see if it is really working? Well, the simplest way to find out which address is returned by the name server is to send a ping to the application. And then another. And then another. And finally tally up the result to see what has occurred. Now this is boring!

In the interests of sanity, we have devised a little REXX EXEC that will send as many pings as you like, keep a tally of the responses for each address returned by the name server, and write it to a file. The source for this EXEC is provided in Appendix B, “REXX EXECs” on page 181 and comes in two flavors. The first, called SYSPLEX, is written for an OS/2 Warp workstation and the second, called SYSPLEXW, is for a Windows 95 or Windows NT workstation. REXX is not provided as part of Windows 95 or NT, so for those environments a separate REXX interpreter is required.

The REXX EXEC is executed by the following line commands:

- For OS/2, `SYSPLEX WLM_registered_name number_of_pings extra_delay`

- For Windows, REXX SYSPLEXW WLM_registered_name number_of_pings extra_delay

In these commands:

- WLM_registered_name is the name used to define the application group to WLM (for example, tnral).
- number_of_pings is just that (default = 10).
- extra_delay is an optional value that inserts a delay (in seconds) in the ping loop (default 0).

The first test of the balancing effects of DNS, sysplex and WLM was run when the systems were lightly loaded. The MVS systems for T03ATCP and T28ATCP were running around 15% CPU utilization. In this environment we expected an even balance of system selection. The workstation was connected directly to the sysplex and all the IP hosts were in the same subnetwork and the same domain buddha.ral.ibm.com. The command issued was:

```
SYSPLEX TNRAL 10 0
```

The results of running the SYSPLEX EXEC in this environment are shown in Figure 20:

```
Application or Host Name                IP Address      Time

tnral.ralplex1.buddha.ral.ibm.com       192.168.221.3   0.390000
tnral.ralplex1.buddha.ral.ibm.com       192.168.221.28  0.380000
tnral.ralplex1.buddha.ral.ibm.com       192.168.221.3   0.220000
tnral.ralplex1.buddha.ral.ibm.com       192.168.221.28  0.220000
tnral.ralplex1.buddha.ral.ibm.com       192.168.221.3   0.280000
tnral.ralplex1.buddha.ral.ibm.com       192.168.221.28  0.270000
tnral.ralplex1.buddha.ral.ibm.com       192.168.221.3   0.220000
tnral.ralplex1.buddha.ral.ibm.com       192.168.221.28  0.280000
tnral.ralplex1.buddha.ral.ibm.com       192.168.221.3   0.220000
tnral.ralplex1.buddha.ral.ibm.com       192.168.221.28  0.280000

Summary of Ping responses

Good Responses : 10
Lost Responses : 0
Total Responses: 10

Hits by Canonical Addresses

Number     IP Address      Application or Host Name                Time
5          192.168.221.3   tnral.ralplex1                         0.266
5          192.168.221.28  tnral.ralplex1                         0.286
```

Figure 20. Observations from the Workstation of a Single-Path Sysplex Test

Examining the output of the EXEC, we can see the name TNRAL has been fully qualified as tnral.ralplex1.buddha.ral.ibm.com. This is the name reported by ping. The address assigned is in the next column and we see that the address provided by the DNS started with 192.168.221.3 (T03ATCP) and then toggled between that address and 192.168.221.28 (T28ATCP). 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 summarizes 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 on the 03 and 28 MVS systems.

4.7.2.1 A DNS Dump

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 OS/390 DNS. One is to use the USS ISPF panels to issue the commands and the second is to log on to the TCP/IP stack using standard (line mode) Telnet and issue UNIX commands. The second method is shown here.

To dump the DNS, connect to a USS Telnet server on the system running the DNS. In our case, T28ATCP controlled the DNS and we had set up a USS Telnet server on a second stack, T28CTCP.

```
EZYTE27I login: thomas
EZYTE28I thomas Password:     1
IBM
Licensed Material - Property of IBM
5647-A01 (C) Copyright IBM Corp. 1993, 1998
(C) Copyright Mortice Kern Systems, Inc., 1985, 1996.
(C) Copyright Software Development Group, University of Waterloo, 1989.
                              .
                              .
                              .
                              .
THOMAS @ RA03:/u/thomas>
THOMAS @ RA03:/u/thomas>su    2
THOMAS @ RA03:/u/thomas>
THOMAS @ RA03:/u/thomas>kill -int $(cat /etc/named.pid)   3
THOMAS @ RA03:/u/thomas>

THOMAS @ RA03:/u/thomas>cd /tmp    4
THOMAS @ RA03:/tmp>ls na*          5
named.run               named.run.great         named.stats
named.run.dhcpnsupdate  named.run.nt            named_dump.db
THOMAS @ RA03:/tmp>
```

Figure 21. UNIX Commands to Dump the DNS

Figure 21 shows the sequence of the commands and the responses. Some of the logon messages have been eliminated for clarity, but the first prompt is shown.

1. First we need to log on to Telnet on the MVS system **1** .
2. Then we need to be in superuser state. Enter su **2** .
3. Next we need to issue a command to dump the DNS. The task name is EZANSNMD and we could look through a process listing to get its process ID (PID). There is an easier way, however. There is a file called /etc/named.pid that contains the PID for the current NAMED (DNS) task.
4. So issue the command `kill -int $(cat /etc/named.pid)` **3** which goes out and reads the PID from the file and issues the command `kill -int nnnnnnn`, where nnnnnnn is the PID that was retrieved from the named.pid file.
5. The dump will be taken to a file /tmp/named_dump.db.
6. Change to that directory (cd /tmp) **4** .
7. List a subset of files (ls na*) and we get all files in the TMP directory beginning with ″na″. **5** And the dump is there.
8. To read the dump we have two choices. We can either FTP the file to a workstation and use whatever text editor or word processor we are familiar with or we can use the vi editor under Telnet. For those unfamiliar with vi, the exit command is
   - <esc> <:> <Q> <enter>

   where the brackets < > identify each key and are not typed.
9. We used FTP.

Now that we have a dump, what information can we obtain from it? Figure 22 shows the information at the beginning of the dump. In this section we do not provide a complete dump listing. That can be found in Appendix E, “Dump of T28ATCP Name Server - Single-Path Network” on page 217.

```
; Dumped at Mon May 18 17:24:01 1998
;; ++zone table++
; ralplex1.buddha.ral.ibm.com (type 2, class 1, source fback)   1
; time=895503283, lastupdate=895503223, serial=19980507,
; refresh=3600, retry=3600, expire=14400, minimum=60
; ftime=895503223, xaddr= 0.0.0.0 , state=20041, pid=0
; z_addr 1 :  192.168.236.3
; 168.192.in-addr.arpa (type 2, class 1, source rback)          2
; time=895512204, lastupdate=895512196, serial=10004,
; refresh=10, retry=10, expire=10, minimum=10
; ftime=895512196, xaddr= 192.168.236.3 , state=0041, pid=0
; z_addr 1 :  192.168.236.3
; 0.0.127.in-addr.arpa (type 1, class 1, source mvs28a.lbk)     3
; time=0, lastupdate=895364243, serial=10004,
; refresh=10, retry=10, expire=10, minimum=10
; ftime=895364243, xaddr= 0.0.0.0 , state=0041, pid=0
;; --zone table--
```

Figure 22. DNS Server Zone Transfer for a Single-Path Configuration

The first thing we see in the dump is the zone transfer from the primary DNS on T03ATCP to the secondary DNS on T28ATCP. It defines the domain name, ralplex1.buddha.ral.ibm.com, the time and date of the dump and the zone transfer information. The zone information has three sections:

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

These three sections refer to the three records defined in named.boot on T03DNS. Figure 12 on page 32 shows the corresponding boot records.

```
$ORIGIN buddha.ral.ibm.com.
ralplex1 IN SOA mvs03a.ralplex1.buddha.ral.ibm.com.
   4     garthm@mvs03a.ralplex1.buddha.ral.ibm.com. ( 19980507
               3600 3600 14400 60 ) ;Cl=5
   5     IN NS mvs03a.ralplex1.buddha.ral.ibm.com. ;Cl=5
   6     IN A 192.168.221.3 ;Cl=5
               IN A 192.168.221.4 ;Cl=5
  IN A 192.168.221.28 ;Cl=5
  IN A 192.168.221.29 ;Cl=5
```

Figure 23. More Data from the Primary DNS

In Figure 23 we can see the information obtained from the primary DNS concerning ralplex1. This information comes from the T03DNS forward cluster file pointed to in named.boot. The forward cluster file is shown in Figure 13 on page 33.

The $ORIGIN and SOA records identify the source of the information, mvs03a.ralplex1.buddha.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 in this single-path configuration also identifies a separate IP stack.

```
$ORIGIN ralplex1.buddha.ral.ibm.com.

FTPRAL  IN A 192.168.221.3 ;Cl=5
  IN A 192.168.221.28 ;Cl=5
mvs03a  IN A 192.168.221.3 ;Cl=5
mvsdum  IN A 192.168.221.5 ;Cl=5
mvs03c  IN A 192.168.221.4 ;Cl=5
localhost IN A 127.0.0.1 ;Cl=5
mvs28a  IN A 192.168.221.28 ;Cl=5
FTPOERAL IN A 192.168.221.4 ;Cl=5
  IN A 192.168.221.29 ;Cl=5
TNRAL  IN A 192.168.221.3 ;Cl=5
  IN A 192.168.221.28 ;Cl=5
mvs28c  IN A 192.168.221.29 ;Cl=5
ralplex1 IN CNAME ralplex1.buddha.ral.ibm.com. ;Cl=5
```

Figure 24. Hosts Defined in the Sysplex Domain

Figure 24 shows the domain statements for ralplex1.buddha.ral.ibm.com. There we can see entries for the three WLM managed applications: TNRAL, FTPRAL

and FTPOERAL. Both FTPRAL and TNRAL are available from both the 03 and 28 stacks while FTPOERAL is available from the 04 and 29 stacks. Even though we are looking at the DNS entries for the sysplex domain, not all of the DNS entries necessarily reflect applications that are sysplex capable. Host entries come either as part of a zone transfer from the primary name server or dynamically from WLM. Only those defined to WLM will have multiple host entries.

The information in this part of the dump will be updated dynamically to reflect the application availability. Should one instance of a WLM managed application be brought down, then this DNS record would show only one address. Similarly, should an additional instance of the application be started, we would then see three records.

The definition of applications to WLM is described in 4.2.3, “Data File Definitions for DNS/WLM” on page 27.

4.7.2.2 A DNS Trace of WLM Data

The DNS dump showed us which applications have multiple instances. How can we determine the balance DNS will use when building the application selection table shown in Figure 19 on page 40? There are no direct 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 the entire trace is available in Appendix F, “Trace of DNS for the Single-Path Network” on page 219. We will show excerpts as we go along.

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

The first area of the trace to look at is the data retrieved from WLM. Figure 25 shows the trace beginning from line 72, for those who are counting:

```
wlm_load ralplex1.buddha.ral.ibm.com          1
 group list retcode = -1 rsncode = 1034
 group list retcode = 0 rsncode = 0
 group list entry_count = 4
 querying group =  TCPIP                      2
 server list retcode = -1 rsncode = 1034
 server list retcode = 0 rsncode = 0
 querying group =  FTPRAL                     3
 server list retcode = -1 rsncode = 1034
 server list retcode = 0 rsncode = 0
 querying group =  TNRAL                      4
 server list retcode = -1 rsncode = 1034
 server list retcode = 0 rsncode = 0
 querying group =  FTPOERAL                   5
 server list retcode = -1 rsncode = 1034
 server list retcode = 0 rsncode = 0
```

Figure 25. WLM Data from DNS Trace - 1

We can see the WLM query 1 for ralplex1.buddha.ral.ibm.com. There are four entries returned by WLM: TCPIP 2, FTPRAL 3, TNRAL 4 and FTPOERAL 5. Earlier we mentioned there were three applications in the DNS dump. The fourth, TCPIP, is automatically generated by each stack when it starts up, but DNS *will not* respond to this application name. It will not be shown in a dump of DNS.

```
 processing group = ralplex1.buddha.ral.ibm.com count = 4  6
 minimum weight = 15
 processing server = TO3ATCP weight = 16 host = mvsO3a     7
 adding mvsO3a to list
db_update(ralplex1.buddha.ral.ibm.com, 0x14dce57O, 0x14dce57O, O1, 0x14dcd880)
match(0x14dcd9O8, 1, 6) 1, 6
match(0x14dcd9O8, 1, 1) 1, 6
match(0x14dcdb2O, 1, 1) 1, 2
db_update: adding 14dce57O
 processing server = TO3CTCP weight = 16 host = MVSO3C     8
 adding MVSO3C to list
db_update(ralplex1.buddha.ral.ibm.com, 0x14dce5a8, 0x14dce5a8, O1, 0x14dcd880)
match(0x14dcd9O8, 1, 6) 1, 6
match(0x14dcd9O8, 1, 1) 1, 6
match(0x14dcdb2O, 1, 1) 1, 2
match(0x14dce57O, 1, 1) 1, 1
db_update: flags = Ox1, sizes = 4, 4 (cmp -1)
credibility for ralplex1.buddha.ral.ibm.com is 4(5) from  0.0.0.0 .0, is 4(5) in cache
db_update: adding 14dce5a8
 processing server = T28ATCP weight = 15 host = MVS28A     9
 adding MVS28A to list
db_update(ralplex1.buddha.ral.ibm.com, 0x14dce5eO, 0x14dce5eO, O1, 0x14dcd880)
match(0x14dcd9O8, 1, 6) 1, 6
match(0x14dcd9O8, 1, 1) 1, 6
match(0x14dcdb2O, 1, 1) 1, 2
match(0x14dce57O, 1, 1) 1, 1
db_update: flags = Ox1, sizes = 4, 4 (cmp -1)
credibility for ralplex1.buddha.ral.ibm.com is 4(5) from  0.0.0.0 .0, is 4(5) in cache
match(0x14dce5a8, 1, 1) 1, 1
db_update: flags = Ox1, sizes = 4, 4 (cmp -1)
credibility for ralplex1.buddha.ral.ibm.com is 4(5) from  0.0.0.0 .0, is 4(5) in cache
db_update: adding 14dce5eO
 processing server = T28CTCP weight = 15 host = MVS28C     10
 adding MVS28C to list
```

Figure 26. WLM Data from DNS Trace - 2

In Figure 26 we can see the entries for each of the stacks and the relative weights associated with each system. The entry for the processing group 6 shows the minimum weight available for a server to be 15. Once the processing group parameters have been established, each of the servers is assigned a weight. The assigned weights of the servers are shown in Table 2. From this we can see the hosts are fairly evenly balanced.

Table 2 (Page 1 of 2). Host Weights

Server Name	Host Name	Identifier	Minimum Weight
T03ATCP	MVS03A	7	16

Table 2 (Page 2 of 2). Host Weights

Server Name	Host Name	Identifier	Minimum Weight
T03CTCP	MVS03C	8	16
T28ATCP	MVS28A	9	15
T28CTCP	MVS28C	10	15

We are not load balancing upon the hosts (in TCP/IP terms), but on the applications we have registered with WLM. These have entries a bit further down in the DNS trace. If we look at the entry for TNRAL in Figure 27 we can see that TNRAL is supported on only two of the four hosts. As WLM recognizes each host is on a different MVS system, the full weight for that system is allocated to the application, rather than the weight for the host (or stack, if you like).

```
 processing group = TNRAL.ralplex1.buddha.ral.ibm.com count = 2
 minimum weight = 31                                          11
 processing server = mvs03a weight = 32 host = mvs03a         12
db_update(TNRAL.ralplex1.buddha.ral.ibm.com, 0x14dce7f0, 0x14dce7f0, 01, 0x14dcd880)
match(0x14dcd908, 1, 6) 1, 6
db_update: adding 14dce7f0
db_update(mvs03a.TNRAL.ralplex1.buddha.ral.ibm.com, 0x14dce860, 0x14dce860, 01, 0x14dcd880)
savehash GROWING to 2
match(0x14dce7f0, 1, 6) 1, 1
match(0x14dce7f0, 1, 2) 1, 1
match(0x14dcd908, 1, 6) 1, 6
db_update: adding 14dce860
 processing server = MVS28A weight = 31 host = MVS28A         13
db_update(TNRAL.ralplex1.buddha.ral.ibm.com, 0x14dce8e8, 0x14dce8e8, 01, 0x14dcd880)
match(0x14dce7f0, 1, 6) 1, 1
match(0x14dce7f0, 1, 2) 1, 1
match(0x14dcd908, 1, 6) 1, 6
match(0x14dce7f0, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for TNRAL.ralplex1.buddha.ral.ibm.com is 4(5) from  0.0.0.0 .0, is 4(5) in cache
db_update: adding 14dce8e8
db_update(MVS28A.TNRAL.ralplex1.buddha.ral.ibm.com, 0x14dce920, 0x14dce920, 01, 0x14dcd880)
match(0x14dce7f0, 1, 6) 1, 1
match(0x14dce8e8, 1, 6) 1, 1
match(0x14dce7f0, 1, 2) 1, 1
match(0x14dce8e8, 1, 2) 1, 1
match(0x14dcd908, 1, 6) 1, 6
db_update: adding 14dce920
```

Figure 27. Application Weights

We can see that the minimum weight for the TNRAL application is 31 **11** and that there are two instances of the application. The application on MVS03A **12** is given a weight of 32 while the instance on MVS28A **13** is given the minimum weight of 31. This would provide a fairly even load balance and the difference between weights of 31 and 32 may not even be noticed if the WLM data is refreshed before 62 connections are made.

Our testing at the time this workload data was valid always showed an even distribution. The sample we are examining would have been valid for only 60 seconds as that is the default timer for DNS to refresh WLM data.

4.7.2.3 A DNS Trace for Datagram Traffic

Another common use of a trace is to look for data flowing between session partners. The DNS trace also provides that information and can be used to determine what data is received and sent. Previously we examined data that flowed over an API, and now we can look at the packet data.

Figure 28 on page 49 shows a datagram arriving from our workstation, 192.168.221.120, and the processing that occurs:

```
datagram from  192.168.221.120 .1089, fd 5, len 32; now Mon May 18 17:22:35 1998
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1, type = A, class = IN                  1


;; ...truncated
ns_req(from= 192.168.221.120 .1089)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1, type = A, class = IN


;; ...truncated
req: nlookup(tnral.ralplex1) id 1 type=1 class=1
req: missed 'tnral.ralplex1' as '' (cname=0)
findns: using cache
findns: using hints
findns: No root nameservers for class IN?
ns_req: answer ->  192.168.221.120 .1089 fd=5 id=1 size=32 Local
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 1       2
;; flags: qr rd ra; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1, type = A, class = IN


;; ...truncated

datagram from  192.168.221.120 .1090, fd 5, len 51; now Mon May 18 17:22:35 1998
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0             3
;; QUESTIONS:
;; tnral.ralplex1.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
ns_req(from= 192.168.221.120 .1090)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1.buddha.ral.ibm.com, type = A, class = IN
```

Figure 28 (Part 1 of 3). First Datagram from the Workstation

```
;; ...truncated
req: nlookup(tnral.ralplex1.buddha.ral.ibm.com) id 2 type=1 class=1
req: found 'tnral.ralplex1.buddha.ral.ibm.com' as 'tnral.ralplex1.buddha.ral.ibm.com' (cname=0)
******* Remove_Expired_RRs  Entered *******
name: TNRAL
  type=1 size=4
  type=1 size=4
Remove_Expired_RRs rc=0
 Looking for WLM generated Address
make_rr(tnral.ralplex1.buddha.ral.ibm.com, 14dcf008, bf3a3, 449, 1) 4 zone 1 ttl 0
match(0x14dcf008, 1, 24) 1, 1
match(0x14dcf100, 1, 24) 1, 1
 Weight = 0
 Current Server Weight 0
req: foundname=1, count=1, founddata=1, cname=0
sort_response(1)
findns: np 0x14dcf040 'TNRAL'
match(0x14dcf008, 1, 6) 1, 1
match(0x14dcf100, 1, 6) 1, 1
match(0x14dcf008, 1, 2) 1, 1
match(0x14dcf100, 1, 2) 1, 1
findns: np 0x14dcdaf0 'ralplex1'
match(0x14dcd918, 1, 6) 1, 6
findns: SOA found
req: leaving (tnral.ralplex1.buddha.ral.ibm.com, rcode 0)
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
findns: np 0x14dcdaf0 'ralplex1'
match(0x14dcd918, 1, 2) 1, 6
match(0x14dcdb30, 1, 2) 1, 2
match(0x14dceac0, 1, 2) 1, 1
match(0x14dceaf8, 1, 2) 1, 1
match(0x14dcedf8, 1, 2) 1, 1
match(0x14dcee30, 1, 2) 1, 1
findns: 1 NS's added for 'ralplex1'
make_rr(ralplex1.buddha.ral.ibm.com, 14dcdb30, bf3b3, 433, 1) 35 zone 1 ttl 60
match(0x14dcd918, 1, 24) 1, 6
match(0x14dcdb30, 1, 24) 1, 2
match(0x14dceac0, 1, 24) 1, 1
match(0x14dceaf8, 1, 24) 1, 1
match(0x14dcedf8, 1, 24) 1, 1
match(0x14dcee30, 1, 24) 1, 1
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
doaddinfo() addcount = 1
do additional "mvs03a.ralplex1.buddha.ral.ibm.com" (from "ralplex1.buddha.ral.ibm.com")
found it
match(0x14dcdb88, 1, 5) 1, 1
match(0x14dcdb88, 1, 5) 1, 1
match(0x14dcdb88, 1, 1) 1, 1
make_rr(mvs03a.ralplex1.buddha.ral.ibm.com, 14dcdb88, bf3e3, 385, 0) 4 zone 1 ttl 60
match(0x14dcdb88, 1, 24) 1, 1
addinfo: adding address data n = 16
```

Figure 28 (Part 2 of 3). First Datagram from the Workstation

```
ns_req: answer ->  192.168.221.120 .1090 fd=5 id=2 size=131 Local
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2        4
;; flags: qr aa rd ra; Ques: 1, Ans: 1, Auth: 1, Addit: 1
;; QUESTIONS:
;; tnral.ralplex1.buddha.ral.ibm.com, type = A, class = IN

;; ANSWERS:
tnral.ralplex1.buddha.ral.ibm.com.  IN A  192.168.221.3

;; AUTHORITY RECORDS:
ralplex1.buddha.ral.ibm.com.            60      IN      NS
                         mvs03a.ralplex1.buddha.ral.ibm.com.

;; ADDITIONAL RECORDS:
mvs03a.ralplex1.buddha.ral.ibm.com. 60 IN A
                         192.168.221.3
```

Figure 28 (Part 3 of 3). First Datagram from the Workstation

As this is a DNS trace we will not get the IP or UDP header details. In fact, all we see is the *QUESTIONS* and eventually the *ANSWERS*. In **1** the question translates to ″where is tnral.ralplex1?″ And the DNS checks through all its tables and finally responds with a status of SERVFAIL **2**, which translates as ″I don′t know, never heard of it.″

That is not what we expected, but the workstation is not surprised because it immediately sends another request **3**. This time it asks for tnral.ralplex1.buddha.ral.ibm.com, which is the full name of the application that we are after. This time the DNS thinks a lot longer and finally comes up with an answer **4**. This time DNS returns ANSWERS that include:

1. The address of tnral.ralplex1.buddha.ral.ibm.com is 192.168.221.3.
2. The authority for this came from the primary DNS for the ralplex1.buddha.ral.ibm.com domain which is mvs03a.ralplex1.buddha.ral.ibm.com.
3. An additional record that tells the workstation that the IP address of the primary DNS is at 192.168.221.3.

Since we had a ping loop going, the workstation immediately came back with another QUESTION, where is tnral.... The ANSWERS are in the back in Appendix F, “Trace of DNS for the Single-Path Network” on page 219. The ANSWERS for the second query can be summed up thus:

1. The address of tnral.ralplex1.buddha.ral.ibm.com is 192.168.221.28.
2. The authority for this came from the primary DNS for the ralplex1.buddha.ral.ibm.com domain which is mvs03a.ralplex1.buddha.ral.ibm.com.
3. An additional record that tells the workstation that the IP address of the primary DNS is at 192.168.221.3.

If that looks very similar to the first response, it′s because it is. The only difference between the DNS responses is the IP address associated with tnral.ralplex1.buddha.ral.ibm.com. The first time we saw 192.168.221.3 and the

second time 192.168.221.28. And that is what the printout from the REXX EXEC told us had happened. Now we can see why it happened.

4.7.3 Using an External DNS

Earlier in this section we stated we used the sysplex DNS as the default name server for the workstation rather than an external name server. The reason for this was to ensure the subject of workload balancing was addressed rather than worrying about other DNS operations. We found that the external DNS servers available to us exhibited some quirks that masked the objectives of this section.

We did not get the toggling of the returned addresses as we expected for an equally balanced two-image sysplex. Investigation showed that BIND-based name servers will by design not honor a TTL of 0. They will always store the received data in their cache and delete it a bit later. This will result in an effective TTL of ″less than 1 second″. Therefore, if you send multiple queries within this time frame the server will return cached data. This will not affect normal operations but distorted our test results. Figure 29 on page 53 shows the output of the REXX PING EXEC when using an external name server, MVS25O. You can see that, overall, the load balancing is maintained but it is not as granular as in the case where the workstation used the sysplex name server directly. If you refer to 8.5, “Tests Performed” on page 105 you will see a similar example where the external name server′s TTL was set deliberately to three seconds.

When a Windows NT name server was used as the external name server, it refused to point to T28DNS to retrieve ralplex1 data and always tried to point to the primary sysplex name server, T03DNS, instead. We understand there is a fix available for this.

For the above reasons we used T28DNS as the workstation′s name server so we could properly show the workload management provided by OS/390 DNS, WLM and sysplex.

```
Application or Host Name                  IP Address      Time

tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.071000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.080000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.3   0.090000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.3   0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.3   0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.3   0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.3   0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.3   0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.3   0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.3   0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.3   0.071000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.3   0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.3   0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.3   0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.3   0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.3   0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.090000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.071000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.080000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.090000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.080000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.080000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.28  0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.3   0.090000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.3   0.070000
tnral.ralplex1.buddha.ral.ibm.com         192.168.221.3   0.070000
 :

Summary of Ping responses

Good Responses : 200
Lost Responses : 0
Total Responses: 200

Hits by Canonical Addresses

Number     IP Address      Applicaton or Host Name                Time
98         192.168.221.3   tnral                                  0.07392380
102        192.168.221.28  tnral                                  0.07333333
```

Figure 29. Client Using External Name Server - SYSPLEXW Output

Chapter 5. Programs and Procedures for Checking Load Balancing

In Chapter 4, “Basic Sysplex with DNS” on page 21 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 OS/390 V2R8 IP and the network dispatcher, we need something more than that. Indeed, new functions such as dynamic VIPA and sysplex sockets, introduced in Release 8 and Release 7, respectively, required more coding effort to demonstrate their operation effectively.

In this chapter, therefore, we introduce some simple coding that we used to good effect in subsequent tests:

- A new REXX EXEC, SYSPLEX2.

  The simple EXECs in Chapter 4, “Basic Sysplex with DNS” on page 21 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 new 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.

- 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.

- 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.

- Finally, we have written a program to utilize the new 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.

5.1 Collecting Statistics Using REXX

In 4.7.2, “Observing the Effects of WLM and DNS” on page 40 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 introduced in 5.4, “SOCSRVR, a Simple Socket Server Program” on page 62. 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 30:

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

Figure 30. Resolving Host Name in REXX Sockets API

It then allocates a standard stream socket and connects to it as shown in the code excerpt in Figure 31:

```
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 31. 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 32 on page 57:

```
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 32. 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 B.3, "EXEC to Connect to Server Using TCP" on page 187 and the subroutine which is called at the end of the REXX program to sort and output the statistics, sysstats, is in B.4, "REXX Statistics Subroutine" on page 190. 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.

5.2 Registering Your Own Applications with WLM

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 as we can simply call a generic registration program before starting a server and benefit from DNS/WLM connection balancing very easily.

5.2.1 WLMREG, a Sample Registration Program

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 A.1, “WLMREG Registration Sample” on page 161.

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

5.2.2 The Registration Call

The IWMDNREG C function is documented in *OS/390 eNetwork Communications Server 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.

5.2.3 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, as 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.

5.2.4 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.

5.3 WLMQ, a WLM Query Program

Before introducing the new server program, we will look first at 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 cater for 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 #defined 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 33 on page 62 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 33. 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.

5.4 SOCSRVR, a Simple Socket Server Program

Here we introduce a simple TCP/IP server program, SOCSRVR, that we will 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 34:

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

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

Figure 34. Find Host Name 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 35 on page 63:

```
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 35. 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 36:

```
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 36. Accept Conversation, Send Data and Close

The complete source for the sample is available in A.3, “SOCSRVR Single Threading Server” on page 168.

See Figure 37 on page 64 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 37. Sample JCL to Run the WLMREG Then the SOCSRVR Programs

5.4.1 Modifying SOCSRVR for Dynamic VIPA

To modify the SOCSRVR (see 5.4, "SOCSRVR, a Simple Socket Server Program" on page 62) sample to exploit dynamic VIPA is very simple. First we change the parameter checking so that two arguments are required as in Figure 38:

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

Figure 38. 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 39:

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

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

Figure 39. Binding to a Specific VIPA

5.5 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.

5.5.1 Discovering Partner Information

This is done by way of a new option on the socket call getsockopt(), which is described in more detail in *OS/390 eNetwork Communications Server IP Application Programming Interface Guide*, SC31-8516. This new option is SO_CLUSTERCONNTYPE and is coded as shown in Figure 40:

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

Figure 40. getsockopt() Call

The call can return any of the values shown in Table 3. In this context a *cluster* is a sysplex:

Table 3. 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.
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 41 on page 66:

```
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 41. getsockopt() Call

5.5.2 SSOCCLNT, a Sample Sysplex Sockets Program

The sample sysplex sockets program very simply connects to the SOCSRVR program introduced in 5.4, “SOCSRVR, a Simple Socket Server Program” on page 62 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 42:

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

Figure 42. Output from SSOCCLNT

The complete source for the sample is available in A.4, “SSOCCLNT Sysplex Sockets Sample” on page 170.

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.

5.6 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.

5.6.1 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 *OS/390 eNetwork Communications Server 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 A.5, "MTCSRVR Multitasking Socket Program" on page 172, 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 43:

```
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 43. 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 44:

```
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 44. 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 45 on page 68:

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

Figure 45. 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 A.6, "MTCSUBT Subtask for the Multitasking Socket Program" on page 178.

5.6.2 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 shoud 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 5.1, "Collecting Statistics Using REXX" on page 55. 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 B.3, "EXEC to Connect to Server Using TCP" on page 187.

Chapter 6. Sysplex with RIP

In our next scenario, we introduced a slightly more complex network into the sysplex environment. We introduced a 3746-900 router, and we implemented RIP on the T03ATCP, T03CTCP and T28ATCP stacks using the OROUTED (the new RouteD) server. In this network, we decided not to use the T28CTCP stack at all. Since we are using RIP, we have enabled as many interfaces as possible to take advantage of dynamic route selection.

We used IP over XCF between the two ″A″ stacks and a same host connection between T03ATCP and T03CTCP. Both of these are point-to-point connections (in fact, the same host connection behaves like a point-to-multipoint connection if there are three or more stacks in the same MVS image).

See Figure 46 for a complete diagram:

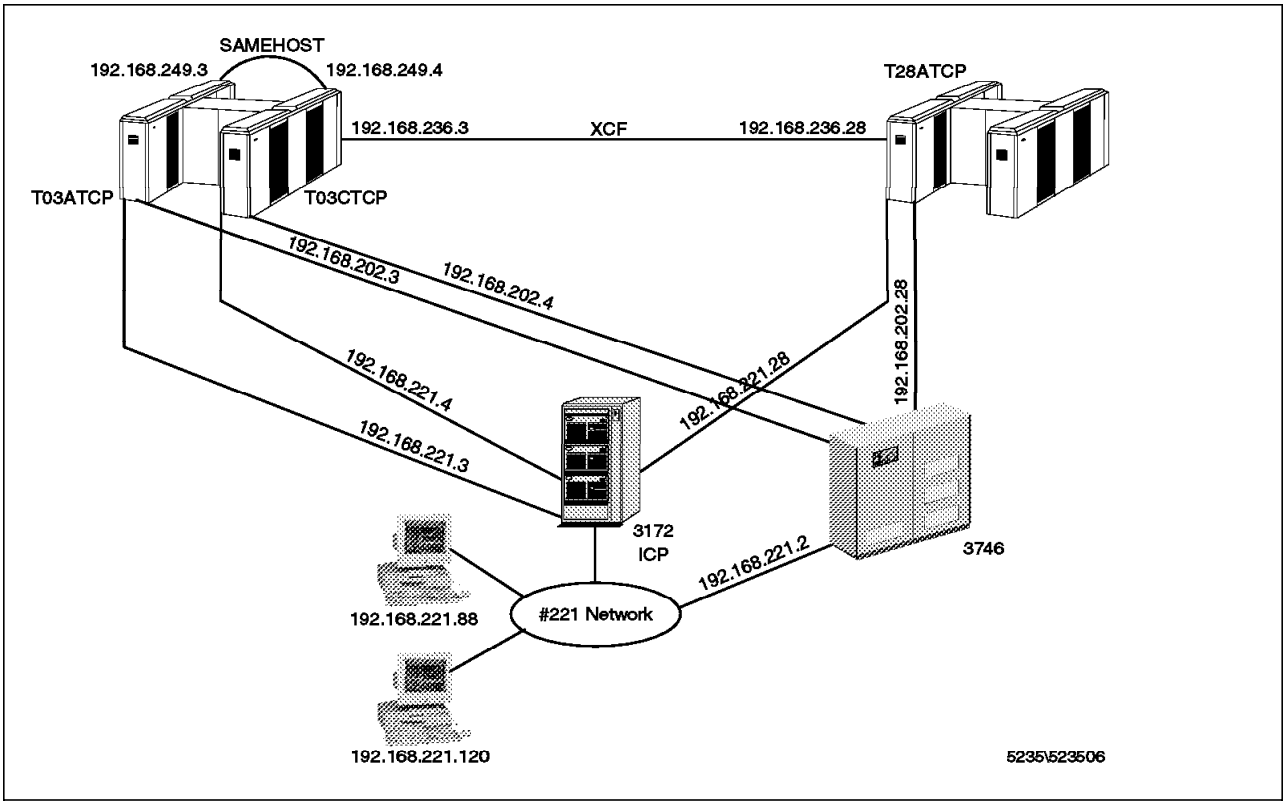


Figure 46. The Multipath Network

6.1 Configuring OROUTED

We added the following procedure and parameter files to enable RIP for our testing. Figure 47 on page 70 shows the JCL that runs OROUTED under UNIX System Services.

```
//TO3AROUT  PROC MODULE='BPXBATCH'
//OROUTED  EXEC PGM=&MODULE,REGION=4096K,TIME=NOLIMIT,
//         PARM='PGM /usr/lpp/tcpip/sbin/orouted -h -hv'
//STDOUT   DD PATH='/tmp/orouted.03a.stdout',
//         PATHOPTS=(OWRONLY,OCREAT,OAPPEND),
//         PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
//STDERR   DD PATH='/tmp/orouted.03a.stderr',
//         PATHOPTS=(OWRONLY,OCREAT,OAPPEND),
//         PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
//STDENV   DD DSN=TCP.TCPPARMS(RDO3AENV),DISP=SHR
//CEEDUMP  DD SYSOUT=*,DCB=(RECFM=FB,LRECL=132,BLKSIZE=132)
//SYSERR   DD PATH='/tmp/orouted.03a.syserr',
//         PATHOPTS=(OWRONLY,OCREAT,OAPPEND),
//         PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
```

Figure 47. Procedure for T03AROU

The -h parameter allowed OROUTED to handle host routes, which was necessary for our point-to-point connections to be broadcast. The -hv parameter enables host virtual routing, which was not necessary for this network, since we were not using VIPA capabilities. We left this parameter in, since it would do no harm and was needed for our next network example.

```
RESOLVER_CONFIG=//'TCP.TCPPARMS(TDATA03A)'
ROUTED_PROFILE=//'TCP.TCPPARMS(RDO3APR)'
GATEWAYS_FILE=//'TCP.TCPPARMS(RDO3AGW)'
```

Figure 48. STDENV Parameter File RD03AENV

The contents of our standard environment file are listed in Figure 48. This file is pointed to by the STDENV DD statement. This DD statement 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. We preferred using this method of choosing data files. The primary reason was that it worked, but it also eliminated all the complications of having to use JCL for coding USS variables on the ENVAR parameter. See Appendix C, "Configuration Files for Base and RIP Examples" on page 193 for the values used for our data files.

```
RIP_SUPPLY_CONTROL: RIP1
RIP_RECEIVE_CONTROL: ANY
```

Figure 49. RouteD Profile

Next we defined the profile used for the RouteD daemon. The contents can be found in Figure 49. We had no routers on our network using RIP-2, but we enabled our router to receive RIP-2 packets anyway. We only sent out RIP-1 packets.

```
options interface MPCTO25  192.168.235.3  ripoff
;options interface MPCTO25  *  ripoff
```

Figure 50. OROUTED Gateway File

Although we did not use it as part of our testing, we left a connection to our T25OTCP host active in this latest network. This host had a connection to our external network, and hence a huge number of potential routers. The options statement found in Figure 50 was used to stop OROUTED from accepting any RIP updates over that interface. The RIPOFF parameter stops OROUTED from accepting updates.

```
EZZ4922I Option(s): interface MPCTO25  *              ripoff
res_querydomain(*, buddha.ral.ibm.com, 1, 1)
res_query(*.buddha.ral.ibm.com, 1, 1)
res_mkquery(0, *.buddha.ra\.ibm.com, 1, 1)
res_send()
HEADER:
.opcode = QUERY, id = 1, rcode = NOERROR
.header flags:  rd
.qdcount = 1, ancount = 0, nscount = 0, arcount = 0

QUESTIONS:
.*.buddha.ral.ibm.com, type = A, class = IN

rcode = 3, ancount=0
res_search failed, errno = 0
EZZ4973I Unknown interface name (EN1) and/or address (*)
res_querydomain(, buddha.ral.ibm.com, 1, 1)
res_query(.buddha.ral.ibm.com, 1, 1)
res_mkquery(0, .buddha.ral.ibm.com, 1, 1)
res_query: mkquery failed
res_search failed, errno = 0
```

Figure 51. OROUTED Trace Excerpt

For an example of our BSDROUTINGPARMS settings, see Appendix C, “Configuration Files for Base and RIP Examples” on page 193.

Our procedures and data files were identical for our T03CTCP and T28ATCP stacks. The only changes were the necessary stack, file and address name differences. For complete examples, see Appendix C, “Configuration Files for Base and RIP Examples” on page 193.

Note that we have not used any VIPA addressing. We will use VIPA on this network in Chapter 7, “Sysplex with OROUTED and VIPA” on page 79.

6.1.1 The Workstation

We used two workstations on our LAN, one running Windows NT and one running OS/2. In each case, we had to configure TCP/IP to point to the name server at 192.168.221.28.

6.2 DNS Configuration, T03ATCP and T28ATCP

For this network, several changes were made to our DNS configuration. We began by getting rid of our non-sysplex DNS altogether. We considered this a viable migration path, since there are many advantages to running your name server on an OS/390 host (for example, when we implement VIPA, we can use a VIPA address on the host to point to the name server, decreasing the chances of an unreachable name server). Our DNS layout now looks like the one shown in Figure 52:

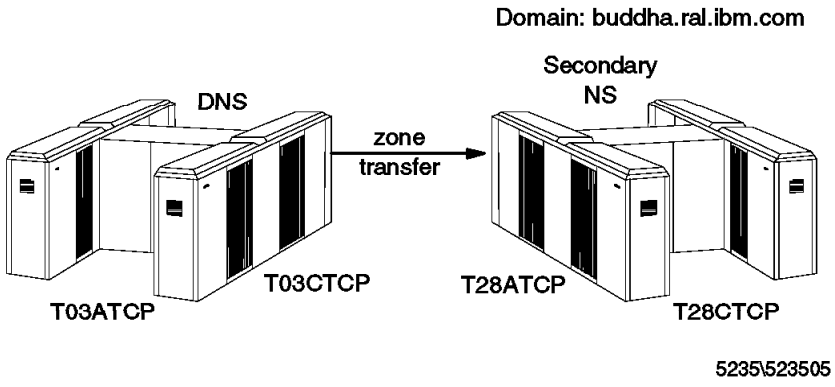


Figure 52. The DNS Layout, Multipath Network

Another thing we did was remove our ralplex1 subdomain from our DNS. This is not a recommended method of implementing a sysplex in an IP network, but we wanted to see if it caused any problems. We will talk about the effects of this change as we examine our configuration files in more detail.

In Figure 53 you can see the boot file used on our primary name server on our T03ATCP stack. This time, this server is a primary server for buddha.ral.ibm.com; the ralplex1 subdomain has been removed. Otherwise, this file is unchanged from our previous network.

```
;
;    /etc/named.boot for T03ATCP
;
;  TYPE      DOMAIN                    HOST        FILE
;
directory    /etc/dnsdata
;
primary    buddha.ral.ibm.com                     buddha.for   cluster
primary    168.192.in-addr.arpa                   buddha.rev
primary    0.0.127.in-addr.arpa                   mvs03a.lbk
```

Figure 53. T03ATCP DNS Boot File, Multipath Network

In Figure 54 on page 73 we have coded all of the addresses that we want to be available as addresses for our registered applications (remember, if we want an address to be used, it must be in the data file for the sysplex name server).

Again, we have included a mapping for one workstation on the LAN called thatpc. If we wanted to include more mappings for workstations on the LAN, we would include them here.

```
;
;  /etc/dnsdata/buddha.for for TO3ATCP
;
$ORIGIN buddha.ral.ibm.com.
@ IN      SOA     mvs03a.buddha.ral.ibm.com. garthm@mvs03a  (
            1998051904  ; 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
mvs03a          IN      A       192.168.249.3
                IN      A       192.168.202.3
                IN      A       192.168.236.3
                IN      A       192.168.221.3
mvs03c          IN      A       192.168.202.4
                IN      A       192.168.221.4
                IN      A       192.168.249.4
mvs28a          IN      A       192.168.236.28
                IN      A       192.168.202.28
                IN      A       192.168.221.28
thatpc          IN      A       192.168.221.88
localhost       IN      A       127.0.0.1
```

Figure 54. T03ATCP Forward File, Multipath Network

You may have noticed another limitation of this single domain example: we will no longer have hosts registered specifically in the sysplex domain. Because we must code an entry for mvs03a.buddha.ral.ibm.com in Figure 54, this host name will be the same as the name used when the T03ATCP stack registers through WLM. Previously the registered host was called mvs03a.ralplex1.buddha.ibm.com. Remember that in order for the name server to accept an address, it must be an active device (READY status) and it must also be listed in the name server forward data file. Basically, this means we do not have WLM management of the host names with this DNS configuration. But, we do still get a CNAME record added that matches our domain name. This means if you wanted to access any host in the sysplex, you could do so by using the host name buddha. The CNAME record maps it to buddha.buddha.ral.ibm.com, and normal sysplex/WLM management of this host name applies.

No static coding of CNAME records for our applications (TNRAL, FTPRAL and FTPOERAL) is necessary. Because we have included our sysplex in the domain buddha.ral.ibm.com, the WLM registered hosts will be found within the buddha domain directly.

In Figure 55 on page 74 we have the values for our reverse mapping file. We have added the addresses for all of the interfaces we will be using. Of course, we have also removed the ralplex1 domain qualifier from all entries.

```
;
;    /etc/dnsdata/buddha.rev for T03ATCP
;
@     IN    SOA     mvs03a.buddha.ral.ibm.com. garthm@mvs03a (
            1998051904  ; 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.buddha.ral.ibm.com.
3.221        IN  PTR     mvs03a.buddha.ral.ibm.com.
3.202        IN  PTR     mvs03a.buddha.ral.ibm.com.
3.249        IN  PTR     mvs03a.buddha.ral.ibm.com.
3.236        IN  PTR     mvs03a.buddha.ral.ibm.com.
4.221        IN  PTR     mvs03c.buddha.ral.ibm.com.
4.202        IN  PTR     mvs03c.buddha.ral.ibm.com.
4.249        IN  PTR     mvs03c.buddha.ral.ibm.com.
28.221       IN  PTR     mvs28a.buddha.ral.ibm.com.
28.202       IN  PTR     mvs28a.buddha.ral.ibm.com.
28.236       IN  PTR     mvs28a.buddha.ral.ibm.com.
```

Figure 55. T03ATCP Reverse File, Multipath Network

The loopback file for T03ATCP remained unchanged from the example shown in Figure 15 on page 34, except for the removal of the ralplex1 domain qualifier. See Figure 56 for an example:

```
;
;    /etc/dnsdata/mvs03a.lbk for T03ATCP
;
@   IN      SOA     mvs03a.buddha.ral.ibm.com. garthm@mvs03a  (
            1998051901  ; 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.buddha.ral.ibm.com.
1               IN      PTR     loopback.
```

Figure 56. T03ATCP Loopback Reverse Mapping File, Multipath Network

The coding of T28ATCP's DNS configuration included even fewer changes than T03ATCP. The boot file was unchanged except for the removal of the ralplex1 qualifier. See Figure 57:

```
;
;    /etc/named.boot for T28ATCP
;
;  TYPE      DOMAIN                      FILE OR HOST
directory      /etc/dnsdata
;
secondary   buddha.ral.ibm.com           192.168.236.3  fback  cluster
secondary   168.192.in-addr.arpa         192.168.236.3  rback
primary     0.0.127.in-addr.arpa         mvs28a.lbk
```

Figure 57. T28ATCP Boot File, Multipath Network

The two secondary statements will cause a zone transfer of the buddha.ral.ibm.com and 168.192.in-addr.arpa zones from the name server on 192.168.236.3. Backup files will be written to fback and rback respectively. Do not forget to either delete these files or change your serial number in the SOA on these zones to receive updated information. Also be sure to confirm that the zones have been successfully loaded; see 4.6, "Do Not Forget Your SOA Records" on page 35 for complete details on zone transfers and SOA records. The cluster keyword configures the name server to query WLM.

The loopback file was changed to have the ralplex1 domain qualifier removed, and was otherwise unchanged. See Figure 58:

```
;
;  /etc/dnsdata/mvs28a.lbk for T28ATCP
;
@   IN      SOA     mvs28a.buddha.ral.ibm.com. garthm@mvs03a  (
              1998051901  ; 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.buddha.ral.ibm.com.
1               IN      PTR     loopback.
```

Figure 58. T28ATCP Loopback Reverse Mapping File, Multipath Network

6.3 Observations of a Multipath Network

This scenario differs from that in 4.7, "Observation of TCP/IP Sysplex Operation" on page 37 in that each of the IP stacks now has multiple interfaces to the network. Figure 46 on page 69 shows this environment. How does this affect the way DNS balances the load between the systems?

The ICP connections (192.168.21.3 -.4 and -.28) remained in the configuration, but other interfaces were added. OROUTED was enabled to allow connection of a 3746-900 router to all three stacks on point-to-point links using the 192.168.202.0 network. Now we were able to see two additional routes. The XCF connection from T03ATCP to T28ATCP used the 192.168.236.0 network and the *samehost* connection from T03ATCP to T03CTCP used the 192.168.249.0 network.

No static routes were defined. DNS is, however, not a router, so DNS considered the connection of the 900 and the other connections as just more networks connected to the IP stacks via serial links. DNS does not take the routing paths or hop count into consideration when selecting IP addresses for a connection.

In our original tests with CS for OS/390 Release 5, we saw that the DNS algorithm did *not* balance the connection load evenly between the available interface addresses, as we had expected. Figure 59 on page 76 shows why.

```
 D:\ nslookup tnral
Server:  mvs28a.buddha.ral.ibm.com                          1
Address:  192.168.221.28

Name:    tnral.buddha.ral.ibm.com
Addresses:  192.168.221.28, 192.168.202.28, 192.168.236.28  2


 D:\ nslookup tnral
Server:  mvs28a.buddha.ral.ibm.com
Address:  192.168.221.28

Name:    tnral.buddha.ral.ibm.com                           3
Addresses:  192.168.221.3, 192.168.249.3, 192.168.202.3, 192.168.236.3
```

Figure 59. NSLOOKUP for the Multipath Configuration, Release 5

We used the nslookup command to find out what interface addresses DNS had in its cache for our generic sysplex application tnral. When nslookup tnral is entered, the program provides two responses. The first, **1**, gives the name and IP address of the name server. We can see from this display that we were using the name server on MVS28 for name resolution.

The second part of the response shows a list of addresses **2**. The list has been sorted by DNS and lists the direct attached path first. In this case it is the 3172 ICP with address 192.168.221.28. The list has two more addresses: 192.168.202.28 is the 900's ESCON attachment and 192.168.236.28 is the MVS28 end of the XCF connection. Since DNS does not know about routing, it cannot determine that the 236.28 connection would require a path through the T03ATCP stack.

When we repeated the nslookup command, we saw that DNS had toggled to the second host and again placed the directly attached 3172 ICP connection, 192.168.221.3, **3** at the head of the list. There are three other addresses: 249.3 is the same host link between T03ATCP and T03CTCP, 202.3 is the MVS03 end of the 900 ESCON connection and 236.3 is the MVS03 end of the XCF connection.

When the DNS/WLM combination was first implemented, DNS always favored what it perceived as a direct connection to its chosen host. It compared the subnet address of the client with the subnet address of each host interface, and any that matched were placed at the top of the list of addresses returned to the client. Thus, the connection requests were distributed evenly between the applications, but *always* on the 3172 interfaces. A 3172 running ICP has no routing function and no IP identity of its own, so its LAN interface appears to be directly attached to the host. Thus, the host and the client workstation are in the same IP subnetwork and DNS would favor this connection.

This behavior has since been changed, because it was deemed to be inefficient. Now DNS ignores any subnet matches between client and server, and distributes requests strictly according to the instructions of WLM. We reran this test using CS for OS/390 Release 7 and, as Figure 60 on page 77 shows, we saw an even distribution of connections across *all* the interfaces to the sysplex servers.

```
 Application or Host Name                 IP Address      Time

 tnral.ralplex1.buddha.ral.ibm.com        192.168.236.3   0.172000
 tnral.ralplex1.buddha.ral.ibm.com        192.168.236.28  0.140000
 tnral.ralplex1.buddha.ral.ibm.com        192.168.221.3   0.156000
 tnral.ralplex1.buddha.ral.ibm.com        192.168.236.28  0.156000
 tnral.ralplex1.buddha.ral.ibm.com        192.168.221.3   0.156000
 tnral.ralplex1.buddha.ral.ibm.com        192.168.221.28  0.172000
 tnral.ralplex1.buddha.ral.ibm.com        192.168.221.5   0.172000
 tnral.ralplex1.buddha.ral.ibm.com        192.168.236.28  0.141000
 tnral.ralplex1.buddha.ral.ibm.com        192.168.249.3   0.157000
 tnral.ralplex1.buddha.ral.ibm.com        192.168.221.28  0.156000

 Summary of Ping responses

Good Responses : 100
Lost Responses : 0
Total Responses: 100

Hits by Canonical Addresses

Number     IP Address      Applicaton or Host Name                 Time
26         192.168.236.28  tnral                                   0.16
13         192.168.249.3   tnral                                   0.16
25         192.168.221.28  tnral                                   0.15
12         192.168.236.3   tnral                                   0.16
12         192.168.221.3   tnral                                   0.16
12         192.168.221.5   tnral                                   0.17
```

Figure 60. SYSPLEXW Output for Multipath Configuration, Release 7

In Chapter 7, “Sysplex with OROUTED and VIPA” on page 79 we look at a configuration that permits the load balancing benefits of DNS and still allows routing code to make the routing decisions.

Chapter 7. Sysplex with OROUTED and VIPA

For this example, we have left the topology of the network the same as shown in Figure 46 on page 69. However, we did implement VIPA addressing on all of our hosts. In Figure 61, you can see the only change is the addition of three new VIPA addresses on T03ATCP, T03CTCP and T28ATCP. Because we are now using VIPA, we also had to add the SOURCEVIPA statement to all of the profiles. This statement will force the TCP/IP stack to insert the VIPA address as the source field in outbound packets. The VIPA definitions can be found in Appendix C, "Configuration Files for Base and RIP Examples" on page 193.

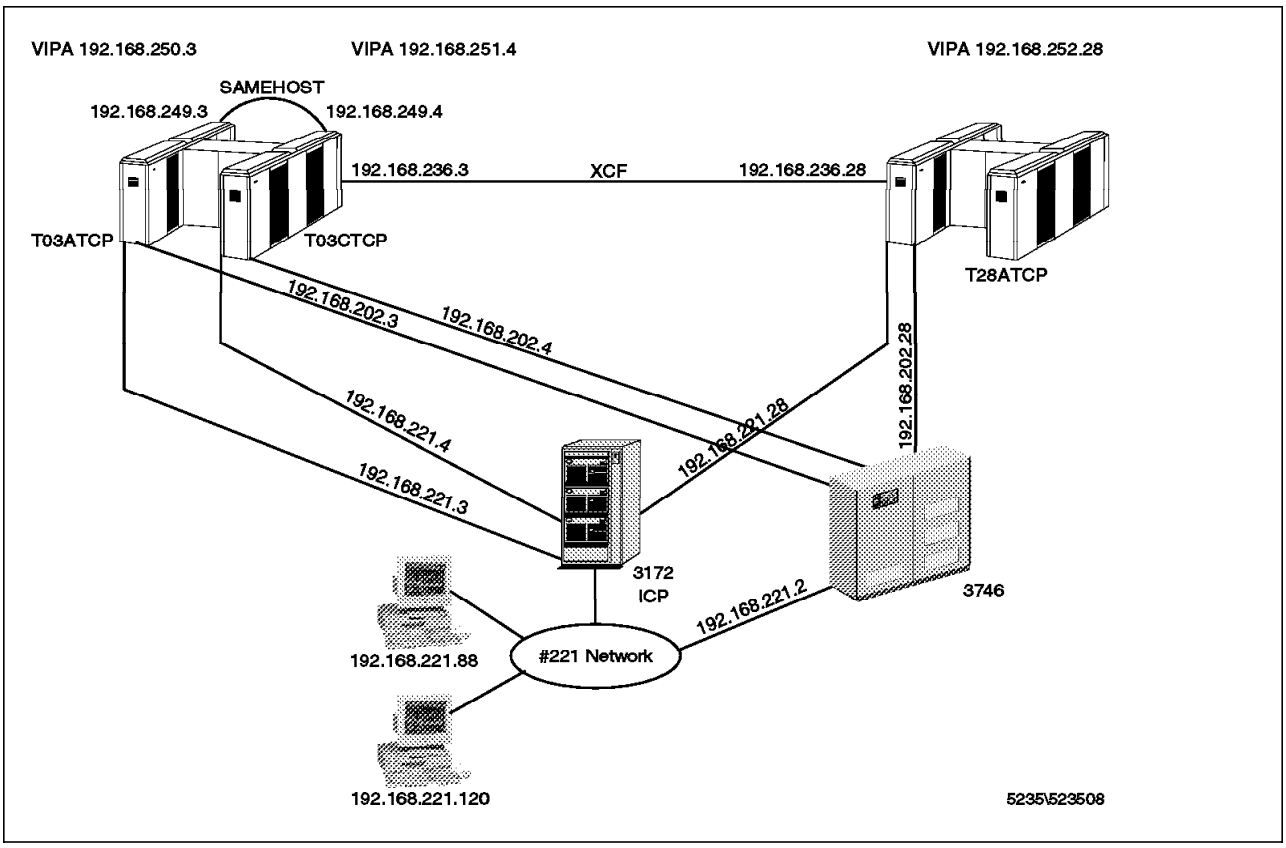


Figure 61. The VIPA/OROUTED Network

Because we have an ICP connection to our hosts, VIPA routing can present a problem. If the port is shared, the ICP box functions the same as an OSA card, in that it checks to see what the destination address is in a packet, and forwards it to the appropriate LPAR and/or stack if you are running multiple stacks. When we were using physical addresses, this was never a concern. If a packet comes in for 192.168.221.3, the ICP has a definition for this link and the packet goes to T03ATCP. But, what happens when we try to ping the VIPA address for T03ATCP, 192.168.250.3? Our router sends the packet to 192.168.221.3 because it knows that the VIPA subnet sits behind that interface. When the ICP box does its routine check of the destination address, it discards the packet because no such link exists!

The way around this is to define an extra link between the 3172 and the TCP/IP stacks on the host. On the 3172 end of the link, we defined the VIPA addresses. Yes, we put the VIPA addresses out there on the 3172 itself. Then, on each

TCP/IP stack, we added the device statement found in Figure 62 on page 80 to our profile for T03ATCP.

```
DEVICE icp11 LCS  328 autorestart   ; for VIPA on LAN
LINK icp11 IBMTR    0 icp11
```

Figure 62. Link Statement for VIPA Addressing

We then coded a HOME statement using a dummy IP address and of course a START statement. We made the corresponding changes to the T03CTCP and T28ATCP stacks as well. See Appendix C, “Configuration Files for Base and RIP Examples” on page 193 for complete details. With these changes, our VIPA connections were now possible throughout our network.

Again we used two workstations on our LAN, one running Windows NT and one running OS/2. In each case, we had to configure TCP/IP to point to the name server at 192.168.252.28. Because this is a VIPA address, we had to make sure that we added a reverse address in our 168.192.in-addr.arpa domain so that NSLOOKUP commands would not report any errors.

7.1 DNS Layout

In order to give different perspectives on how the DNS configuration might be done, we decided to continue to use only sysplex-based name servers, as we did in Chapter 6, “Sysplex with RIP” on page 69 (see Figure 52 on page 72). However, this time we returned to implementing our sysplex in the ralplex1 subdomain. This was the same domain configuration used in Chapter 4, “Basic Sysplex with DNS” on page 21.

7.1.1 DNS Configuration, T03ATCP with VIPA

In Figure 63 you can see the boot file used for the name server on T03ATCP:

```
;
;    /etc/named.boot for T03ATCP
;
;  TYPE      DOMAIN                    HOST        FILE
;
directory    /etc/dnsdata
;
primary    buddha.ral.ibm.com                     buddha.for
primary    ralplex1.buddha.ral.ibm.com            ralplex1.for  cluster
primary    168.192.in-addr.arpa                   buddha.rev
primary    0.0.127.in-addr.arpa                   mvs03a.lbk
```

Figure 63. T03ATCP DNS Boot File, VIPA Network

Two significant changes were made to this boot file. We added another primary statement for the ralplex1.buddha.ral.ibm.com subdomain. This is our WLM managed subdomain, so we added a cluster keyword. The second change was to remove the cluster keyword from the primary statement for the buddha domain. We want this domain to remain separate from our sysplex-managed subdomain, even though it is defined within the same name server.

```
;
;  /etc/dnsdata/buddha.for for TO3ATCP
;
$ORIGIN buddha.ral.ibm.com.
@ IN      SOA     mvs03a.buddha.ral.ibm.com. garthm@mvs03a  (
            1998051904  ; 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
mvs03a          IN      A       192.168.249.3
                IN      A       192.168.202.3
                IN      A       192.168.236.3
                IN      A       192.168.221.3
mvs03c          IN      A       192.168.202.4
                IN      A       192.168.221.4
                IN      A       192.168.249.4
mvs28a          IN      A       192.168.236.28
                IN      A       192.168.202.28
                IN      A       192.168.221.28
tnral           IN      CNAME   tnral.ralplex1.buddha.ral.ibm.com.
ftpral          IN      CNAME   ftpral.ralplex1.buddha.ral.ibm.com.
ftpoeral        IN      CNAME   ftpoeral.ralplex1.buddha.ral.ibm.com.
thatpc          IN      A       192.168.221.88
localhost       IN      A       127.0.0.1
```

Figure 64. T03ATCP Buddha Domain Forward File, VIPA Network

The forward file in Figure 64 looks very similar to the forward file we used for the T25OTCP name server in Chapter 4, “Basic Sysplex with DNS” on page 21. This file describes our non-sysplex mappings. These addresses will never be used for matching with addresses supplied from WLM because there is no `cluster` keyword on the primary statement in the boot file (see Figure 63 on page 80). Even so, these addresses should be included since they allow us to reach these hosts regardless of whether any applications have registered (that is what the `buddha` domain is for).

Again, we have `CNAME` records to allow us to reach our `ralplex1` domain applications by specifying a simple alias. This is necessary, as it was in Chapter 4, “Basic Sysplex with DNS” on page 21, because all of our queries to the name server will be coming from the `buddha.ral.ibm.com` domain.

If you wanted name resolution for workstations on your LAN, or other hosts in the `buddha.ral.ibm.com` domain, this would be the place to add those mappings. We have included a mapping only for `thatpc`, one of our test machines on the LAN.

```
;
;  /etc/dnsdata/named.for for TO3ATCP
;
$ORIGIN ralplex1.buddha.ral.ibm.com.
@ IN      SOA     mvs03a.ralplex1.buddha.ral.ibm.com. garthm@mvs03a  (
            1998051901  ; 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
mvs03a          IN      A       192.168.250.3
mvs03c          IN      A       192.168.251.4
mvs28a          IN      A       192.168.252.28
```

Figure 65. T03ATCP Ralplex1 Domain Forward File, VIPA Network

Figure 65 contains the forward mappings for our ralplex1.buddha.ral.ibm.com subdomain. VIPA addressing makes our ralplex1.buddha.ral.ibm.com domain quite straightforward. We do not want any of the real addresses to be used for our WLM-managed applications. Therefore, we put only our VIPA addresses into this file. Remember, when a stack registers with WLM, it passes the first 15 active (ready) addresses from the HOME list to WLM. When the name server queries WLM, it will receive all of these addresses. Then, the name server matches these addresses with the addresses listed in the forward data file for the primary statement with the cluster keyword. So, for us, only those three VIPA addresses will ever be given mappings in the name server. All the active physical addresses passed from WLM are ignored.

Note that for this network, we already have nine addresses in our home statement. If you have more than 15, you should make sure the ones you want passed to WLM are in the first 15 listed.

```
;
;   /etc/dnsdata/buddha.rev for T03ATCP
;
@     IN    SOA     mvs03a.buddha.ral.ibm.com. garthm@mvs03a (
            1998051904  ; 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.buddha.ral.ibm.com.
3.221        IN  PTR     mvs03a.buddha.ral.ibm.com.
3.202        IN  PTR     mvs03a.buddha.ral.ibm.com.
3.249        IN  PTR     mvs03a.buddha.ral.ibm.com.
3.250        IN  PTR     mvs03a.buddha.ral.ibm.com.
3.236        IN  PTR     mvs03a.buddha.ral.ibm.com.
4.221        IN  PTR     mvs03c.buddha.ral.ibm.com.
4.202        IN  PTR     mvs03c.buddha.ral.ibm.com.
4.249        IN  PTR     mvs03c.buddha.ral.ibm.com.
4.251        IN  PTR     mvs03c.buddha.ral.ibm.com.
28.221       IN  PTR     mvs28a.buddha.ral.ibm.com.
28.202       IN  PTR     mvs28a.buddha.ral.ibm.com.
28.236       IN  PTR     mvs28a.buddha.ral.ibm.com.
28.252       IN  PTR     mvs28a.buddha.ral.ibm.com.
```

Figure 66. T03ATCP Reverse File, VIPA Network

The reverse mapping file for T03ATCP, listed in Figure 66, is a little more interesting. We have included all physical addresses and all VIPA addresses. In particular, the reverse mapping for VIPA was necessary to allow us to use NSLOOKUP while pointing to the VIPA address for our name server (NSLOOKUP does a reverse query to the name server when it is first invoked). We did not include any reverse mappings to the `ralplex1` subdomain. There is no need for this mapping.

The loopback reverse mapping file remains unchanged from the one used in Chapter 6, “Sysplex with RIP” on page 69.

7.1.2 DNS Configuration, T28ATCP with VIPA

The boot file for T28ATCP is listed in Figure 67 on page 84. We have three secondary statements corresponding to the three primary zones we want to pull from T03ATCP’s name server. Since we will go directly to WLM instead of transferring that information, the secondary statement for `ralplex1.buddha.ral.ibm.com` has the `cluster` keyword coded. Backup files will be written to `fback`, `xback` and `rback`, respectively. Do not forget to delete these files or to change your serial number in the SOA on these zones to receive updated information. Also be sure to confirm that the zones have been successfully loaded; see 4.6, “Do Not Forget Your SOA Records” on page 35 for complete details on zone transfers and SOA records.

```
;
;    /etc/named.boot for T28ATCP
;
;  TYPE      DOMAIN                        FILE OR HOST
directory      /etc/dnsdata
;
secondary   buddha.ral.ibm.com            192.168.250.3  fback
secondary   ralplex1.buddha.ral.ibm.com   192.168.250.3  xback  cluster
secondary   168.192.in-addr.arpa          192.168.250.3  rback
primary     0.0.127.in-addr.arpa          mvs28a.lbk
```

Figure 67. T28ATCP DNS Boot File, VIPA Network

Our only primary zone is for the loopback address reverse mapping file. This file remained unchanged from the example in Chapter 6, “Sysplex with RIP” on page 69. See Figure 58 on page 75 for the actual coding.

7.2 Observing VIPA in a Sysplex Environment

In this DNS configuration we have the same interfaces as in the previous chapter, but in addition we have enabled virtual IP addresses for each stack. The VIPA data is shown in Table 4:

Table 4. Host Addresses

MVS Name	Stack Name	VIPA Address	Home Addresses
MVS03	T03ATCP	192.168.250.3	• 192.168.221.3 • 192.168.202.3 • 192.168.236.3 • 192.168.249.3
MVS03	T03CTCP	192.168.251.4	• 192.168.221.4 • 192.168.202.4 • 192.168.249.4
MVS28	T28ATCP	192.168.252.28	• 192.168.221.28 • 192.168.202.28 • 192.168.236.28

7.2.1 Workload Management

We have configured two domains with the DNS on MVS03 acting as primary DNS for both domains and the DNS on MVS28 acting as the secondary. All of the HOME addresses are defined in domain buddha.ral.ibm.com to provide individual addressability while the VIPA addresses are defined in the subdomain ralplex1.buddha.ral.ibm.com.

The objective of this configuration is to associate the sysplex application names with the VIPA addresses only, to see if we can combine the MVS load balancing capabilities of sysplex with the availability and routing functions associated with VIPA. Since all of our previous examples showed lightly loaded systems and even load balancing, now we show an unbalanced environment where one of the hosts has a noticeably heavier workload than the other.

When we ran the tests this time, 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 make no allowance for other resource utilization such as paging, I/O rate or memory.

The output from the SYSPLEXW EXEC for this run is shown in Figure 68. The run was for 50 samples at 0 delay, but only the first 20 or so are shown:

```
Application or Host Name                 IP Address      Time

tnral.ralplex1.buddha.ral.ibm.com        192.168.250.3   0.141000
tnral.ralplex1.buddha.ral.ibm.com        192.168.250.3   0.140000
tnral.ralplex1.buddha.ral.ibm.com        192.168.252.28  0.141000
tnral.ralplex1.buddha.ral.ibm.com        192.168.250.3   0.140000
tnral.ralplex1.buddha.ral.ibm.com        192.168.250.3   0.141000
tnral.ralplex1.buddha.ral.ibm.com        192.168.252.28  0.140000
tnral.ralplex1.buddha.ral.ibm.com        192.168.250.3   0.141000
tnral.ralplex1.buddha.ral.ibm.com        192.168.250.3   0.141000
tnral.ralplex1.buddha.ral.ibm.com        192.168.252.28  0.140000
tnral.ralplex1.buddha.ral.ibm.com        192.168.250.3   0.141000
tnral.ralplex1.buddha.ral.ibm.com        192.168.250.3   0.140000
tnral.ralplex1.buddha.ral.ibm.com        192.168.252.28  0.140000
tnral.ralplex1.buddha.ral.ibm.com        192.168.250.3   0.141000
tnral.ralplex1.buddha.ral.ibm.com        192.168.250.3   0.140000
tnral.ralplex1.buddha.ral.ibm.com        192.168.252.28  0.141000
tnral.ralplex1.buddha.ral.ibm.com        192.168.250.3   0.141000
tnral.ralplex1.buddha.ral.ibm.com        192.168.250.3   0.156000
tnral.ralplex1.buddha.ral.ibm.com        192.168.252.28  0.156000
tnral.ralplex1.buddha.ral.ibm.com        192.168.250.3   0.140000
tnral.ralplex1.buddha.ral.ibm.com        192.168.250.3   0.141000
tnral.ralplex1.buddha.ral.ibm.com        192.168.252.28  0.140000

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          192.168.250.3   tnral                                  0.14617647
16          192.168.252.28  tnral                                  0.1463125
```

Figure 68. SYSPLEXW Output for VIPA Configuration

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 69 on page 86 shows a heavily edited version of the trace showing the load balance information provided by WLM. A more complete version of the trace is shown in Appendix G, “DNS Trace for VIPA Configuration” on page 231.

```
wlm_load ralplex1.buddha.ral.ibm.com
 querying group =  FTPRAL                1
 querying group =  TNRAL
 querying group =  TCPIP
 processing group = FTPRAL.ralplex1.buddha.ral.ibm.com count = 2
 minimum weight = 21                2
 processing server = MVS03A weight = 42 host = MVS03A
savehash GROWING to 2               3
 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
 processing server = MVS03A weight = 42 host = MVS03A
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
db_update(TNRAL.ralplex1.buddha.ral.ibm.com, 0x14dceb98, 0x14dceb98, 01, 0x14dcd8b8)
credibility for TNRAL.ralplex1.buddha.ral.ibm.com is 4(5) from  0.0.0.0 .0, is 4(5) in cache
db_update: adding 14dceb98
db_update(MVS28A.TNRAL.ralplex1.buddha.ral.ibm.com, 0x14dcebd0, 0x14dcebd0, 01, 0x14dcd8b8)
```

Figure 69. DNS Trace for the VIPA Configuration

At 1 we see the beginning of the WLM update. The two sysplex applications FTPRAL and TNRAL can be seen. The data for FTPRAL 2 shows FTPRAL on two hosts, MVS03A and MVS28A. MVS03A has been assigned a weight of 42 while MVS28A has been assigned 21 3. The data for TNRAL is the same as for FTPRAL since both instances of TNRAL run on the same hosts as FTPRAL.

From the weights assigned by WLM, we can see the basis for the 2 to 1 ratio observed when running the SYSPLEXW REXX EXEC.

7.2.2 Routing Benefits

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. In this scenario, we tested a connection from the workstation that could go via a 3746-900 router or via a 3172 ICP connection to the OS/390 host. We originally set up the routing tables so that the 900 did not know it had a connection from its token-ring interface to the 3172 token-ring interface.

The workstation used the 3746-900 as its default router. Traffic from the workstation to the host went to the 3746 and then up the ESCON connection to the TCP/IP stack. However, traffic from the host to the workstation was routed via the 3172 since OROUTED knew that the workstation and the 3172 were both part of the same network.

When we stopped the 3172 ESCON connection, our TN3270E session continued to function since OROUTED simply used the alternative route via the 3746-900. Had we been using the 3172 (ICP) home address instead of the VIPA address for TCP/IP, our session would have failed since this address was no longer available.

There is enough evidence to suggest the most viable configuration is to use the DNS sysplex functions with the 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 Telnet (or whatever) 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.

7.2.2.1 Dynamic Routing Does Not Fix Everything

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 RIP is disabled on the workstation. With an NT server on the same network as the 3172 and the 3746-900, we found a route created dynamically on the NT server. We had set the 3746-900 as the default router for the NT server. The 3746-900 was RIP enabled on its LAN interface and was aware of the existence of the 3172 ICP adapter on the same network. The 3746-900 set the route to the VIPA via the 3172 rather than its own ESCON.

   We set up a TN3270 session from NT to the VIPA and then killed the ICP link. Our session died. When we looked at the routes *on the NT server* we found a host route for the VIPA via the 3172 address! It took about 10 minutes for this route to disappear. TN3270 sessions do not always stay alive for 10 minutes.

   This scenario would be uncommon for a user′s workstation, but many installations may well have NT servers on the same network as their OS/390 servers. In this case, it may be necessary to configure a channel-attached default router with static routes to the VIPA so that connections via 3172 ICP or OSA will not show up in other servers′ routing tables.

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. If we are talking about host links into the OS/390 stack, then only one or two levels of routers need to change tables to make it work. That will normally succeed, but you should keep this in mind.

Chapter 8. Sysplex with OMPROUTE and VIPA

Our next configuration was set up to perform some tests with OMPROUTE and dynamic XCF, which were introduced in CS for OS/390 Release 6 and Release 7, respectively. We configured three systems in separate LPARs. Each system had an XCF connection (to each other), a LAN connection (via an OSA), and an MPC connection to a channel-attached 2216. One TCP/IP stack ran in each system. See Figure 70 on page 90 for a complete network diagram.

Dynamic routing was enabled by starting the OMPROUTE daemon on each stack. We configured OMPROUTE to use the OSPF routing protocol (see 3.2, “OSPF” on page 18). The two external routers were also configured to run OSPF.

Since we were connected to a production network (the 9.0.0.0 network), we also received some OSPF information from other production routers.

We used dynamic XCF to create dynamic IP definitions over XCF among the three stacks. Dynamic XCF is achieved by coding the DYNAMICXCF parameter in the IPCONFIG section of the profile data set for TCP/IP. The configuration for each stack also included SOURCEVIPA in the IPCONFIG section.

We have provided the new TCP/IP configuration files in Appendix D, “Profiles, Data Files and Parameter Files” on page 205.

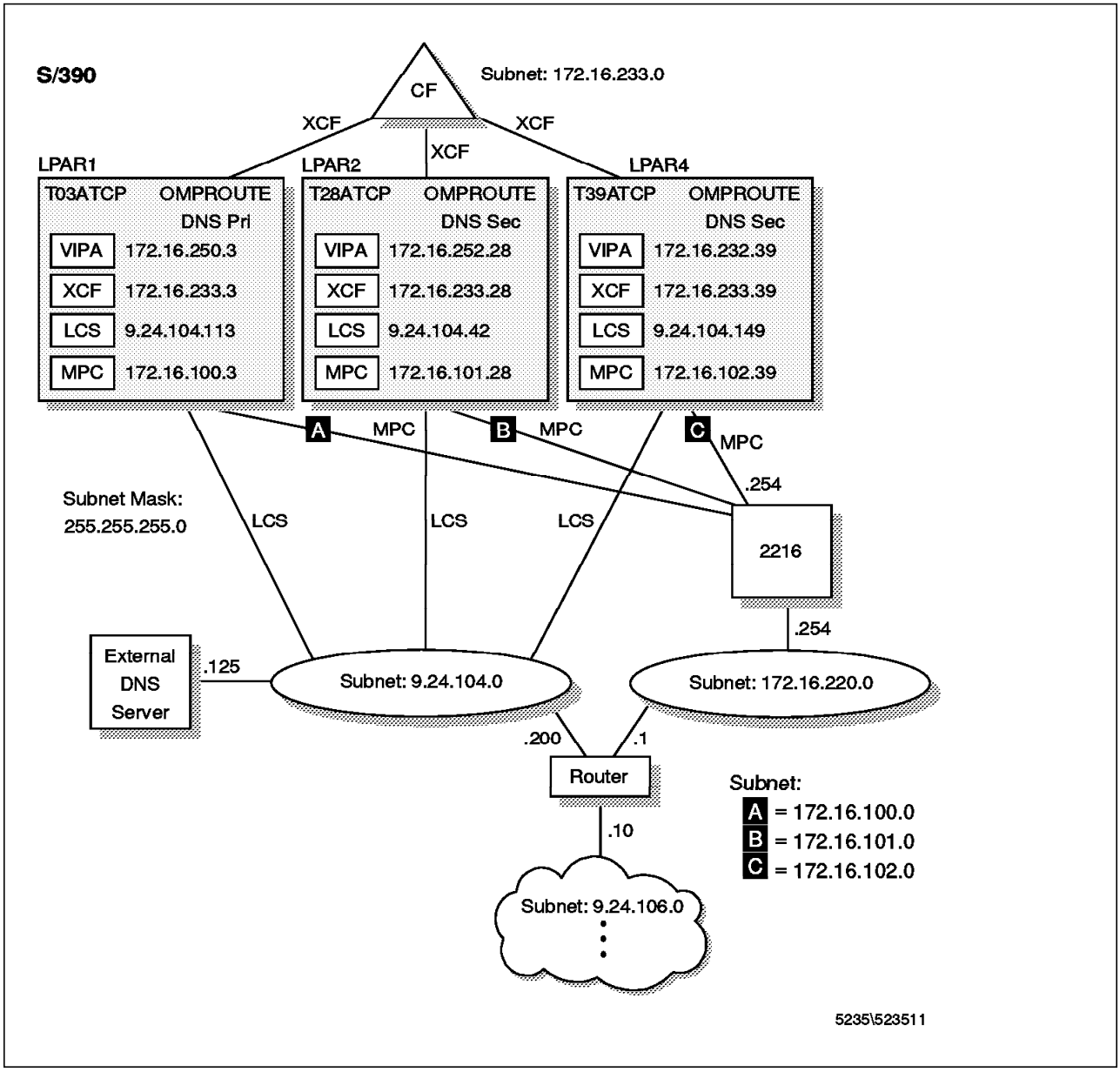


Figure 70. The OMPROUTE Network

8.1 Configuring OMPROUTE

In this section we show sample configurations for an OMPROUTE server in the test environment at ITSO Raleigh.

Figure 71 on page 91 is our sample cataloged procedure for an OMPROUTE server in our OS/390 system.

```
//OMPROUTE EXEC PGM=BPXBATCH,REGION=4096K,TIME=NOLIMIT,
//          PARM='PGM /usr/lpp/tcpip/sbin/omproute '
//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=TCP.TCPPARMS(OM03AENV),DISP=SHR
//CEEDUMP   DD DUMMY
```

Figure 71. Sample OMPROUTE Catalog Procedure - T03AOMPR

The contents of our standard environment file are listed in Figure 72:

```
RESOLVER_CONFIG=//'TCP.TCPPARMS(TDATA03A)'
OMPROUTE_FILE=//'TCP.TCPPARMS(OM03ACF)'
```

Figure 72. STDENV Parameter File OM03AENV

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

In Figure 73 on page 92 the contents of our TCPIP.DATA file are shown:

```
TCPIPJOBNAME TO3ATCP
HOSTNAME  MVSO3A
DOMAINORIGIN  buddha.ral.ibm.com
NSINTERADDR  9.24.104.125
NSPORTADDR 53
RESOLVEVIA UDP
RESOLVERTIMEOUT 10
RESOLVERUDPRETRIES 1
DATASETPREFIX TCP
MESSAGECASE MIXED
```

Figure 73. TCPDATA File TDATA03A

The contents of the OMPROUTE configuration file are found in Figure 74. The OMPROUTE routing application reads the file to configure the OSPF and RIP routing protocols to be used over each interface.

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

```
Area    Area_Number=0.0.0.0                    1
              Stub_Area=NO
              Authentication_type=None;
OSPF_Interface IP_Address=172.16.100.3         2
              Name=mO32216b
              Subnet_mask=255.255.255.0
              MTU=32768;
OSPF_Interface IP_Address=172.16.233.*         3
              Subnet_mask=255.255.255.0
              MTU=32768;
OSPF_Interface IP_Address=9.24.104.113         4
              Name=tr1
              CostO=6
              Subnet_mask=255.255.255.0
              MTU=4082;
Interface     IP_Address=172.16.250.3          5
              Name=VIPA3A
              Subnet_mask=255.255.255.0
              MTU=32768;
AS_Boundary_routing                            6
              Import_Direct_Routes=YES;
```

Figure 74. OMPROUTE Configuration File OM03ACF

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

3 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.

4 This statement sets the OSPF parameters for the OSA token-ring interface. We configured a higher cost for this interface than the other interfaces (default value of Cost0 is 1) to ensure that we used the route over XCF links rather than though the OSA token-ring, for traffic in between the sysplex hosts.

5 We configured VIPA using the Interface statement. A point-to-point interface that is neither an OSPF nor a RIP interface should be configured to OMPROUTE via the Interface statement.

6 The AS_Boundary_routing statement is used to enable the AS boundary routing capability, which allows you to import routes learned from other methods (RIP, statically configured, and direct routes) into the 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.

2, **3**, **4**, **5** 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.

8.1.1 OMPROUTE Commands

OMPROUTE is controlled from the MVS operator console using MVS system commands. We will introduce some DISPLAY commands to display OSPF configuration and state information, and MODIFY commands to control OMPROUTE. You will find more details in *OS/390 eNetwork Communications Server IP Configuration*, SC31-8513.

To display all OSPF configuration information, you can issue `Display TCPIP,<tcpipjobname>,OMPROUTE,OSPF,LIST,ALL` with the result as in Figure 75 on page 94:

```
D TCPIP,TO3ATCP,OMPROUTE,OSPF,LIST,ALL
EZZ7831I GLOBAL CONFIGURATION 250
    TRACE: 0, DEBUG: 0, SADEBUG LEVEL: 0
    STACK AFFINITY:          TO3ATCP
    OSPF PROTOCOL:           ENABLED
    EXTERNAL COMPARISON:     TYPE 2
    AS BOUNDARY CAPABILITY:  ENABLED
    IMPORT EXTERNAL ROUTES:  DIR SUB
    ORIG. DEFAULT ROUTE:     NO
    DEFAULT ROUTE COST:      (1, TYPE 2)
    DEFAULT FORWARD. ADDR.:  0.0.0.0
    DEMAND CIRCUITS:         ENABLED

EZZ7832I AREA CONFIGURATION
AREA ID          AUTYPE          STUB? DEFAULT-COST IMPORT-SUMMARIES?
0.0.0.0          0=NONE           NO          N/A           N/A

EZZ7833I INTERFACE CONFIGURATION
IP ADDRESS      AREA             COST  RTRNS  TRNSDLY PRI  HELLO   DEAD
172.16.233.3    0.0.0.0             1      5      1     1     10     40
172.16.233.3    0.0.0.0             1      5      1     1     10     40
172.16.100.3    0.0.0.0             1      5      1     1     10     40
9.24.104.113    0.0.0.0             6      5      1     1     10     40
```

Figure 75. Display Command to List OSPF Definitions

To display current run-time statistics related to the OSPF interface use Display TCPIP,<tcpipjobname>,OMPROUTE,OSPF,InterFaces, NAME=<if_name>. Figure 76 shows a typical result:

```
D TCPIP,TO3ATCP,OMPR,OSPF,IF,NAME=MO32216B
EZZ7850I INTERFACE DETAILS 812
                INTERFACE ADDRESS:      172.16.100.3
                ATTACHED AREA:          0.0.0.0
                PHYSICAL INTERFACE:     MO32216B
                INTERFACE MASK:         255.255.255.0
                INTERFACE TYPE:         P-2-MP
                STATE:                  16
                DESIGNATED ROUTER:      0.0.0.0
                BACKUP DR:              0.0.0.0

  DR PRIORITY:       1  HELLO INTERVAL:   10  RXMT INTERVAL:     5
  DEAD INTERVAL:    40  TX DELAY:          1  POLL INTERVAL:     0
  DEMAND CIRCUIT:  OFF  HELLO SUPPRESS:  OFF  SUPPRESS REQ:    OFF
  MAX PKT SIZE:  32768  TOS 0 COST:        1

  # NEIGHBORS:       1  # ADJACENCIES:     1  # FULL ADJS.:      1
  # MCAST FLOODS: 2980  # MCAST ACKS:    592  DL UNICAST:      OFF
  MC FORWARDING:   OFF

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

Figure 76. Display Command to Display Statistics for the OSPF Interface

To display all of the routes in the OMPROUTE routing table, you can issue Display TCPIP,<tcpipjobname>,OMPROUTE,RTTABLE as shown in Figure 77 on page 95:.

```
D TCPIP,TO3ATCP,OMPROUTE,RTTABLE
EZZ7847I ROUTING TABLE
TYPE   DEST NET         MASK      COST    AGE     NEXT HOP(S)

SBNT   9.0.0.0          FF000000  1       7023    NONE
 SPF*  9.24.104.0       FFFFFF00  6       3997    TR1
 SPF   9.24.105.0       FFFFFF00  18      3981    172.16.100.254
 SPF   9.24.106.0       FFFFFF00  8       3981    172.16.100.254
SBNT   172.16.0.0       FFFF0000  1       372     NONE
 DIR*  172.16.100.0     FFFFFF00  1       4010    172.16.100.3
 SPF   172.16.100.3     FFFFFFFF  0       4007    MO32216B
 SPF*  172.16.100.254   FFFFFFFF  1       3997    172.16.100.254
SPE2   172.16.101.0     FFFFFF00  1       51      172.16.233.28
 SPF   172.16.101.28    FFFFFFFF  1       83      172.16.233.28
 SPF   172.16.101.254   FFFFFFFF  1       3997    172.16.100.254
SPE2   172.16.102.0     FFFFFF00  1       51      172.16.233.39
 SPF   172.16.102.39    FFFFFFFF  1       3989    172.16.233.39
 SPF   172.16.102.254   FFFFFFFF  1       3997    172.16.100.254
 SPF   172.16.220.0     FFFFFF00  2       3981    172.16.100.254
 SPF   172.16.220.1     FFFFFFFF  2       3981    172.16.100.254
 SPF   172.16.220.254   FFFFFFFF  1       3981    172.16.100.254
SPE2   172.16.232.0     FFFFFF00  1       51      172.16.233.39
SPE2   172.16.232.39    FFFFFFFF  1       51      172.16.233.39
 DIR*  172.16.233.0     FFFFFF00  1       4010    172.16.233.3
 SPF   172.16.233.3     FFFFFFFF  0       4007    EZAXCF28
STAT*  172.16.233.28    FFFFFFFF  0       4010    172.16.233.3
STAT*  172.16.233.39    FFFFFFFF  0       4010    172.16.233.3
 DIR*  172.16.250.0     FFFFFF00  1       4010    172.16.250.3
 DIR*  172.16.250.3     FFFFFFFF  1       4010    VIPA3A
SPE2   172.16.252.0     FFFFFF00  1       51      172.16.233.28
SPE2   172.16.252.28    FFFFFFFF  1       51      172.16.233.28
                      0 NETS DELETED, 1 NETS INACTIVE
```

Figure 77. Display Command to List OMPROUTE Routing Table

To stop OMPROUTE, use P <procname> or F <procname>,KILL.

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

8.2 DNS Layout

We had a primary DNS server for the domain buddha.ral.ibm.com, located outside the sysplex. The sysplex itself was represented by subdomain ralplex1. In the sysplex we ran three name servers, one on each stack. A primary name server ran on stack T03ATCP and secondary name servers ran on T28ATCP and T39ATCP.

The conceptual DNS layout is the same as the one described in Figure 7 on page 28; we just added one secondary name server to the sysplex domain in the third sysplex host.

8.2.1 DNS Configuration, External Name Server

The external name server boot file had four primary statements pointing to a forward mapping file, two reverse mapping files and a loopback reverse mapping file. The following files are the external name server's boot (Figure 78) and forward mapping (Figure 79) files:

```
;
;    /etc/named.boot
;
;  TYPE      DOMAIN                    FILE OR HOST
;
directory    /etc/dnsdata
;
primary    buddha.ral.ibm.com            buddha.for
primary    104.24.9.in-addr.arpa         buddha.rev9
primary    16.172.in-addr.arpa           buddha.rev
primary    0.0.127.in-addr.arpa          mvs25o.lbk
```

Figure 78. External Name Server DNS Boot File

```
;
;  /etc/dnsdata/buddha.for
;
$ORIGIN buddha.ral.ibm.com.
@ IN      SOA     mvs25o.buddha.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      mvs25o
mvs25o          IN      A       9.24.104.125
localhost       IN      A       127.0.0.1
mvs03a          IN      A       172.16.250.3
mvs28a          IN      A       172.16.252.28
mvs39a          IN      A       172.16.232.39
TNRAL           IN      CNAME   tnral.ralplex1.buddha.ral.ibm.com.
TESTRAL         IN      CNAME   testral.ralplex1.buddha.ral.ibm.com.      1
TESTRALMT       IN      CNAME   testralmt.ralplex1.buddha.ral.ibm.com.    1
mvs03a.ralplex1 IN      A       172.16.250.3                              2
mvs28a.ralplex1 IN      A       172.16.252.28                             2
mvs39a.ralplex1 IN      A       172.16.232.39                             2
ralplex1        IN      NS      mvs03a.ralplex1.buddha.ral.ibm.com.
ralplex1        IN      NS      mvs28a.ralplex1.buddha.ral.ibm.com.
ralplex1        IN      NS      mvs39a.ralplex1.buddha.ral.ibm.com.
```

Figure 79. External Name Server Domain Forward File

1 Shows CNAME definitions for the sysplex socket test applications. Our socket test applications were registered to WLM with these names.

2 The external name server was configured to use the VIPA addresses of the subdomain ralplex1 name servers.

8.2.2 DNS Configuration, Sysplex Primary Name Server

The sysplex primary name server that ran on T03ATCP was also configured with four primary statements in its boot file. In addition to the primary statements we coded a cache statement. Also, the cluster keyword was set in the boot file. See Figure 80.

All the name servers in the sysplex were started with the parameter -l 3 in the PARM field of the procedure. This parameter set the time to live to three seconds. After the external name server sends a DNS request to the sysplex name server, the sysplex name server will send a DNS response with TTL set to 3. The external name server will then keep the entry in its cache for three seconds. You can see this in our test output Figure 92 on page 106. We thought TTL set to 3 would be a good value to start with to avoid too much extra load on the network.

The following are the configuration files for the boot (Figure 80), forward (Figure 81), loopback reverse mapping (Figure 82 on page 98) and cache (Figure 83 on page 98) configurations:

```
;
;    /etc/named.boot for TO3ATCP
;
;  TYPE      DOMAIN                    HOST        FILE
;
directory    /etc/dnsdata
;
primary    ralplex1.buddha.ral.ibm.com           ralplex1.for  cluster
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          1
```

Figure 80. T03ATCP DNS Boot File

1 Refers to the cache configuration file. This file holds the information on root name servers needed to initialize the caches of the domain name servers.

```
;
;  /etc/dnsdata/ralplex1.for for TO3ATCP
;
$ORIGIN ralplex1.buddha.ral.ibm.com.
@ IN      SOA     mvs03a.ralplex1.buddha.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
mvs03a          IN      A       172.16.250.3
mvs28a          IN      A       172.16.252.28
mvs39a          IN      A       172.16.232.39
```

Figure 81. T03ATCP Domain Forward File

```
;
;    /etc/dnsdata/mvs03a.lbk for T03ATCP
;
@     IN    SOA     mvs03a.ralplex1.buddha.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.buddha.ral.ibm.com.
1            IN  PTR     loopback.
128          IN  PTR     ralplex1.buddha.ral.ibm.com.                    1
```

Figure 82. T03ATCP Loopback Reverse Mapping File

1 This entry, called the Sysplex Loopback Address (SLA) is used by some client/server applications to make it possible to connect to the same server instance after interruption. Although we coded this entry, it was not used in any of our tests.

```
;
;  /etc/dnsdata/ralplex1.ca for T03ATCP
;
.                   3600000  IN  NS    mvs25o.buddha.ral.ibm.com.
mvs25o.buddha.ral.ibm.com. 3600000      A     9.24.104.125
```

Figure 83. T03ATCP Domain Cache File

8.2.3 DNS Configuration, Sysplex Secondary Name Servers

The name servers that ran on T28ATCP and T39ATCP were both secondary name servers for the ralplex1 subdomain. The secondary name servers imported their zone information from the primary name server on the T03ATCP stack; note that we pointed to the VIPA address of the primary name server. We also coded the cluster keyword. Please see Figure 84 for the boot file and Figure 85 on page 99 for the loopback reverse mapping file.

```
;
;    /etc/named.boot for T28ATCP
;
;  TYPE      DOMAIN                        FILE OR HOST
;
directory    /etc/dnsdata
;
secondary  ralplex1.buddha.ral.ibm.com     172.16.250.3  fback  cluster
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 84. T28ATCP DNS Boot File

```
;
;    /etc/dnsdata/mvs28a.lbk for T28ATCP
;
@     IN    SOA     mvs28a.ralplex1.buddha.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.buddha.ral.ibm.com.
1            IN  PTR     loopback.
128          IN  PTR     ralplex1.buddha.ral.ibm.com.
```

Figure 85. T28ATCP Loopback Reverse Mapping File

The configuration files for the name server that ran on the T39ATCP stack were almost the same, except that mvs28a was changed to mvs39a.

8.3 VTAM and I/O Definitions

We set up the 2216 MPC+ connection for our three 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 86 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 86. I/O Definitions for 2216 ESCON Channel

Since this is an MPC+ connection we needed to configure it via 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 87 contains the VTAM TRL definition on RA03, which was to be used by the T03ATCP stack:

```
         VBUILD TYPE=TRL
MO32216B TRLE   LNCTL=MPC,                              1
               MAXBFRU=9,
               READ=381,                                2
               WRITE=380,                               2
               MPCLEVEL=HPDT,
               REPLYTO=3.0
```

Figure 87. VTAM TRL Major Node for 2216 MPC+ on T03ATCP

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 87 on page 99. 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 88. 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).

8.4 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.

8.4.1 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 adapters. See Figure 89 on page 101. 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 89 (Part 1 of 2). 2216 Device Configuration Console Log

```
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 89 (Part 2 of 2). 2216 Device Configuration Console Log

In Figure 89 on page 101:

1 The talk 6 command enables configuration mode.

2 The values within [ ] are the defaults. You can confirm the default by presing 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.

8.4.2 2216 IP Configuration

After the device configuration we configured the IP protocol via the 2216 console. Figure 90 illustrates:

```
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 90. 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.

8.4.3 2216 OSPF Configuration

Next, we configured OSPF on the router as shown in Figure 91:

```
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 91. 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 interfaces 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.

8.5 Tests Performed

In this test round we made use of our new applications described in chapter 5.2, "Registering Your Own Applications with WLM" on page 57.

We ran our client programs from two different PCs. One of them was configured to use the external name server and the other one to use the sysplex name server. The PC that used the sysplex name server pointed to the sysplex primary name server in the T03ATCP stack. Both PCs were connected to the 9.24.106.0 subnet.

We started our external name server with a time to live of three seconds to illustrate the differences in operation between the sysplex DNS (which always returns TTL=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).

8.5.1 SYSPLEXW Test

We used the TNRAL host name as a server when we ran the SYSPLEXW EXEC. We invoked `sysplexw tnral 330` from the client with the external name server to see Figure 92 on page 106. Note the "deleted" entries where we have removed the responses that were identical to the previous line.

```
Application or Host Name                  IP Address      Time

tnral.ralplex1.buddha.ral.ibm.com         172.16.250.3    0.094000
tnral.ralplex1.buddha.ral.ibm.com         172.16.252.28   0.125000
: deleted 35 rows to 172.16.252.28
tnral.ralplex1.buddha.ral.ibm.com         172.16.232.39   0.125000
: deleted 35 rows to 172.16.232.39
tnral.ralplex1.buddha.ral.ibm.com         172.16.250.3    0.125000
: deleted 35 rows to 172.16.250.3
tnral.ralplex1.buddha.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 92. SYSPLEXW 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 93 on page 107 shows the corresponding output on the client using the sysplex name server. In this case the lines deleted looked like the first three repeated over and over again.

```
Application or Host Name                  IP Address      Time

tnral.ralplex1.buddha.ral.ibm.com         172.16.232.39   0.070000
tnral.ralplex1.buddha.ral.ibm.com         172.16.252.28   0.071000
tnral.ralplex1.buddha.ral.ibm.com         172.16.250.3    0.060000
tnral.ralplex1.buddha.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 93. SYSPLEXW Result - Sysplex Name Server

The client configured to use the sysplex name server got 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 was telling it.

8.5.2 Socket Test Application

Next, we ran a job (Figure 37 on page 64) 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.

On the client side we used the client test program described in 5.1, “Collecting Statistics Using REXX” on page 55 to perform our tests.

We executed `sysplex2 testral 1234 -c 350 -b 0.1` from both of our clients, one at a time. Figure 94 on page 108 shows the results for the client with the external name server, and Figure 95 on page 108 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            |
| 172.16.252.28  | 112            |
| 172.16.232.39  | 123            |
```

Figure 94. 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 95. Socket Application Client Result - Sysplex Name Server

As we expected, the results from the socket application were similar to the one from the SYSPLEXW REXX EXEC.

8.5.3 Socket 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 96 shows how the DNS/WLM dealt with the failure in the external name server case; Figure 97 on page 110 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 96. Socket Application Server Failure Case - External Name Server

After we stopped the application server on T03ATCP, 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 97. 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 9. Network Dispatcher

In this chapter we first present an introduction to the network dispatcher feature, then we provide some test configurations that we set up to try out the load balancing capabilities of the network dispatcher.

9.1 Overview of Network Dispatcher

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. Thus, it attempts to balance the traffic across all the servers according to the load on them. The forwarding is transparent to the users and to applications. Network dispatcher may be used for server applications such as HTTP, FTP, and Telnet; it can be used also 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 find automatically the optimal server to handle incoming requests.

The network dispatcher function does not use a DNS name server for load balancing. 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 a loopback address to the servers. This address is externally advertised 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 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.
- 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 and directs packets on the corresponding data connection to the same server. With passive 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.

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 the network dispatcher and not the server.
- If the clients, the servers and the NDR are all on the same subnetwork, the servers (in addition to the NDR) will respond to an ARP for the cluster address. NDR will not work under these circumstances; the only inbound packets for the servers′ cluster addresses must come from NDR for the function to operate correctly.
- 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.

9.2 Balancing TCP and UDP Traffic Using Network Dispatcher

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.

Network dispatcher supports advisors for FTP, HTTP, SMTP, NNTP, POP3, and Telnet, plus a TN3270 advisor that works with TN3270 servers in the IBM 221X routers and an MVS advisor that works with WLM on MVS systems. WLM manages the amount of workload on an individual MVS address space. Network dispatcher can use WLM to help load balance requests to MVS servers running OS/390 V1R3 or above.

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 (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 use only servers that use the supplied source IP address to respond to the client. Network dispatcher substitutes the client's source address for its own when forwarding IP datagrams to the server.

9.3 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), 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 host 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

9.3.1 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 obtained by pinging the hosts in question.

Switchover 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.

9.3.2 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 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.

9.3.3 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 switchover is performed across the alternate path.

9.3.4 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, 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.

9.4 Network Dispatcher Base Example

In this section we describe our implementation of NDR 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 show also some of the 2216 console log files we produced during the tests.

We used the same network configuration as described in Chapter 8, “Sysplex with OMPROUTE and VIPA” on page 89 when we did our network dispatcher test. The network diagram is shown in Figure 98 on page 116. The network dispatcher was configured on the channel-attached 2216.

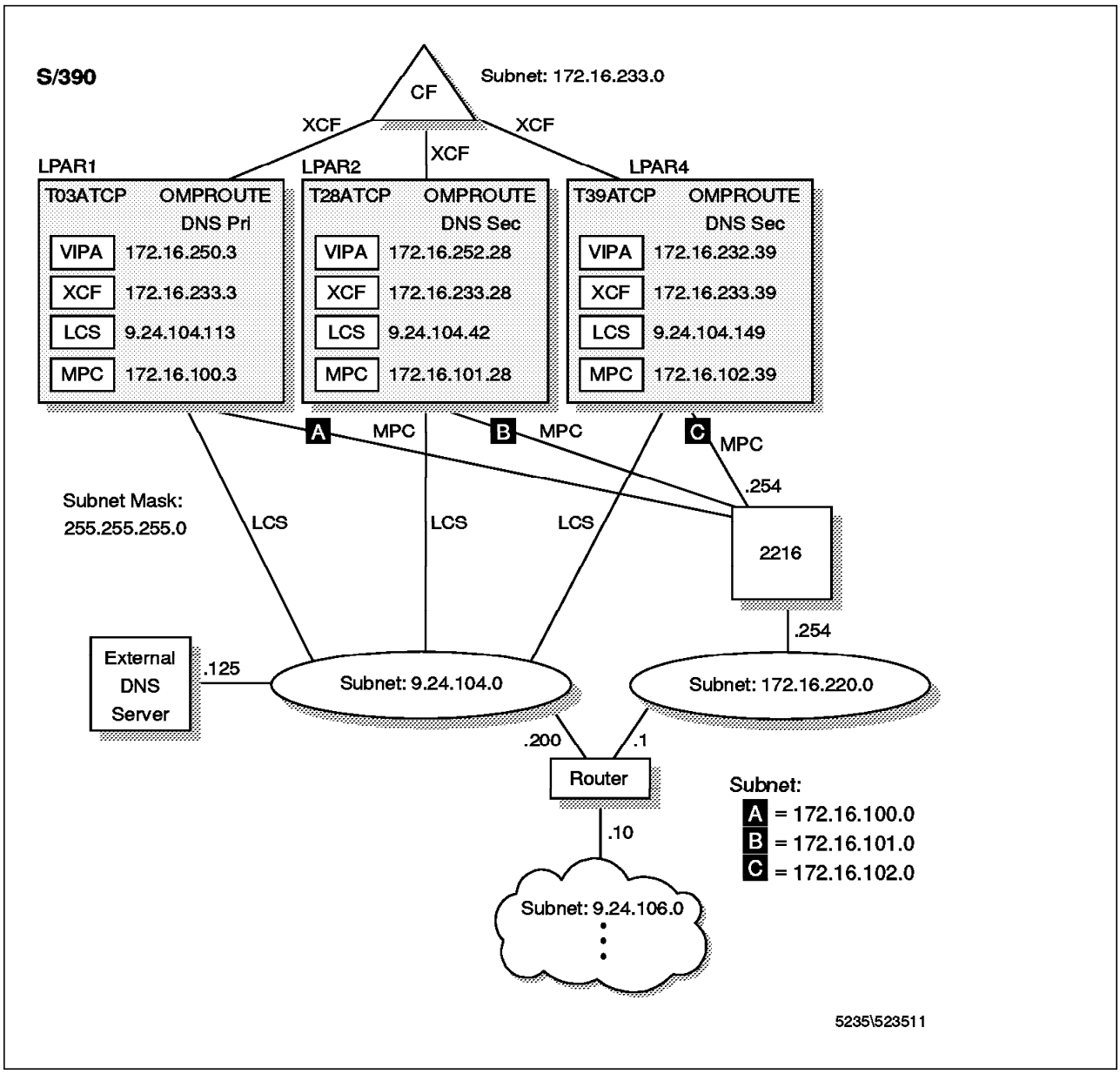


Figure 98. Network Dispatcher Configuration

9.4.1 2216 NDR Executor Configuration

We first configured the cluster using a Telnet session to the 2216; see Figure 99 on page 117.

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 99. 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 101 on page 119.

9.4.1.1 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 5.4, “SOCSRVR, a Simple Socket Server Program” on page 62 in all of our stacks. The servers were set to listen on port 1234.

Then we ran our client test program described in 5.1, “Collecting Statistics Using REXX” on page 55. 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 100. 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 101 on page 119 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 101. 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.

9.4.1.2 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 102 on page 120 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 102. 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 9.5, “NDR Protocol Advisors” on page 131 to get more information about protocol advisors and how they affect the distribution of the connections.

9.4.2 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 103 on page 121 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 103. 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).

9.4.2.1 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 104 on page 122.

```
+----------------------+------------------+----------------+--------+--------+
| 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 104. 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 you would expect; 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 105 on page 123.

```
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 105. 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 103 on page 121) 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.

9.4.2.2 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 106 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 106. 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 107 on page 125.

```
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 107. 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 be (eventually) 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 9.5, “NDR Protocol Advisors” on page 131.

9.4.3 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 103 on page 121. After the configuration changes were made we disabled the manager using DISABLE MANAGER, and reenabled it by issuing ENABLE MANAGER, from the NDR Config> prompt. The new manager parameters were now in use.

Figure 108 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 108. 2216 NDR 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 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 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 109. 2216 NDR Config List All Command Console Log

9.4.3.1 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 110 on page 128 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 110. 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 111:

```
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 111. 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 112 on page 129).

```
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 112. 2216 Report Advisor Console Log

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

9.4.3.2 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 113 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 113. NDR Base Sample Advisor Test 2

The outcome of this test was similar to the one in 9.4.2.2, “NDR Base Sample Manager Test 2” on page 124.

9.4.3.3 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 114 on page 130.

```
+----------------------+------------------+---------------+--------+--------+
| 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 114. 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 115 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 115. 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 116 on page 131 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 116. 2216 Console Log Quiesce/Unquiesce

9.5 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 117 on page 132 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 117. 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 118 on page 133 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 118. 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 119 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 119. 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.

9.6 NDR High Availability Example

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 120 on page 134 shows the network diagram with the second 2216 added.

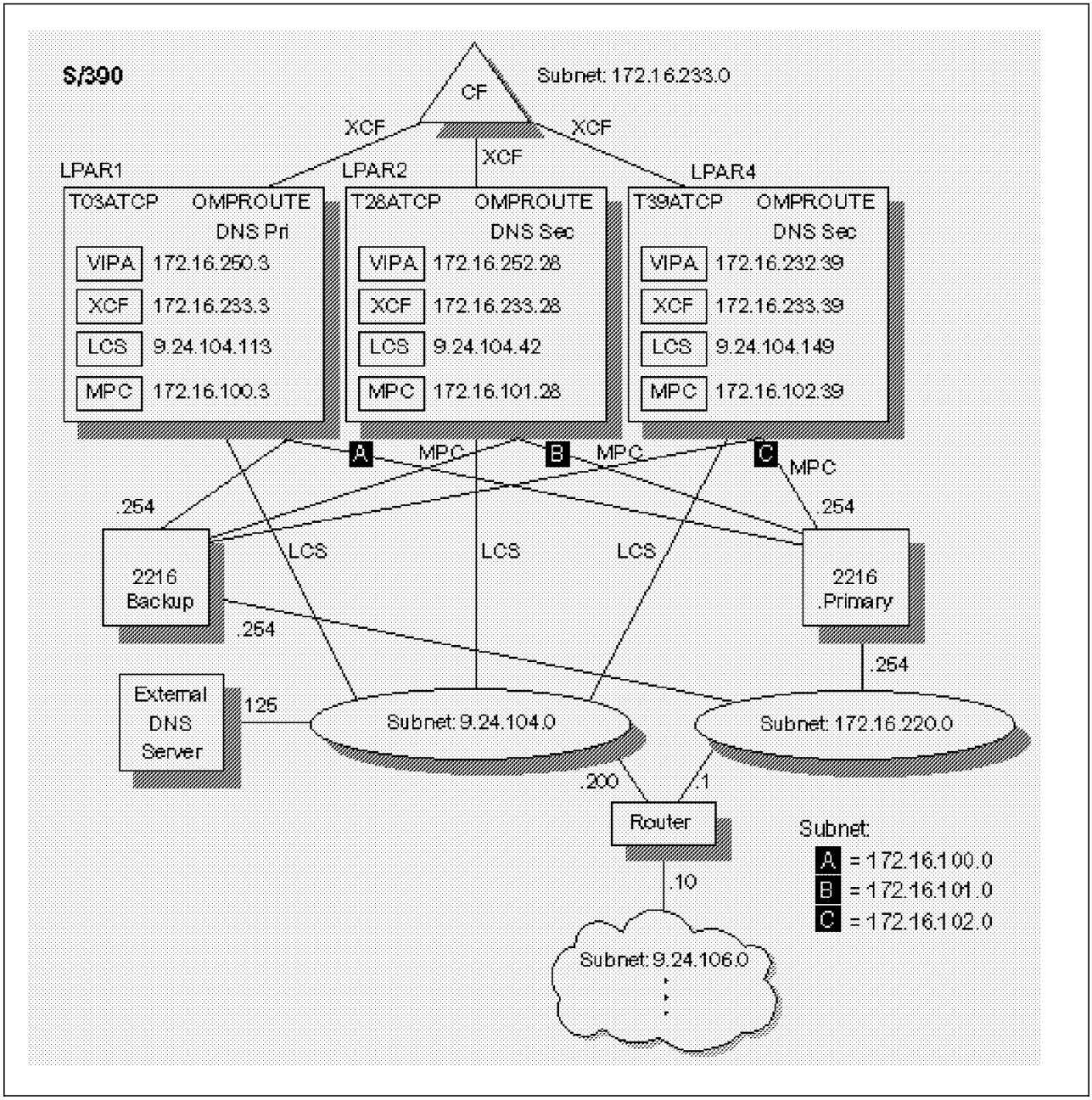


Figure 120. High-Availability Network Dispatcher Configuration

The TCP/IP definitions we made were very similar to those in Chapter 8, “Sysplex with OMPROUTE and VIPA” on page 89 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.

9.6.1 VTAM and IOCP Definitions

This time we needed only one IOCP definition; all LPARs used the same I/O addresses to reach the backup 2216. Figure 121 shows the IOCP setup:

```
                          --- 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
   85    CNC           SHR           CD        0200   3172           1              00   0F     0200,16      00    3172
```

Figure 121. IOCP Definitions for Backup 2216 ESCON Channel

We added and activated new TRL definitions in all of our sysplex VTAMs for the backup 2216.

9.6.2 TCP/IP Stack Configuration

We added and started new DEVICE, LINK, HOME and OSPF definitions for the backup 2216 in the TCP/IP stacks.

9.6.3 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 switchback 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 122 on page 136.

```
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 122. 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.

9.6.4 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 123 (primary) and Figure 124 on page 137 (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 ...
.....Heartbeat target : 172.16.103.253 Status : REACHABLE
.....Heartbeat target : 172.16.104.253 Status : REACHABLE
.....Heartbeat target : 172.16.105.253 Status : REACHABLE
.....Heartbeat target : 172.16.220.253 Status : REACHABLE

Dumping Reachability Status ...
```

Figure 123. 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 ...
.....Heartbeat target : 172.16.220.254 Status : REACHABLE
.....Heartbeat target : 172.16.100.254 Status : REACHABLE
.....Heartbeat target : 172.16.101.254 Status : REACHABLE
.....Heartbeat target : 172.16.102.254 Status : REACHABLE

Dumping Reachability Status ...
```

Figure 124. 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 5.6, “Loading the System” on page 66. This application was configured to listen on port 2345.

The client program syntax is described in 5.1, “Collecting Statistics Using REXX” on page 55. 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 125 on page 138.

```
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 125. 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 126:

```
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 126. 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 127 on page 139.

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

Dumping HeartBeat Status ...
.....Heartbeat target : 172.16.220.254 Status : UNREACHABLE
.....Heartbeat target : 172.16.100.254 Status : UNREACHABLE
.....Heartbeat target : 172.16.101.254 Status : UNREACHABLE
.....Heartbeat target : 172.16.102.254 Status : UNREACHABLE

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 127. 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 switchback 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 128 on page 140.

```
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 ...
.....Heartbeat target : 172.16.103.253 Status : REACHABLE
.....Heartbeat target : 172.16.104.253 Status : REACHABLE
.....Heartbeat target : 172.16.105.253 Status : REACHABLE
.....Heartbeat target : 172.16.220.253 Status : REACHABLE

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 128. 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 10. Dynamic VIPA and VIPA Takeover

OS/390 SecureWay Communications Server Release 8 introduces enhancements to the virtual IP addressing (VIPA) function. 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, CS for OS/390 has provided the following improvements to allow more systematic VIPA takeover operation:

- VIPA can be dynamically activated.
- VIPA can be taken over automatically by another stack.

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.

10.1 Overview of Dynamic VIPA and VIPA Takeover

In this section we explain the theory of VIPA and the new improvements.

10.1.1 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 or the network adapter or the 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.

CS for OS/390 addresses the requirement for 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. 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 now 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.

10.1.2 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. Therefore, CS for OS/390 Release 8 provides improvements to the VIPA takeover function. VIPA takeover builds on the VIPA concept, but automates the movement of the VIPA to an appropriate surviving stack. There are two forms of VIPA takeover:

- *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 going to the failed stack/node.
- *Dynamic VIPA for an application server* allows 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. This means that application instances on a failing cluster node may be distributed among many (or all) of the surviving nodes, reducing the amount of spare processing capacity that must be provided for each node that may participate in takeover.

To support these new enhancements, CS for OS/390 Release 8 now provides three ways to define a VIPA:

- Traditional static VIPA definition, as in Release 7
- Dynamic VIPA definition for automatic VIPA takeover
- Dynamic VIPA activation by application action

The last two ways are new in Release 8 and are configured via the new VIPADYnamic block in the TCP/IP profile.

10.1.3 VIPA Takeover

Automatic VIPA takeover requires that dynamic VIPAs (as opposed to traditional static VIPAs) be defined as having a normal ″home″ stack, and optionally one or more backup stacks. The stacks all share information on 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 active VIPA
- Which stack(s), and in what order, will participate in backup if the active one fails

When a failure of a stack hosting 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, to be passed on to the routing network via dynamic routing protocols.

When the original ″normal home″ stack is reactivated, the dynamic VIPA remains on the current backup stack as long as there are active connections to that dynamic VIPA on that stack. Thus, takeback is nondisruptive to existing connections at reactivation of the original stack; however, takeback may be delayed, potentially indefinitely.

The application-related form of dynamic VIPA allows the 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 a new IOCTL to the stack. The stack must be configured appropriately to permit activation of such a dynamic VIPA, with a VIPA subnet range defined to ensure that illegal 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 the sysplex. 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.

10.1.4 Benefits of Sysplex-Wide VIPA Takeover

When a stack or its underlying OS/390 fails, it does not make sense to try to restart the stack with the same configuration profile on a different OS/390 image, because the configuration almost always contains DEVICE, LINK, and HOME definitions for physical devices that do not exist as such in other OS/390 images. On the other hand, a VIPA is not associated with a real device and could be moved to another, functioning, stack in the event of a failure.

In the ideal world, this would not be needed, because clients would detect the failure and would go back to DNS (which would end up at the authoritative DNS/WLM) to get an address representing a server that is still available. However, there are large numbers of clients that ignore the zero time to live from DNS/WLM, and even some intermediate DNS implementations that cache irrespective of the time to live received from more authoritative DNSs. Finally, some networks do not want all DNS traffic to go all the way back to the sysplex, and are willing to trade failed reconnect attempts for reduced network bandwidth consumption. VIPA takeover is intended to address all of these scenarios.

VIPA takeover allows complete flexibility in 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.

10.2 Preparing Dynamic VIPA and VIPA Takeover

In this section we give an overview of how the dynamic VIPA functions can be defined; 10.4, “Examples of Dynamic VIPA and VIPA Takeover” on page 149 shows practical examples.

10.2.1 Automatic VIPA Takeover Configuration

Each VIPA has a preferred home stack and set of backup stacks, and 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.

Automatic VIPA backup requires you to define the primary and backup VIPA addresses to each stack that will participate; everything else is automatic. Figure 129 on page 144 shows the concept:

```
VIPADynamic
    VIPADEFine   address_mask   vipa_address1 vipa_address2 ...
    VIPABackup   rank           vipa_address1 vipa_address2 ...
 ENDVIPADynamic
```

Figure 129. Definition Format for VIPA Backup

The new VIPA-related parameters are coded in a VIPADynamic block in the TCP/IP profile as shown. 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. 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. Instead, there is a new parameter, the *rank* value. Valid values are from 0 to 255, and 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.

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

VIPADELete may also be used to delete an application-related dynamic VIPA established via BIND to a specific address or via IOCTL.

10.2.2 Dynamic VIPA Configuration for an Application Instance

Activation of a dynamic VIPA associated with a specific application instance occurs only via an application program′s API call, by either of the following ways:

- By the application issuing BIND to that particular IP address
- By a utility, EZBXFDVP, issuing a new IOCTL to the stack

Since the VIPA address comes from the application in this case, we do not define it in the TCP/IP profile. However, we must ensure that the addresses defined correspond to our own IP addressing scheme, so we use the VIPARange statement in the TCP/IP profile as shown in Figure 130:

```
VIPADynamic
     VIPARange    DEFINE  address_mask  network_prefix
 ENDVIPADynamic
```

Figure 130. Definition Format for Dynamic VIPA

The VIPARange statement defines an IP subnetwork using the network address (prefix) and the subnet mask. If there is no VIPARange statement corresponding to the address requested in an application call, the call is rejected. Multiple application-defined VIPA addresses can be active on a stack.

An application may issue a BIND to INADDR_ANY to accept connection requests to any IP address associated with the stack. In that case it is not possible to determine which VIPA should be associated with it. The solution is to modify the application to BIND to a specific address or to use the utility EZBXFDVP (see 10.2.3, “Modifying an Application for Dynamic VIPA” on page 145).

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. There is no configuration difference between activation by BIND or IOCTL, and the same VIPARange may be used for both if desired. Since the same VIPA 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.

Once activated on a stack via BIND or IOCTL, a dynamic VIPA remains active unless the VIPA is moved to another stack. The system operator may delete an active dynamic VIPA by issuing VARY OBEY with a profile containing a VIPADELete statement referencing the specific VIPA. To remove a whole VIPARANGE, the VIPARange DELETE statement may be used.

10.2.3 Modifying an Application for Dynamic VIPA

If the application can be modified, change the target address for the BIND from INADDR_ANY to a specific address for dynamic VIPA. In 5.4.1, “Modifying SOCSRVR for Dynamic VIPA” on page 64 we show how we did this for the sample Sockets application used in many of our tests.

To address the case where the application that desires to BIND(INADDR_ANY) cannot be modified, CS for OS/390 Release R8 provides a utility EZBXFDVP to create a dynamic VIPA using the IOCTL call. The utility can be initiated via JCL, from the OMVS command line, or from a shell script. Figure 131 is our sample procedure to invoke the utility:

```
//TCPDVP   PROC
//TCPDVP   EXEC PGM=EZBXFDVP,REGION=OK,TIME=1440,PARM='-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 131. Sample JCL to Run EZBXFDVP

The utility expects a parameter specifying the VIPA to be activated. It may also be used to delete a VIPA, as an alternative to VARY OBEY 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 address 172.16.240.193. Activation of the dynamic VIPA will succeed as long as the desired IP address is not claimed by one of the other stacks as an IP address for a physical interface or a static VIPA, or defined via VIPADEFine or VIPABackup in a VIPADynamic block.

Note that the issuer must be APF authorized or have root authority. If the dynamic VIPA was activated earlier on another stack via BIND or IOCTL, the VIPA will be deleted on the other stack, even if this means that existing connections are broken.

10.3 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.

10.3.1 Display Sysplex Command

A new display command D TCPIP,<tcpipjobname>,SYSplex,VIPADyn is available to show you the status of the dynamic VIPAs (both application related and takeover) in the sysplex. Figure 132 shows an example with VIPAs configured via VIPADEFine and VIPABackup. The origin (definition statement) and status of each dynamic VIPA on the stack (TCPNAME) and system (MVSNAME) in the sysplex are shown. The RANK value indicates which of the backup stacks will be chosen if the stack on which the dynamic VIPA is active is stopped. The system with the highest rank is the one that will take over the dynamic VIPA.

```
D TCPIP,T28ATCP,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V2R8
VIPA DYNAMIC DISPLAY FROM T28ATCP  AT RA28
IPADDR: 172.16.240.28  LINKNAME: VIPLAC1OF01C
  ORIGIN: VIPADEFINE
  TCPNAME   MVSNAME   STATUS  RANK  ADDRESS MASK     NETWORK PREFIX
  --------  --------  ------  ----  ---------------  ---------------
  T28ATCP   RA28      ACTIVE        255.255.255.192  172.16.240.0
  T39ATCP   RA39      BACKUP  050
IPADDR: 172.16.240.39
  ORIGIN: VIPABACKUP
  TCPNAME   MVSNAME   STATUS  RANK  ADDRESS MASK     NETWORK PREFIX
  --------  --------  ------  ----  ---------------  ---------------
  T39ATCP   RA39      ACTIVE        255.255.255.192  172.16.240.0
  T28ATCP   RA28      BACKUP  100
3 OF 3 RECORDS DISPLAYED
```

Figure 132. Display VIPA Backup Configuration in the Sysplex

Figure 133 on page 147 is a similar example showing dynamic VIPAs created by BIND or IOCTL:

```
D TCPIP,T28ATCP,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V2R8
VIPA DYNAMIC DISPLAY FROM T28ATCP  AT RA28
IPADDR: 172.16.240.193  LINKNAME: VIPLAC10F0C1                          1
  ORIGIN: VIPARANGE IOCTL
  TCPNAME   MVSNAME   STATUS  RANK  ADDRESS MASK     NETWORK PREFIX
  --------  --------  ------  ----  ---------------  ---------------
  T28ATCP   RA28      ACTIVE        255.255.255.192  172.16.240.192
IPADDR: 172.16.240.194                                                  2
  TCPNAME   MVSNAME   STATUS  RANK  ADDRESS MASK     NETWORK PREFIX
  --------  --------  ------  ----  ---------------  ---------------
  T39ATCP   RA39      ACTIVE        255.255.255.192  172.16.240.192
IPADDR: 172.16.240.200  LINKNAME: VIPLAC10F0C8                          3
  ORIGIN: VIPARANGE BIND
  TCPNAME   MVSNAME   STATUS  RANK  ADDRESS MASK     NETWORK PREFIX
  --------  --------  ------  ----  ---------------  ---------------
  T28ATCP   RA28      ACTIVE        255.255.255.192  172.16.240.192
9 OF 9 RECORDS DISPLAYED
```

Figure 133. Display Dynamic VIPA in the Sysplex

The ORIGIN field shows how the dynamic VIPA was created. Here 1 is a dynamic VIPA activated via IOCTL on this stack, whereas 3 is a dynamic VIPA activated by BIND on this stack. 2 is actually a VIPA activated by IOCTL on the other stack, but no origin is shown because this stack has no knowledge of how the address was created.

10.3.2 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:

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

all have a new section in the CONFIG (-f) report and 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 of the CONFIG report is displayed whether there are any dynamic VIPAs known to this system (see 1 in Figure 134 on page 148). The Dynamic VIPA Information section is displayed only when there are dynamic VIPAs known to this system. The VIPA Range section, displayed only if a VIPARANGE statement was processed in this stack's profile (or obey file), indicates only that a range was configured. It does not indicate whether any BIND or IOCTL has actually created any address within the range.

```
D TCPIP,T28ATCP,N,CONFIG
EZZ2500I NETSTAT CS V2R8 T28ATCP
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:           10
UDP CONFIGURATION TABLE:
DEFAULTRCVBUFSIZE: 00016384  DEFAULTSNDBUFSIZE: 00016384
CHECKSUM:          00000001  LOGPROTOERR:       01
UDPFLAGS:          00
IP CONFIGURATION TABLE:
FORWARDING: YES    TIMETOLIVE: 00060  RSMTIMEOUT:  00060
FIREWALL:   00000  ARPTIMEOUT: 01200  MAXRSMSIZE:  65535
IGREDIRECT: 00001  SYSPLXROUT: 00001  DOUBLENOP:   00000
STOPCLAWER: 00000  SOURCEVIPA: 00001  VARSUBNET:   00001
MULTIPATH:  CONN   PATHMTUDSC: 00000
DYNAMICXCF: 00001
  IPADDR: 172.16.233.28    SUBNET: 255.255.255.0    METRIC: 01
SMF PARAMETERS:
INITTYPE: 00  TERMTYPE: 00  CLIENTTYPE: 00  TCPIPSTATS: 00
GLOBAL CONFIGURATION INFORMATION:                                  1
TCPIPSTATS: 01          ARMRESTART: 01
DYNAMIC VIPA INFORMATION:
  VIPA BACKUP:
    IP ADDRESS       RANK
    ----------       ----
    172.16.240.3     000002
    172.16.240.39    000001
  VIPA DEFINE:
    IP ADDRESS       ADDRESSMASK
    ----------       -----------
    172.16.240.28    255.255.255.192
  VIPA RANGE:
    ADDRESSMASK      IP ADDRESS
    -----------      ----------
    255.255.255.192  172.16.240.192
    ----------       -----------
    201.2.10.11      255.255.255.192
    201.2.10.12      255.255.255.192
  VIPA Range:
    AddressMask      IP Address
    -----------      ----------
    255.255.255.192  201.2.10.192
```

Figure 134. Display NETSTAT, CONFIG Command

Figure 135 on page 149 shows the results of a Display N,VIPADYN command. The same display can be achieved using onetstat -v from a USS prompt.

```
D TCPIP,T28ATCP,N,VIPADYN
EZZ2500I NETSTAT CS V2R8 T28ATCP 985
IP ADDRESS       ADDRESSMASK      STATUS  ORIGINATION
172.16.240.3     <NONE>           BACKUP  VIPABACKUP
172.16.240.28    255.255.255.192  ACTIVE  VIPADEFINE
172.16.240.39    <NONE>           BACKUP  VIPABACKUP
172.16.240.193   255.255.255.192  ACTIVE  VIPARANGE IOCTL
172.16.240.200   255.255.255.192  ACTIVE  VIPARANGE BIND
5 OF 5 RECORDS DISPLAYED
```

Figure 135. Display NETSTAT,VIPADYN(-v) Commands

10.4 Examples of Dynamic VIPA and VIPA Takeover

In this section we demonstrate examples of VIPA takeover using three different methods:

- VIPA takeover. VIPA definitions are coded on multiple stacks, so that the failure of one stack results in the takeover of its VIPA address on another.
- Application-related dynamic VIPA. An application issues BIND to a specific VIPA address, and the stacks (which have suitably coded range definitions) ensure that only one stack owns the VIPA at any one time.
- Application-related dynamic VIPA. An application issues BIND to INADDR_ANY and cannot be modified. Thus, we run the IOCTL utility to associate a dynamic VIPA address with it when it needs to be moved to another stack.

The base network configuration is the same as the OMPROUTE network in Figure 70 on page 90.

10.4.1 Automatic VIPA Takeover

For this test, we used two TCP/IP stacks: T28ATCP and T39ATCP. On each stack we configured a primary VIPA address and a backup address for the other stack's primary VIPA. The test we performed was as follows:

1. Define the primary VIPAs: 172.16.240.39 on T39ATCP and 172.16.240.28 on T28ATCP.
2. Define the backup VIPAs: 172.16.240.39 on T28ATCP and 172.16.240.28 on T39ATCP.
3. Start our server applications on both of the stacks, and establish connections from a client to the server on T39ATCP using the VIPA 172.16.240.39.
4. Log on to the TN3270 server on T39ATCP using the VIPA.
5. Stop the T39ATCP stack.
6. Make sure the backup stack T28ATCP takes over the VIPA 172.16.240.39.
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 T39ATCP stack and the server application.
9. Check that the VIPA moves back to the original stack.

Figure 136 on page 150 shows our VIPA definitions on T28ATCP, and Figure 137 on page 150 shows the corresponding definitions on T39ATCP.

```
VIPADynamic
   VIPABackup    100                172.16.240.39
   VIPADEFine    255.255.255.192    172.16.240.28
ENDVIPADYNAMIC
```

Figure 136. Dynamic VIPA Definition for T28ATCP

```
VIPADynamic
   VIPABackup    2                  172.16.240.28
   VIPADEFine    255.255.255.192    172.16.240.39
ENDVIPADYNAMIC
```

Figure 137. Dynamic VIPA Definition for T39ATCP

Before we broke anything, we used the Display SYSplex command on T28ATCP to see what dynamic VIPAs were known. Figure 138 matches our definitions:

```
D TCPIP,T28ATCP,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V2R8
VIPA DYNAMIC DISPLAY FROM T28ATCP  AT RA28
IPADDR: 172.16.240.28  LINKNAME: VIPLAC1OF01C
  ORIGIN: VIPADEFINE
  TCPNAME   MVSNAME   STATUS  RANK  ADDRESS MASK     NETWORK PREFIX
  --------  --------  ------  ----  --------------  --------------
  T28ATCP   RA28      ACTIVE        255.255.255.192  172.16.240.0
  T39ATCP   RA39      BACKUP  002
IPADDR: 172.16.240.39
  ORIGIN: VIPABACKUP
  TCPNAME   MVSNAME   STATUS  RANK  ADDRESS MASK     NETWORK PREFIX
  --------  --------  ------  ----  --------------  --------------
  T39ATCP   RA39      ACTIVE        255.255.255.192  172.16.240.0
  T28ATCP   RA28      BACKUP  100
2 OF 2 RECORDS DISPLAYED
```

Figure 138. Display VIPA Definition for T28ATCP before T39ATCP Stack Failure

We also displayed the OSPF routing table on T28ATCP, as shown in Figure 139 on page 151. Note the presence of the local dynamic VIPA **1** with its stack-generated link name, and the remote VIPA **2** on T39ATCP.

```
D TCPIP,T28ATCP,OMPROUTE,RTTABLE
EZZ7847I ROUTING TABLE
TYPE   DEST NET         MASK      COST    AGE     NEXT HOP(S)
STAT*  172.16.233.3     FFFFFFFF  0       240     172.16.233.28
STAT*  172.16.233.4     FFFFFFFF  0       240     172.16.233.28
 SPF   172.16.233.28    FFFFFFFF  0       238     EZAXCF39
STAT*  172.16.233.29    FFFFFFFF  0       240     172.16.233.28
STAT*  172.16.233.39    FFFFFFFF  0       240     172.16.233.28
 DIR*  172.16.240.28    FFFFFFFF  1       240     VIPLAC1OF01C  1
SPE2   172.16.240.39    FFFFFFFF  1       147     172.16.233.39 2
SPE2   172.16.250.0     FFFFFF00  1       167     172.16.233.3
SPE2   172.16.250.3     FFFFFFFF  1       167     172.16.233.3
SPE2   172.16.251.4     FFFFFFFF  1       167     172.16.233.4
 DIR*  172.16.252.0     FFFFFF00  1       240     172.16.252.28
 DIR*  172.16.252.28    FFFFFFFF  1       240     VIPA28A
```

Figure 139. Display OMPROUTE Table before T39ATCP Stack Failure

We then stopped the TCP/IP stack T39ATCP, which caused the servers (TN3270 and our own application) 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 172.16.240.39 was T28ATCP. Therefore, T28ATCP defined and activated this VIPA address itself. The console log on RA28 confirmed this, as shown in Figure 140:

```
EZZ8301I VIPA 172.16.240.39 TAKEN OVER FROM T39ATCP ON RA39
EZZ4323I CONNECTION TO 172.16.233.39 CLEARED FOR DEVICE RA39M
EZZ4323I CONNECTION TO 172.16.233.39 CLEARED FOR DEVICE RA39M
```

Figure 140. 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 141 on page 152:

```
D TCPIP,T28ATCP,N,HOME
EZZ2500I NETSTAT CS V2R8 T28ATCP
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
172.16.252.28    VIPA28A          P
9.24.104.42      TR1
172.16.101.28    M282216B
172.16.104.28    M282216C
172.16.220.50    LOOPBACK
172.16.220.51    LOOPBACK
172.16.220.52    LOOPBACK
172.16.233.28    EZAXCF39
172.16.233.28    EZAXCF03
172.16.233.28    EZASAMEMVS
172.16.240.28    VIPLAC1OF01C
172.16.240.39    VIPLAC1OF027                                          1
127.0.0.1        LOOPBACK
13 OF 13 RECORDS DISPLAYED

D TCPIP,T28ATCP,SYSPLEX,VIPAD
EZZ8260I SYSPLEX CS V2R8
VIPA DYNAMIC DISPLAY FROM T28ATCP  AT RA28
IPADDR: 172.16.240.28  LINKNAME: VIPLAC1OF01C
  ORIGIN: VIPADEFINE
  TCPNAME   MVSNAME   STATUS  RANK  ADDRESS MASK     NETWORK PREFIX
  --------  --------  ------  ----  ---------------  ---------------
  T28ATCP   RA28      ACTIVE        255.255.255.192  172.16.240.0
IPADDR: 172.16.240.39  LINKNAME: VIPLAC1OF027                         2
  ORIGIN: VIPABACKUP
  TCPNAME   MVSNAME   STATUS  RANK  ADDRESS MASK     NETWORK PREFIX
  --------  --------  ------  ----  ---------------  ---------------
  T28ATCP   RA28      ACTIVE        255.255.255.192  172.16.240.0
2 OF 2 RECORDS DISPLAYED

D TCPIP,T28ATCP,N,CON
EZZ2500I NETSTAT CS V2R8 T28ATCP
USER ID  CONN     LOCAL SOCKET           FOREIGN SOCKET         STATE
GORGMT2  00000322 172.16.240.39..2345    9.24.106.197..1695     FINWAIT   3
GORGMT2  00000321 172.16.240.39..2345    9.24.106.197..1694     FINWAIT   3
GORGMT2  00000094 0.0.0.0..2345          0.0.0.0..0             LISTEN
GOWLMRG2 0000002D 0.0.0.0..1234          0.0.0.0..0             LISTEN
T28ATCP  000002DF 172.16.240.39..23      9.24.106.197..1274     ESTABLS   4
:

D TCPIP,T28ATCP,N,SOCKETS
EZZ2500I NETSTAT CS V2R8 T28ATCP
SOCKETS INTERFACE STATUS:
TYPE   BOUND TO              CONNECTED TO           STATE    CONN
NAME: GORGMT2   SUBTASK: 007E7388
STREAM 172.16.240.39..2345    9.24.106.197..1597     TIMEWAIT 00000311
:
STREAM 172.16.240.39..2345    9.24.106.197..1849     ESTABLSH 00000347
STREAM 172.16.240.39..2345    9.24.106.197..1842     ESTABLSH 00000346
:
```

Figure 141. Displays after VIPA Takeover on T28ATCP

In this display:

1 Shows the VIPA address 172.16.240.39 in T28ATCP'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 T28ATCP. The recovered address 172.16.240.39 was defined as VIPABackup but is now active.

3 Shows that new connections to our server application were connected to the instance on T28ATCP.

4 Shows that the TN3270 connection was dropped and reconnected to its new home T28ATCP.

Our client on the workstation reported (Figure 142) that all connections were successfully established, but that the actual IP address returned by the server is now 172.16.252.28. This address is the static VIPA for T28ATCP, and is returned here because of the SOURCEVIPA definition in T28ATCP's profile.

```
sysplex2 172.16.240.39 2345 -c 5 -b 0.1 -t 1
+---------------------+-----------------+----------------+--------+--------+
| Hostname            | Resolved Addr   | Connected Addr | Resolv | Connec |
+---------------------+-----------------+----------------+--------+--------+
| 172.16.240.39       | 172.16.240.39   | 172.16.252.28  | 0.0910 | 9.9840 |
| 172.16.240.39       | 172.16.240.39   | 172.16.252.28  | 0.0800 | 1.0120 |
| 172.16.240.39       | 172.16.240.39   | 172.16.252.28  | 0.0700 | 1.0110 |
| 172.16.240.39       | 172.16.240.39   | 172.16.252.28  | 0.0810 | 1.0110 |
| 172.16.240.39       | 172.16.240.39   | 172.16.252.28  | 0.0700 | 1.0120 |

+----------------+---------------+
| Resolved  Addr | Resolved  Count|
+----------------+---------------+
| 172.16.240.39  | 5             |

+----------------+---------------+
| Connected Addr | Connected Count|
+----------------+---------------+
| 172.16.252.28  | 5             |
```

Figure 142. Output of Socket Application Using the VIPA 172.16.240.39

Some time later, we restarted the failed TCP/IP stack T39ATCP and the server application on T39ATCP. At that time T28ATCP not only had the VIPA 172.16.240.39 active, but also had some active connections through the VIPA. T39ATCP learned this via XCF, so it did not activate the primary dynamic VIPA; instead, T28ATCP set a flag to indicate that another stack should take over 172.16.240.39 when all connections were terminated. T39ATCP then declared itself as a backup stack for that address with a rank of 255. This put T39ATCP at the head of the backup chain, since the highest rank that can be configured is 254. Figure 143 on page 154 shows the results of a Display SYSplex,VIPAD command on T28ATCP.

```
D TCPIP,T28ATCP,SYSPLEX,VIPAD
EZZ8260I SYSPLEX CS V2R8
VIPA DYNAMIC DISPLAY FROM T28ATCP  AT RA28
IPADDR: 172.16.240.28  LINKNAME: VIPLAC1OF01C
  ORIGIN: VIPADEFINE
  TCPNAME   MVSNAME   STATUS  RANK  ADDRESS MASK     NETWORK PREFIX
  --------  --------  ------  ----  --------------  --------------
  T28ATCP   RA28      ACTIVE        255.255.255.192  172.16.240.0
  T39ATCP   RA39      BACKUP  002
IPADDR: 172.16.240.39  LINKNAME: VIPLAC1OF027
  ORIGIN: VIPABACKUP
  TCPNAME   MVSNAME   STATUS  RANK  ADDRESS MASK     NETWORK PREFIX
  --------  --------  ------  ----  --------------  --------------
  T28ATCP   RA28      ACTIVE        255.255.255.192  172.16.240.0
  T39ATCP   RA39      BACKUP  255
4 OF 4 RECORDS DISPLAYED
```

Figure 143. Display SYSplex,VIPAD after T39ATCP Restart

When the last connection to the VIPA 172.16.240.39 on T28ATCP was closed, T28ATCP deleted the VIPA, reverted to backup status, and notified the other stacks. T39ATCP noted that the VIPA had ceased to be active, and that it was the first stack in the backup chain. So it defined and activated the VIPA. The sysplex now reverted to its pre-failure condition.

On RA28, we saw the message:

```
EZZ8303I VIPA 172.16.240.39 GIVEN TO T39ATCP ON RA39
```

On RA39, the following message appeared:

```
EZZ8302I VIPA 172.16.240.39 TAKEN FROM T28ATCP ON RA28
```

Finally, we issued Display SYSplex,VIPAD again on T28ATCP (Figure 144) to confirm that we were back where we had started:

```
D TCPIP,T28ATCP,SYSPLEX,VIPAD
EZZ8260I SYSPLEX CS V2R8
VIPA DYNAMIC DISPLAY FROM T28ATCP  AT RA28
IPADDR: 172.16.240.28  LINKNAME: VIPLAC1OF01C
  ORIGIN: VIPADEFINE
  TCPNAME   MVSNAME   STATUS  RANK  ADDRESS MASK     NETWORK PREFIX
  --------  --------  ------  ----  --------------  --------------
  T28ATCP   RA28      ACTIVE        255.255.255.192  172.16.240.0
  T39ATCP   RA39      BACKUP  002
IPADDR: 172.16.240.39
  ORIGIN: VIPABACKUP
  TCPNAME   MVSNAME   STATUS  RANK  ADDRESS MASK     NETWORK PREFIX
  --------  --------  ------  ----  --------------  --------------
  T39ATCP   RA39      ACTIVE        255.255.255.192  172.16.240.0
  T28ATCP   RA28      BACKUP  100
4 OF 4 RECORDS DISPLAYED
```

Figure 144. Displays after VIPA Giveback to T39ATCP

10.4.2 Dynamic VIPA for Application BINDing to a Specific Address

For our second test, we used our socket server application. The sequence of actions was as follows:

1. Define VIPARANGE in a VIPADynamic block on both of the stacks.
2. Start the server application on the T39ATCP stack.
3. Start the client application using the dynamic VIPA address.
4. Stop the T39ATCP stack.
5. Start the application server on the T28ATCP stack.
6. Check the dynamic VIPA address on T28ATCP.
7. Restart T39ATCP.
8. Cancel the server application on T28ATCP.
9. Restart the server application on T39ATCP.
10. Check the dynamic VIPA on T39ATCP.

To create a dynamic VIPA for the socket application, we coded the profile statements shown in Figure 145 on each stack:

```
VIPADynamic
   VIPARange    DEFINE  255.255.255.192  172.16.240.192
ENDVIPADYNAMIC
```

Figure 145. Definition for Dynamic VIPA

When we started the server application on T39ATCP, it issued BIND to the IP address 172.16.240.200. Since this address is within the range defined by VIPARANGE and was not active elsewhere in the sysplex, a VIPA with the address was dynamically created and activated. The display in Figure 146 confirms this.

```
D TCPIP,T39ATCP,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V2R8
VIPA DYNAMIC DISPLAY FROM T39ATCP  AT RA39
IPADDR: 172.16.240.200  LINKNAME: VIPLAC10F0C8
  ORIGIN: VIPARANGE BIND
  TCPNAME   MVSNAME   STATUS  RANK  ADDRESS MASK     NETWORK PREFIX
  --------  --------  ------  ----  ---------------  ---------------
  T39ATCP   RA39      ACTIVE        255.255.255.192  172.16.240.192
1 OF 1 RECORDS DISPLAYED
```

Figure 146. Display Dynamic VIPA on T39ATCP

Next, we started the client application pointing to the new dynamic VIPA address. Then we stopped the T39ATCP stack along with the server application. T28ATCP was notified of the failure of T39ATCP and took note of the fact that 172.16.240.200 was no longer active anywhere.

Then we started the server application on T28ATCP. T28ATCP had already been configured with the appropriate VIPARange for the application, so the dynamic VIPA was defined and activated on T28ATCP. See Figure 147 on page 156 for a display that shows this.

```
D TCPIP,T28ATCP,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V2R8
VIPA DYNAMIC DISPLAY FROM T28ATCP  AT RA28
IPADDR: 172.16.240.200  LINKNAME: VIPLAC10F0C8
  ORIGIN: VIPARANGE BIND
  TCPNAME   MVSNAME   STATUS  RANK  ADDRESS MASK     NETWORK PREFIX
  --------  --------  ------  ----  ---------------  ---------------
  T28ATCP   RA28      ACTIVE        255.255.255.192  172.16.240.192
1 OF 1 RECORDS DISPLAYED
```

Figure 147. Display Dynamic VIPA on T28ATCP

Some time later, we restarted T39ATCP. Then, before restarting the server application on T39ATCP, we canceled the one on T28ATCP. This resulted in the deletion of the dynamic VIPA address on T28ATCP. Note that only one stack may own a dynamic VIPA at any one time, and a BIND will create a dynamic VIPA only when there are no active connections to that same dynamic VIPA on another stack.

We restarted the server application on T39ATCP and found that the dynamic VIPA was again active on T39ATCP.

Figure 148 shows the results of the socket application before and after the T39ATCP stack failure. You can see that the connected host address for the dynamic VIPA, 172.16.240.200, was changed from T39ATCP to T28ATCP when T39ATCP failed and the server application was started on T28ATCP.

```
sysplex2 172.16.240.200 8062 -c 30 -b 0.1 -t 1

+----------------------+------------------+----------------+--------+--------+
| Hostname             | Resolved Addr    | Connected Addr | Resolv | Connec |
+----------------------+------------------+----------------+--------+--------+
| 172.16.240.200       | 172.16.240.200   | 172.16.232.39  | 1.3440 | 1.0790 |
| 172.16.240.200       | 172.16.240.200   | 172.16.232.39  | 0.1400 | 1.0000 |
Error on connecting socket to '172.16.240.200': EHOSTUNREACH
| 172.16.240.200       | 172.16.240.200   | 172.16.252.28  | 1.4220 | 1.0000 |
| 172.16.240.200       | 172.16.240.200   | 172.16.252.28  | 0.0630 | 1.0000 |
| :
| :

+----------------+----------------+
| Resolved  Addr | Resolved  Count|
+----------------+----------------+
| 172.16.240.200 | 30             |

+----------------+----------------+
| Connected Addr | Connected Count|
+----------------+----------------+
| 172.16.232.39  | 8              |
| 172.16.252.28  | 21             |
| EHOSTUNREACH   | 1              |
```

Figure 148. Socket Application at T39ATCP Failure

Figure 149 on page 157 is the output of the socket application before and after the server application was restarted on T39ATCP.

```
sysplex2 172.16.240.200 8062 -c 30 -b 0.1 -t 1

+----------------------+------------------+---------------+--------+--------+
| Hostname             | Resolved Addr    | Connected Addr | Resolv | Connec |
+----------------------+------------------+---------------+--------+--------+
| 172.16.240.200       | 172.16.240.200   | 172.16.252.28  | 0.8120 | 1.2650 |
| 172.16.240.200       | 172.16.240.200   | 172.16.252.28  | 0.1250 | 1.1250 |
| :
| 172.16.240.200       | 172.16.240.200   | 172.16.232.39  | 0.0310 | 22.984 |
| 172.16.240.200       | 172.16.240.200   | 172.16.232.39  | 0.0310 | 1.0150 |
| :
| :

+----------------+----------------+
| Resolved  Addr | Resolved  Count|
+----------------+----------------+
| 172.16.240.200 | 30             |

+----------------+----------------+
| Connected Addr | Connected Count|
+----------------+----------------+
| 172.16.232.39  | 16             |
| 172.16.252.28  | 14             |
```

Figure 149. Socket Application at T39ATCP and Server Restart

10.4.3 Dynamic VIPA for Application Using IOCTL

Our final scenario is the case where the application issues BIND to INADDR_ANY and cannot be modified to specify a given address. We used the IOCTL utility EZBXFDVP to activate a dynamic VIPA and make the socket application use it. Our sequence of actions was:

1. Define VIPARANGE in a VIPADynamic block on both of the stacks.
2. Create a dynamic VIPA 172.16.240.193 via IOCTL on T28ATCP.
3. Start the server application on T28ATCP.
4. Start the client application specifying the dynamic VIPA address.
5. Stop the T28ATCP stack.
6. Create the failed VIPA 172.16.240.193 by IOCTL on T39ATCP.
7. Start the server application on T39ATCP.
8. Check the VIPA on T39ATCP.
9. Restart T28ATCP.
10. Run the utility EZBXFDVP and restart the server application on T28ATCP.
11. Check the VIPA on T28ATCP.

To create a dynamic VIPA for the socket application, we defined the VIPARANGE in the TCP/IP profile for both of the stacks as illustrated in Figure 145 on page 155.

We then ran the utility EZBXFDVP as shown in Figure 131 on page 145 to activate the VIPA 172.16.240.193 on T28ATCP. Figure 150 on page 158 confirms that this has happened.

```
D TCPIP,T28ATCP,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V2R8
VIPA DYNAMIC DISPLAY FROM T28ATCP  AT RA28
IPADDR: 172.16.240.193  LINKNAME: VIPLAC1OF0C1
  ORIGIN: VIPARANGE IOCTL
  TCPNAME   MVSNAME   STATUS  RANK  ADDRESS MASK     NETWORK PREFIX
  --------  --------  ------  ----  ---------------  ---------------
  T28ATCP   RA28      ACTIVE        255.255.255.192  172.16.240.192
1 OF 1 RECORDS DISPLAYED
```

Figure 150. Displaying Dynamic VIPA on T28ATCP

We then started the server application on T28ATCP and invoked the client on our workstation specifying the new dynamic VIPA address 172.16.240.193 as follows: `sysplex2 172.16.240.193 1234 -c 30 -b 0.1 -t 1`

Next, we stopped the T28ATCP stack. T39ATCP was notified of the failure and took note of the fact that 172.16.240.193 was no longer active anywhere.

We ran the utility EZBXFDVP and activated the VIPA 172.16.240.193 on T39ATCP, followed by restarting the server application on T39ATCP. Figure 151 shows how the dynamic VIPA has moved to its new home:

```
D TCPIP,T39ATCP,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V2R8
VIPA DYNAMIC DISPLAY FROM T39ATCP  AT RA39
:
IPADDR: 172.16.240.193
  TCPNAME   MVSNAME   STATUS  RANK  ADDRESS MASK     NETWORK PREFIX
  --------  --------  ------  ----  ---------------  ---------------
  T39ATCP   RA39      ACTIVE        255.255.255.192  172.16.240.192
1 OF 1 RECORDS DISPLAYED
```

Figure 151. Displaying Dynamic VIPA on T39TCP

Some time later, we restarted the T28ATCP stack. To restart the server application associated with the VIPA on T28ATCP, we ran the utility again on T28ATCP to activate the VIPA there. Finally we started the server application on the stack.

Note that if the same dynamic VIPA is active on another stack, invoking the utility forces the other VIPA to be inactivated. In this case, TCP connections through the VIPA on T39ATCP were deleted as a part of moving the VIPA to T28ATCP, and new connection requests through the VIPA were addressed to T28ATCP.

Figure 152 on page 159 is the output of our client application.

```
sysplex2 172.16.240.193 1234 -c 30 -b 0.1 -t 1

+---------------------+-----------------+---------------+--------+--------+
| Hostname            | Resolved Addr   | Connected Addr | Resolv | Connec |
+---------------------+-----------------+---------------+--------+--------+
| 172.16.240.193      | 172.16.240.193  | 172.16.252.28  | 0.0620 | 0.0160 |
| 172.16.240.193      | 172.16.240.193  | 172.16.252.28  | 0.0470 | 0.0160 |
| :
| :
| 172.16.240.193      | 172.16.240.193  | 172.16.252.28  | 0.0780 | 0.0150 |
Error on connecting socket to '172.16.240.193': EHOSTUNREACH                    1
| 172.16.240.193      | 172.16.240.193  | 172.16.232.39  | 0.0460 | 0.0160 |    2
| 172.16.240.193      | 172.16.240.193  | 172.16.232.39  | 0.0310 | 0.0160 |
| :
| :
| 172.16.240.193      | 172.16.240.193  | 172.16.232.39  | 0.0310 | 0.0160 |
| 172.16.240.193      | 172.16.240.193  | 172.16.252.28  | 0.0310 | 0.0310 |    3
| 172.16.240.193      | 172.16.240.193  | 172.16.252.28  | 0.0310 | 0.0320 |
| :
| :
```

Figure 152. Socket Application Results

In this display, **1** shows the connection failure when T28ATCP was taken down. At **2**, the server was restarted on T39ATCP and the new connections through the VIPA were addressed to the T39ATCP stack. At **3**, T28ATCP and its server were restarted and the connections moved back to the T28ATCP stack.

Appendix A. Sample C Programs

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.

A.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);
}
```

A.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" );
}
```

A.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);
    }
}
```

A.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);
}
```

A.5 MTCSRVR Multitasking Socket 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;
}
```

A.6 MTCSUBT Subtask for the Multitasking Socket 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 B. REXX EXECs

In this section we show the three REXX EXECs used to invoke server functions in the sysplex.

B.1 OS/2 EXEC to Issue PING

```
/*Rexx - Exec to perform TCP/IP Sysplex validation and tracing */
call RxFuncAdd 'SysLoadFuncs','rexxutil','SysLoadFuncs'
call SysLoadFuncs
'@ECHO OFF'
/*                                                             */
/*    Syntax:  SYSPLEX  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 <> '' ] p1 = '?' Then pingname = p1
Else
  Do
    Say
    Say  'Syntax:  SYSPLEX  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' ]] mmv ]] ssv ]] Substr(therest,1,2) ]] '.dat'
pingfile = '\pf' ]] mmv ]] ssv ]] Substr(therest,1,2) ]] '.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 ]] '. 10 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 = '' 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
Start 'EPM '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
  Select
     When w1 = 'PING' Then
       Do
         thispingname = Substr(w2,1,Length(w2) -1)
       End
     When w2 w3 = 'bytes from' Then
       Do
         thisaddr = Substr(w4,1,length(w4)-1)
         goodpings = goodpings + 1
       End
     When w1 = 'Ping:' & w2 w3 = 'unknown host' Then
       Do
         lostpingflag = 'YES'    h
         lostpings = lostpings + 1
         dataline = Left(pingname,40)  Left('no response',15) Left(endtime,10)
         Call disp_prt
       End
    Otherwise 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
```

B.2 32-Bit Windows EXEC to Issue PING

```
/*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 <> '' ] 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' || mmv || ssv || Substr(therest,1,2) || '.dat'
pingfile = '\pf' || mmv || ssv || Substr(therest,1,2) || '.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 || ' -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
```

B.3 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"
```

B.4 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 C. Configuration Files for Base and RIP Examples

This section details the TCP/IP profiles and other configuration files used in the original tests under CS for OS/390 V2R5.

C.1 Profile for T03ATCP Stack - PROF03A

```
DATASETPREFIX TCP
;
TCPCONFIG
   UNRESTRICTLowports
   TCPSENDBfrsize  16384   ; Range is 256-256K - Default is 16K
   TCPRCVBufrsize  16384   ; Range is 256-256K - Default is 16K
   SENDGARBAGE FALSE       ; Packet contains no data
UDPCONFIG
   UNRESTRICTLowports
   UDPCHKsum               ; Do checksum
   UDPSENDBfrsize 16384    ; Range is ???-???K (Default is 16K)
   UDPRCVBufrsize  16384   ; Range is ???-???K (Default is 16K)
IPCONFig
 ARPTO  1200           ; In seconds
 DATAGRamfwd
 NOSOURCEVIPA
 VARSUBNETTING         ; For RIPV2
 SYSPLEXRouting
 IGNORERedirect
 REASSEMBLytimeout   15 ; In seconds
 STOPONclawerror
 TTL    60             ; In seconds, but actually Hop count
;
SACONFIG
  OSASF 760
  ENABLED
  ATMENABLED
;
 AUTOLOG 5
    TO3AFTP JOBNAME TO3AFTP1   ; FTP Server
    TO3DNS  JOBNAME TO3DNS1    ; Domain Name Server
    TO3SNMPD             ; SNMP Agent Server
 ENDAUTOLOG
;
PORT
     7 UDP MISCSERV            ; Miscellaneous Server
     7 TCP MISCSERV
     9 UDP MISCSERV
     9 TCP MISCSERV
    19 UDP MISCSERV
    19 TCP MISCSERV
    20 TCP OMVS      NOAUTOLOG ; FTP Server
    21 TCP TO3AFTP1            ; FTP Server
    25 TCP SMTP                ; SMTP Server
    53 TCP TO3DNS              ; Domain Name Server - Parent Process
    53 UDP TO3DNS              ; Domain Name Server - Parent Process
    80 TCP OMVS                ; Domino webserver
   111 TCP OMVS                ; Portmap Server
   111 UDP OMVS                ; Portmap Server
```

```
   135 UDP LLBD               ; NCS Location Broker
   161 UDP TO3SNMPD           ; SNMP Agent
   162 UDP TO3SNMPQ           ; SNMP Query Engine
   443 TCP OMVS               ; Domino webserver
   512 TCP TO3AREX            ; Remote Execution Server
   514 TCP TO3AREX            ; Remote Execution Server
   515 TCP TO3ALPD            ; LPD Server
   520 UDP TO3AROU            ; RouteD Server
   580 UDP NCPROUT            ; NCPROUTE Server
   750 TCP MVSKERB            ; Kerberos
   750 UDP MVSKERB            ; Kerberos
   751 TCP ADM@SRV            ; Kerberos Admin Server
   751 UDP ADM@SRV            ; Kerberos Admin Server
   760 UDP IOASNMP            ; osa/sf
   760 TCP IOASNMP            ; osa/sf
 20000 TCP CICS410             ; CICS Sockets
 20001 TCP CICS410             ; CICS Sockets Server
 3005 TCP TO3AHWS            ; IMS Sockets (OTMA)
 3012 TCP TO3AIMSL           ; IMS Sockets (Listener)
;
; ************************************************************
; VIPA Definition  (For V2R5)
; ************************************************************
   DEVICE VIPA3A   VIRTUAL     0
   LINK   VIPA3A   VIRTUAL     0     VIPA3A
;
; ************************************************************
; SAMEHOST Definition (For V2R5)
; ************************************************************
   DEVICE IUTSAMEH    MPCPTP
   LINK   TO3CTCP     MPCPTP      IUTSAMEH
;
; ************************************************************
; LCS Definition
; 3172 T/R attach - Rel Adapter 0 - Device # 320-321
; ************************************************************
 DEVICE icp1 LCS  320 autorestart
 LINK icp1 IBMTR    0 icp1
 DEVICE icp11 LCS  328 autorestart   ; for VIPA addr on LAN
 LINK icp11 IBMTR    0 icp11
;
; ************************************************************
; MPC Definitions
; ************************************************************
  DEVICE MPC25 MPCPTP AUTORESTART
  LINK   MPC25 MPCPTP MPC25
  DEVICE MPC2216 MPCPTP AUTORESTART
  LINK   MPC2216 MPCPTP MPC2216
;
; ************************************************************
; XCF Definition
; ************************************************************
   DEVICE RA28M MPCPTP autorestart
   LINK RA28M  MPCPTP RA28M
;
 DEVICE DEVT25A SNAIUCV SNALINK RAPT25A  TO3AT25A
 LINK LINKT25A SAMEHOST 0 DEVT25A
```

```
;
; DEV3746 is an 3746 IP over Channel attachment
;         To 3746-9x0 IP Router

 DEVICE A3746 CDLC 901 15 15 4096 4096
 LINK A3746 CDLC 0 A3746
;
HOME
    192.168.250.3    VIPA3A      ; 1st VIPA Link (for V2R5)
    192.168.210.3    ICP11       ; for VIPA address on LAN...
    192.168.249.3    TO3CTCP     ; For SAMEHOST - IUTSAMEH - Connection
    192.168.236.3    RA28M       ; XCF TO MVS28
    192.168.239.3    LINKT25A
    192.168.202.3    A3746
    192.168.221.03   icp1
    192.168.235.3    MPC25     ; MPC TO MVS25
    192.168.230.3    MPC2216     ; MPC TO 2216
;
; PRO3AGW used for static routing only.....
;
;INCLUDE TCP.TCPPARMS(PRO3AGW)
;
; PRO3ABSD used for RouteD only
;
 INCLUDE TCP.TCPPARMS(PRO3ABSD)
;
;  the VTAM parameters are in telnet3a
;
 INCLUDE TCP.TCPPARMS(TELNO3A)
 START icp1
 START ICP11
 START DEVT25A
 START A3746
 START MPC25
 START RA28M
 START IUTSAMEH            ; SAMEHOST LINK (IUTSAMEH)
 START MPC2216             ; mpc to 2216
```

C.2 TCPIP.DATA File for T03ATCP Stack - TDATA03A

```
TCPIPJOBNAME TO3ATCP
;
HOSTNAME  mvs03a
DOMAINORIGIN  buddha.ral.ibm.com
;
NSINTERADDR  192.168.221.25       ;SA25 nameserver...
NSPORTADDR 53
;
RESOLVEVIA UDP
RESOLVERTIMEOUT 30
RESOLVERUDPRETRIES 2
;
DATASETPREFIX TCP
;
MESSAGECASE MIXED
```

C.3 Telnet Parameters for T03ATCP Stack - TELN03A

```
TELNETPARMS
    PORT 23
    INACTIVE 7200
    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
      RA3ATN01..RA3ATN50
      RA3ATP01..RA3ATP10  ; printers
  ENDDEFAULTLUS
;
  LUGROUP SPECLU
      RA3ATN90..RA3ATN99  ; specials
  ENDLUGROUP
;
  PRTGROUP PRINTERS
      RA3ATPR8..RA3ATPR9  ; printers
  ENDPRTGROUP
;
  MSG07             ; Error messages will be issued
  LUSESSIONPEND
;
  USSTCP TELNUST
  LINEMODEAPPL RA03T ; Send all line-mode terminals directly to TSO.
  ALLOWAPPL RA*  ;* DISCONNECTABLE  Allow all users access to TSO
ENDVTAM
```

C.4 Gateway Parameters for T03ATCP Stack - PR03AGW

```
 GATEWAY
;
; Network  First Hop  Link Name Packet Size  Subnet Mask  Subnet Value
  192.168.221    =       icp1       4000       0
  192.168.236.28 =      RA28M       32768     HOST
  192.168.236.29 =      RA28M       32768     HOST
  192.168.235.25 =      MPC25       32768     HOST
;
```

C.5 BSD Routing Parameters for T03ATCP Stack - PR03ABSD

```
;    Link     Maxmtu   Metric   Subnet Mask     Dest Addr
  BSDROUTINGPARMS false
    VIPA3A   DEFAULTSIZE   0     255.255.255.0   0
    TO3CTCP     4096       0     0               192.168.249.4
    A3746       4092       0     0               192.168.202.1
    ICP1        4000       0     255.255.255.0   0
    LINKT25A    2000       0     0               192.168.239.27
    RA28M      32768       0     0               192.168.236.28
    MPC2216    32768       0     0               192.168.230.13
    MPC25      32768       0     0               192.168.235.25
  ENDBSDROUTINGPARMS
```

C.6 FTP Data Parameters for T03ATCP FTP Server

```
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      MVS             ; 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 FTPRAL         ; group name registered in DNS/WLM sysplex
WRAPRECORD    FALSE           ; data is NOT wrapped to next record
```

C.7 Profile for T03CTCP Stack - PROF03C

```
DATASETPREFIX TCP
;
TCPCONFIG
   UNRESTRICTLowports
   TCPSENDBfrsize  16384   ; Range is 256-256K - Default is 16K
   TCPRCVBufrsize  16384   ; Range is 256-256K - Default is 16K
   SENDGARBAGE FALSE       ; Packet contains no data
UDPCONFIG
   UNRESTRICTLowports
   UDPCHKsum               ; Do checksum
   UDPSENDBfrsize 16384    ; Range is ???-???K (Default is 16K)
   UDPRCVBufrsize  16384   ; Range is ???-???K (Default is 16K)
IPCONFig
 ARPTO  1200           ; In seconds
 DATAGRamfwd
 SOURCEVIPA
 VARSUBNETTING         ; For RIPV2
 SYSPLEXRouting
 IGNORERedirect
 STOPONclawerror
 TTL    60             ; In seconds, but actually Hop count
;
AUTOLOG 1
    T03CFTP JOBNAME T03CFTP1   ; FTP Server
    T03INETD             ; jobname not require, 8 characters....
 ENDAUTOLOG
;
PORT
    514 UDP SYSLOGD1
; ********************************************************
; SAMEHOST Definition (For V2R5)
; ********************************************************
   DEVICE IUTSAMEH    MPCPTP
   LINK   T03ATCP     MPCPTP      IUTSAMEH
;
; ********************************************************
; MPC Definition
; ********************************************************
  DEVICE MPCT025 MPCPTP AUTORESTART
  LINK   MPCT025 MPCPTP MPCT025
  DEVICE M3C2216 MPCPTP AUTORESTART
  LINK   M3C2216 MPCPTP M3C2216
;
; ********************************************************
; VIPA Definition  (For V2R5)
; ********************************************************
   DEVICE VIPA3C      VIRTUAL     0
   LINK   VIPA3C      VIRTUAL     0     VIPA3C
;
; ICP defintion
;
 DEVICE icp1   LCS  322 autorestart
 LINK icp1   IBMTR    0 icp1
 DEVICE icp11 LCS  32A autorestart   ; for VIPA addr on LAN
 LINK icp11 IBMTR    0 icp11
;
```

```
; 3746 Definitions
;
 DEVICE A3746 CDLC 902 15 15 4096 4096
 LINK A3746 CDLC 0 A3746
;
HOME
    192.168.251.4    VIPA3C      ; 1st VIPA Link (for V2R5)
    192.168.111.4    ICP11       ; address for VIPA on LAN
    192.168.249.4    TO3ATCP     ; For SAMEHOST - IUTSAMEH - Connection
    192.168.221.4    icp1
    192.168.235.4    MPCTO25     ; MPC TO MVS25
    192.168.230.4    MPC2216
;
;GATEWAY  commented out when using RouteD
;
;  Network  First Hop  Link Name Packet Size  Subnet Mask  Subnet Value
;
; 192.168.252.3   =      TO3ATCP     4096         HOST
; 192.168.236.28  =      RA28M       32768        HOST
; 192.168.236.29  =      RA28M       32768        HOST
; 192.168.235.25  =      MPCTO25     32768        HOST
; 192.168.221     =      icp1        4000          0
;
; PRO3CBSD used when using RouteD
;
INCLUDE TCP.TCPPARMS(PRO3CBSD)
;
 START MPCTO25
 START IUTSAMEH            ; SAMEHOST LINK (IUTSAMEH)
 START ICP1
 START ICP11
 START A3746
 START M3C2216
```

C.8 TCPIP.DATA File for T03CTCP Stack - TDATA03C

```
TCPIPJOBNAME TO3CTCP
;
HOSTNAME  MVSO3C
DOMAINORIGIN  buddha.ral.ibm.com
;
NSINTERADDR  192.168.221.25     ;  MVS25o stack...
NSPORTADDR 53
;
RESOLVEVIA UDP
RESOLVERTIMEOUT 30
RESOLVERUDPRETRIES 2
;
DATASETPREFIX TCP
;
MESSAGECASE MIXED
```

C.9 BSD Routing Parameters for T03CTCP Stack - PR03CBSD

```
;    Link     Maxmtu   Metric   Subnet Mask     Dest Addr
  BSDROUTINGPARMS false
    VIPA3C   DEFAULTSIZE   0     255.255.255.0   0
    T03ATCP     4096       0     0               192.168.249.3
    A3746       4092       0     0               192.168.202.1
    ICP1        4000       0     255.255.255.0   0
    MPC2216    32768       0     0               192.168.230.13
    MPCT025    32768       0     0               192.168.235.25
  ENDBSDROUTINGPARMS
```

C.10 Profile for T28ATCP Stack - PROF28A

```
DATASETPREFIX TCP
;
TCPCONFIG
   UNRESTRICTLowports
   TCPSENDBfrsize  16384   ; Range is 256-256K - Default is 16K
   TCPRCVBufrsize  16384   ; Range is 256-256K - Default is 16K
   SENDGARBAGE FALSE       ; Packet contains no data
UDPCONFIG
   UNRESTRICTLowports
   UDPCHKsum               ; Do checksum
   UDPSENDBfrsize 16384    ; Range is ???-???K (Default is 16K)
   UDPRCVBufrsize  16384   ; Range is ???-???K (Default is 16K)
IPCONFig
 ARPTO  1200           ; In seconds
 DATAGRamfwd
 TTL    60             ; In seconds, but actually Hop count
 SYSPLEXRouting
 IGNORERedirect
 noSOURCEVIPA
 VARSUBNETTING
;
AUTOLOG 1
   T28AFTP JOBNAME T28AFTP1   ; FTP Server
   T28DNS  JOBNAME T28DNS1    ; Domain Name Server
  PORTMAP
 ENDAUTOLOG
; ********************************************************
; VIPA Definition  (For V2R5)
; ********************************************************
   DEVICE VIPA28A  VIRTUAL     0
   LINK   VIPA28A  VIRTUAL     0     VIPA28A
;
; ********************************************************
; XCF Definition to ra03
; ********************************************************
   DEVICE RA03M MPCPTP autorestart
   LINK RA03M  MPCPTP RA03M
; ********************************************************
; LCS Definition
; 3172 T/R attach - Rel Adapter 0 - Device # 320-321
; ********************************************************
 DEVICE icp1   LCS  330 autorestart
 LINK  icp1   IBMTR    0 icp1
 DEVICE icp11 LCS  338 autorestart   ; for VIPA addr on LAN
```

```
 LINK icp11 IBMTR    0 icp11
;
HOME
    192.168.252.28   VIPA28A
    192.168.212.28   ICP11
    192.168.236.28   RA03M
    192.168.221.28   icp1
    192.168.229.28   MPC25
    192.168.231.28   M282216
    192.168.202.28   A3746
  PRIMARYINTERFACE ICP1
;
;GATEWAY, commenetd out when using RouteD
;
; Network  First Hop  Link Name Packet Size  Subnet Mask  Subnet Value
; 192.168.236.1  =      RA03M       32768        HOST
; 192.168.236.3  =      RA03M       32768        HOST
; 192.168.221    =       icp1        4000          0
;
; PR28ABSD used when running RouteD
 INCLUDE TCP.TCPPARMS(PR28ABSD)
;
 INCLUDE TCP.TCPPARMS(TELN28A)
;
 START RA03M
 START icp1
 START icp11
 START T28CTCP
```

C.11 TCPIP.DATA File for T28ATCP Stack - TDATA28A

```
TCPIPJOBNAME T28ATCP
;
HOSTNAME  MVS28A
DOMAINORIGIN  buddha.ral.ibm.com
;
NSINTERADDR  192.168.221.25
NSPORTADDR 53
;
RESOLVEVIA UDP
RESOLVERTIMEOUT 30
RESOLVERUDPRETRIES 1
;
DATASETPREFIX TCP
MESSAGECASE MIXED
```

C.12 Telnet Parameters for T28ATCP Stack - TELN28A

```
TELNETPARMS
    PORT 23
    INACTIVE 7200
    TIMEMARK 7200
    SCANINTERVAL 300
    SMFINIT STD
    SMFTERM STD
```

```
    WLMCLUSTERNAME TN3270E TN3270 TELNET 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
; Define the LUs to be used for general users.
  DEFAULTLUS
      RASATN01..RASATN50
  ENDDEFAULTLUS
;
  LUGROUP SPECLU
      RASATN90..RASATN99  ; specials
  ENDLUGROUP
  PRTGROUP PRINTERS
      RASATPR8..RASATPR9  ; printers
  ENDPRTGROUP
;
  MSG07             ; Error messages will be issued
  LUSESSIONPEND     ; On termination of a Telnet server connection,
;                   ; the user will revert to the DEFAULTAPPL
  LINEMODEAPPL RASAT ; Send all line-mode terminals directly to TSO.
  ALLOWAPPL RA*
ENDVTAM
```

C.13 BSD Routing Parameters for T28ATCP Stack - PR28ABSD

```
;    Link     Maxmtu   Metric   Subnet Mask     Dest Addr
;    Link     Maxmtu   Metric   Subnet Mask     Dest Addr
  BSDROUTINGPARMS false
    VIPA28A  DEFAULTSIZE   0     255.255.255.0   0
    RA03M      32768       0     0               192.168.236.3
    A3746       4092       0     0               192.168.202.1
    ICP1        4000       0     255.255.255.0   0
    M282216    32768       0     0               192.168.231.128
  ENDBSDROUTINGPARMS
```

C.14 Profile for T28CTCP Stack - PROF28C

```
DATASETPREFIX TCP
TCPCONFIG
   UNRESTRICTLowports
   TCPSENDBfrsize 131072   ; Range is 256-256K - Default is 16K
   TCPRCVBufrsize 131072   ; Range is 256-256K - Default is 16K
   SENDGARBAGE FALSE       ; Packet contains no data
UDPCONFIG
   UNRESTRICTLowports
   UDPCHKsum               ; Do checksum
   UDPSENDBfrsize 131072   ; Range is ???-???K (Default is 16K)
   UDPRCVBufrsize 131072   ; Range is ???-???K (Default is 16K)
IPCONFig
 ARPTO  1200           ; In seconds
 DATAGRamfwd
 TTL    60             ; In seconds, but actually Hop count
 SYSPLEXRouting
 VARSUBNETTING
 firewall
 AUTOLOG 1
     T28CFTP JOBNAME T28CFTP1   ; FTP Server
     T28INETD             ; jobname not require, 8 characters....
  ENDAUTOLOG
 ;
 DEVICE icp1   LCS  332 autorestart
 LINK  icp1   IBMTR    0 icp1
;
HOME
    192.168.221.29   icp1
;
 START icp1
```

C.15 TCPIP.DATA File for T28CTCP Stack - TDATA28C

```
TCPIPJOBNAME T28CTCP
;
HOSTNAME  MVS28C
DOMAINORIGIN  buddha.ral.ibm.com
;
NSINTERADDR  192.168.221.25
NSPORTADDR 53
;
RESOLVEVIA UDP
RESOLVERTIMEOUT 30
RESOLVERUDPRETRIES 1
;
DATASETPREFIX TCP
;
MESSAGECASE MIXED
```

Appendix D. Profiles, Data Files and Parameter Files

This section shows the profiles and configuration files used in the OSPF tests described in Chapter 8, “Sysplex with OMPROUTE and VIPA” on page 89 and Chapter 9, “Network Dispatcher” on page 111.

D.1 Profile for T03ATCP Stack - PRO03A

```
DATASETPREFIX TCP

TCPCONFIG
   UNRESTRICTLowports
   TCPSENDBfrsize  65536       ; Range is 256-256K - Default is 16K
   TCPRCVBufrsize  65536       ; Range is 256-256K - Default is 16K
   SENDGARBAGE FALSE           ; Packet contains no data

UDPCONFIG
   UNRESTRICTLowports
   UDPCHKsum                   ; Do checksum

IPCONFig
   ARPTO  1200                 ; In seconds
   DATAGRamfwd
   SOURCEVIPA
   VARSUBNETTING
   SYSPLEXRouting
   DYNAMICXCF 172.16.233.3 255.255.255.0 1    ; Dynamic XCF definitions
   IGNORERedirect
   REASSEMBLytimeout   15      ; In seconds
   STOPONclawerror
   TTL    60                   ; In seconds, but actually Hop count
   MULTIPATH

SACONFIG  COMMUNITY publicv2c
   OSASF 760
   ENABLED
   ATMENABLED
   SETSENABLED

AUTOLOG 1
    TO3AFTP JOBNAME TO3AFTP1   ; FTP Server
    TO3DNS  JOBNAME TO3DNS1    ; Domain Name Server
    TO3AOMPR                   ; OMPROUTE Server
    SYSLOGD JOBNAME SYSLOGD1   ; SYSLOG daemon
ENDAUTOLOG


PORT
     7 UDP MISCSERV            ; Miscellaneous Server
     7 TCP MISCSERV
     9 UDP MISCSERV
     9 TCP MISCSERV
    19 UDP MISCSERV
    19 TCP MISCSERV
```

```
    20 TCP OMVS      NOAUTOLOG ; FTP Server
;   21 TCP TO3AFTP1            ; FTP Server
;   23 TCP INTCLIEN            ; Telnet Server
    25 TCP SMTP                ; SMTP Server
    53 TCP TO3DNS              ; Domain Name Server - Parent Process
    53 UDP TO3DNS              ; Domain Name Server - Parent Process
    80 TCP WEBQM               ; Domino webserver
   111 TCP OMVS                ; Portmap Server
   111 UDP OMVS                ; Portmap Server
   135 UDP LLBD                ; NCS Location Broker
   161 UDP TO3SNMPQ            ; SNMP Agent
   162 UDP TO3SBNPQ            ; SNMP Query Engine
   443 TCP OMVS                ; Domino webserver
   512 TCP TO3AREX             ; Remote Execution Server
   514 TCP TO3AREX             ; Remote Execution Server
   514 UDP OMVS                ; SYSLOG Daemon
   515 TCP TO3ALPD             ; LPD Server
   520 UDP TO3AOMPR            ; OMPROUTE Server
   580 UDP NCPROUT             ; NCPROUTE Server
   750 TCP MVSKERB             ; Kerberos
   750 UDP MVSKERB             ; Kerberos
   751 TCP ADM@SRV             ; Kerberos Admin Server
   751 UDP ADM@SRV             ; Kerberos Admin Server
   760 UDP IOASNMP             ; osa/sf
   760 TCP IOASNMP             ; osa/sf

DEVICE VIPA3A   VIRTUAL     0
LINK   VIPA3A   VIRTUAL     0     VIPA3A

DEVICE TR1    LCS  2060 autorestart
LINK  TR1    IBMTR    0 TR1

DEVICE MO32216B MPCPTP AUTORESTART
LINK   MO32216B MPCPTP MO32216B

HOME
    172.16.250.3     VIPA3A
    9.24.104.113     TR1
    172.16.100.3     MO32216B    ; for 2216 ESCON MPC+
    172.16.220.50    loopback    ; ndr cluster
    172.16.220.51    loopback    ; ndr cluster

ITRACE OFF

INCLUDE TCP.TCPPARMS(TELNO3A)

START TR1
START MO32216B
```

D.2 TCPIP.DATA File for T03ATCP Stack - TDATA03A

```
TCPIPJOBNAME TO3ATCP
HOSTNAME  MVSO3A
DOMAINORIGIN  buddha.ral.ibm.com
NSINTERADDR  9.24.104.125
NSPORTADDR 53
RESOLVEVIA UDP
```

```
RESOLVERTIMEOUT 10
RESOLVERUDPRETRIES 1
DATASETPREFIX TCP
MESSAGECASE MIXED
```

D.3 Telnet Parameters for T03ATCP Stack - TELN03A

```
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
; Define the LUs to be used for general users.
  DEFAULTLUS
      RA3ATN01..RA3ATN50
      RA3ATP01..RA3ATP10  ; printers
  ENDDEFAULTLUS
;
  LUGROUP SPECLU
      RA3ATN90..RA3ATN99  ; specials
  ENDLUGROUP
;
  PRTGROUP PRINTERS
      RA3ATPR8..RA3ATPR9  ; 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.4 FTP Data Parameters for FTP Server - FDATA03A, 28A and 39A

```
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       20               ; 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     10               ; new data set allocation secondary space
SPACETYPE     CYL              ; new data set allocation space type
SPREAD        FALSE            ; sql output format
SQLCOL        NAMES            ; sql output uses column names as headings
STARTDIR      MVS              ; 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 FTPRAL          ; group name registered in DNS/WLM sysplex
WRAPRECORD    FALSE            ; data is NOT wrapped to next record
```

D.5 OMPROUTE Configuration File for T03ATCP Stack - OM03ACF

```
Area    Area_Number=0.0.0.0
              Stub_Area=NO
              Authentication_type=None;
OSPF_Interface IP_Address=172.16.100.3
              Name=m032216b
              Subnet_mask=255.255.255.0
              MTU=32768;
OSPF_Interface IP_Address=172.16.233.*
              Subnet_mask=255.255.255.0
              MTU=32768;
Interface     IP_Address=9.24.104.113
              Name=tr1
              Subnet_mask=255.255.255.0
              MTU=4000;
Interface     IP_Address=172.16.250.3
```

```
                Name=VIPA3A
                Subnet_mask=255.255.255.0
                MTU=32768;
AS_Boundary_routing
                Import_Direct_Routes=YES;
```

D.6 Profile for T28ATCP Stack - PRO28A

```
DATASETPREFIX TCP

TCPCONFIG
   UNRESTRICTLowports
   SENDGARBAGE FALSE           ; Packet contains no data

UDPCONFIG
   UNRESTRICTLowports
   UDPCHKsum                   ; Do checksum

IPCONFig
   ARPTO  1200                 ; In seconds
   DATAGRamfwd
   SOURCEVIPA
   VARSUBNETTING
   SYSPLEXRouting
   DYNAMICXCF 172.16.233.28 255.255.255.0 1    ; Dynamic XCF definitions
   IGNORERedirect
   REASSEMBLytimeout   15      ; In seconds
   STOPONclawerror
   TTL    60                   ; In seconds, but actually Hop count
   MULTIPATH

SACONFIG  COMMUNITY publicv2c
   OSASF 760
   ENABLED
   ATMENABLED
   SETSENABLED

AUTOLOG 1
   T28AFTP JOBNAME T28AFTP1   ; FTP Server
   T28DNS  JOBNAME T28DNS1    ; Domain Name Server
   T28AOMPR                   ; OMPROUTE Server
   SYSLOGD JOBNAME SYSLOGD1   ; SYSLOG daemon
ENDAUTOLOG


PORT
     7 UDP MISCSERV            ; Miscellaneous Server
     7 TCP MISCSERV
     9 UDP MISCSERV
     9 TCP MISCSERV
    19 UDP MISCSERV
    19 TCP MISCSERV
    20 TCP OMVS      NOAUTOLOG ; FTP Server
    21 TCP T28AFTP1            ; FTP Server
;   23 TCP INTCLIEN            ; Telnet Server
    25 TCP SMTP                ; SMTP Server
    53 TCP T28DNS              ; Domain Name Server - Parent Process
    53 UDP T28DNS              ; Domain Name Server - Parent Process
```

```
    80 TCP WEBQM               ; Domino webserver
   111 TCP OMVS                ; Portmap Server
   111 UDP OMVS                ; Portmap Server
   135 UDP LLBD                ; NCS Location Broker
   161 UDP T28SNMPD            ; SNMP Agent
   162 UDP T28SNMPQ            ; SNMP Query Engine
   443 TCP OMVS                ; Domino webserver
   512 TCP T28AREX             ; Remote Execution Server
   514 TCP T28AREX             ; Remote Execution Server
   514 UDP OMVS                ; SYSLOG daemon
   515 TCP T28ALPD             ; LPD Server
   520 UDP T28AOMPR            ; OMPROUTE Server
   580 UDP NCPROUT             ; NCPROUTE Server
   750 TCP MVSKERB             ; Kerberos
   750 UDP MVSKERB             ; Kerberos
   751 TCP ADM@SRV             ; Kerberos Admin Server
   751 UDP ADM@SRV             ; Kerberos Admin Server
   760 UDP IOASNMP             ; osa/sf
   760 TCP IOASNMP             ; osa/sf

DEVICE VIPA28A   VIRTUAL     0
LINK   VIPA28A   VIRTUAL     0     VIPA28A

DEVICE TR1    LCS  2060 autorestart  ; for OSA TR
LINK  TR1   IBMTR    0 TR1

DEVICE M282216B MPCPTP AUTORESTART   ; for 2216 ESCON MPC+
LINK   M282216B MPCPTP M282216B

HOME
     172.16.252.28    VIPA28A
     9.24.104.42      TR1
     172.16.101.28    M282216B
     172.16.220.50    loopback       ; for NDR Cluster
     172.16.220.51    loopback

ITRACE OFF

INCLUDE TCP.TCPPARMS(TELN28A)

START TR1
START M282216B
```

D.7 TCPIP.DATA File for T28ATCP Stack - TDATA28A

```
TCPIPJOBNAME T28ATCP
HOSTNAME  MVS28A
DOMAINORIGIN  buddha.ral.ibm.com
NSINTERADDR  9.24.104.125
NSPORTADDR 53
RESOLVEVIA UDP
RESOLVERTIMEOUT 10
RESOLVERUDPRETRIES 1
DATASETPREFIX TCP
MESSAGECASE MIXED
```

D.8 Telnet Parameters for T28ATCP Stack - TELN28A

```
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
; Define the LUs to be used for general users.
  DEFAULTLUS
      RASATN01..RASATN50
  ENDDEFAULTLUS
;
  LUGROUP SPECLU
      RASATN90..RASATN99  ; specials
  ENDLUGROUP
;
  PRTGROUP PRINTERS
      RASATPR8..RASATPR9  ; printers
  ENDPRTGROUP
;
  LUMAP SPECLU   9.24.104.201
;
  MSG07            ; Error messages will be issued
  LUSESSIONPEND    ; On termination of a Telnet server connection,
  USSTCP TELNUST
  LINEMODEAPPL RASAT ; Send all line-mode terminals directly to TSO.
  ALLOWAPPL RA*
  ALLOWAPPL AD*
  ALLOWAPPL A2*
  ALLOWAPPL FD*
  ALLOWAPPL X6*
  ALLOWAPPL X7*
ENDVTAM
```

D.9 OMPROUTE Configuration File for T28ATCP Stack - OM28ACF

```
Area    Area_Number=0.0.0.0
              Stub_Area=NO
              Authentication_type=None;
OSPF_Interface IP_Address=172.16.101.28
              Name=M282216B
              Subnet_mask=255.255.255.0
              MTU=32768;
OSPF_Interface IP_Address=172.16.233.*
              Subnet_mask=255.255.255.0
              MTU=32768;
Interface     IP_Address=9.24.104.42
              Name=tr1
              Subnet_mask=255.255.255.0
              MTU=4000;
Interface     IP_Address=172.16.252.28
              Name=VIPA28A
              Subnet_mask=255.255.255.0
              MTU=32768;
AS_Boundary_routing
              Import_Direct_Routes=YES;
```

D.10 Profile for T39ATCP Stack - PRO39A

```
DATASETPREFIX TCP

TCPCONFIG
   UNRESTRICTLowports
   SENDGARBAGE FALSE           ; Packet contains no data

UDPCONFIG
   UNRESTRICTLowports
   UDPCHKsum                   ; Do checksum

IPCONFig
   ARPTO  1200                 ; In seconds
   DATAGRamfwd
   SOURCEVIPA
   VARSUBNETTING
   SYSPLEXRouting
   DYNAMICXCF 172.16.233.39 255.255.255.0 1    ; Dynamic XCF definitions
   IGNORERedirect
   REASSEMBLytimeout   15      ; In seconds
   STOPONclawerror
   TTL    60                   ; In seconds, but actually Hop count
   MULTIPATH

SACONFIG  COMMUNITY publicv2c
   OSASF 760
   ENABLED
   ATMENABLED
   SETSENABLED

AUTOLOG 1
   T39AFTP JOBNAME T39AFTP1   ; FTP Server
   T39DNS  JOBNAME T39DNS1    ; Domain Name Server
   T39AOMPR                   ; OMPROUTE Server
```

```
   SYSLOGD JOBNAME SYSLOGD1   ; SYSLOG daemon
ENDAUTOLOG


PORT
     7 UDP MISCSERV            ; Miscellaneous Server
     7 TCP MISCSERV
     9 UDP MISCSERV
     9 TCP MISCSERV
    19 UDP MISCSERV
    19 TCP MISCSERV
    20 TCP OMVS      NOAUTOLOG ; FTP Server
    21 TCP TO3AFTP1            ; FTP Server
;   23 TCP INTCLIEN            ; Telnet Server
    25 TCP omvs                ; SMTP Server
    53 TCP omvs                ; Domain Name Server - Parent Process
    53 UDP omvs                ; Domain Name Server - Parent Process
    80 TCP WEBQM               ; Domino webserver
   111 TCP OMVS                ; Portmap Server
   111 UDP OMVS                ; Portmap Server
   135 UDP LLBD                ; NCS Location Broker
   161 UDP TO3SNMPD            ; SNMP Agent
   162 UDP TO3SNMPQ            ; SNMP Query Engine
   443 TCP OMVS                ; Domino webserver
   512 TCP TO3AREX             ; Remote Execution Server
   514 TCP TO3AREX             ; Remote Execution Server
   514 UDP OMVS                ; SYSLOG daemon
   515 TCP TO3ALPD             ; LPD Server
   520 UDP T39AOMPR            ; OMPROUTE Server
   580 UDP NCPROUT             ; NCPROUTE Server
   750 TCP MVSKERB             ; Kerberos
   750 UDP MVSKERB             ; Kerberos
   751 TCP ADM@SRV             ; Kerberos Admin Server
   751 UDP ADM@SRV             ; Kerberos Admin Server
   760 UDP IOASNMP             ; osa/sf
   760 TCP IOASNMP             ; osa/sf

DEVICE VIPA39A   VIRTUAL     0
LINK   VIPA39A   VIRTUAL     0     VIPA39A

DEVICE TR1    LCS  2060 autorestart       ; for OSA TR
LINK  TR1    IBMTR    0 TR1

DEVICE M392216B MPCPTP AUTORESTART        ; for 2216 ESCON MPC+
LINK   M392216B MPCPTP M392216B

HOME
    172.16.232.39 VIPA39A     ; VIPA
    9.24.104.149   TR1        ; osa TO public network
    172.16.102.39 M392216b    ; MPC TO 2216 mae
    172.16.220.50 loopback    ; ndr cluster
    172.16.220.51 loopback    ; ndr cluster

ITRACE OFF

INCLUDE TCP.TCPPARMS(TELN39A)

START TR1
START M392216b
```

D.11 TCPIP.DATA File for T39ATCP Stack - TDATA39A

```
TCPIPJOBNAME T39ATCP
HOSTNAME  MVS39A
DOMAINORIGIN  buddha.ral.ibm.com
NSINTERADDR   9.24.104.125
NSPORTADDR 53
RESOLVEVIA UDP
RESOLVERTIMEOUT 10
RESOLVERUDPRETRIES 1
DATASETPREFIX TCP
MESSAGECASE MIXED
DATASETPREFIX TCP
MESSAGECASE MIXED
```

D.12 Telnet Parameters for T39ATCP Stack - TELN39A

```
TELNETPARMS
    PORT 23
    INACTIVE 0
    TIMEMARK 7200
    SCANINTERVAL 300
    SMFINIT STD
    SMFTERM STD
    WLMCLUSTERNAME TNRAL ENDWLMCLUSTERNAME
ENDTELNETPARMS
BEGINVTAM
    PORT 23 233
  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
  LUGROUP LU1
      RA39TN01..RA39TN05
  ENDLUGROUP
;
  IPGROUP IP1
      255.255.255.0:9.24.104.0
  ENDIPGROUP
;
  PRTGROUP PRT1
      RA39TP01..RA39TP05
```

```
  ENDPRTGROUP
;
  LUMAP LU1 IP1 GENERIC PRT1
;
  LUGROUP LU2
      RA39TN06..RA39TN10
  ENDLUGROUP
  IPGROUP IP2
      255.255.255.0:9.24.105.0
  ENDIPGROUP
;
  PRTGROUP PRT2
      RA39TP06..RA39TP10
  ENDPRTGROUP
;
  LUMAP LU2 IP2 GENERIC PRT2
;
  LUGROUP LU3
      RA39TN11..RA39TN15
  ENDLUGROUP
;
  IPGROUP IP3
      255.255.255.0:9.24.106.0
  ENDIPGROUP
  PRTGROUP PRT3
      RA39TP11..RA39TP15
  ENDPRTGROUP
;
  LUMAP LU3 IP3 GENERIC PRT3
;
  LUGROUP LU4
      RA39TN16..RA39TN20
  ENDLUGROUP
;
  IPGROUP IP4
      255.0.0.0:9.0.0.0
  ENDIPGROUP
;
  PRTGROUP PRT4
      RA39TP16..RA39TP20
  ENDPRTGROUP
;
  LUMAP LU4 IP4 GENERIC PRT4
;
  LUGROUP LU5
      RA39TN26..RA39TN40
  ENDLUGROUP
;
  IPGROUP IP5
      255.255.0.0:172.16.0.0
  ENDIPGROUP
;
  PRTGROUP PRT5
      RA39TP26..RA39TP40
  ENDPRTGROUP
;
  LUMAP LU5 IP5 GENERIC PRT5
;
  MSG07            ; Error messages will be issued
```

```
  LUSESSIONPEND    ; On termination of a Telnet server connection,
  USSTCP TELNUST
  LINEMODEAPPL RAO3T ; Send all line-mode terminals directly to TSO.
  ALLOWAPPL RA*
  ALLOWAPPL AD*
  ALLOWAPPL A2*
  ALLOWAPPL FD*
  ALLOWAPPL X6*
  ALLOWAPPL X7*
ENDVTAM
TELNETPARMS
    SECUREPORT 223 KEYRING hfs /etc/telnetssl/keyfile.kyr
    INACTIVE 7200
    TIMEMARK 7200
    SCANINTERVAL 300
    SMFINIT STD
    SMFTERM STD
    WLMCLUSTERNAME TNRALSSL2 ENDWLMCLUSTERNAME
ENDTELNETPARMS
```

D.13 OMPROUTE Configuration File for T39ATCP Stack - OM39ACF

```
Area    Area_Number=0.0.0.0
              Stub_Area=NO
              Authentication_type=None;
OSPF_Interface IP_Address=172.16.102.39
              Name=m392216b
              Subnet_mask=255.255.255.0
              MTU=32768;
OSPF_Interface IP_Address=172.16.233.*
              Subnet_mask=255.255.255.0
              MTU=32768;
Interface     IP_Address=9.24.104.149
              Name=tr1
              Subnet_mask=255.255.255.0
              MTU=4000;
Interface     IP_Address=172.16.232.39
              name=VIPA39A
              Subnet_mask=255.255.255.0
              MTU=32768;
AS_Boundary_routing
              Import_Direct_Routes=YES;
```

Appendix E. Dump of T28ATCP Name Server - Single-Path Network

```
;; --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 buddha.ral.ibm.com.
ralplex1 IN SOA mvs03a.ralplex1.buddha.ral.ibm.com. garthm@mvs03a.ralplex1.buddha.ral.ibm.com. (
  19980507 3600 3600 14400 60 ) ;Cl=5
  IN NS mvs03a.ralplex1.buddha.ral.ibm.com. ;Cl=5
  IN A 192.168.221.3 ;Cl=5
  IN A 192.168.221.4 ;Cl=5
  IN A 192.168.221.28 ;Cl=5
  IN A 192.168.221.29 ;Cl=5
$ORIGIN ralplex1.buddha.ral.ibm.com.
FTPRAL  IN A 192.168.221.3 ;Cl=5
  IN A 192.168.221.28 ;Cl=5
mvs03a  IN A 192.168.221.3 ;Cl=5
mvsdum  IN A 192.168.221.5 ;Cl=5
mvs03c  IN A 192.168.221.4 ;Cl=5
localhost IN A 127.0.0.1 ;Cl=5
mvs28a  IN A 192.168.221.28 ;Cl=5
FTPOERAL IN A 192.168.221.4 ;Cl=5
  IN A 192.168.221.29 ;Cl=5
TNRAL  IN A 192.168.221.3 ;Cl=5
  IN A 192.168.221.28 ;Cl=5
mvs28c  IN A 192.168.221.29 ;Cl=5
ralplex1 IN CNAME ralplex1.buddha.ral.ibm.com. ;Cl=5
$ORIGIN FTPRAL.ralplex1.buddha.ral.ibm.com.
MVS28A  IN A 192.168.221.28 ;Cl=5
mvs03a  IN A 192.168.221.3 ;Cl=5
$ORIGIN FTPOERAL.ralplex1.buddha.ral.ibm.com.
MVS28C  IN A 192.168.221.29 ;Cl=5
MVS03C  IN A 192.168.221.4 ;Cl=5
$ORIGIN TNRAL.ralplex1.buddha.ral.ibm.com.
MVS28A  IN A 192.168.221.28 ;Cl=5
mvs03a  IN A 192.168.221.3 ;Cl=5
$ORIGIN 192.in-addr.arpa.
168  IN SOA mvs03a.ralplex1.buddha.ral.ibm.com. garthm@mvs03a.168.192.in-addr.arpa. (
  10004 10 10 10 10 ) ;Cl=4
  IN NS mvs03a.ralplex1.buddha.ral.ibm.com. ;Cl=4
$ORIGIN 221.168.192.in-addr.arpa.
4  IN PTR mvs03c.ralplex1.buddha.ral.ibm.com. ;Cl=4
28  IN PTR mvs28a.ralplex1.buddha.ral.ibm.com. ;Cl=4
3  IN PTR mvs03a.ralplex1.buddha.ral.ibm.com. ;Cl=4
29  IN PTR mvs28c.ralplex1.buddha.ral.ibm.com. ;Cl=4
$ORIGIN 0.127.in-addr.arpa.
0  IN SOA mvs28a.ralplex1.buddha.ral.ib.com.0.0.127.in-addr.arpa. garthm@mvs03a.0.0.127.in-addr.arpa. (
  10004 10 10 10 10 ) ;Cl=5
  IN NS mvs28a.raplplex1.buddha.ral.ibm.com. ;Cl=5
$ORIGIN 0.0.127.in-addr.arpa.
1  IN PTR loopback. ;Cl=5
```

Appendix F. Trace of DNS for the Single-Path Network

```
Debug turned ON, Level 11
Version = @(#) ddns/ns/ns_main.c, dns_ns, dns_r1.1 1.62  9/23/97 10:57:21
main: __iptcpn() returns TCPIPjobname 'T28ATCP'
considering  192.168.251.28
considering  9.24.104.158
considering  192.168.236.28
considering  192.168.221.28
considering  192.168.221.26
considering  192.168.109.28
considering  192.168.100.99
loopback address: x7f000001
dqp->dq_addr 0.0.0.0 d_dfd 5

ns_init(/etc/named.boot)
savehash GROWING to 2
savehash GROWING to 2

 content of zones before loading
zone origin ralplex1.buddha.ral.ibm.com, new zone
 addrs: 192.168.236.3, addrcnt = 1
purge_zone(ralplex1.buddha.ral.ibm.com,1)
db_load(fback, ralplex1.buddha.ral.ibm.com, 1, Nil)
db_load: origin buddha.ral.ibm.com., buf ralplex1.buddha.ral.ibm.com
db_load: origin now buddha.ral.ibm.com
d='ralplex1.buddha.ral.ibm.com', c=1, t=6, ttl=0, data='mvs03a.ralplex1.buddha.ral.ibm.com.'
db_update(ralplex1.buddha.ral.ibm.com, 0x14dcd908, 0x14dcd908, 01, 0x14dcd880)
savehash GROWING to 2
savehash GROWING to 2
savehash GROWING to 2
savehash GROWING to 2
db_update: adding 14dcd908
d='ralplex1.buddha.ral.ibm.com', c=1, t=2, ttl=0, data='mvs03a.ralplex1.buddha.ral.ibm.com.'
db_update(ralplex1.buddha.ral.ibm.com, 0x14dcdb20, 0x14dcdb20, 01, 0x14dcd880)
match(0x14dcd908, 1, 6) 1, 6
match(0x14dcd908, 1, 2) 1, 6
db_update: adding 14dcdb20
db_load: origin ralplex1.buddha.ral.ibm.com., buf buddha.ral.ibm.com
db_load: origin now ralplex1.buddha.ral.ibm.com
d='mvs03a.ralplex1.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.221.3'
db_update(mvs03a.ralplex1.buddha.ral.ibm.com, 0x14dcdb78, 0x14dcdb78, 01, 0x14dcd880)
savehash GROWING to 2
match(0x14dcd908, 1, 6) 1, 6
db_update: adding 14dcdb78
d='mvsdum.ralplex1.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.221.5'
db_update(mvsdum.ralplex1.buddha.ral.ibm.com, 0x14dcdc00, 0x14dcdc00, 01, 0x14dcd880)
match(0x14dcd908, 1, 6) 1, 6
db_update: adding 14dcdc00
d='mvs03c.ralplex1.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.221.4'
db_update(mvs03c.ralplex1.buddha.ral.ibm.com, 0x14dcdc70, 0x14dcdc70, 01, 0x14dcd880)
match(0x14dcd908, 1, 6) 1, 6
db_update: adding 14dcdc70
d='localhost.ralplex1.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='127.0.0.1'
db_update(localhost.ralplex1.buddha.ral.ibm.com, 0x14dcdce0, 0x14dcdce0, 01, 0x14dcd880)
match(0x14dcd908, 1, 6) 1, 6
db_update: adding 14dcdce0
d='mvs28a.ralplex1.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.221.28'
db_update(mvs28a.ralplex1.buddha.ral.ibm.com, 0x14dcdd58, 0x14dcdd58, 01, 0x14dcd880)
savehash GROWING to 11
savehash(0x14dcdbb0) cnt=5, sz=2, newsz=11
match(0x14dcd908, 1, 6) 1, 6
db_update: adding 14dcdd58
d='ralplex1.ralplex1.buddha.ral.ibm.com', c=1, t=5, ttl=0, data='ralplex1.buddha.ral.ibm.com.'
```

```
db_update(ralplex1.ralplex1.buddha.ral.ibm.com, 0x14dcde08, 0x14dcde08, 01, 0x14dcd880)
match(0x14dcd908, 1, 6) 1, 6
db_update: adding 14dcde08
d='mvs28c.ralplex1.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.221.29'
db_update(mvs28c.ralplex1.buddha.ral.ibm.com, 0x14dcde80, 0x14dcde80, 01, 0x14dcd880)
match(0x14dcd908, 1, 6) 1, 6
db_update: adding 14dcde80
wlm_load ralplex1.buddha.ral.ibm.com
 group list retcode = -1 rsncode = 1034
 group list retcode = 0 rsncode = 0
 group list entry_count = 4
 querying group =  TCPIP
 server list retcode = -1 rsncode = 1034
 server list retcode = 0 rsncode = 0
 querying group =  FTPRAL
 server list retcode = -1 rsncode = 1034
 server list retcode = 0 rsncode = 0
 querying group =  TNRAL
 server list retcode = -1 rsncode = 1034
 server list retcode = 0 rsncode = 0
 querying group =  FTPOERAL
 server list retcode = -1 rsncode = 1034
 server list retcode = 0 rsncode = 0
 processing group = ralplex1.buddha.ral.ibm.com count = 4
 minimum weight = 15
 processing server = TO3ATCP weight = 16 host = mvs03a
 adding mvs03a to list
db_update(ralplex1.buddha.ral.ibm.com, 0x14dce570, 0x14dce570, 01, 0x14dcd880)
match(0x14dcd908, 1, 6) 1, 6
match(0x14dcd908, 1, 1) 1, 6
match(0x14dcdb20, 1, 1) 1, 2
db_update: adding 14dce570
 processing server = TO3CTCP weight = 16 host = MVS03C
 adding MVS03C to list
db_update(ralplex1.buddha.ral.ibm.com, 0x14dce5a8, 0x14dce5a8, 01, 0x14dcd880)
match(0x14dcd908, 1, 6) 1, 6
match(0x14dcd908, 1, 1) 1, 6
match(0x14dcdb20, 1, 1) 1, 2
match(0x14dce570, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for ralplex1.buddha.ral.ibm.com is 4(5) from  0.0.0.0 .0, is 4(5) in cache
db_update: adding 14dce5a8
 processing server = T28ATCP weight = 15 host = MVS28A
 adding MVS28A to list
db_update(ralplex1.buddha.ral.ibm.com, 0x14dce5e0, 0x14dce5e0, 01, 0x14dcd880)
match(0x14dcd908, 1, 6) 1, 6
match(0x14dcd908, 1, 1) 1, 6
match(0x14dcdb20, 1, 1) 1, 2
match(0x14dce570, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for ralplex1.buddha.ral.ibm.com is 4(5) from  0.0.0.0 .0, is 4(5) in cache
match(0x14dce5a8, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for ralplex1.buddha.ral.ibm.com is 4(5) from  0.0.0.0 .0, is 4(5) in cache
db_update: adding 14dce5e0
 processing server = T28CTCP weight = 15 host = MVS28C
 adding MVS28C to list
db_update(ralplex1.buddha.ral.ibm.com, 0x14dce618, 0x14dce618, 01, 0x14dcd880)
match(0x14dcd908, 1, 6) 1, 6
match(0x14dcd908, 1, 1) 1, 6
match(0x14dcdb20, 1, 1) 1, 2
match(0x14dce570, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for ralplex1.buddha.ral.ibm.com is 4(5) from  0.0.0.0 .0, is 4(5) in cache
match(0x14dce5a8, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
```

```
credibility for ralplex1.buddha.ral.ibm.com is 4(5) from  0.0.0.0 .0, is 4(5) in cache
match(0x14dce5e0, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for ralplex1.buddha.ral.ibm.com is 4(5) from  0.0.0.0 .0, is 4(5) in cache
db_update: adding 14dce618
 processing group = FTPRAL.ralplex1.buddha.ral.ibm.com count = 2
 minimum weight = 31
 processing server = mvs03a weight = 32 host = mvs03a
db_update(FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14dce650, 0x14dce650, 01, 0x14dcd880)
match(0x14dcd908, 1, 6) 1, 6
db_update: adding 14dce650
db_update(mvs03a.FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14dce6c0, 0x14dce6c0, 01, 0x14dcd880)
savehash GROWING to 2
match(0x14dce650, 1, 6) 1, 1
match(0x14dce650, 1, 2) 1, 1
match(0x14dcd908, 1, 6) 1, 6
db_update: adding 14dce6c0
 processing server = MVS28A weight = 31 host = MVS28A
db_update(FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14dce748, 0x14dce748, 01, 0x14dcd880)
match(0x14dce650, 1, 6) 1, 1
match(0x14dce650, 1, 2) 1, 1
match(0x14dcd908, 1, 6) 1, 6
match(0x14dce650, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for FTPRAL.ralplex1.buddha.ral.ibm.com is 4(5) from  0.0.0.0 .0, is 4(5) in cache
db_update: adding 14dce748
db_update(MVS28A.FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14dce780, 0x14dce780, 01, 0x14dcd880)
match(0x14dce650, 1, 6) 1, 1
match(0x14dce748, 1, 6) 1, 1
match(0x14dce650, 1, 2) 1, 1
match(0x14dce748, 1, 2) 1, 1
match(0x14dcd908, 1, 6) 1, 6
db_update: adding 14dce780
 processing group = TNRAL.ralplex1.buddha.ral.ibm.com count = 2
 minimum weight = 31
 processing server = mvs03a weight = 32 host = mvs03a
db_update(TNRAL.ralplex1.buddha.ral.ibm.com, 0x14dce7f0, 0x14dce7f0, 01, 0x14dcd880)
match(0x14dcd908, 1, 6) 1, 6
db_update: adding 14dce7f0
db_update(mvs03a.TNRAL.ralplex1.buddha.ral.ibm.com, 0x14dce860, 0x14dce860, 01, 0x14dcd880)
savehash GROWING to 2
match(0x14dce7f0, 1, 6) 1, 1
match(0x14dce7f0, 1, 2) 1, 1
match(0x14dcd908, 1, 6) 1, 6
db_update: adding 14dce860
 processing server = MVS28A weight = 31 host = MVS28A
db_update(TNRAL.ralplex1.buddha.ral.ibm.com, 0x14dce8e8, 0x14dce8e8, 01, 0x14dcd880)
match(0x14dce7f0, 1, 6) 1, 1
match(0x14dce7f0, 1, 2) 1, 1
match(0x14dcd908, 1, 6) 1, 6
match(0x14dce7f0, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for TNRAL.ralplex1.buddha.ral.ibm.com is 4(5) from  0.0.0.0 .0, is 4(5) in cache
db_update: adding 14dce8e8
db_update(MVS28A.TNRAL.ralplex1.buddha.ral.ibm.com, 0x14dce920, 0x14dce920, 01, 0x14dcd880)
match(0x14dce7f0, 1, 6) 1, 1
match(0x14dce8e8, 1, 6) 1, 1
match(0x14dce7f0, 1, 2) 1, 1
match(0x14dce8e8, 1, 2) 1, 1
match(0x14dcd908, 1, 6) 1, 6
db_update: adding 14dce920
 processing group = FTPOERAL.ralplex1.buddha.ral.ibm.com count = 2
 minimum weight = 31
 processing server = MVS03C weight = 32 host = MVS03C
db_update(FTPOERAL.ralplex1.buddha.ral.ibm.com, 0x14dce990, 0x14dce990, 01, 0x14dcd880)
match(0x14dcd908, 1, 6) 1, 6
```

```
db_update: adding 14dce990
db_update(MVS03C.FTPOERAL.ralplex1.buddha.ral.ibm.com, 0x14dcea08, 0x14dcea08, 01, 0x14dcd880)
savehash GROWING to 2
match(0x14dce990, 1, 6) 1, 1
match(0x14dce990, 1, 2) 1, 1
match(0x14dcd908, 1, 6) 1, 6
db_update: adding 14dcea08
 processing server = MVS28C weight = 31 host = MVS28C
db_update(FTPOERAL.ralplex1.buddha.ral.ibm.com, 0x14dcea90, 0x14dcea90, 01, 0x14dcd880)
match(0x14dce990, 1, 6) 1, 1
match(0x14dce990, 1, 2) 1, 1
match(0x14dcd908, 1, 6) 1, 6
match(0x14dce990, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for FTPOERAL.ralplex1.buddha.ral.ibm.com is 4(5) from  0.0.0.0 .0, is 4(5) in cache
db_update: adding 14dcea90
db_update(MVS28C.FTPOERAL.ralplex1.buddha.ral.ibm.com, 0x14dceac8, 0x14dceac8, 01, 0x14dcd880)
match(0x14dce990, 1, 6) 1, 1
match(0x14dcea90, 1, 6) 1, 1
match(0x14dce990, 1, 2) 1, 1
match(0x14dcea90, 1, 2) 1, 1
match(0x14dcd908, 1, 6) 1, 6
db_update: adding 14dceac8
db_update(ralplex1.ralplex1.buddha.ral.ibm.com, 0x14dcdef0, 0x14dcdef0, 01, 0x14dcd880)
match(0x14dcde08, 1, 6) 1, 5
match(0x14dcde08, 1, 2) 1, 5
match(0x14dcd908, 1, 6) 1, 6
match(0x14dcde08, 1, 5) 1, 5
db_update: flags = 0x1, sizes = 28, 28 (cmp 0)
credibility for ralplex1.ralplex1.buddha.ral.ibm.com is 4(5) from  0.0.0.0 .0, is 4(5) in cache
fclose(7) succeeded
zone 1  type 2: 'ralplex1.buddha.ral.ibm.com' z_time 895503283, z_refresh 3600
zone origin 168.192.in-addr.arpa, new zone
 addrs: 192.168.236.3, addrcnt = 1
purge_zone(168.192.in-addr.arpa,1)
db_load(rback, 168.192.in-addr.arpa, 2, Nil)
db_load: origin 192.in-addr.arpa., buf 168.192.in-addr.arpa
db_load: origin now 192.in-addr.arpa
d='168.192.in-addr.arpa',  c=1, t=6, ttl=0, data='mvs03a.ralplex1.buddha.ral.ibm.com.'
db_update(168.192.in-addr.arpa, 0x14dcdf80, 0x14dcdf80, 01, 0x14dcd880)
savehash GROWING to 2
savehash GROWING to 2
savehash GROWING to 2
db_update: adding 14dcdf80
d='168.192.in-addr.arpa',  c=1, t=2, ttl=0, data='mvs03a.ralplex1.buddha.ral.ibm.com.'
db_update(168.192.in-addr.arpa, 0x14dce138, 0x14dce138, 01, 0x14dcd880)
match(0x14dcdf80, 1, 6) 1, 6
match(0x14dcdf80, 1, 2) 1, 6
db_update: adding 14dce138
db_load: origin 221.168.192.in-addr.arpa., buf 192.in-addr.arpa
db_load: origin now 221.168.192.in-addr.arpa
d='28.221.168.192.in-addr.arpa',  c=1, t=12, ttl=0, data='mvs28a.ralplex1.buddha.ral.ibm.com.'
db_update(28.221.168.192.in-addr.arpa, 0x14dce190, 0x14dce190, 01, 0x14dcd880)
savehash GROWING to 2
savehash GROWING to 2
match(0x14dcdf80, 1, 6) 1, 6
db_update: adding 14dce190
d='4.221.168.192.in-addr.arpa',  c=1, t=12, ttl=0, data='mvs03c.ralplex1.buddha.ral.ibm.com.'
db_update(4.221.168.192.in-addr.arpa, 0x14dce288, 0x14dce288, 01, 0x14dcd880)
match(0x14dcdf80, 1, 6) 1, 6
db_update: adding 14dce288
d='29.221.168.192.in-addr.arpa',  c=1, t=12, ttl=0, data='mvs28c.ralplex1.buddha.ral.ibm.com.'
db_update(29.221.168.192.in-addr.arpa, 0x14dce318, 0x14dce318, 01, 0x14dcd880)
match(0x14dcdf80, 1, 6) 1, 6
db_update: adding 14dce318
d='3.221.168.192.in-addr.arpa',  c=1, t=12, ttl=0, data='mvs03a.ralplex1.buddha.ral.ibm.com.'
```

```
db_update(3.221.168.192.in-addr.arpa, 0x14dce3a8, 0x14dce3a8, 01, 0x14dcd880)
match(0x14dcdf80, 1, 6) 1, 6
db_update: adding 14dce3a8
fclose(7) succeeded
zone 2  type 2: '168.192.in-addr.arpa' z_time 895512064, z_refresh 10
zone origin 0.0.127.in-addr.arpa, new zone, source = mvs28a.lbk
purge_zone(0.0.127.in-addr.arpa,1)
reloading zone
db_load(mvs28a.lbk, 0.0.127.in-addr.arpa, 3, Nil)
d='0.0.127.in-addr.arpa',  c=1, t=6, ttl=0, data='mvs28a.ralplex1.buddha.ral.ib.com'
db_update(0.0.127.in-addr.arpa, 0x14dce480, 0x14dce480, 01, 0x14dcd880)
savehash GROWING to 2
savehash GROWING to 2
db_update: adding 14dce480
d='0.0.127.in-addr.arpa',  c=1, t=2, ttl=0, data='mvs28a.raplplex1.buddha.ral.ibm.com.'
db_update(0.0.127.in-addr.arpa, 0x14dcebc0, 0x14dcebc0, 01, 0x14dcd880)
match(0x14dce480, 1, 6) 1, 6
match(0x14dce480, 1, 2) 1, 6
db_update: adding 14dcebc0
d='1.0.0.127.in-addr.arpa',  c=1, t=12, ttl=0, data='loopback.'
db_update(1.0.0.127.in-addr.arpa, 0x14dcec18, 0x14dcec18, 01, 0x14dcd880)
savehash GROWING to 2
match(0x14dce480, 1, 6) 1, 6
db_update: adding 14dcec18
fclose(7) succeeded
zone 3  type 1: '0.0.127.in-addr.arpa' z_time 895512121, z_refresh 10
fclose(6) succeeded

 content of zones after loading
printzoneinfo(1):
origin ='ralplex1.buddha.ral.ibm.com',  class = 1, type = Static secondary, source = fback
z_refresh = 3600, retry = 3600, expire = 14400, minimum = 60, serial = 19980507
z_time = 895503283, now time : 895512116 sec, time left: 4294958463 sec; flags 20041
printzoneinfo(2):
origin ='168.192.in-addr.arpa',  class = 1, type = Static secondary, source = rback
z_refresh = 10, retry = 10, expire = 10, minimum = 10, serial = 10004
z_time = 895512064, now time : 895512116 sec, time left: 4294967244 sec; flags 41
printzoneinfo(3):
origin ='0.0.127.in-addr.arpa',  class = 1, type = Static primary, source = mvs28a.lbk
z_refresh = 10, retry = 10, expire = 10, minimum = 10, serial = 10004
z_time = 895512121, now time : 895512116 sec, time left: 5 sec; flags 41
exit ns_init(), need maintenance immediately

bootfile = /etc/named.boot
Network and sort list:
addr xc0a8fb00 mask xffffff00 my_addr xc0a8fb1c 192.168.251.28
addr x9000000 mask xff000000 my_addr x918689e 9.24.104.158
addr xc0a8ec03 mask xffffffff my_addr xc0a8ec1c 192.168.236.28
addr xc0a8dd00 mask xffffff00 my_addr xc0a8dd1c 192.168.221.28
addr xc0a8dd00 mask xffffff00 my_addr xc0a8dd1a 192.168.221.26
addr xc0a86d02 mask xffffffff my_addr xc0a86d1c 192.168.109.28
addr xc0a86400 mask xffffff00 my_addr xc0a86463 192.168.100.99
addr xc0a8ec00 mask xffffff00 my_addr xc0a8ec1c 192.168.236.28
addr xc0a86d00 mask xffffff00 my_addr xc0a86d1c 192.168.109.28
database initialized
fclose(6) succeeded
prime_cache: priming = 0
.
.
.
.
.
.
.
.
datagram from  192.168.221.120 .1089, fd 5, len 32; now Mon May 18 17:22:35 1998
```

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1, type = A, class = IN


;; ...truncated
ns_req(from= 192.168.221.120 .1089)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1, type = A, class = IN


;; ...truncated
req: nlookup(tnral.ralplex1) id 1 type=1 class=1
req: missed 'tnral.ralplex1' as '' (cname=0)
findns: using cache
findns: using hints
findns: No root nameservers for class IN?
ns_req: answer ->  192.168.221.120 .1089 fd=5 id=1 size=32 Local
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 1
;; flags: qr rd ra; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1, type = A, class = IN


;; ...truncated

datagram from  192.168.221.120 .1090, fd 5, len 51; now Mon May 18 17:22:35 1998
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
ns_req(from= 192.168.221.120 .1090)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
req: nlookup(tnral.ralplex1.buddha.ral.ibm.com) id 2 type=1 class=1
req: found 'tnral.ralplex1.buddha.ral.ibm.com' as 'tnral.ralplex1.buddha.ral.ibm.com' (cname=0)
******* Remove_Expired_RRs  Entered *******
name: TNRAL
  type=1 size=4
  type=1 size=4
Remove_Expired_RRs rc=0
 Looking for WLM generated Address
make_rr(tnral.ralplex1.buddha.ral.ibm.com, 14dcf008, bf3a3, 449, 1) 4 zone 1 ttl 0
match(0x14dcf008, 1, 24) 1, 1
match(0x14dcf100, 1, 24) 1, 1
 Weight = 0
 Current Server Weight 0
req: foundname=1, count=1, founddata=1, cname=0
sort_response(1)
findns: np 0x14dcf040 'TNRAL'
match(0x14dcf008, 1, 6) 1, 1
match(0x14dcf100, 1, 6) 1, 1
match(0x14dcf008, 1, 2) 1, 1
match(0x14dcf100, 1, 2) 1, 1
findns: np 0x14dcdaf0 'ralplex1'
```

```
match(0x14dcd918, 1, 6) 1, 6
findns: SOA found
req: leaving (tnral.ralplex1.buddha.ral.ibm.com, rcode 0)
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
findns: np 0x14dcdaf0 'ralplex1'
match(0x14dcd918, 1, 2) 1, 6
match(0x14dcdb30, 1, 2) 1, 2
match(0x14dceac0, 1, 2) 1, 1
match(0x14dceaf8, 1, 2) 1, 1
match(0x14dcedf8, 1, 2) 1, 1
match(0x14dcee30, 1, 2) 1, 1
findns: 1 NS's added for 'ralplex1'
make_rr(ralplex1.buddha.ral.ibm.com, 14dcdb30, bf3b3, 433, 1) 35 zone 1 ttl 60
match(0x14dcd918, 1, 24) 1, 6
match(0x14dcdb30, 1, 24) 1, 2
match(0x14dceac0, 1, 24) 1, 1
match(0x14dceaf8, 1, 24) 1, 1
match(0x14dcedf8, 1, 24) 1, 1
match(0x14dcee30, 1, 24) 1, 1
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
doaddinfo() addcount = 1
do additional "mvs03a.ralplex1.buddha.ral.ibm.com" (from "ralplex1.buddha.ral.ibm.com")
found it
match(0x14dcdb88, 1, 5) 1, 1
match(0x14dcdb88, 1, 5) 1, 1
match(0x14dcdb88, 1, 1) 1, 1
make_rr(mvs03a.ralplex1.buddha.ral.ibm.com, 14dcdb88, bf3e3, 385, 0) 4 zone 1 ttl 60
match(0x14dcdb88, 1, 24) 1, 1
addinfo: adding address data n = 16
ns_req: answer ->  192.168.221.120 .1090 fd=5 id=2 size=131 Local
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2
;; flags: qr aa rd ra; Ques: 1, Ans: 1, Auth: 1, Addit: 1
;; QUESTIONS:
;; tnral.ralplex1.buddha.ral.ibm.com, type = A, class = IN

;; ANSWERS:
tnral.ralplex1.buddha.ral.ibm.com. 0 IN A 192.168.221.3

;; AUTHORITY RECORDS:
ralplex1.buddha.ral.ibm.com. 60 IN NS mvs03a.ralplex1.buddha.ral.ibm.com.

;; ADDITIONAL RECORDS:
mvs03a.ralplex1.buddha.ral.ibm.com. 60 IN A 192.168.221.3


datagram from  192.168.221.120 .1091, fd 5, len 32; now Mon May 18 17:22:36 1998
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1, type = A, class = IN


;; ...truncated
ns_req(from= 192.168.221.120 .1091)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1, type = A, class = IN


;; ...truncated
req: nlookup(tnral.ralplex1) id 1 type=1 class=1
req: missed 'tnral.ralplex1' as '' (cname=0)
findns: using cache
findns: using hints
findns: No root nameservers for class IN?
```

```
ns_req: answer ->  192.168.221.120 .1091 fd=5 id=1 size=32 Local
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 1
;; flags: qr rd ra; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1, type = A, class = IN


;; ...truncated

datagram from  192.168.221.120 .1092, fd 5, len 51; now Mon May 18 17:22:36 1998
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
ns_req(from= 192.168.221.120 .1092)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
req: nlookup(tnral.ralplex1.buddha.ral.ibm.com) id 2 type=1 class=1
req: found 'tnral.ralplex1.buddha.ral.ibm.com' as 'tnral.ralplex1.buddha.ral.ibm.com' (cname=0)
******* Remove_Expired_RRs  Entered *******
name: TNRAL
  type=1 size=4
  type=1 size=4
Remove_Expired_RRs rc=0
 Looking for WLM generated Address
make_rr(tnral.ralplex1.buddha.ral.ibm.com, 14dcf100, bf3a3, 449, 1) 4 zone 1 ttl 0
match(0x14dcf008, 1, 24) 1, 1
match(0x14dcf100, 1, 24) 1, 1
 Weight = 0
 Current Server Weight 0
 Initializing all Weights
req: foundname=1, count=1, founddata=1, cname=0
sort_response(1)
findns: np 0x14dcf040 'TNRAL'
match(0x14dcf008, 1, 6) 1, 1
match(0x14dcf100, 1, 6) 1, 1
match(0x14dcf008, 1, 2) 1, 1
match(0x14dcf100, 1, 2) 1, 1
findns: np 0x14dcdaf0 'ralplex1'
match(0x14dcd918, 1, 6) 1, 6
findns: SOA found
req: leaving (tnral.ralplex1.buddha.ral.ibm.com, rcode 0)
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
findns: np 0x14dcdaf0 'ralplex1'
match(0x14dcd918, 1, 2) 1, 6
match(0x14dcdb30, 1, 2) 1, 2
match(0x14dceac0, 1, 2) 1, 1
match(0x14dceaf8, 1, 2) 1, 1
match(0x14dcedf8, 1, 2) 1, 1
match(0x14dcee30, 1, 2) 1, 1
findns: 1 NS's added for 'ralplex1'
make_rr(ralplex1.buddha.ral.ibm.com, 14dcdb30, bf3b3, 433, 1) 35 zone 1 ttl 60
match(0x14dcd918, 1, 24) 1, 6
match(0x14dcdb30, 1, 24) 1, 2
match(0x14dceac0, 1, 24) 1, 1
match(0x14dceaf8, 1, 24) 1, 1
match(0x14dcedf8, 1, 24) 1, 1
match(0x14dcee30, 1, 24) 1, 1
```

```
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
doaddinfo() addcount = 1
do additional "mvs03a.ralplex1.buddha.ral.ibm.com" (from "ralplex1.buddha.ral.ibm.com")
found it
match(0x14dcdb88, 1, 5) 1, 1
match(0x14dcdb88, 1, 5) 1, 1
match(0x14dcdb88, 1, 1) 1, 1
make_rr(mvs03a.ralplex1.buddha.ral.ibm.com, 14dcdb88, bf3e3, 385, 0) 4 zone 1 ttl 60
match(0x14dcdb88, 1, 24) 1, 1
addinfo: adding address data n = 16
ns_req: answer ->  192.168.221.120 .1092 fd=5 id=2 size=131 Local
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2
;; flags: qr aa rd ra; Ques: 1, Ans: 1, Auth: 1, Addit: 1
;; QUESTIONS:
;; tnral.ralplex1.buddha.ral.ibm.com, type = A, class = IN

;; ANSWERS:
tnral.ralplex1.buddha.ral.ibm.com. 0 IN A 192.168.221.28

;; AUTHORITY RECORDS:
ralplex1.buddha.ral.ibm.com. 60 IN NS mvs03a.ralplex1.buddha.ral.ibm.com.

;; ADDITIONAL RECORDS:
mvs03a.ralplex1.buddha.ral.ibm.com. 60 IN A 192.168.221.3


datagram from  192.168.221.120 .1093, fd 5, len 32; now Mon May 18 17:22:36 1998
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1, type = A, class = IN


;; ...truncated
ns_req(from= 192.168.221.120 .1093)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1, type = A, class = IN


;; ...truncated
req: nlookup(tnral.ralplex1) id 1 type=1 class=1
req: missed 'tnral.ralplex1' as '' (cname=0)
findns: using cache
findns: using hints
findns: No root nameservers for class IN?
ns_req: answer ->  192.168.221.120 .1093 fd=5 id=1 size=32 Local
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 1
;; flags: qr rd ra; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1, type = A, class = IN


;; ...truncated

datagram from  192.168.221.120 .1094, fd 5, len 51; now Mon May 18 17:22:36 1998
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
ns_req(from= 192.168.221.120 .1094)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2
```

```
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
req: nlookup(tnral.ralplex1.buddha.ral.ibm.com) id 2 type=1 class=1
req: found 'tnral.ralplex1.buddha.ral.ibm.com' as 'tnral.ralplex1.buddha.ral.ibm.com' (cname=0)
******* Remove_Expired_RRs  Entered *******
name: TNRAL
  type=1 size=4
  type=1 size=4
Remove_Expired_RRs rc=0
 Looking for WLM generated Address
make_rr(tnral.ralplex1.buddha.ral.ibm.com, 14dcf008, bf3a3, 449, 1) 4 zone 1 ttl 0
match(0x14dcf008, 1, 24) 1, 1
match(0x14dcf100, 1, 24) 1, 1
 Weight = 0
 Current Server Weight 0
req: foundname=1, count=1, founddata=1, cname=0
sort_response(1)
findns: np 0x14dcf040 'TNRAL'
match(0x14dcf008, 1, 6) 1, 1
match(0x14dcf100, 1, 6) 1, 1
match(0x14dcf008, 1, 2) 1, 1
match(0x14dcf100, 1, 2) 1, 1
findns: np 0x14dcdaf0 'ralplex1'
match(0x14dcd918, 1, 6) 1, 6
findns: SOA found
req: leaving (tnral.ralplex1.buddha.ral.ibm.com, rcode 0)
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
findns: np 0x14dcdaf0 'ralplex1'
match(0x14dcd918, 1, 2) 1, 6
match(0x14dcdb30, 1, 2) 1, 2
match(0x14dceac0, 1, 2) 1, 1
match(0x14dceaf8, 1, 2) 1, 1
match(0x14dcedf8, 1, 2) 1, 1
match(0x14dcee30, 1, 2) 1, 1
findns: 1 NS's added for 'ralplex1'
make_rr(ralplex1.buddha.ral.ibm.com, 14dcdb30, bf3b3, 433, 1) 35 zone 1 ttl 60
match(0x14dcd918, 1, 24) 1, 6
match(0x14dcdb30, 1, 24) 1, 2
match(0x14dceac0, 1, 24) 1, 1
match(0x14dceaf8, 1, 24) 1, 1
match(0x14dcedf8, 1, 24) 1, 1
match(0x14dcee30, 1, 24) 1, 1
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
doaddinfo() addcount = 1
do additional "mvs03a.ralplex1.buddha.ral.ibm.com" (from "ralplex1.buddha.ral.ibm.com")
found it
match(0x14dcdb88, 1, 5) 1, 1
match(0x14dcdb88, 1, 5) 1, 1
match(0x14dcdb88, 1, 1) 1, 1
make_rr(mvs03a.ralplex1.buddha.ral.ibm.com, 14dcdb88, bf3e3, 385, 0) 4 zone 1 ttl 60
match(0x14dcdb88, 1, 24) 1, 1
addinfo: adding address data n = 16
ns_req: answer ->  192.168.221.120 .1094 fd=5 id=2 size=131 Local
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2
;; flags: qr aa rd ra; Ques: 1, Ans: 1, Auth: 1, Addit: 1
;; QUESTIONS:
;; tnral.ralplex1.buddha.ral.ibm.com, type = A, class = IN

;; ANSWERS:
tnral.ralplex1.buddha.ral.ibm.com. 0 IN A 192.168.221.3
```

```
;; AUTHORITY RECORDS:
ralplex1.buddha.ral.ibm.com. 60 IN NS mvs03a.ralplex1.buddha.ral.ibm.com.

;; ADDITIONAL RECORDS:
mvs03a.ralplex1.buddha.ral.ibm.com. 60 IN A 192.168.221.3


datagram from  192.168.221.120 .1095, fd 5, len 32; now Mon May 18 17:22:36 1998
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1, type = A, class = IN


;; ...truncated
ns_req(from= 192.168.221.120 .1095)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1, type = A, class = IN


;; ...truncated
req: nlookup(tnral.ralplex1) id 1 type=1 class=1
req: missed 'tnral.ralplex1' as '' (cname=0)
findns: using cache
findns: using hints
findns: No root nameservers for class IN?
ns_req: answer ->  192.168.221.120 .1095 fd=5 id=1 size=32 Local
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 1
;; flags: qr rd ra; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1, type = A, class = IN


;; ...truncated

datagram from  192.168.221.120 .1096, fd 5, len 51; now Mon May 18 17:22:36 1998
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
ns_req(from= 192.168.221.120 .1096)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.ralplex1.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
req: nlookup(tnral.ralplex1.buddha.ral.ibm.com) id 2 type=1 class=1
req: found 'tnral.ralplex1.buddha.ral.ibm.com' as 'tnral.ralplex1.buddha.ral.ibm.com' (cname=0)
******* Remove_Expired_RRs  Entered *******
name: TNRAL
  type=1 size=4
  type=1 size=4
Remove_Expired_RRs rc=0
 Looking for WLM generated Address
make_rr(tnral.ralplex1.buddha.ral.ibm.com, 14dcf100, bf3a3, 449, 1) 4 zone 1 ttl 0
match(0x14dcf008, 1, 24) 1, 1
match(0x14dcf100, 1, 24) 1, 1
 Weight = 0
 Current Server Weight 0
```

```
 Initializing all Weights
req: foundname=1, count=1, founddata=1, cname=0
sort_response(1)
findns: np 0x14dcf040 'TNRAL'
match(0x14dcf008, 1, 6) 1, 1
match(0x14dcf100, 1, 6) 1, 1
match(0x14dcf008, 1, 2) 1, 1
match(0x14dcf100, 1, 2) 1, 1
findns: np 0x14dcdaf0 'ralplex1'
match(0x14dcd918, 1, 6) 1, 6
findns: SOA found
req: leaving (tnral.ralplex1.buddha.ral.ibm.com, rcode 0)
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
findns: np 0x14dcdaf0 'ralplex1'
match(0x14dcd918, 1, 2) 1, 6
match(0x14dcdb30, 1, 2) 1, 2
match(0x14dceac0, 1, 2) 1, 1
match(0x14dceaf8, 1, 2) 1, 1
match(0x14dcedf8, 1, 2) 1, 1
match(0x14dcee30, 1, 2) 1, 1
findns: 1 NS's added for 'ralplex1'
make_rr(ralplex1.buddha.ral.ibm.com, 14dcdb30, bf3b3, 433, 1) 35 zone 1 ttl 60
match(0x14dcd918, 1, 24) 1, 6
match(0x14dcdb30, 1, 24) 1, 2
match(0x14dceac0, 1, 24) 1, 1
match(0x14dceaf8, 1, 24) 1, 1
match(0x14dcedf8, 1, 24) 1, 1
match(0x14dcee30, 1, 24) 1, 1
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
doaddinfo() addcount = 1
do additional "mvs03a.ralplex1.buddha.ral.ibm.com" (from "ralplex1.buddha.ral.ibm.com")
found it
match(0x14dcdb88, 1, 5) 1, 1
match(0x14dcdb88, 1, 5) 1, 1
match(0x14dcdb88, 1, 1) 1, 1
make_rr(mvs03a.ralplex1.buddha.ral.ibm.com, 14dcdb88, bf3e3, 385, 0) 4 zone 1 ttl 60
match(0x14dcdb88, 1, 24) 1, 1
addinfo: adding address data n = 16
ns_req: answer ->  192.168.221.120 .1096 fd=5 id=2 size=131 Local
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2
;; flags: qr aa rd ra; Ques: 1, Ans: 1, Auth: 1, Addit: 1
;; QUESTIONS:
;; tnral.ralplex1.buddha.ral.ibm.com, type = A, class = IN

;; ANSWERS:
tnral.ralplex1.buddha.ral.ibm.com. 0 IN A 192.168.221.28

;; AUTHORITY RECORDS:
ralplex1.buddha.ral.ibm.com. 60 IN NS mvs03a.ralplex1.buddha.ral.ibm.com.

;; ADDITIONAL RECORDS:
mvs03a.ralplex1.buddha.ral.ibm.com. 60 IN A 192.168.221.3
```

Appendix G. DNS Trace for VIPA Configuration

The following is the trace of DNS including the first 10 datagrams received from the workstation at 192.168.221.88.

```
Debug turned ON, Level 11
Version = @(#) ddns/ns/ns_main.c, dns_ns, dns_r1.1 1.62  9/23/97 10:57:21
main: __iptcpn() returns TCPIPjobname 'T28ATCP'
considering  192.168.252.28
considering  192.168.212.28
considering  192.168.236.28
considering  9.24.104.158
considering  192.168.221.28
considering  192.168.231.28
considering  192.168.241.28
considering  192.168.109.28
considering  192.168.100.99
considering  192.168.202.28
loopback address: x7f000001
dqp->dq_addr 0.0.0.0 d_dfd 5

ns_init(/etc/named.boot)
savehash GROWING to 2
savehash GROWING to 2

 content of zones before loading
zone origin buddha.ral.ibm.com, new zone
 addrs: 192.168.250.3, addrcnt = 1
purge_zone(buddha.ral.ibm.com,1)
db_load(fback, buddha.ral.ibm.com, 1, Nil)
db_load: origin ral.ibm.com., buf buddha.ral.ibm.com
db_load: origin now ral.ibm.com
d='buddha.ral.ibm.com',  c=1, t=6, ttl=0, data='mvs03a.buddha.ral.ibm.com.'
db_update(buddha.ral.ibm.com, 0x14dcd938, 0x14dcd938, 01, 0x14dcd8b8)
savehash GROWING to 2
savehash GROWING to 2
savehash GROWING to 2
db_update: adding 14dcd938
d='buddha.ral.ibm.com',  c=1, t=2, ttl=0, data='mvs03a.buddha.ral.ibm.com.'
db_update(buddha.ral.ibm.com, 0x14dcdae0, 0x14dcdae0, 01, 0x14dcd8b8)
match(0x14dcd938, 1, 6) 1, 6
match(0x14dcd938, 1, 2) 1, 6
db_update: adding 14dcdae0
db_load: origin buddha.ral.ibm.com., buf ral.ibm.com
db_load: origin now buddha.ral.ibm.com
d='ftpral.buddha.ral.ibm.com',  c=1, t=5, ttl=0, data='ftpral.ralplex1.buddha.ral.ibm.com.'
db_update(ftpral.buddha.ral.ibm.com, 0x14dcdb30, 0x14dcdb30, 01, 0x14dcd8b8)
savehash GROWING to 2
match(0x14dcd938, 1, 6) 1, 6
db_update: adding 14dcdb30
d='mvsdum.buddha.ral.ibm.com',  c=1, t=1, ttl=0, data='192.168.221.99'
db_update(mvsdum.buddha.ral.ibm.com, 0x14dcdbd8, 0x14dcdbd8, 01, 0x14dcd8b8)
match(0x14dcd938, 1, 6) 1, 6
db_update: adding 14dcdbd8
d='mvs03a.buddha.ral.ibm.com',  c=1, t=1, ttl=0, data='192.168.249.3'
db_update(mvs03a.buddha.ral.ibm.com, 0x14dcdc48, 0x14dcdc48, 01, 0x14dcd8b8)
match(0x14dcd938, 1, 6) 1, 6
db_update: adding 14dcdc48
d='mvs03a.buddha.ral.ibm.com',  c=1, t=1, ttl=0, data='192.168.202.3'
db_update(mvs03a.buddha.ral.ibm.com, 0x14dcdcb8, 0x14dcdcb8, 01, 0x14dcd8b8)
match(0x14dcdc48, 1, 6) 1, 1
match(0x14dcdc48, 1, 2) 1, 1
match(0x14dcd938, 1, 6) 1, 6
match(0x14dcdc48, 1, 1) 1, 1
```

```
db_update: flags = 0x1, sizes = 4, 4 (cmp 1)
credibility for mvs03a.buddha.ral.ibm.com is 4(4) from  0.0.0.0 .0, is 4(4) in cache
db_update: adding 14dcdcb8
d='mvs03a.buddha.ral.ibm.com',  c=1, t=1, ttl=0, data='192.168.236.3'
db_update(mvs03a.buddha.ral.ibm.com, 0x14dcdcf0, 0x14dcdcf0, 01, 0x14dcd8b8)
match(0x14dcdc48, 1, 6) 1, 1
match(0x14dcdcb8, 1, 6) 1, 1
match(0x14dcdc48, 1, 2) 1, 1
match(0x14dcdcb8, 1, 2) 1, 1
match(0x14dcd938, 1, 6) 1, 6
match(0x14dcdc48, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp 1)
credibility for mvs03a.buddha.ral.ibm.com is 4(4) from  0.0.0.0 .0, is 4(4) in cache
match(0x14dcdcb8, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for mvs03a.buddha.ral.ibm.com is 4(4) from  0.0.0.0 .0, is 4(4) in cache
db_update: adding 14dcdcf0
d='mvs03a.buddha.ral.ibm.com',  c=1, t=1, ttl=0, data='192.168.221.3'
db_update(mvs03a.buddha.ral.ibm.com, 0x14dcdd28, 0x14dcdd28, 01, 0x14dcd8b8)
match(0x14dcdc48, 1, 6) 1, 1
match(0x14dcdcb8, 1, 6) 1, 1
match(0x14dcdcf0, 1, 6) 1, 1
match(0x14dcdc48, 1, 2) 1, 1
match(0x14dcdcb8, 1, 2) 1, 1
match(0x14dcdcf0, 1, 2) 1, 1
match(0x14dcd938, 1, 6) 1, 6
match(0x14dcdc48, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp 1)
credibility for mvs03a.buddha.ral.ibm.com is 4(4) from  0.0.0.0 .0, is 4(4) in cache
match(0x14dcdcb8, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for mvs03a.buddha.ral.ibm.com is 4(4) from  0.0.0.0 .0, is 4(4) in cache
match(0x14dcdcf0, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp 1)
credibility for mvs03a.buddha.ral.ibm.com is 4(4) from  0.0.0.0 .0, is 4(4) in cache
db_update: adding 14dcdd28
d='mvs03c.buddha.ral.ibm.com',  c=1, t=1, ttl=0, data='192.168.202.4'
db_update(mvs03c.buddha.ral.ibm.com, 0x14dcdd60, 0x14dcdd60, 01, 0x14dcd8b8)
match(0x14dcd938, 1, 6) 1, 6
db_update: adding 14dcdd60
d='mvs03c.buddha.ral.ibm.com',  c=1, t=1, ttl=0, data='192.168.221.4'
db_update(mvs03c.buddha.ral.ibm.com, 0x14dcddd0, 0x14dcddd0, 01, 0x14dcd8b8)
match(0x14dcdd60, 1, 6) 1, 1
match(0x14dcdd60, 1, 2) 1, 1
match(0x14dcd938, 1, 6) 1, 6
match(0x14dcdd60, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for mvs03c.buddha.ral.ibm.com is 4(4) from  0.0.0.0 .0, is 4(4) in cache
db_update: adding 14dcddd0
d='mvs03c.buddha.ral.ibm.com',  c=1, t=1, ttl=0, data='192.168.249.4'
db_update(mvs03c.buddha.ral.ibm.com, 0x14dcde08, 0x14dcde08, 01, 0x14dcd8b8)
match(0x14dcdd60, 1, 6) 1, 1
match(0x14dcddd0, 1, 6) 1, 1
match(0x14dcdd60, 1, 2) 1, 1
match(0x14dcddd0, 1, 2) 1, 1
match(0x14dcd938, 1, 6) 1, 6
match(0x14dcdd60, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for mvs03c.buddha.ral.ibm.com is 4(4) from  0.0.0.0 .0, is 4(4) in cache
match(0x14dcddd0, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for mvs03c.buddha.ral.ibm.com is 4(4) from  0.0.0.0 .0, is 4(4) in cache
db_update: adding 14dcde08
d='localhost.buddha.ral.ibm.com',  c=1, t=1, ttl=0, data='127.0.0.1'
db_update(localhost.buddha.ral.ibm.com, 0x14dcde40, 0x14dcde40, 01, 0x14dcd8b8)
savehash GROWING to 11
```

```
savehash(0x14dcdb88) cnt=5, sz=2, newsz=11
match(0x14dcd938, 1, 6) 1, 6
db_update: adding 14dcde40
d='mvs28a.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.236.28'
db_update(mvs28a.buddha.ral.ibm.com, 0x14dcdef8, 0x14dcdef8, 01, 0x14dcd8b8)
match(0x14dcd938, 1, 6) 1, 6
db_update: adding 14dcdef8
d='mvs28a.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.202.28'
db_update(mvs28a.buddha.ral.ibm.com, 0x14dcdf58, 0x14dcdf58, 01, 0x14dcd8b8)
match(0x14dcdef8, 1, 6) 1, 1
match(0x14dcdef8, 1, 2) 1, 1
match(0x14dcd938, 1, 6) 1, 6
match(0x14dcdef8, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp 1)
credibility for mvs28a.buddha.ral.ibm.com is 4(4) from  0.0.0.0 .0, is 4(4) in cache
db_update: adding 14dcdf58
d='mvs28a.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.221.28'
db_update(mvs28a.buddha.ral.ibm.com, 0x14dcdf90, 0x14dcdf90, 01, 0x14dcd8b8)
match(0x14dcdef8, 1, 6) 1, 1
match(0x14dcdf58, 1, 6) 1, 1
match(0x14dcdef8, 1, 2) 1, 1
match(0x14dcdf58, 1, 2) 1, 1
match(0x14dcd938, 1, 6) 1, 6
match(0x14dcdef8, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp 1)
credibility for mvs28a.buddha.ral.ibm.com is 4(4) from  0.0.0.0 .0, is 4(4) in cache
match(0x14dcdf58, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for mvs28a.buddha.ral.ibm.com is 4(4) from  0.0.0.0 .0, is 4(4) in cache
db_update: adding 14dcdf90
d='ftpoeral.buddha.ral.ibm.com', c=1, t=5, ttl=0, data='ftpoeral.ralplex1.buddha.ral.ibm.com.'
db_update(ftpoeral.buddha.ral.ibm.com, 0x14dcdfc8, 0x14dcdfc8, 01, 0x14dcd8b8)
match(0x14dcd938, 1, 6) 1, 6
db_update: adding 14dcdfc8
d='tnral.buddha.ral.ibm.com', c=1, t=5, ttl=0, data='tnral.ralplex1.buddha.ral.ibm.com.'
db_update(tnral.buddha.ral.ibm.com, 0x14dce060, 0x14dce060, 01, 0x14dcd8b8)
match(0x14dcd938, 1, 6) 1, 6
db_update: adding 14dce060
d='ralplex1.buddha.ral.ibm.com', c=1, t=2, ttl=0, data='mvs03a.ralplex1.buddha.ral.ibm.com.'
db_update(ralplex1.buddha.ral.ibm.com, 0x14dce0f0, 0x14dce0f0, 01, 0x14dcd8b8)
match(0x14dcd938, 1, 6) 1, 6
db_update: adding 14dce0f0
db_load: origin ralplex1.buddha.ral.ibm.com., buf buddha.ral.ibm.com
db_load: origin now ralplex1.buddha.ral.ibm.com
d='mvs03a.ralplex1.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.250.3'
db_update(mvs03a.ralplex1.buddha.ral.ibm.com, 0x14dce188, 0x14dce188, 01, 0x14dcd8b8)
savehash GROWING to 2
match(0x14dce0f0, 1, 6) 1, 2
match(0x14dce0f0, 1, 2) 1, 2
db_update: adding 14dce188
db_load: origin buddha.ral.ibm.com., buf ralplex1.buddha.ral.ibm.com
db_load: origin now buddha.ral.ibm.com
d='thatpc.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.221.88'
db_update(thatpc.buddha.ral.ibm.com, 0x14dce210, 0x14dce210, 01, 0x14dcd8b8)
match(0x14dcd938, 1, 6) 1, 6
db_update: adding 14dce210
fclose(7) succeeded
zone 1  type 2: 'buddha.ral.ibm.com' z_time 896884162, z_refresh 7200
zone origin ralplex1.buddha.ral.ibm.com, new zone
 addrs: 192.168.250.3, addrcnt = 1
purge_zone(ralplex1.buddha.ral.ibm.com,1)
rm_datum(14dce0f0, 14dce0f0, 0) -> 0
bottom_of_zone() == 0
rm_datum(14dce188, 14dce188, 0) -> 0
purge_zone: cleaning cache
purge_zone: found our selves
```

```
db_load(xback, ralplex1.buddha.ral.ibm.com, 2, Nil)
db_load: origin buddha.ral.ibm.com., buf ralplex1.buddha.ral.ibm.com
db_load: origin now buddha.ral.ibm.com
d='ralplex1.buddha.ral.ibm.com', c=1, t=6, ttl=0, data='mvs03a.ralplex1.buddha.ral.ibm.com.'
db_update(ralplex1.buddha.ral.ibm.com, 0x14dce100, 0x14dce100, 01, 0x14dcd8b8)
match(0x14dcd938, 1, 6) 1, 6
db_update: adding 14dce100
d='ralplex1.buddha.ral.ibm.com', c=1, t=2, ttl=0, data='mvs03a.ralplex1.buddha.ral.ibm.com.'
db_update(ralplex1.buddha.ral.ibm.com, 0x14dce2b8, 0x14dce2b8, 01, 0x14dcd8b8)
match(0x14dce100, 1, 6) 1, 6
match(0x14dce100, 1, 2) 1, 6
db_update: adding 14dce2b8
db_load: origin ralplex1.buddha.ral.ibm.com., buf buddha.ral.ibm.com
db_load: origin now ralplex1.buddha.ral.ibm.com
d='mvs03a.ralplex1.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.250.3'
db_update(mvs03a.ralplex1.buddha.ral.ibm.com, 0x14dce1d8, 0x14dce1d8, 01, 0x14dcd8b8)
savehash GROWING to 2
match(0x14dce100, 1, 6) 1, 6
db_update: adding 14dce1d8
d='mvs03c.ralplex1.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.251.4'
db_update(mvs03c.ralplex1.buddha.ral.ibm.com, 0x14dce360, 0x14dce360, 01, 0x14dcd8b8)
match(0x14dce100, 1, 6) 1, 6
db_update: adding 14dce360
d='mvs28a.ralplex1.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.252.28'
db_update(mvs28a.ralplex1.buddha.ral.ibm.com, 0x14dce3d0, 0x14dce3d0, 01, 0x14dcd8b8)
match(0x14dce100, 1, 6) 1, 6
db_update: adding 14dce3d0
d='ralplex1.ralplex1.buddha.ral.ibm.com', c=1, t=5, ttl=0, data='ralplex1.buddha.ral.ibm.com.'
db_update(ralplex1.ralplex1.buddha.ral.ibm.com, 0x14dce440, 0x14dce440, 01, 0x14dcd8b8)
match(0x14dce100, 1, 6) 1, 6
db_update: adding 14dce440
wlm_load ralplex1.buddha.ral.ibm.com
 group list retcode = -1 rsncode = 1034
 group list retcode = 0 rsncode = 0
 group list entry_count = 3
 querying group =  FTPRAL
 server list retcode = -1 rsncode = 1034
 server list retcode = 0 rsncode = 0
 querying group =  TNRAL
 server list retcode = -1 rsncode = 1034
 server list retcode = 0 rsncode = 0
 querying group =  TCPIP
 server list retcode = -1 rsncode = 1034
 server list retcode = 0 rsncode = 0
 processing group = FTPRAL.ralplex1.buddha.ral.ibm.com count = 2
 minimum weight = 21
 processing server = MVS03A weight = 42 host = MVS03A
db_update(FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14dce8d8, 0x14dce8d8, 01, 0x14dcd8b8)
savehash GROWING to 11
savehash(0x14dce310) cnt=5, sz=2, newsz=11
match(0x14dce100, 1, 6) 1, 6
db_update: adding 14dce8d8
db_update(MVS03A.FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14dce988, 0x14dce988, 01, 0x14dcd8b8)
savehash GROWING to 2
match(0x14dce8d8, 1, 6) 1, 1
match(0x14dce8d8, 1, 2) 1, 1
match(0x14dce100, 1, 6) 1, 6
db_update: adding 14dce988
 processing server = MVS28A weight = 21 host = MVS28A
db_update(FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14dce9f8, 0x14dce9f8, 01, 0x14dcd8b8)
match(0x14dce8d8, 1, 6) 1, 1
match(0x14dce8d8, 1, 2) 1, 1
match(0x14dce100, 1, 6) 1, 6
match(0x14dce8d8, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for FTPRAL.ralplex1.buddha.ral.ibm.com is 4(5) from  0.0.0.0 .0, is 4(5) in cache
```

```
db_update: adding 14dce9f8
db_update(MVS28A.FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14dcea30, 0x14dcea30, 01, 0x14dcd8b8)
match(0x14dce8d8, 1, 6) 1, 1
match(0x14dce9f8, 1, 6) 1, 1
match(0x14dce8d8, 1, 2) 1, 1
match(0x14dce9f8, 1, 2) 1, 1
match(0x14dce100, 1, 6) 1, 6
db_update: adding 14dcea30
 processing group = TNRAL.ralplex1.buddha.ral.ibm.com count = 2
 minimum weight = 21
 processing server = MVS03A weight = 42 host = MVS03A
db_update(TNRAL.ralplex1.buddha.ral.ibm.com, 0x14dceaa0, 0x14dceaa0, 01, 0x14dcd8b8)
match(0x14dce100, 1, 6) 1, 6
db_update: adding 14dceaa0
db_update(MVS03A.TNRAL.ralplex1.buddha.ral.ibm.com, 0x14dceb10, 0x14dceb10, 01, 0x14dcd8b8)
savehash GROWING to 2
match(0x14dceaa0, 1, 6) 1, 1
match(0x14dceaa0, 1, 2) 1, 1
match(0x14dce100, 1, 6) 1, 6
db_update: adding 14dceb10
 processing server = MVS28A weight = 21 host = MVS28A
db_update(TNRAL.ralplex1.buddha.ral.ibm.com, 0x14dceb98, 0x14dceb98, 01, 0x14dcd8b8)
match(0x14dceaa0, 1, 6) 1, 1
match(0x14dceaa0, 1, 2) 1, 1
match(0x14dce100, 1, 6) 1, 6
match(0x14dceaa0, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for TNRAL.ralplex1.buddha.ral.ibm.com is 4(5) from  0.0.0.0 .0, is 4(5) in cache
db_update: adding 14dceb98
db_update(MVS28A.TNRAL.ralplex1.buddha.ral.ibm.com, 0x14dcebd0, 0x14dcebd0, 01, 0x14dcd8b8)
match(0x14dceaa0, 1, 6) 1, 1
match(0x14dceb98, 1, 6) 1, 1
match(0x14dceaa0, 1, 2) 1, 1
match(0x14dceb98, 1, 2) 1, 1
match(0x14dce100, 1, 6) 1, 6
db_update: adding 14dcebd0
 processing group = ralplex1.buddha.ral.ibm.com count = 2
 minimum weight = 21
 processing server = TO3ATCP weight = 42 host = MVS03A
 adding MVS03A to list
db_update(ralplex1.buddha.ral.ibm.com, 0x14dcec40, 0x14dcec40, 01, 0x14dcd8b8)
match(0x14dce100, 1, 6) 1, 6
match(0x14dce100, 1, 1) 1, 6
match(0x14dce2b8, 1, 1) 1, 2
db_update: adding 14dcec40
 processing server = T28ATCP weight = 21 host = MVS28A
 adding MVS28A to list
db_update(ralplex1.buddha.ral.ibm.com, 0x14dcec78, 0x14dcec78, 01, 0x14dcd8b8)
match(0x14dce100, 1, 6) 1, 6
match(0x14dce100, 1, 1) 1, 6
match(0x14dce2b8, 1, 1) 1, 2
match(0x14dcec40, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for ralplex1.buddha.ral.ibm.com is 4(5) from  0.0.0.0 .0, is 4(5) in cache
db_update: adding 14dcec78
db_update(ralplex1.ralplex1.buddha.ral.ibm.com, 0x14dce4d0, 0x14dce4d0, 01, 0x14dcd8b8)
match(0x14dce440, 1, 6) 1, 5
match(0x14dce440, 1, 2) 1, 5
match(0x14dce100, 1, 6) 1, 6
match(0x14dce440, 1, 5) 1, 5
db_update: flags = 0x1, sizes = 28, 28 (cmp 0)
credibility for ralplex1.ralplex1.buddha.ral.ibm.com is 4(5) from  0.0.0.0 .0, is 4(5) in cache
fclose(7) succeeded
zone 2  type 2: 'ralplex1.buddha.ral.ibm.com' z_time 896822324, z_refresh 7200
zone origin 168.192.in-addr.arpa, new zone
```

```
 addrs: 192.168.250.3, addrcnt = 1
purge_zone(168.192.in-addr.arpa,1)
db_load(rback, 168.192.in-addr.arpa, 3, Nil)
db_load: origin 192.in-addr.arpa., buf 168.192.in-addr.arpa
db_load: origin now 192.in-addr.arpa
d='168.192.in-addr.arpa', c=1, t=6, ttl=0, data='mvs03a.buddha.ral.ibm.com.'
db_update(168.192.in-addr.arpa, 0x14dce560, 0x14dce560, 01, 0x14dcd8b8)
savehash GROWING to 2
savehash GROWING to 2
savehash GROWING to 2
db_update: adding 14dce560
d='168.192.in-addr.arpa', c=1, t=2, ttl=0, data='mvs03a.buddha.ral.ibm.com.'
db_update(168.192.in-addr.arpa, 0x14dce710, 0x14dce710, 01, 0x14dcd8b8)
match(0x14dce560, 1, 6) 1, 6
match(0x14dce560, 1, 2) 1, 6
db_update: adding 14dce710
db_load: origin 202.168.192.in-addr.arpa., buf 192.in-addr.arpa
db_load: origin now 202.168.192.in-addr.arpa
d='28.202.168.192.in-addr.arpa', c=1, t=12, ttl=0, data='mvs28a.buddha.ral.ibm.com.'
db_update(28.202.168.192.in-addr.arpa, 0x14dce760, 0x14dce760, 01, 0x14dcd8b8)
savehash GROWING to 2
savehash GROWING to 2
match(0x14dce560, 1, 6) 1, 6
db_update: adding 14dce760
d='4.202.168.192.in-addr.arpa', c=1, t=12, ttl=0, data='mvs03c.buddha.ral.ibm.com.'
db_update(4.202.168.192.in-addr.arpa, 0x14dce850, 0x14dce850, 01, 0x14dcd8b8)
match(0x14dce560, 1, 6) 1, 6
db_update: adding 14dce850
d='3.202.168.192.in-addr.arpa', c=1, t=12, ttl=0, data='mvs03a.buddha.ral.ibm.com.'
db_update(3.202.168.192.in-addr.arpa, 0x14dcecb0, 0x14dcecb0, 01, 0x14dcd8b8)
match(0x14dce560, 1, 6) 1, 6
db_update: adding 14dcecb0
db_load: origin 221.168.192.in-addr.arpa., buf 202.168.192.in-addr.arpa
db_load: origin now 221.168.192.in-addr.arpa
d='28.221.168.192.in-addr.arpa', c=1, t=12, ttl=0, data='mvs28a.buddha.ral.ibm.com.'
db_update(28.221.168.192.in-addr.arpa, 0x14dced38, 0x14dced38, 01, 0x14dcd8b8)
savehash GROWING to 2
match(0x14dce560, 1, 6) 1, 6
db_update: adding 14dced38
d='4.221.168.192.in-addr.arpa', c=1, t=12, ttl=0, data='mvs03c.buddha.ral.ibm.com.'
db_update(4.221.168.192.in-addr.arpa, 0x14dcee10, 0x14dcee10, 01, 0x14dcd8b8)
match(0x14dce560, 1, 6) 1, 6
db_update: adding 14dcee10
d='3.221.168.192.in-addr.arpa', c=1, t=12, ttl=0, data='mvs03a.buddha.ral.ibm.com.'
db_update(3.221.168.192.in-addr.arpa, 0x14dcee98, 0x14dcee98, 01, 0x14dcd8b8)
match(0x14dce560, 1, 6) 1, 6
db_update: adding 14dcee98
db_load: origin 250.168.192.in-addr.arpa., buf 221.168.192.in-addr.arpa
db_load: origin now 250.168.192.in-addr.arpa
d='3.250.168.192.in-addr.arpa', c=1, t=12, ttl=0, data='mvs03a.buddha.ral.ibm.com.'
db_update(3.250.168.192.in-addr.arpa, 0x14dcef20, 0x14dcef20, 01, 0x14dcd8b8)
savehash GROWING to 2
match(0x14dce560, 1, 6) 1, 6
db_update: adding 14dcef20
db_load: origin 251.168.192.in-addr.arpa., buf 250.168.192.in-addr.arpa
db_load: origin now 251.168.192.in-addr.arpa
d='4.251.168.192.in-addr.arpa', c=1, t=12, ttl=0, data='mvs03c.buddha.ral.ibm.com.'
db_update(4.251.168.192.in-addr.arpa, 0x14dceff8, 0x14dceff8, 01, 0x14dcd8b8)
savehash GROWING to 2
match(0x14dce560, 1, 6) 1, 6
db_update: adding 14dceff8
db_load: origin 249.168.192.in-addr.arpa., buf 251.168.192.in-addr.arpa
db_load: origin now 249.168.192.in-addr.arpa
d='4.249.168.192.in-addr.arpa', c=1, t=12, ttl=0, data='mvs03c.buddha.ral.ibm.com.'
db_update(4.249.168.192.in-addr.arpa, 0x14dcf0d0, 0x14dcf0d0, 01, 0x14dcd8b8)
savehash GROWING to 11
```

```
savehash(0x14dce7b0) cnt=5, sz=2, newsz=11
savehash GROWING to 2
match(0x14dce560, 1, 6) 1, 6
db_update: adding 14dcf0d0
d='3.249.168.192.in-addr.arpa', c=1, t=12, ttl=0, data='mvs03a.buddha.ral.ibm.com.'
db_update(3.249.168.192.in-addr.arpa, 0x14dcf1d0, 0x14dcf1d0, 01, 0x14dcd8b8)
match(0x14dce560, 1, 6) 1, 6
db_update: adding 14dcf1d0
db_load: origin 236.168.192.in-addr.arpa., buf 249.168.192.in-addr.arpa
db_load: origin now 236.168.192.in-addr.arpa
d='28.236.168.192.in-addr.arpa', c=1, t=12, ttl=0, data='mvs28a.buddha.ral.ibm.com.'
db_update(28.236.168.192.in-addr.arpa, 0x14dcf258, 0x14dcf258, 01, 0x14dcd8b8)
savehash GROWING to 2
match(0x14dce560, 1, 6) 1, 6
db_update: adding 14dcf258
d='3.236.168.192.in-addr.arpa', c=1, t=12, ttl=0, data='mvs03a.buddha.ral.ibm.com.'
db_update(3.236.168.192.in-addr.arpa, 0x14dcf330, 0x14dcf330, 01, 0x14dcd8b8)
match(0x14dce560, 1, 6) 1, 6
db_update: adding 14dcf330
db_load: origin 252.168.192.in-addr.arpa., buf 236.168.192.in-addr.arpa
db_load: origin now 252.168.192.in-addr.arpa
d='28.252.168.192.in-addr.arpa', c=1, t=12, ttl=0, data='mvs28a.buddha.ral.ibm.com.'
db_update(28.252.168.192.in-addr.arpa, 0x14dcf3b8, 0x14dcf3b8, 01, 0x14dcd8b8)
savehash GROWING to 2
match(0x14dce560, 1, 6) 1, 6
db_update: adding 14dcf3b8
fclose(7) succeeded
zone 3  type 2: '168.192.in-addr.arpa' z_time 896888713, z_refresh 7200
zone origin 0.0.127.in-addr.arpa, new zone, source = mvs28a.lbk
purge_zone(0.0.127.in-addr.arpa,1)
reloading zone
db_load(mvs28a.lbk, 0.0.127.in-addr.arpa, 4, Nil)
d='0.0.127.in-addr.arpa', c=1, t=6, ttl=0, data='mvs28a.buddha.ral.ib.com'
db_update(0.0.127.in-addr.arpa, 0x14dcf4d8, 0x14dcf4d8, 01, 0x14dcd8b8)
savehash GROWING to 2
savehash GROWING to 2
db_update: adding 14dcf4d8
d='0.0.127.in-addr.arpa', c=1, t=2, ttl=0, data='mvs28a.buddha.ral.ibm.com.'
db_update(0.0.127.in-addr.arpa, 0x14dcf648, 0x14dcf648, 01, 0x14dcd8b8)
match(0x14dcf4d8, 1, 6) 1, 6
match(0x14dcf4d8, 1, 2) 1, 6
db_update: adding 14dcf648
d='1.0.0.127.in-addr.arpa', c=1, t=12, ttl=0, data='loopback.'
db_update(1.0.0.127.in-addr.arpa, 0x14dcf698, 0x14dcf698, 01, 0x14dcd8b8)
savehash GROWING to 2
match(0x14dcf4d8, 1, 6) 1, 6
db_update: adding 14dcf698
fclose(7) succeeded
zone 4  type 1: '0.0.127.in-addr.arpa' z_time 896890560, z_refresh 7200
fclose(6) succeeded

 content of zones after loading
printzoneinfo(1):
origin ='buddha.ral.ibm.com', class = 1, type = Static secondary, source = fback
z_refresh = 7200, retry = 3600, expire = 604800, minimum = 3600, serial = 1998051904
z_time = 896884162, now time : 896886430 sec, time left: 4294965028 sec; flags 41
printzoneinfo(2):
origin ='ralplex1.buddha.ral.ibm.com', class = 1, type = Static secondary, source = xback
z_refresh = 7200, retry = 3600, expire = 604800, minimum = 3600, serial = 1998051901
z_time = 896822324, now time : 896886430 sec, time left: 4294903190 sec; flags 20041
printzoneinfo(3):
origin ='168.192.in-addr.arpa', class = 1, type = Static secondary, source = rback
z_refresh = 7200, retry = 3600, expire = 604800, minimum = 3600, serial = 1998051904
z_time = 896888713, now time : 896886430 sec, time left: 2283 sec; flags 41
printzoneinfo(4):
origin ='0.0.127.in-addr.arpa', class = 1, type = Static primary, source = mvs28a.lbk
```

```
z_refresh = 7200, retry = 3600, expire = 604800, minimum = 3600, serial = 1998051901
z_time = 896890560, now time : 896886430 sec, time left: 4130 sec; flags 41
exit ns_init(), need maintenance immediately

bootfile = /etc/named.boot
Network and sort list:
addr xc0a8fc00 mask xffffff00 my_addr xc0a8fc1c 192.168.252.28
addr x0 mask x0 my_addr xc0a8d41c 192.168.212.28
addr xc0a8ec03 mask xffffffff my_addr xc0a8ec1c 192.168.236.28
addr x9186800 mask xffffff00 my_addr x918689e 9.24.104.158
addr xc0a8dd00 mask xffffff00 my_addr xc0a8dd1c 192.168.221.28
addr xc0a8e780 mask xffffffff my_addr xc0a8e71c 192.168.231.28
addr xc0a8f180 mask xffffffff my_addr xc0a8f11c 192.168.241.28
addr xc0a86d02 mask xffffffff my_addr xc0a86d1c 192.168.109.28
addr xc0a86400 mask xffffff00 my_addr xc0a86463 192.168.100.99
addr xc0a8ca01 mask xffffffff my_addr xc0a8ca1c 192.168.202.28
addr xc0a8d400 mask xffffff00 my_addr xc0a8d41c 192.168.212.28
addr xc0a8ec00 mask xffffff00 my_addr xc0a8ec1c 192.168.236.28
addr x9000000 mask xff000000 my_addr x918689e 9.24.104.158
addr xc0a8e700 mask xffffff00 my_addr xc0a8e71c 192.168.231.28
addr xc0a8f100 mask xffffff00 my_addr xc0a8f11c 192.168.241.28
addr xc0a86d00 mask xffffff00 my_addr xc0a86d1c 192.168.109.28
addr xc0a8ca00 mask xffffff00 my_addr xc0a8ca1c 192.168.202.28
database initialized
fclose(6) succeeded
prime_cache: priming = 0

ns_maint(); now Wed Jun  3 15:07:11 1998
printzoneinfo(0):
origin ='.',  class = 0, type = Static cachez_refresh = 0, retry = 0, expire = 0, minimum = 0, serial = 0
z_time = 0; flags 0
printzoneinfo(1):
origin ='buddha.ral.ibm.com',  class = 1, type = Static secondary, source = fback
z_refresh = 7200, retry = 3600, expire = 604800, minimum = 3600, serial = 1998051904
z_time = 896884162, now time : 896886431 sec, time left: 4294965027 sec; flags 41
qserial_query(buddha.ral.ibm.com)
sysquery(buddha.ral.ibm.com, 1, 6, 0x14dcb668, 1)
qnew(x14dcf738)
retrytime: nstime-1ms t4 nretry0 u4 : v4
schedretry(0x14dcf738, 4 sec)
sysquery: send ->  192.168.250.3 .53 dfd=5 nsid=53508 id=0 retry=896886435
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53508
;; flags:; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; buddha.ral.ibm.com, type = SOA, class = IN


;; ...truncated
qserial_query(buddha.ral.ibm.com) QUEUED
printzoneinfo(2):
origin ='ralplex1.buddha.ral.ibm.com',  class = 1, type = Static secondary, source = xback
z_refresh = 7200, retry = 3600, expire = 604800, minimum = 3600, serial = 1998051901
z_time = 896822324, now time : 896886432 sec, time left: 4294903188 sec; flags 20041
printzoneinfo(3):
origin ='168.192.in-addr.arpa',  class = 1, type = Static secondary, source = rback
z_refresh = 7200, retry = 3600, expire = 604800, minimum = 3600, serial = 1998051904
z_time = 896888713, now time : 896886432 sec, time left: 2281 sec; flags 41
printzoneinfo(4):
origin ='0.0.127.in-addr.arpa',  class = 1, type = Static primary, source = mvs28a.lbk
z_refresh = 7200, retry = 3600, expire = 604800, minimum = 3600, serial = 1998051901
z_time = 896890560, now time : 896886432 sec, time left: 4128 sec; flags 41
next_refresh: 896886492, next_alarm: 0, tt.tv_sec: 896886432
sched_maint: Next interrupt in 60 sec
exit ns_maint()
```

```
datagram from  192.168.250.3 .53, fd 5, len 171; now Wed Jun  3 15:07:12 1998
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53508
;; flags: qr aa ra; Ques: 1, Ans: 1, Auth: 1, Addit: 4
;; QUESTIONS:
;; buddha.ral.ibm.com, type = SOA, class = IN

;; ANSWERS:
buddha.ral.ibm.com. 3600 IN SOA mvs03a.buddha.ral.ibm.com. garthm@mvs03a.buddha.ral.ibm.com. (
   1998051904 ; serial
   7200 ; refresh (2 hours)
   3600 ; retry (1 hour)
   604800 ; expire (7 days)
   3600 ) ; minimum (1 hour)

;; AUTHORITY RECORDS:
buddha.ral.ibm.com. 3600 IN NS mvs03a.buddha.ral.ibm.com.

;; ADDITIONAL RECORDS:
mvs03a.buddha.ral.ibm.com. 3600 IN A 192.168.249.3
mvs03a.buddha.ral.ibm.com. 3600 IN A 192.168.202.3
mvs03a.buddha.ral.ibm.com. 3600 IN A 192.168.236.3
mvs03a.buddha.ral.ibm.com. 3600 IN A 192.168.221.3

ns_req(from= 192.168.250.3 .53)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53508
;; flags: qr aa ra; Ques: 1, Ans: 1, Auth: 1, Addit: 4
;; QUESTIONS:
;; buddha.ral.ibm.com, type = SOA, class = IN

;; ANSWERS:
buddha.ral.ibm.com. 3600 IN SOA mvs03a.buddha.ral.ibm.com. garthm@mvs03a.buddha.ral.ibm.com. (
   1998051904 ; serial
   7200 ; refresh (2 hours)
   3600 ; retry (1 hour)
   604800 ; expire (7 days)
   3600 ) ; minimum (1 hour)

;; AUTHORITY RECORDS:
buddha.ral.ibm.com. 3600 IN NS mvs03a.buddha.ral.ibm.com.

;; ADDITIONAL RECORDS:
mvs03a.buddha.ral.ibm.com. 3600 IN A 192.168.249.3
mvs03a.buddha.ral.ibm.com. 3600 IN A 192.168.202.3
mvs03a.buddha.ral.ibm.com. 3600 IN A 192.168.236.3
mvs03a.buddha.ral.ibm.com. 3600 IN A 192.168.221.3

qfindid(53508)
Response (SYSTEM NORMAL ZSERIAL) nsid=53508 id=0
stime 896886431/668518  now 896886432/348693 rtt 681
resp: ancount 1, aucount 1, arcount 4
qserial_answer(buddha.ral.ibm.com, 1998051904)
qserial_answer: zone serial is still OK
qremove(x14dcf738)
unsched(0x14dcf738, 0)
Qfree(x14dcf738)
loadxfer() "ralplex1.buddha.ral.ibm.com"
rm_datum(14dce100, 14dce100, 0) -> 14dce2b8
rm_datum(14dce2b8, 14dce2b8, 0) -> 14dcec40
rm_datum(14dcec40, 14dcec40, 0) -> 14dcec78
rm_datum(14dcec78, 14dcec78, 0) -> 0
rm_datum(14dce8d8, 14dce8d8, 0) -> 14dce9f8
rm_datum(14dce9f8, 14dce9f8, 0) -> 0
rm_datum(14dcea30, 14dcea30, 0) -> 0
rm_datum(14dce988, 14dce988, 0) -> 0
rm_datum(14dce1d8, 14dce1d8, 0) -> 0
rm_datum(14dce360, 14dce360, 0) -> 0
```

```
rm_datum(14dce3d0, 14dce3d0, 0) -> 0
rm_datum(14dceaa0, 14dceaa0, 0) -> 14dceb98
rm_datum(14dceb98, 14dceb98, 0) -> 0
rm_datum(14dcebd0, 14dcebd0, 0) -> 0
rm_datum(14dceb10, 14dceb10, 0) -> 0
rm_datum(14dce440, 14dce440, 0) -> 0
purge_zone(ralplex1.buddha.ral.ibm.com,1)
db_load(xback, ralplex1.buddha.ral.ibm.com, 2, Nil)
db_load: origin buddha.ral.ibm.com., buf ralplex1.buddha.ral.ibm.com
db_load: origin now buddha.ral.ibm.com
d='ralplex1.buddha.ral.ibm.com', c=1, t=6, ttl=0, data='mvs03a.ralplex1.buddha.ral.ibm.com.'
db_update(ralplex1.buddha.ral.ibm.com, 0x14e5a4f0, 0x14e5a4f0, 01, 0x14dcd8b8)
match(0x14dcd938, 1, 6) 1, 6
db_update: adding 14e5a4f0
d='ralplex1.buddha.ral.ibm.com', c=1, t=2, ttl=0, data='mvs03a.ralplex1.buddha.ral.ibm.com.'
db_update(ralplex1.buddha.ral.ibm.com, 0x14e5a5c8, 0x14e5a5c8, 01, 0x14dcd8b8)
match(0x14e5a4f0, 1, 6) 1, 6
match(0x14e5a4f0, 1, 2) 1, 6
db_update: adding 14e5a5c8
db_load: origin ralplex1.buddha.ral.ibm.com., buf buddha.ral.ibm.com
db_load: origin now ralplex1.buddha.ral.ibm.com
d='mvs03a.ralplex1.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.250.3'
db_update(mvs03a.ralplex1.buddha.ral.ibm.com, 0x14e5a620, 0x14e5a620, 01, 0x14dcd8b8)
savehash GROWING to 2
match(0x14e5a4f0, 1, 6) 1, 6
db_update: adding 14e5a620
d='mvs03c.ralplex1.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.251.4'
db_update(mvs03c.ralplex1.buddha.ral.ibm.com, 0x14e5a6a8, 0x14e5a6a8, 01, 0x14dcd8b8)
match(0x14e5a4f0, 1, 6) 1, 6
db_update: adding 14e5a6a8
d='mvs28a.ralplex1.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.252.28'
db_update(mvs28a.ralplex1.buddha.ral.ibm.com, 0x14e5a718, 0x14e5a718, 01, 0x14dcd8b8)
match(0x14e5a4f0, 1, 6) 1, 6
db_update: adding 14e5a718
d='ralplex1.ralplex1.buddha.ral.ibm.com', c=1, t=5, ttl=0, data='ralplex1.buddha.ral.ibm.com.'
db_update(ralplex1.ralplex1.buddha.ral.ibm.com, 0x14e5a788, 0x14e5a788, 01, 0x14dcd8b8)
match(0x14e5a4f0, 1, 6) 1, 6
db_update: adding 14e5a788
wlm_load ralplex1.buddha.ral.ibm.com
 group list retcode = -1 rsncode = 1034
 group list retcode = 0 rsncode = 0
 group list entry_count = 3
 querying group =  FTPRAL
 server list retcode = -1 rsncode = 1034
 server list retcode = 0 rsncode = 0
 querying group =  TNRAL
 server list retcode = -1 rsncode = 1034
 server list retcode = 0 rsncode = 0
 querying group =  TCPIP
 server list retcode = -1 rsncode = 1034
 server list retcode = 0 rsncode = 0
 processing group = FTPRAL.ralplex1.buddha.ral.ibm.com count = 2
 minimum weight = 21
 processing server = MVS03A weight = 42 host = MVS03A
db_update(FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14e5ac20, 0x14e5ac20, 01, 0x14dcd8b8)
savehash GROWING to 11
savehash(0x14e5a658) cnt=5, sz=2, newsz=11
match(0x14e5a4f0, 1, 6) 1, 6
db_update: adding 14e5ac20
db_update(MVS03A.FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14e5acd0, 0x14e5acd0, 01, 0x14dcd8b8)
savehash GROWING to 2
match(0x14e5ac20, 1, 6) 1, 1
match(0x14e5ac20, 1, 2) 1, 1
match(0x14e5a4f0, 1, 6) 1, 6
db_update: adding 14e5acd0
```

```
 processing server = MVS28A weight = 21 host = MVS28A
db_update(FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14e5ad40, 0x14e5ad40, 01, 0x14dcd8b8)
match(0x14e5ac20, 1, 6) 1, 1
match(0x14e5ac20, 1, 2) 1, 1
match(0x14e5a4f0, 1, 6) 1, 6
match(0x14e5ac20, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for FTPRAL.ralplex1.buddha.ral.ibm.com is 4(5) from  192.168.250.3 .53, is 4(5) in cache
db_update: adding 14e5ad40
db_update(MVS28A.FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14e5ad78, 0x14e5ad78, 01, 0x14dcd8b8)
match(0x14e5ac20, 1, 6) 1, 1
match(0x14e5ad40, 1, 6) 1, 1
match(0x14e5ac20, 1, 2) 1, 1
match(0x14e5ad40, 1, 2) 1, 1
match(0x14e5a4f0, 1, 6) 1, 6
db_update: adding 14e5ad78
 processing group = TNRAL.ralplex1.buddha.ral.ibm.com count = 2
 minimum weight = 21
 processing server = MVSO3A weight = 42 host = MVSO3A
db_update(TNRAL.ralplex1.buddha.ral.ibm.com, 0x14e5ade8, 0x14e5ade8, 01, 0x14dcd8b8)
match(0x14e5a4f0, 1, 6) 1, 6
db_update: adding 14e5ade8
db_update(MVSO3A.TNRAL.ralplex1.buddha.ral.ibm.com, 0x14e5ae58, 0x14e5ae58, 01, 0x14dcd8b8)
savehash GROWING to 2
match(0x14e5ade8, 1, 6) 1, 1
match(0x14e5ade8, 1, 2) 1, 1
match(0x14e5a4f0, 1, 6) 1, 6
db_update: adding 14e5ae58
 processing server = MVS28A weight = 21 host = MVS28A
db_update(TNRAL.ralplex1.buddha.ral.ibm.com, 0x14e5aee0, 0x14e5aee0, 01, 0x14dcd8b8)
match(0x14e5ade8, 1, 6) 1, 1
match(0x14e5ade8, 1, 2) 1, 1
match(0x14e5a4f0, 1, 6) 1, 6
match(0x14e5ade8, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for TNRAL.ralplex1.buddha.ral.ibm.com is 4(5) from  192.168.250.3 .53, is 4(5) in cache
db_update: adding 14e5aee0
db_update(MVS28A.TNRAL.ralplex1.buddha.ral.ibm.com, 0x14e5af18, 0x14e5af18, 01, 0x14dcd8b8)
match(0x14e5ade8, 1, 6) 1, 1
match(0x14e5aee0, 1, 6) 1, 1
match(0x14e5ade8, 1, 2) 1, 1
match(0x14e5aee0, 1, 2) 1, 1
match(0x14e5a4f0, 1, 6) 1, 6
db_update: adding 14e5af18
 processing group = ralplex1.buddha.ral.ibm.com count = 2
 minimum weight = 21
 processing server = TO3ATCP weight = 42 host = MVSO3A
db_update(ralplex1.buddha.ral.ibm.com, 0x14e5af88, 0x14e5af88, 01, 0x14dcd8b8)
match(0x14e5a4f0, 1, 6) 1, 6
match(0x14e5a4f0, 1, 1) 1, 6
match(0x14e5a5c8, 1, 1) 1, 2
db_update: adding 14e5af88
 processing server = T28ATCP weight = 21 host = MVS28A
db_update(ralplex1.buddha.ral.ibm.com, 0x14e5afc0, 0x14e5afc0, 01, 0x14dcd8b8)
match(0x14e5a4f0, 1, 6) 1, 6
match(0x14e5a4f0, 1, 1) 1, 6
match(0x14e5a5c8, 1, 1) 1, 2
match(0x14e5af88, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for ralplex1.buddha.ral.ibm.com is 4(5) from  192.168.250.3 .53, is 4(5) in cache
db_update: adding 14e5afc0
db_update(ralplex1.ralplex1.buddha.ral.ibm.com, 0x14e5a818, 0x14e5a818, 01, 0x14dcd8b8)
match(0x14e5a788, 1, 6) 1, 5
match(0x14e5a788, 1, 2) 1, 5
match(0x14e5a4f0, 1, 6) 1, 6
match(0x14e5a788, 1, 5) 1, 5
```

```
db_update: flags = 0x1, sizes = 28, 28 (cmp 0)
credibility for ralplex1.ralplex1.buddha.ral.ibm.com is 4(5) from  192.168.250.3 .53, is 4(5) in cache
fclose(7) succeeded
next_refresh: 896822324, next_alarm: 896886492, tt.tv_sec: 896886432
sched_maint: Next interrupt in 60 sec

ns_maint(); now Wed Jun  3 15:08:12 1998
printzoneinfo(0):
origin ='.',  class = 0, type = Static cachez_refresh = 0, retry = 0, expire = 0, minimum = 0, serial = 0
z_time = 0; flags 0
printzoneinfo(1):
origin ='buddha.ral.ibm.com',  class = 1, type = Static secondary, source = fback
z_refresh = 7200, retry = 3600, expire = 604800, minimum = 3600, serial = 1998051904
z_time = 896890066, now time : 896886492 sec, time left: 3574 sec; flags 41
printzoneinfo(2):
origin ='ralplex1.buddha.ral.ibm.com',  class = 1, type = Static secondary, source = xback
z_refresh = 7200, retry = 3600, expire = 604800, minimum = 3600, serial = 1998051901
z_time = 896822324, now time : 896886492 sec, time left: 4294903128 sec; flags 20041
printzoneinfo(3):
origin ='168.192.in-addr.arpa',  class = 1, type = Static secondary, source = rback
z_refresh = 7200, retry = 3600, expire = 604800, minimum = 3600, serial = 1998051904
z_time = 896888713, now time : 896886492 sec, time left: 2221 sec; flags 41
printzoneinfo(4):
origin ='0.0.127.in-addr.arpa',  class = 1, type = Static primary, source = mvs28a.lbk
z_refresh = 7200, retry = 3600, expire = 604800, minimum = 3600, serial = 1998051901
z_time = 896890560, now time : 896886492 sec, time left: 4068 sec; flags 41
next_refresh: 896886552, next_alarm: 896886492, tt.tv_sec: 896886492
sched_maint: Next interrupt in 60 sec
exit ns_maint()
loadxfer() "ralplex1.buddha.ral.ibm.com"
rm_datum(14e5a4f0, 14e5a4f0, 0) -> 14e5a5c8
rm_datum(14e5a5c8, 14e5a5c8, 0) -> 14e5af88
rm_datum(14e5af88, 14e5af88, 0) -> 14e5afc0
rm_datum(14e5afc0, 14e5afc0, 0) -> 0
rm_datum(14e5ac20, 14e5ac20, 0) -> 14e5ad40
rm_datum(14e5ad40, 14e5ad40, 0) -> 0
rm_datum(14e5ad78, 14e5ad78, 0) -> 0
rm_datum(14e5acd0, 14e5acd0, 0) -> 0
rm_datum(14e5a620, 14e5a620, 0) -> 0
rm_datum(14e5a6a8, 14e5a6a8, 0) -> 0
rm_datum(14e5a718, 14e5a718, 0) -> 0
rm_datum(14e5ade8, 14e5ade8, 0) -> 14e5aee0
rm_datum(14e5aee0, 14e5aee0, 0) -> 0
rm_datum(14e5af18, 14e5af18, 0) -> 0
rm_datum(14e5ae58, 14e5ae58, 0) -> 0
rm_datum(14e5a788, 14e5a788, 0) -> 0
purge_zone(ralplex1.buddha.ral.ibm.com,1)
db_load(xback, ralplex1.buddha.ral.ibm.com, 2, Nil)
db_load: origin buddha.ral.ibm.com., buf ralplex1.buddha.ral.ibm.com
db_load: origin now buddha.ral.ibm.com
d='ralplex1.buddha.ral.ibm.com',  c=1, t=6, ttl=0, data='mvs03a.ralplex1.buddha.ral.ibm.com.'
db_update(ralplex1.buddha.ral.ibm.com, 0x14e5a500, 0x14e5a500, 01, 0x14dcd8b8)
match(0x14dcd938, 1, 6) 1, 6
db_update: adding 14e5a500
d='ralplex1.buddha.ral.ibm.com',  c=1, t=2, ttl=0, data='mvs03a.ralplex1.buddha.ral.ibm.com.'
db_update(ralplex1.buddha.ral.ibm.com, 0x14e5a5d8, 0x14e5a5d8, 01, 0x14dcd8b8)
match(0x14e5a500, 1, 6) 1, 6
match(0x14e5a500, 1, 2) 1, 6
db_update: adding 14e5a5d8
db_load: origin ralplex1.buddha.ral.ibm.com., buf buddha.ral.ibm.com
db_load: origin now ralplex1.buddha.ral.ibm.com
d='mvs03a.ralplex1.buddha.ral.ibm.com',  c=1, t=1, ttl=0, data='192.168.250.3'
db_update(mvs03a.ralplex1.buddha.ral.ibm.com, 0x14e5a630, 0x14e5a630, 01, 0x14dcd8b8)
savehash GROWING to 2
match(0x14e5a500, 1, 6) 1, 6
db_update: adding 14e5a630
```

```
d='mvs03c.ralplex1.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.251.4'
db_update(mvs03c.ralplex1.buddha.ral.ibm.com, 0x14e5a6b8, 0x14e5a6b8, 01, 0x14dcd8b8)
match(0x14e5a500, 1, 6) 1, 6
db_update: adding 14e5a6b8
d='mvs28a.ralplex1.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.252.28'
db_update(mvs28a.ralplex1.buddha.ral.ibm.com, 0x14e5a728, 0x14e5a728, 01, 0x14dcd8b8)
match(0x14e5a500, 1, 6) 1, 6
db_update: adding 14e5a728
d='ralplex1.ralplex1.buddha.ral.ibm.com', c=1, t=5, ttl=0, data='ralplex1.buddha.ral.ibm.com.'
db_update(ralplex1.ralplex1.buddha.ral.ibm.com, 0x14e5a798, 0x14e5a798, 01, 0x14dcd8b8)
match(0x14e5a500, 1, 6) 1, 6
db_update: adding 14e5a798
wlm_load ralplex1.buddha.ral.ibm.com
 group list retcode = -1 rsncode = 1034
 group list retcode = 0 rsncode = 0
 group list entry_count = 3
 querying group =  FTPRAL
 server list retcode = -1 rsncode = 1034
 server list retcode = 0 rsncode = 0
 querying group =  TNRAL
 server list retcode = -1 rsncode = 1034
 server list retcode = 0 rsncode = 0
 querying group =  TCPIP
 server list retcode = -1 rsncode = 1034
 server list retcode = 0 rsncode = 0
 processing group = FTPRAL.ralplex1.buddha.ral.ibm.com count = 2
 minimum weight = 21
 processing server = MVS03A weight = 42 host = MVS03A
db_update(FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14e5ac88, 0x14e5ac88, 01, 0x14dcd8b8)
savehash GROWING to 11
savehash(0x14e5a668) cnt=5, sz=2, newsz=11
match(0x14e5a500, 1, 6) 1, 6
db_update: adding 14e5ac88
db_update(MVS03A.FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14e5ad38, 0x14e5ad38, 01, 0x14dcd8b8)
savehash GROWING to 2
match(0x14e5ac88, 1, 6) 1, 1
match(0x14e5ac88, 1, 2) 1, 1
match(0x14e5a500, 1, 6) 1, 6
db_update: adding 14e5ad38
 processing server = MVS28A weight = 21 host = MVS28A
db_update(FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14e5ada8, 0x14e5ada8, 01, 0x14dcd8b8)
match(0x14e5ac88, 1, 6) 1, 1
match(0x14e5ac88, 1, 2) 1, 1
match(0x14e5a500, 1, 6) 1, 6
match(0x14e5ac88, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for FTPRAL.ralplex1.buddha.ral.ibm.com is 4(5) from  192.168.250.3 .53, is 4(5) in cache
db_update: adding 14e5ada8
db_update(MVS28A.FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14e5ade0, 0x14e5ade0, 01, 0x14dcd8b8)
match(0x14e5ac88, 1, 6) 1, 1
match(0x14e5ada8, 1, 6) 1, 1
match(0x14e5ac88, 1, 2) 1, 1
match(0x14e5ada8, 1, 2) 1, 1
match(0x14e5a500, 1, 6) 1, 6
db_update: adding 14e5ade0
 processing group = TNRAL.ralplex1.buddha.ral.ibm.com count = 2
 minimum weight = 21
 processing server = MVS03A weight = 42 host = MVS03A
db_update(TNRAL.ralplex1.buddha.ral.ibm.com, 0x14e5ae50, 0x14e5ae50, 01, 0x14dcd8b8)
match(0x14e5a500, 1, 6) 1, 6
db_update: adding 14e5ae50
db_update(MVS03A.TNRAL.ralplex1.buddha.ral.ibm.com, 0x14e5aec0, 0x14e5aec0, 01, 0x14dcd8b8)
savehash GROWING to 2
match(0x14e5ae50, 1, 6) 1, 1
match(0x14e5ae50, 1, 2) 1, 1
```

```
match(0x14e5a500, 1, 6) 1, 6
db_update: adding 14e5aec0
 processing server = MVS28A weight = 21 host = MVS28A
db_update(TNRAL.ralplex1.buddha.ral.ibm.com, 0x14e5af48, 0x14e5af48, 01, 0x14dcd8b8)
match(0x14e5ae50, 1, 6) 1, 1
match(0x14e5ae50, 1, 2) 1, 1
match(0x14e5a500, 1, 6) 1, 6
match(0x14e5ae50, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for TNRAL.ralplex1.buddha.ral.ibm.com is 4(5) from  192.168.250.3 .53, is 4(5) in cache
db_update: adding 14e5af48
db_update(MVS28A.TNRAL.ralplex1.buddha.ral.ibm.com, 0x14e5af80, 0x14e5af80, 01, 0x14dcd8b8)
match(0x14e5ae50, 1, 6) 1, 1
match(0x14e5af48, 1, 6) 1, 1
match(0x14e5ae50, 1, 2) 1, 1
match(0x14e5af48, 1, 2) 1, 1
match(0x14e5a500, 1, 6) 1, 6
db_update: adding 14e5af80
 processing group = ralplex1.buddha.ral.ibm.com count = 2
 minimum weight = 21
 processing server = TO3ATCP weight = 42 host = MVS03A
db_update(ralplex1.buddha.ral.ibm.com, 0x14e5aff0, 0x14e5aff0, 01, 0x14dcd8b8)
match(0x14e5a500, 1, 6) 1, 6
match(0x14e5a500, 1, 1) 1, 6
match(0x14e5a5d8, 1, 1) 1, 2
db_update: adding 14e5aff0
 processing server = T28ATCP weight = 21 host = MVS28A
db_update(ralplex1.buddha.ral.ibm.com, 0x14e5b028, 0x14e5b028, 01, 0x14dcd8b8)
match(0x14e5a500, 1, 6) 1, 6
match(0x14e5a500, 1, 1) 1, 6
match(0x14e5a5d8, 1, 1) 1, 2
match(0x14e5aff0, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for ralplex1.buddha.ral.ibm.com is 4(5) from  192.168.250.3 .53, is 4(5) in cache
db_update: adding 14e5b028
db_update(ralplex1.ralplex1.buddha.ral.ibm.com, 0x14e5a880, 0x14e5a880, 01, 0x14dcd8b8)
match(0x14e5a798, 1, 6) 1, 5
match(0x14e5a798, 1, 2) 1, 5
match(0x14e5a500, 1, 6) 1, 6
match(0x14e5a798, 1, 5) 1, 5
db_update: flags = 0x1, sizes = 28, 28 (cmp 0)
credibility for ralplex1.ralplex1.buddha.ral.ibm.com is 4(5) from  192.168.250.3 .53, is 4(5) in cache
fclose(7) succeeded

ns_maint(); now Wed Jun  3 15:09:12 1998
printzoneinfo(0):
origin ='.',  class = 0, type = Static cachez_refresh = 0, retry = 0, expire = 0, minimum = 0, serial = 0
z_time = 0; flags 0
printzoneinfo(1):
origin ='buddha.ral.ibm.com',  class = 1, type = Static secondary, source = fback
z_refresh = 7200, retry = 3600, expire = 604800, minimum = 3600, serial = 1998051904
z_time = 896890066, now time : 896886552 sec, time left: 3514 sec; flags 41
printzoneinfo(2):
origin ='ralplex1.buddha.ral.ibm.com',  class = 1, type = Static secondary, source = xback
z_refresh = 7200, retry = 3600, expire = 604800, minimum = 3600, serial = 1998051901
z_time = 896822324, now time : 896886552 sec, time left: 4294903068 sec; flags 20041
printzoneinfo(3):
origin ='168.192.in-addr.arpa',  class = 1, type = Static secondary, source = rback
z_refresh = 7200, retry = 3600, expire = 604800, minimum = 3600, serial = 1998051904
z_time = 896888713, now time : 896886552 sec, time left: 2161 sec; flags 41
printzoneinfo(4):
origin ='0.0.127.in-addr.arpa',  class = 1, type = Static primary, source = mvs28a.lbk
z_refresh = 7200, retry = 3600, expire = 604800, minimum = 3600, serial = 1998051901
z_time = 896890560, now time : 896886552 sec, time left: 4008 sec; flags 41
next_refresh: 896886612, next_alarm: 896886552, tt.tv_sec: 896886552
```

```
sched_maint: Next interrupt in 60 sec
exit ns_maint()

datagram from  192.168.221.88 .1096, fd 5, len 42; now Wed Jun  3 15:09:33 1998
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
ns_req(from= 192.168.221.88 .1096)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
req: nlookup(tnral.buddha.ral.ibm.com) id 1 type=1 class=1
req: found 'tnral.buddha.ral.ibm.com' as 'tnral.buddha.ral.ibm.com' (cname=0)
******* Remove_Expired_RRs  Entered *******
name: tnral
  type=5 size=34
name: ftpoeral
  type=5 size=37
Remove_Expired_RRs rc=0
wanted(0x14dce060, 1, 1)  IN CNAME
make_rr(tnral.buddha.ral.ibm.com, 14dce060, bf39a, 458, 1) 34 zone 1 ttl 3600
match(0x14dce060, 1, 24) 1, 5
finddata: added 1 class 1 type 1 RRs
req: nlookup(tnral.ralplex1.buddha.ral.ibm.com) id 1 type=1 class=1
req: found 'tnral.ralplex1.buddha.ral.ibm.com' as 'tnral.ralplex1.buddha.ral.ibm.com' (cname=1)
******* Remove_Expired_RRs  Entered *******
name: TNRAL
  type=1 size=4
  type=1 size=4
Remove_Expired_RRs rc=0
 Looking for WLM generated Address
make_rr(tnral.ralplex1.buddha.ral.ibm.com, 14e5ae50, bf3c9, 411, 1) 4 zone 2 ttl 0
match(0x14e5ae50, 1, 24) 1, 1
match(0x14e5af48, 1, 24) 1, 1
 Weight = 1
req: foundname=2, count=1, founddata=1, cname=1
sort_response(1)
findns: np 0x14e5ae88 'TNRAL'
match(0x14e5ae50, 1, 6) 1, 1
match(0x14e5af48, 1, 6) 1, 1
match(0x14e5ae50, 1, 2) 1, 1
match(0x14e5af48, 1, 2) 1, 1
findns: np 0x14e5a598 'ralplex1'
match(0x14e5a500, 1, 6) 1, 6
findns: SOA found
req: leaving (tnral.ralplex1.buddha.ral.ibm.com, rcode 0)
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
findns: np 0x14e5a598 'ralplex1'
match(0x14e5a500, 1, 2) 1, 6
match(0x14e5a5d8, 1, 2) 1, 2
match(0x14e5aff0, 1, 2) 1, 1
match(0x14e5b028, 1, 2) 1, 1
findns: 1 NS's added for 'ralplex1'
make_rr(ralplex1.buddha.ral.ibm.com, 14e5a5d8, bf3d9, 395, 1) 35 zone 2 ttl 3600
match(0x14e5a500, 1, 24) 1, 6
match(0x14e5a5d8, 1, 24) 1, 2
match(0x14e5aff0, 1, 24) 1, 1
match(0x14e5b028, 1, 24) 1, 1
```

```
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
doaddinfo() addcount = 1
do additional "mvs03a.ralplex1.buddha.ral.ibm.com" (from "ralplex1.buddha.ral.ibm.com")
found it
match(0x14e5a630, 1, 5) 1, 1
match(0x14e5a630, 1, 5) 1, 1
match(0x14e5a630, 1, 1) 1, 1
make_rr(mvs03a.ralplex1.buddha.ral.ibm.com, 14e5a630, bf3ee, 374, 0) 4 zone 2 ttl 3600
match(0x14e5a630, 1, 24) 1, 1
addinfo: adding address data n = 16
ns_req: answer ->  192.168.221.88 .1096 fd=5 id=1 size=142 Local
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: qr aa rd ra; Ques: 1, Ans: 2, Auth: 1, Addit: 1
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN

;; ANSWERS:
tnral.buddha.ral.ibm.com. 3600 IN CNAME tnral.ralplex1.buddha.ral.ibm.com.
tnral.ralplex1.buddha.ral.ibm.com. 0 IN A 192.168.250.3

;; AUTHORITY RECORDS:
ralplex1.buddha.ral.ibm.com. 3600 IN NS mvs03a.ralplex1.buddha.ral.ibm.com.

;; ADDITIONAL RECORDS:
mvs03a.ralplex1.buddha.ral.ibm.com. 3600 IN A 192.168.250.3

loadxfer() "ralplex1.buddha.ral.ibm.com"
rm_datum(14e5a500, 14e5a500, 0) -> 14e5a5d8
rm_datum(14e5a5d8, 14e5a5d8, 0) -> 14e5aff0
rm_datum(14e5aff0, 14e5aff0, 0) -> 14e5b028
rm_datum(14e5b028, 14e5b028, 0) -> 0
rm_datum(14e5ac88, 14e5ac88, 0) -> 14e5ada8
rm_datum(14e5ada8, 14e5ada8, 0) -> 0
rm_datum(14e5ade0, 14e5ade0, 0) -> 0
rm_datum(14e5ad38, 14e5ad38, 0) -> 0
rm_datum(14e5a630, 14e5a630, 0) -> 0
rm_datum(14e5a6b8, 14e5a6b8, 0) -> 0
rm_datum(14e5a728, 14e5a728, 0) -> 0
rm_datum(14e5ae50, 14e5ae50, 0) -> 14e5af48
rm_datum(14e5af48, 14e5af48, 0) -> 0
rm_datum(14e5af80, 14e5af80, 0) -> 0
rm_datum(14e5aec0, 14e5aec0, 0) -> 0
rm_datum(14e5a798, 14e5a798, 0) -> 0
purge_zone(ralplex1.buddha.ral.ibm.com,1)
db_load(xback, ralplex1.buddha.ral.ibm.com, 2, Nil)
db_load: origin buddha.ral.ibm.com., buf ralplex1.buddha.ral.ibm.com
db_load: origin now buddha.ral.ibm.com
d='ralplex1.buddha.ral.ibm.com', c=1, t=6, ttl=0, data='mvs03a.ralplex1.buddha.ral.ibm.com.'
db_update(ralplex1.buddha.ral.ibm.com, 0x14e5a510, 0x14e5a510, 01, 0x14dcd8b8)
match(0x14dcd938, 1, 6) 1, 6
db_update: adding 14e5a510
d='ralplex1.buddha.ral.ibm.com', c=1, t=2, ttl=0, data='mvs03a.ralplex1.buddha.ral.ibm.com.'
db_update(ralplex1.buddha.ral.ibm.com, 0x14e5a5e8, 0x14e5a5e8, 01, 0x14dcd8b8)
match(0x14e5a510, 1, 6) 1, 6
match(0x14e5a510, 1, 2) 1, 6
db_update: adding 14e5a5e8
db_load: origin ralplex1.buddha.ral.ibm.com., buf buddha.ral.ibm.com
db_load: origin now ralplex1.buddha.ral.ibm.com
d='mvs03a.ralplex1.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.250.3'
db_update(mvs03a.ralplex1.buddha.ral.ibm.com, 0x14e5a640, 0x14e5a640, 01, 0x14dcd8b8)
savehash GROWING to 2
match(0x14e5a510, 1, 6) 1, 6
db_update: adding 14e5a640
d='mvs03c.ralplex1.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.251.4'
db_update(mvs03c.ralplex1.buddha.ral.ibm.com, 0x14e5a6c8, 0x14e5a6c8, 01, 0x14dcd8b8)
match(0x14e5a510, 1, 6) 1, 6
```

```
db_update: adding 14e5a6c8
d='mvs28a.ralplex1.buddha.ral.ibm.com', c=1, t=1, ttl=0, data='192.168.252.28'
db_update(mvs28a.ralplex1.buddha.ral.ibm.com, 0x14e5a738, 0x14e5a738, 01, 0x14dcd8b8)
match(0x14e5a510, 1, 6) 1, 6
db_update: adding 14e5a738
d='ralplex1.ralplex1.buddha.ral.ibm.com', c=1, t=5, ttl=0, data='ralplex1.buddha.ral.ibm.com.'
db_update(ralplex1.ralplex1.buddha.ral.ibm.com, 0x14e5a7a8, 0x14e5a7a8, 01, 0x14dcd8b8)
match(0x14e5a510, 1, 6) 1, 6
db_update: adding 14e5a7a8
wlm_load ralplex1.buddha.ral.ibm.com
 group list retcode = -1 rsncode = 1034
 group list retcode = 0 rsncode = 0
 group list entry_count = 3
 querying group =  FTPRAL
 server list retcode = -1 rsncode = 1034
 server list retcode = 0 rsncode = 0
 querying group =  TNRAL
 server list retcode = -1 rsncode = 1034
 server list retcode = 0 rsncode = 0
 querying group =  TCPIP
 server list retcode = -1 rsncode = 1034
 server list retcode = 0 rsncode = 0
 processing group = FTPRAL.ralplex1.buddha.ral.ibm.com count = 2
 minimum weight = 21
 processing server = MVS03A weight = 42 host = MVS03A
db_update(FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14e5ad90, 0x14e5ad90, 01, 0x14dcd8b8)
savehash GROWING to 11
savehash(0x14e5a678) cnt=5, sz=2, newsz=11
match(0x14e5a510, 1, 6) 1, 6
db_update: adding 14e5ad90
db_update(MVS03A.FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14e5ae30, 0x14e5ae30, 01, 0x14dcd8b8)
savehash GROWING to 2
match(0x14e5ad90, 1, 6) 1, 1
match(0x14e5ad90, 1, 2) 1, 1
match(0x14e5a510, 1, 6) 1, 6
db_update: adding 14e5ae30
 processing server = MVS28A weight = 21 host = MVS28A
db_update(FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14e5aea0, 0x14e5aea0, 01, 0x14dcd8b8)
match(0x14e5ad90, 1, 6) 1, 1
match(0x14e5ad90, 1, 2) 1, 1
match(0x14e5a510, 1, 6) 1, 6
match(0x14e5ad90, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for FTPRAL.ralplex1.buddha.ral.ibm.com is 4(5) from  192.168.221.88 .1096, is 4(5) in cache
db_update: adding 14e5aea0
db_update(MVS28A.FTPRAL.ralplex1.buddha.ral.ibm.com, 0x14e5aed8, 0x14e5aed8, 01, 0x14dcd8b8)
match(0x14e5ad90, 1, 6) 1, 1
match(0x14e5aea0, 1, 6) 1, 1
match(0x14e5ad90, 1, 2) 1, 1
match(0x14e5aea0, 1, 2) 1, 1
match(0x14e5a510, 1, 6) 1, 6
db_update: adding 14e5aed8
 processing group = TNRAL.ralplex1.buddha.ral.ibm.com count = 2
 minimum weight = 21
 processing server = MVS03A weight = 42 host = MVS03A
db_update(TNRAL.ralplex1.buddha.ral.ibm.com, 0x14e5af48, 0x14e5af48, 01, 0x14dcd8b8)
match(0x14e5a510, 1, 6) 1, 6
db_update: adding 14e5af48
db_update(MVS03A.TNRAL.ralplex1.buddha.ral.ibm.com, 0x14e5afb8, 0x14e5afb8, 01, 0x14dcd8b8)
savehash GROWING to 2
match(0x14e5af48, 1, 6) 1, 1
match(0x14e5af48, 1, 2) 1, 1
match(0x14e5a510, 1, 6) 1, 6
db_update: adding 14e5afb8
 processing server = MVS28A weight = 21 host = MVS28A
db_update(TNRAL.ralplex1.buddha.ral.ibm.com, 0x14e5b040, 0x14e5b040, 01, 0x14dcd8b8)
```

```
match(0x14e5af48, 1, 6) 1, 1
match(0x14e5af48, 1, 2) 1, 1
match(0x14e5a510, 1, 6) 1, 6
match(0x14e5af48, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for TNRAL.ralplex1.buddha.ral.ibm.com is 4(5) from  192.168.221.88 .1096, is 4(5) in cache
db_update: adding 14e5b040
db_update(MVS28A.TNRAL.ralplex1.buddha.ral.ibm.com, 0x14e5b078, 0x14e5b078, 01, 0x14dcd8b8)
match(0x14e5af48, 1, 6) 1, 1
match(0x14e5b040, 1, 6) 1, 1
match(0x14e5af48, 1, 2) 1, 1
match(0x14e5b040, 1, 2) 1, 1
match(0x14e5a510, 1, 6) 1, 6
db_update: adding 14e5b078
 processing group = ralplex1.buddha.ral.ibm.com count = 2
 minimum weight = 21
 processing server = TO3ATCP weight = 42 host = MVS03A
db_update(ralplex1.buddha.ral.ibm.com, 0x14e5b0e8, 0x14e5b0e8, 01, 0x14dcd8b8)
match(0x14e5a510, 1, 6) 1, 6
match(0x14e5a510, 1, 1) 1, 6
match(0x14e5a5e8, 1, 1) 1, 2
db_update: adding 14e5b0e8
 processing server = T28ATCP weight = 21 host = MVS28A
db_update(ralplex1.buddha.ral.ibm.com, 0x14e5b120, 0x14e5b120, 01, 0x14dcd8b8)
match(0x14e5a510, 1, 6) 1, 6
match(0x14e5a510, 1, 1) 1, 6
match(0x14e5a5e8, 1, 1) 1, 2
match(0x14e5b0e8, 1, 1) 1, 1
db_update: flags = 0x1, sizes = 4, 4 (cmp -1)
credibility for ralplex1.buddha.ral.ibm.com is 4(5) from  192.168.221.88 .1096, is 4(5) in cache
db_update: adding 14e5b120
db_update(ralplex1.ralplex1.buddha.ral.ibm.com, 0x14e5a988, 0x14e5a988, 01, 0x14dcd8b8)
match(0x14e5a7a8, 1, 6) 1, 5
match(0x14e5a7a8, 1, 2) 1, 5
match(0x14e5a510, 1, 6) 1, 6
match(0x14e5a7a8, 1, 5) 1, 5
db_update: flags = 0x1, sizes = 28, 28 (cmp 0)
credibility for ralplex1.ralplex1.buddha.ral.ibm.com is 4(5) from  192.168.221.88 .1096, is 4(5) in cache
fclose(7) succeeded
next_refresh: 896822324, next_alarm: 896886612, tt.tv_sec: 896886573
sched_maint: Next interrupt in 60 sec

datagram from  192.168.221.88 .1097, fd 5, len 42; now Wed Jun  3 15:09:33 1998
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
ns_req(from= 192.168.221.88 .1097)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
req: nlookup(tnral.buddha.ral.ibm.com) id 1 type=1 class=1
req: found 'tnral.buddha.ral.ibm.com' as 'tnral.buddha.ral.ibm.com' (cname=0)
******* Remove_Expired_RRs  Entered *******
name: tnral
  type=5 size=34
name: ftpoeral
  type=5 size=37
Remove_Expired_RRs rc=0
```

```
wanted(0x14dce060, 1, 1)  IN CNAME
make_rr(tnral.buddha.ral.ibm.com, 14dce060, bf39a, 458, 1) 34 zone 1 ttl 3600
match(0x14dce060, 1, 24) 1, 5
finddata: added 1 class 1 type 1 RRs
req: nlookup(tnral.ralplex1.buddha.ral.ibm.com) id 1 type=1 class=1
req: found 'tnral.ralplex1.buddha.ral.ibm.com' as 'tnral.ralplex1.buddha.ral.ibm.com' (cname=1)
******* Remove_Expired_RRs  Entered *******
name: TNRAL
  type=1 size=4
  type=1 size=4
Remove_Expired_RRs rc=0
 Looking for WLM generated Address
make_rr(tnral.ralplex1.buddha.ral.ibm.com, 14e5af48, bf3c9, 411, 1) 4 zone 2 ttl 0
match(0x14e5af48, 1, 24) 1, 1
match(0x14e5b040, 1, 24) 1, 1
 Weight = 1
req: foundname=2, count=1, founddata=1, cname=1
sort_response(1)
findns: np 0x14e5af80 'TNRAL'
match(0x14e5af48, 1, 6) 1, 1
match(0x14e5b040, 1, 6) 1, 1
match(0x14e5af48, 1, 2) 1, 1
match(0x14e5b040, 1, 2) 1, 1
findns: np 0x14e5a5a8 'ralplex1'
match(0x14e5a510, 1, 6) 1, 6
findns: SOA found
req: leaving (tnral.ralplex1.buddha.ral.ibm.com, rcode 0)
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
findns: np 0x14e5a5a8 'ralplex1'
match(0x14e5a510, 1, 2) 1, 6
match(0x14e5a5e8, 1, 2) 1, 2
match(0x14e5b0e8, 1, 2) 1, 1
match(0x14e5b120, 1, 2) 1, 1
findns: 1 NS's added for 'ralplex1'
make_rr(ralplex1.buddha.ral.ibm.com, 14e5a5e8, bf3d9, 395, 1) 35 zone 2 ttl 3600
match(0x14e5a510, 1, 24) 1, 6
match(0x14e5a5e8, 1, 24) 1, 2
match(0x14e5b0e8, 1, 24) 1, 1
match(0x14e5b120, 1, 24) 1, 1
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
doaddinfo() addcount = 1
do additional "mvs03a.ralplex1.buddha.ral.ibm.com" (from "ralplex1.buddha.ral.ibm.com")
found it
match(0x14e5a640, 1, 5) 1, 1
match(0x14e5a640, 1, 5) 1, 1
match(0x14e5a640, 1, 1) 1, 1
make_rr(mvs03a.ralplex1.buddha.ral.ibm.com, 14e5a640, bf3ee, 374, 0) 4 zone 2 ttl 3600
match(0x14e5a640, 1, 24) 1, 1
addinfo: adding address data n = 16
ns_req: answer ->  192.168.221.88 .1097 fd=5 id=1 size=142 Local
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: qr aa rd ra; Ques: 1, Ans: 2, Auth: 1, Addit: 1
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN

;; ANSWERS:
tnral.buddha.ral.ibm.com. 3600 IN CNAME tnral.ralplex1.buddha.ral.ibm.com.
tnral.ralplex1.buddha.ral.ibm.com. 0 IN A 192.168.250.3

;; AUTHORITY RECORDS:
ralplex1.buddha.ral.ibm.com. 3600 IN NS mvs03a.ralplex1.buddha.ral.ibm.com.

;; ADDITIONAL RECORDS:
mvs03a.ralplex1.buddha.ral.ibm.com. 3600 IN A 192.168.250.3
```

```
datagram from  192.168.221.88 .1098, fd 5, len 42; now Wed Jun  3 15:09:33 1998
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
ns_req(from= 192.168.221.88 .1098)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
req: nlookup(tnral.buddha.ral.ibm.com) id 1 type=1 class=1
req: found 'tnral.buddha.ral.ibm.com' as 'tnral.buddha.ral.ibm.com' (cname=0)
******* Remove_Expired_RRs  Entered *******
name: tnral
  type=5 size=34
name: ftpoeral
  type=5 size=37
Remove_Expired_RRs rc=0
wanted(0x14dce060, 1, 1)  IN CNAME
make_rr(tnral.buddha.ral.ibm.com, 14dce060, bf39a, 458, 1) 34 zone 1 ttl 3600
match(0x14dce060, 1, 24) 1, 5
finddata: added 1 class 1 type 1 RRs
req: nlookup(tnral.ralplex1.buddha.ral.ibm.com) id 1 type=1 class=1
req: found 'tnral.ralplex1.buddha.ral.ibm.com' as 'tnral.ralplex1.buddha.ral.ibm.com' (cname=1)
******* Remove_Expired_RRs  Entered *******
name: TNRAL
  type=1 size=4
  type=1 size=4
Remove_Expired_RRs rc=0
 Looking for WLM generated Address
make_rr(tnral.ralplex1.buddha.ral.ibm.com, 14e5af48, bf3c9, 411, 1) 4 zone 2 ttl 0
match(0x14e5af48, 1, 24) 1, 1
match(0x14e5b040, 1, 24) 1, 1
 Weight = 0
 Current Server Weight 0
req: foundname=2, count=1, founddata=1, cname=1
sort_response(1)
findns: np 0x14e5af80 'TNRAL'
match(0x14e5af48, 1, 6) 1, 1
match(0x14e5b040, 1, 6) 1, 1
match(0x14e5af48, 1, 2) 1, 1
match(0x14e5b040, 1, 2) 1, 1
findns: np 0x14e5a5a8 'ralplex1'
match(0x14e5a510, 1, 6) 1, 6
findns: SOA found
req: leaving (tnral.ralplex1.buddha.ral.ibm.com, rcode 0)
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
findns: np 0x14e5a5a8 'ralplex1'
match(0x14e5a510, 1, 2) 1, 6
match(0x14e5a5e8, 1, 2) 1, 2
match(0x14e5b0e8, 1, 2) 1, 1
match(0x14e5b120, 1, 2) 1, 1
findns: 1 NS's added for 'ralplex1'
make_rr(ralplex1.buddha.ral.ibm.com, 14e5a5e8, bf3d9, 395, 1) 35 zone 2 ttl 3600
match(0x14e5a510, 1, 24) 1, 6
match(0x14e5a5e8, 1, 24) 1, 2
match(0x14e5b0e8, 1, 24) 1, 1
match(0x14e5b120, 1, 24) 1, 1
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
doaddinfo() addcount = 1
```

```
do additional "mvs03a.ralplex1.buddha.ral.ibm.com" (from "ralplex1.buddha.ral.ibm.com")
found it
match(0x14e5a640, 1, 5) 1, 1
match(0x14e5a640, 1, 5) 1, 1
match(0x14e5a640, 1, 1) 1, 1
make_rr(mvs03a.ralplex1.buddha.ral.ibm.com, 14e5a640, bf3ee, 374, 0) 4 zone 2 ttl 3600
match(0x14e5a640, 1, 24) 1, 1
addinfo: adding address data n = 16
ns_req: answer ->  192.168.221.88 .1098 fd=5 id=1 size=142 Local
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: qr aa rd ra; Ques: 1, Ans: 2, Auth: 1, Addit: 1
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN

;; ANSWERS:
tnral.buddha.ral.ibm.com. 3600 IN CNAME tnral.ralplex1.buddha.ral.ibm.com.
tnral.ralplex1.buddha.ral.ibm.com. 0 IN A 192.168.250.3

;; AUTHORITY RECORDS:
ralplex1.buddha.ral.ibm.com. 3600 IN NS mvs03a.ralplex1.buddha.ral.ibm.com.

;; ADDITIONAL RECORDS:
mvs03a.ralplex1.buddha.ral.ibm.com. 3600 IN A 192.168.250.3


datagram from  192.168.221.88 .1099, fd 5, len 42; now Wed Jun  3 15:09:33 1998
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
ns_req(from= 192.168.221.88 .1099)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
req: nlookup(tnral.buddha.ral.ibm.com) id 1 type=1 class=1
req: found 'tnral.buddha.ral.ibm.com' as 'tnral.buddha.ral.ibm.com' (cname=0)
******* Remove_Expired_RRs  Entered *******
name: tnral
  type=5 size=34
name: ftpoeral
  type=5 size=37
Remove_Expired_RRs rc=0
wanted(0x14dce060, 1, 1)  IN CNAME
make_rr(tnral.buddha.ral.ibm.com, 14dce060, bf39a, 458, 1) 34 zone 1 ttl 3600
match(0x14dce060, 1, 24) 1, 5
finddata: added 1 class 1 type 1 RRs
req: nlookup(tnral.ralplex1.buddha.ral.ibm.com) id 1 type=1 class=1
req: found 'tnral.ralplex1.buddha.ral.ibm.com' as 'tnral.ralplex1.buddha.ral.ibm.com' (cname=1)
******* Remove_Expired_RRs  Entered *******
name: TNRAL
  type=1 size=4
  type=1 size=4
Remove_Expired_RRs rc=0
 Looking for WLM generated Address
make_rr(tnral.ralplex1.buddha.ral.ibm.com, 14e5b040, bf3c9, 411, 1) 4 zone 2 ttl 0
match(0x14e5af48, 1, 24) 1, 1
match(0x14e5b040, 1, 24) 1, 1
 Weight = 0
 Current Server Weight 0
```

```
 Initializing all Weights
req: foundname=2, count=1, founddata=1, cname=1
sort_response(1)
findns: np 0x14e5af80 'TNRAL'
match(0x14e5af48, 1, 6) 1, 1
match(0x14e5b040, 1, 6) 1, 1
match(0x14e5af48, 1, 2) 1, 1
match(0x14e5b040, 1, 2) 1, 1
findns: np 0x14e5a5a8 'ralplex1'
match(0x14e5a510, 1, 6) 1, 6
findns: SOA found
req: leaving (tnral.ralplex1.buddha.ral.ibm.com, rcode 0)
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
findns: np 0x14e5a5a8 'ralplex1'
match(0x14e5a510, 1, 2) 1, 6
match(0x14e5a5e8, 1, 2) 1, 2
match(0x14e5b0e8, 1, 2) 1, 1
match(0x14e5b120, 1, 2) 1, 1
findns: 1 NS's added for 'ralplex1'
make_rr(ralplex1.buddha.ral.ibm.com, 14e5a5e8, bf3d9, 395, 1) 35 zone 2 ttl 3600
match(0x14e5a510, 1, 24) 1, 6
match(0x14e5a5e8, 1, 24) 1, 2
match(0x14e5b0e8, 1, 24) 1, 1
match(0x14e5b120, 1, 24) 1, 1
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
doaddinfo() addcount = 1
do additional "mvs03a.ralplex1.buddha.ral.ibm.com" (from "ralplex1.buddha.ral.ibm.com")
found it
match(0x14e5a640, 1, 5) 1, 1
match(0x14e5a640, 1, 5) 1, 1
match(0x14e5a640, 1, 1) 1, 1
make_rr(mvs03a.ralplex1.buddha.ral.ibm.com, 14e5a640, bf3ee, 374, 0) 4 zone 2 ttl 3600
match(0x14e5a640, 1, 24) 1, 1
addinfo: adding address data n = 16
ns_req: answer ->  192.168.221.88 .1099 fd=5 id=1 size=142 Local
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: qr aa rd ra; Ques: 1, Ans: 2, Auth: 1, Addit: 1
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN

;; ANSWERS:
tnral.buddha.ral.ibm.com. 3600 IN CNAME tnral.ralplex1.buddha.ral.ibm.com.
tnral.ralplex1.buddha.ral.ibm.com. 0 IN A 192.168.252.28

;; AUTHORITY RECORDS:
ralplex1.buddha.ral.ibm.com. 3600 IN NS mvs03a.ralplex1.buddha.ral.ibm.com.

;; ADDITIONAL RECORDS:
mvs03a.ralplex1.buddha.ral.ibm.com. 3600 IN A 192.168.250.3


datagram from  192.168.221.88 .1100, fd 5, len 42; now Wed Jun  3 15:09:33 1998
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
ns_req(from= 192.168.221.88 .1100)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN
```

```
;; ...truncated
req: nlookup(tnral.buddha.ral.ibm.com) id 1 type=1 class=1
req: found 'tnral.buddha.ral.ibm.com' as 'tnral.buddha.ral.ibm.com' (cname=0)
******* Remove_Expired_RRs  Entered *******
name: tnral
  type=5 size=34
name: ftpoeral
  type=5 size=37
Remove_Expired_RRs rc=0
wanted(0x14dce060, 1, 1)  IN CNAME
make_rr(tnral.buddha.ral.ibm.com, 14dce060, bf39a, 458, 1) 34 zone 1 ttl 3600
match(0x14dce060, 1, 24) 1, 5
finddata: added 1 class 1 type 1 RRs
req: nlookup(tnral.ralplex1.buddha.ral.ibm.com) id 1 type=1 class=1
req: found 'tnral.ralplex1.buddha.ral.ibm.com' as 'tnral.ralplex1.buddha.ral.ibm.com' (cname=1)
******* Remove_Expired_RRs  Entered *******
name: TNRAL
  type=1 size=4
  type=1 size=4
Remove_Expired_RRs rc=0
 Looking for WLM generated Address
make_rr(tnral.ralplex1.buddha.ral.ibm.com, 14e5af48, bf3c9, 411, 1) 4 zone 2 ttl 0
match(0x14e5af48, 1, 24) 1, 1
match(0x14e5b040, 1, 24) 1, 1
 Weight = 1
req: foundname=2, count=1, founddata=1, cname=1
sort_response(1)
findns: np 0x14e5af80 'TNRAL'
match(0x14e5af48, 1, 6) 1, 1
match(0x14e5b040, 1, 6) 1, 1
match(0x14e5af48, 1, 2) 1, 1
match(0x14e5b040, 1, 2) 1, 1
findns: np 0x14e5a5a8 'ralplex1'
match(0x14e5a510, 1, 6) 1, 6
findns: SOA found
req: leaving (tnral.ralplex1.buddha.ral.ibm.com, rcode 0)
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
findns: np 0x14e5a5a8 'ralplex1'
match(0x14e5a510, 1, 2) 1, 6
match(0x14e5a5e8, 1, 2) 1, 2
match(0x14e5b0e8, 1, 2) 1, 1
match(0x14e5b120, 1, 2) 1, 1
findns: 1 NS's added for 'ralplex1'
make_rr(ralplex1.buddha.ral.ibm.com, 14e5a5e8, bf3d9, 395, 1) 35 zone 2 ttl 3600
match(0x14e5a510, 1, 24) 1, 6
match(0x14e5a5e8, 1, 24) 1, 2
match(0x14e5b0e8, 1, 24) 1, 1
match(0x14e5b120, 1, 24) 1, 1
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
doaddinfo() addcount = 1
do additional "mvs03a.ralplex1.buddha.ral.ibm.com" (from "ralplex1.buddha.ral.ibm.com")
found it
match(0x14e5a640, 1, 5) 1, 1
match(0x14e5a640, 1, 5) 1, 1
match(0x14e5a640, 1, 1) 1, 1
make_rr(mvs03a.ralplex1.buddha.ral.ibm.com, 14e5a640, bf3ee, 374, 0) 4 zone 2 ttl 3600
match(0x14e5a640, 1, 24) 1, 1
addinfo: adding address data n = 16
ns_req: answer ->  192.168.221.88 .1100 fd=5 id=1 size=142 Local
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: qr aa rd ra; Ques: 1, Ans: 2, Auth: 1, Addit: 1
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN
```

```
;; ANSWERS:
tnral.buddha.ral.ibm.com. 3600 IN CNAME tnral.ralplex1.buddha.ral.ibm.com.
tnral.ralplex1.buddha.ral.ibm.com. 0 IN A 192.168.250.3

;; AUTHORITY RECORDS:
ralplex1.buddha.ral.ibm.com. 3600 IN NS mvs03a.ralplex1.buddha.ral.ibm.com.

;; ADDITIONAL RECORDS:
mvs03a.ralplex1.buddha.ral.ibm.com. 3600 IN A 192.168.250.3


datagram from  192.168.221.88 .1101, fd 5, len 42; now Wed Jun  3 15:09:33 1998
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
ns_req(from= 192.168.221.88 .1101)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
req: nlookup(tnral.buddha.ral.ibm.com) id 1 type=1 class=1
req: found 'tnral.buddha.ral.ibm.com' as 'tnral.buddha.ral.ibm.com' (cname=0)
******* Remove_Expired_RRs  Entered *******
name: tnral
  type=5 size=34
name: ftpoeral
  type=5 size=37
Remove_Expired_RRs rc=0
wanted(0x14dce060, 1, 1)  IN CNAME
make_rr(tnral.buddha.ral.ibm.com, 14dce060, bf39a, 458, 1) 34 zone 1 ttl 3600
match(0x14dce060, 1, 24) 1, 5
finddata: added 1 class 1 type 1 RRs
req: nlookup(tnral.ralplex1.buddha.ral.ibm.com) id 1 type=1 class=1
req: found 'tnral.ralplex1.buddha.ral.ibm.com' as 'tnral.ralplex1.buddha.ral.ibm.com' (cname=1)
******* Remove_Expired_RRs  Entered *******
name: TNRAL
  type=1 size=4
  type=1 size=4
Remove_Expired_RRs rc=0
 Looking for WLM generated Address
make_rr(tnral.ralplex1.buddha.ral.ibm.com, 14e5af48, bf3c9, 411, 1) 4 zone 2 ttl 0
match(0x14e5af48, 1, 24) 1, 1
match(0x14e5b040, 1, 24) 1, 1
 Weight = 0
 Current Server Weight 0
req: foundname=2, count=1, founddata=1, cname=1
sort_response(1)
findns: np 0x14e5af80 'TNRAL'
match(0x14e5af48, 1, 6) 1, 1
match(0x14e5b040, 1, 6) 1, 1
match(0x14e5af48, 1, 2) 1, 1
match(0x14e5b040, 1, 2) 1, 1
findns: np 0x14e5a5a8 'ralplex1'
match(0x14e5a510, 1, 6) 1, 6
findns: SOA found
req: leaving (tnral.ralplex1.buddha.ral.ibm.com, rcode 0)
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
findns: np 0x14e5a5a8 'ralplex1'
match(0x14e5a510, 1, 2) 1, 6
```

```
match(0x14e5a5e8, 1, 2) 1, 2
match(0x14e5b0e8, 1, 2) 1, 1
match(0x14e5b120, 1, 2) 1, 1
findns: 1 NS's added for 'ralplex1'
make_rr(ralplex1.buddha.ral.ibm.com, 14e5a5e8, bf3d9, 395, 1) 35 zone 2 ttl 3600
match(0x14e5a510, 1, 24) 1, 6
match(0x14e5a5e8, 1, 24) 1, 2
match(0x14e5b0e8, 1, 24) 1, 1
match(0x14e5b120, 1, 24) 1, 1
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
doaddinfo() addcount = 1
do additional "mvs03a.ralplex1.buddha.ral.ibm.com" (from "ralplex1.buddha.ral.ibm.com")
found it
match(0x14e5a640, 1, 5) 1, 1
match(0x14e5a640, 1, 5) 1, 1
match(0x14e5a640, 1, 1) 1, 1
make_rr(mvs03a.ralplex1.buddha.ral.ibm.com, 14e5a640, bf3ee, 374, 0) 4 zone 2 ttl 3600
match(0x14e5a640, 1, 24) 1, 1
addinfo: adding address data n = 16
ns_req: answer ->  192.168.221.88 .1101 fd=5 id=1 size=142 Local
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: qr aa rd ra; Ques: 1, Ans: 2, Auth: 1, Addit: 1
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN

;; ANSWERS:
tnral.buddha.ral.ibm.com. 3600 IN CNAME tnral.ralplex1.buddha.ral.ibm.com.
tnral.ralplex1.buddha.ral.ibm.com. 0 IN A 192.168.250.3

;; AUTHORITY RECORDS:
ralplex1.buddha.ral.ibm.com. 3600 IN NS mvs03a.ralplex1.buddha.ral.ibm.com.

;; ADDITIONAL RECORDS:
mvs03a.ralplex1.buddha.ral.ibm.com. 3600 IN A 192.168.250.3


datagram from  192.168.221.88 .1102, fd 5, len 42; now Wed Jun  3 15:09:33 1998
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
ns_req(from= 192.168.221.88 .1102)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
req: nlookup(tnral.buddha.ral.ibm.com) id 1 type=1 class=1
req: found 'tnral.buddha.ral.ibm.com' as 'tnral.buddha.ral.ibm.com' (cname=0)
******* Remove_Expired_RRs  Entered *******
name: tnral
  type=5 size=34
name: ftpoeral
  type=5 size=37
Remove_Expired_RRs rc=0
wanted(0x14dce060, 1, 1)  IN CNAME
make_rr(tnral.buddha.ral.ibm.com, 14dce060, bf39a, 458, 1) 34 zone 1 ttl 3600
match(0x14dce060, 1, 24) 1, 5
finddata: added 1 class 1 type 1 RRs
req: nlookup(tnral.ralplex1.buddha.ral.ibm.com) id 1 type=1 class=1
req: found 'tnral.ralplex1.buddha.ral.ibm.com' as 'tnral.ralplex1.buddha.ral.ibm.com' (cname=1)
```

```
******* Remove_Expired_RRs  Entered *******
name: TNRAL
  type=1 size=4
  type=1 size=4
Remove_Expired_RRs rc=0
 Looking for WLM generated Address
make_rr(tnral.ralplex1.buddha.ral.ibm.com, 14e5b040, bf3c9, 411, 1) 4 zone 2 ttl 0
match(0x14e5af48, 1, 24) 1, 1
match(0x14e5b040, 1, 24) 1, 1
 Weight = 0
 Current Server Weight 0
 Initializing all Weights
req: foundname=2, count=1, founddata=1, cname=1
sort_response(1)
findns: np 0x14e5af80 'TNRAL'
match(0x14e5af48, 1, 6) 1, 1
match(0x14e5b040, 1, 6) 1, 1
match(0x14e5af48, 1, 2) 1, 1
match(0x14e5b040, 1, 2) 1, 1
findns: np 0x14e5a5a8 'ralplex1'
match(0x14e5a510, 1, 6) 1, 6
findns: SOA found
req: leaving (tnral.ralplex1.buddha.ral.ibm.com, rcode 0)
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
findns: np 0x14e5a5a8 'ralplex1'
match(0x14e5a510, 1, 2) 1, 6
match(0x14e5a5e8, 1, 2) 1, 2
match(0x14e5b0e8, 1, 2) 1, 1
match(0x14e5b120, 1, 2) 1, 1
findns: 1 NS's added for 'ralplex1'
make_rr(ralplex1.buddha.ral.ibm.com, 14e5a5e8, bf3d9, 395, 1) 35 zone 2 ttl 3600
match(0x14e5a510, 1, 24) 1, 6
match(0x14e5a5e8, 1, 24) 1, 2
match(0x14e5b0e8, 1, 24) 1, 1
match(0x14e5b120, 1, 24) 1, 1
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
doaddinfo() addcount = 1
do additional "mvs03a.ralplex1.buddha.ral.ibm.com" (from "ralplex1.buddha.ral.ibm.com")
found it
match(0x14e5a640, 1, 5) 1, 1
match(0x14e5a640, 1, 5) 1, 1
match(0x14e5a640, 1, 1) 1, 1
make_rr(mvs03a.ralplex1.buddha.ral.ibm.com, 14e5a640, bf3ee, 374, 0) 4 zone 2 ttl 3600
match(0x14e5a640, 1, 24) 1, 1
addinfo: adding address data n = 16
ns_req: answer ->  192.168.221.88 .1102 fd=5 id=1 size=142 Local
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: qr aa rd ra; Ques: 1, Ans: 2, Auth: 1, Addit: 1
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN

;; ANSWERS:
tnral.buddha.ral.ibm.com. 3600 IN CNAME tnral.ralplex1.buddha.ral.ibm.com.
tnral.ralplex1.buddha.ral.ibm.com. 0 IN A 192.168.252.28

;; AUTHORITY RECORDS:
ralplex1.buddha.ral.ibm.com. 3600 IN NS mvs03a.ralplex1.buddha.ral.ibm.com.

;; ADDITIONAL RECORDS:
mvs03a.ralplex1.buddha.ral.ibm.com. 3600 IN A 192.168.250.3


datagram from  192.168.221.88 .1103, fd 5, len 42; now Wed Jun  3 15:09:34 1998
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
```

```
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
ns_req(from= 192.168.221.88 .1103)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
req: nlookup(tnral.buddha.ral.ibm.com) id 1 type=1 class=1
req: found 'tnral.buddha.ral.ibm.com' as 'tnral.buddha.ral.ibm.com' (cname=0)
******* Remove_Expired_RRs  Entered *******
name: tnral
  type=5 size=34
name: ftpoeral
  type=5 size=37
Remove_Expired_RRs rc=0
wanted(0x14dce060, 1, 1)  IN CNAME
make_rr(tnral.buddha.ral.ibm.com, 14dce060, bf39a, 458, 1) 34 zone 1 ttl 3600
match(0x14dce060, 1, 24) 1, 5
finddata: added 1 class 1 type 1 RRs
req: nlookup(tnral.ralplex1.buddha.ral.ibm.com) id 1 type=1 class=1
req: found 'tnral.ralplex1.buddha.ral.ibm.com' as 'tnral.ralplex1.buddha.ral.ibm.com' (cname=1)
******* Remove_Expired_RRs  Entered *******
name: TNRAL
  type=1 size=4
  type=1 size=4
Remove_Expired_RRs rc=0
 Looking for WLM generated Address
make_rr(tnral.ralplex1.buddha.ral.ibm.com, 14e5af48, bf3c9, 411, 1) 4 zone 2 ttl 0
match(0x14e5af48, 1, 24) 1, 1
match(0x14e5b040, 1, 24) 1, 1
 Weight = 1
req: foundname=2, count=1, founddata=1, cname=1
sort_response(1)
findns: np 0x14e5af80 'TNRAL'
match(0x14e5af48, 1, 6) 1, 1
match(0x14e5b040, 1, 6) 1, 1
match(0x14e5af48, 1, 2) 1, 1
match(0x14e5b040, 1, 2) 1, 1
findns: np 0x14e5a5a8 'ralplex1'
match(0x14e5a510, 1, 6) 1, 6
findns: SOA found
req: leaving (tnral.ralplex1.buddha.ral.ibm.com, rcode 0)
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
findns: np 0x14e5a5a8 'ralplex1'
match(0x14e5a510, 1, 2) 1, 6
match(0x14e5a5e8, 1, 2) 1, 2
match(0x14e5b0e8, 1, 2) 1, 1
match(0x14e5b120, 1, 2) 1, 1
findns: 1 NS's added for 'ralplex1'
make_rr(ralplex1.buddha.ral.ibm.com, 14e5a5e8, bf3d9, 395, 1) 35 zone 2 ttl 3600
match(0x14e5a510, 1, 24) 1, 6
match(0x14e5a5e8, 1, 24) 1, 2
match(0x14e5b0e8, 1, 24) 1, 1
match(0x14e5b120, 1, 24) 1, 1
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
doaddinfo() addcount = 1
do additional "mvs03a.ralplex1.buddha.ral.ibm.com" (from "ralplex1.buddha.ral.ibm.com")
found it
match(0x14e5a640, 1, 5) 1, 1
match(0x14e5a640, 1, 5) 1, 1
```

```
match(0x14e5a640, 1, 1) 1, 1
make_rr(mvs03a.ralplex1.buddha.ral.ibm.com, 14e5a640, bf3ee, 374, 0) 4 zone 2 ttl 3600
match(0x14e5a640, 1, 24) 1, 1
addinfo: adding address data n = 16
ns_req: answer ->  192.168.221.88 .1103 fd=5 id=1 size=142 Local
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: qr aa rd ra; Ques: 1, Ans: 2, Auth: 1, Addit: 1
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN

;; ANSWERS:
tnral.buddha.ral.ibm.com. 3600 IN CNAME tnral.ralplex1.buddha.ral.ibm.com.
tnral.ralplex1.buddha.ral.ibm.com. 0 IN A 192.168.250.3

;; AUTHORITY RECORDS:
ralplex1.buddha.ral.ibm.com. 3600 IN NS mvs03a.ralplex1.buddha.ral.ibm.com.

;; ADDITIONAL RECORDS:
mvs03a.ralplex1.buddha.ral.ibm.com. 3600 IN A 192.168.250.3


datagram from  192.168.221.88 .1104, fd 5, len 42; now Wed Jun  3 15:09:34 1998
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
ns_req(from= 192.168.221.88 .1104)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
req: nlookup(tnral.buddha.ral.ibm.com) id 1 type=1 class=1
req: found 'tnral.buddha.ral.ibm.com' as 'tnral.buddha.ral.ibm.com' (cname=0)
******* Remove_Expired_RRs  Entered *******
name: tnral
  type=5 size=34
name: ftpoeral
  type=5 size=37
Remove_Expired_RRs rc=0
wanted(0x14dce060, 1, 1)  IN CNAME
make_rr(tnral.buddha.ral.ibm.com, 14dce060, bf39a, 458, 1) 34 zone 1 ttl 3600
match(0x14dce060, 1, 24) 1, 5
finddata: added 1 class 1 type 1 RRs
req: nlookup(tnral.ralplex1.buddha.ral.ibm.com) id 1 type=1 class=1
req: found 'tnral.ralplex1.buddha.ral.ibm.com' as 'tnral.ralplex1.buddha.ral.ibm.com' (cname=1)
******* Remove_Expired_RRs  Entered *******
name: TNRAL
  type=1 size=4
  type=1 size=4
Remove_Expired_RRs rc=0
 Looking for WLM generated Address
make_rr(tnral.ralplex1.buddha.ral.ibm.com, 14e5af48, bf3c9, 411, 1) 4 zone 2 ttl 0
match(0x14e5af48, 1, 24) 1, 1
match(0x14e5b040, 1, 24) 1, 1
 Weight = 0
 Current Server Weight 0
req: foundname=2, count=1, founddata=1, cname=1
sort_response(1)
findns: np 0x14e5af80 'TNRAL'
match(0x14e5af48, 1, 6) 1, 1
```

```
match(0x14e5b040, 1, 6) 1, 1
match(0x14e5af48, 1, 2) 1, 1
match(0x14e5b040, 1, 2) 1, 1
findns: np 0x14e5a5a8 'ralplex1'
match(0x14e5a510, 1, 6) 1, 6
findns: SOA found
req: leaving (tnral.ralplex1.buddha.ral.ibm.com, rcode 0)
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
findns: np 0x14e5a5a8 'ralplex1'
match(0x14e5a510, 1, 2) 1, 6
match(0x14e5a5e8, 1, 2) 1, 2
match(0x14e5b0e8, 1, 2) 1, 1
match(0x14e5b120, 1, 2) 1, 1
findns: 1 NS's added for 'ralplex1'
make_rr(ralplex1.buddha.ral.ibm.com, 14e5a5e8, bf3d9, 395, 1) 35 zone 2 ttl 3600
match(0x14e5a510, 1, 24) 1, 6
match(0x14e5a5e8, 1, 24) 1, 2
match(0x14e5b0e8, 1, 24) 1, 1
match(0x14e5b120, 1, 24) 1, 1
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
doaddinfo() addcount = 1
do additional "mvs03a.ralplex1.buddha.ral.ibm.com" (from "ralplex1.buddha.ral.ibm.com")
found it
match(0x14e5a640, 1, 5) 1, 1
match(0x14e5a640, 1, 5) 1, 1
match(0x14e5a640, 1, 1) 1, 1
make_rr(mvs03a.ralplex1.buddha.ral.ibm.com, 14e5a640, bf3ee, 374, 0) 4 zone 2 ttl 3600
match(0x14e5a640, 1, 24) 1, 1
addinfo: adding address data n = 16
ns_req: answer ->  192.168.221.88 .1104 fd=5 id=1 size=142 Local
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: qr aa rd ra; Ques: 1, Ans: 2, Auth: 1, Addit: 1
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN

;; ANSWERS:
tnral.buddha.ral.ibm.com. 3600 IN CNAME tnral.ralplex1.buddha.ral.ibm.com.
tnral.ralplex1.buddha.ral.ibm.com. 0 IN A 192.168.250.3

;; AUTHORITY RECORDS:
ralplex1.buddha.ral.ibm.com. 3600 IN NS mvs03a.ralplex1.buddha.ral.ibm.com.

;; ADDITIONAL RECORDS:
mvs03a.ralplex1.buddha.ral.ibm.com. 3600 IN A 192.168.250.3


datagram from  192.168.221.88 .1105, fd 5, len 42; now Wed Jun  3 15:09:34 1998
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
ns_req(from= 192.168.221.88 .1105)
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: rd; Ques: 1, Ans: 0, Auth: 0, Addit: 0
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN


;; ...truncated
req: nlookup(tnral.buddha.ral.ibm.com) id 1 type=1 class=1
req: found 'tnral.buddha.ral.ibm.com' as 'tnral.buddha.ral.ibm.com' (cname=0)
******* Remove_Expired_RRs  Entered *******
name: tnral
```

```
  type=5 size=34
name: ftpoeral
  type=5 size=37
Remove_Expired_RRs rc=0
wanted(0x14dce060, 1, 1)  IN CNAME
make_rr(tnral.buddha.ral.ibm.com, 14dce060, bf39a, 458, 1) 34 zone 1 ttl 3600
match(0x14dce060, 1, 24) 1, 5
finddata: added 1 class 1 type 1 RRs
req: nlookup(tnral.ralplex1.buddha.ral.ibm.com) id 1 type=1 class=1
req: found 'tnral.ralplex1.buddha.ral.ibm.com' as 'tnral.ralplex1.buddha.ral.ibm.com' (cname=1)
******* Remove_Expired_RRs  Entered *******
name: TNRAL
  type=1 size=4
  type=1 size=4
Remove_Expired_RRs rc=0
 Looking for WLM generated Address
make_rr(tnral.ralplex1.buddha.ral.ibm.com, 14e5b040, bf3c9, 411, 1) 4 zone 2 ttl 0
match(0x14e5af48, 1, 24) 1, 1
match(0x14e5b040, 1, 24) 1, 1
 Weight = 0
 Current Server Weight 0
 Initializing all Weights
req: foundname=2, count=1, founddata=1, cname=1
sort_response(1)
findns: np 0x14e5af80 'TNRAL'
match(0x14e5af48, 1, 6) 1, 1
match(0x14e5b040, 1, 6) 1, 1
match(0x14e5af48, 1, 2) 1, 1
match(0x14e5b040, 1, 2) 1, 1
findns: np 0x14e5a5a8 'ralplex1'
match(0x14e5a510, 1, 6) 1, 6
findns: SOA found
req: leaving (tnral.ralplex1.buddha.ral.ibm.com, rcode 0)
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
findns: np 0x14e5a5a8 'ralplex1'
match(0x14e5a510, 1, 2) 1, 6
match(0x14e5a5e8, 1, 2) 1, 2
match(0x14e5b0e8, 1, 2) 1, 1
match(0x14e5b120, 1, 2) 1, 1
findns: 1 NS's added for 'ralplex1'
make_rr(ralplex1.buddha.ral.ibm.com, 14e5a5e8, bf3d9, 395, 1) 35 zone 2 ttl 3600
match(0x14e5a510, 1, 24) 1, 6
match(0x14e5a5e8, 1, 24) 1, 2
match(0x14e5b0e8, 1, 24) 1, 1
match(0x14e5b120, 1, 24) 1, 1
free_nsp: mvs03a.ralplex1.buddha.ral.ibm.com rcnt 1
doaddinfo() addcount = 1
do additional "mvs03a.ralplex1.buddha.ral.ibm.com" (from "ralplex1.buddha.ral.ibm.com")
found it
match(0x14e5a640, 1, 5) 1, 1
match(0x14e5a640, 1, 5) 1, 1
match(0x14e5a640, 1, 1) 1, 1
make_rr(mvs03a.ralplex1.buddha.ral.ibm.com, 14e5a640, bf3ee, 374, 0) 4 zone 2 ttl 3600
match(0x14e5a640, 1, 24) 1, 1
addinfo: adding address data n = 16
ns_req: answer ->  192.168.221.88 .1105 fd=5 id=1 size=142 Local
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1
;; flags: qr aa rd ra; Ques: 1, Ans: 2, Auth: 1, Addit: 1
;; QUESTIONS:
;; tnral.buddha.ral.ibm.com, type = A, class = IN

;; ANSWERS:
tnral.buddha.ral.ibm.com. 3600 IN CNAME tnral.ralplex1.buddha.ral.ibm.com.
tnral.ralplex1.buddha.ral.ibm.com. 0 IN A 192.168.252.28
```

```
;; AUTHORITY RECORDS:
ralplex1.buddha.ral.ibm.com. 3600 IN NS mvs03a.ralplex1.buddha.ral.ibm.com.

;; ADDITIONAL RECORDS:
mvs03a.ralplex1.buddha.ral.ibm.com. 3600 IN A 192.168.250.3
```

Appendix H. Special Notices

This publication is intended to help implementers of OS/390 SecureWay Communications Server install and customize the product. The information in this publication is not intended as the specification of any programming interfaces that are provided by OS/390 SecureWay Communications Server. See the PUBLICATIONS section of the IBM Programming Announcement for OS/390 SecureWay Communications Server 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.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

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

AIX
CICS
DB2
ESCON
IP Channel Communications
OpenEdition
OS/390
SecureWay
VTAM
AnyNet
DATABASE 2
eNetwork
IBM
MVS
OS/2
Parallel Sysplex
System/390

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Java and HotJava are trademarks of Sun Microsystems, Incorporated.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Pentium, MMX, ProShare, LANDesk, and ActionMedia are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

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

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

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

Appendix I. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

I.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 267.

- *OS/390 eNetwork Communications Server V2R7 TCP/IP Implementation Guide Volume 1: Configuration and Routing*, SG24-5227
- *OS/390 eNetwork Communications Server V2R7 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
- *TCP/IP Tutorial and Technical Overview*, GG24-3376

I.2 Redbooks on CD-ROMs

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at http://www.redbooks.ibm.com/ for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr Format)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037

I.3 Other Publications

These publications are also relevant as further information sources:

- *OS/390 eNetwork Communications Server: IP Planning and Migration Guide*, SC31-8512
- *OS/390 eNetwork Communications Server: IP Configuration*, SC31-8513
- *OS/390 eNetwork Communications Server IP Application Programming Interface Guide*, SC31-8516 (available only on CD-ROM SK2T-6700-12)
- *OS/390 UNIX Systems Services User's Guide*, SC28-1891
- *OS/390 V2R4 Parallel Sysplex Overview: Introducing Data Sharing and Parallelism in a Sysplex*, GC28-1860
- *OS/390 V2R6 MVS Planning: Workload Management*, GC28-1761-05
- *OS/390 V2R4 C/C++ Programming Guide*, SC09-2362

- *OS/390 V1R1 MVS Programming: Workload Management Services*, GC28-1773
- *DNS and BIND* by Paul Albitz and Cricket Liu (O'Reilly & Associates, Inc., 1997), SR23-8771

I.4 Web Sites

The following Web sites contain useful information relevant to the subjects discussed in this book:

- www.ietf.org/rfc/ contains the details of every Request for Comments submitted to the IETF.
- www.software.ibm.com/network/commserver/about/csos390.html is the CS for OS/390 home page.
- www.s390.ibm.com/pso is the Parallel Sysplex home page.

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** http://www.redbooks.ibm.com/

  Search for, view, download, or order hardcopy/CD-ROMs 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 redbook fax order form to:

In United States:	e-mail address: usib6fpl@ibmmail.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 Redbook 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.

List of Abbreviations

AIX	Advanced Interactive Executive
APAR	Authorized Programming Analysis Report
APF	Authorized Program Facility
API	Application Program Interface
APPN	Advanced Peer-to-Peer Networking
ARP	Address Resolution Protocol
ASCII	American national Standard Code for Information Interchange
ATM	Asynchronous Transfer Mode
BIND	Berkeley Internet Name Domain
BSD	Berkeley Software Distribution
CICS	Customer Information Control System
CPU	Central Processing Unit
CS for OS/390	OS/390 SecureWay Communications Server
CSM	Communications Storage Manager
DASD	Direct Access Storage Device
DB2	DATABASE 2
DD	Data Definition
DDNS	Dynamic Domain Name System
DHCP	Dynamic Host Configuration Protocol
DLC	Data Link Control
DNS	Domain Name System (Server)
ESCON	Enterprise Systems CONnection
FTP	File Transfer Protocol
HFS	Hierarchical File System
HPDT	High-Performance Data Transfer
HTTP	HyperText Transfer Protocol
I/O	Input/Output
IBM	International Business Machines Corporation
ICMP	Internet Control Message Protocol
ICP	Interconnect Controller Program
IGP	Interior Gateway Protocol
IP	Internet Protocol
ITSO	International Technical Support Organization
JCL	Job Control Language
LAN	Local Area Network
LCS	LAN Channel Station
LE	Language Environment
LPAR	Logical PARtition
MAC	Medium Access Control
MPC	MultiPath Channel
MQ	Message Queueing
MTU	Maximum Transmission Unit
MVS	Multiple Virtual Storage
NDR	Network DispatcheR
NNTP	Network News Transfer Protocol
OE	OpenEdition
OS/2	Operating System/2
OS/390	Operating System/390
OSA	Open Systems Adapter
OSPF	Open Shortest Path First
PC	Personal Computer
PID	Process IDentifier
POP	Post Office Protocol
PTF	Programming Temporary Fix
REXX	REstructured eXtended eXecutor
RFC	Request For Comments
RIP	Routing Information Protocol
SDSF	Spool Display and Search Facility
SMTP	Simple Mail Transfer Protocol
SNA	Systems Network Architecture
SOA	Start Of Authority
TCP	Transmission Control Protocol
TN3270	Telnet/3270
TRL	Transport Resource List

TRLE Transport Resource List Element

TSO Time Sharing Option

TTL Time To Live

UDP User Datagram Protocol

USS UNIX System Services

VIPA Virtual Internet Protocol Address

VPN Virtual Private Network

VTAM Virtual Telecommunications Access Method

WAN Wide Area Network

WLM WorkLoad Manager

XCF Cross-system Coupling Facility

Index

Special Characters

Numerics

A

B

C

D

E

F

S

T

U

V

W

X

Z

ITSO Redbook Evaluation

TCP/IP in a Sysplex
SG24-5235-01

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com/
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
__Customer __Business Partner __Solution Developer __IBM employee
__None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes____ No____

If no, please explain:

_____

_____

_____

_____

What other redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: **(THANK YOU FOR YOUR FEEDBACK!)**

_____

_____

_____

_____

_____

SG24-5235-01
Printed in the U.S.A.

SG24-5235-01