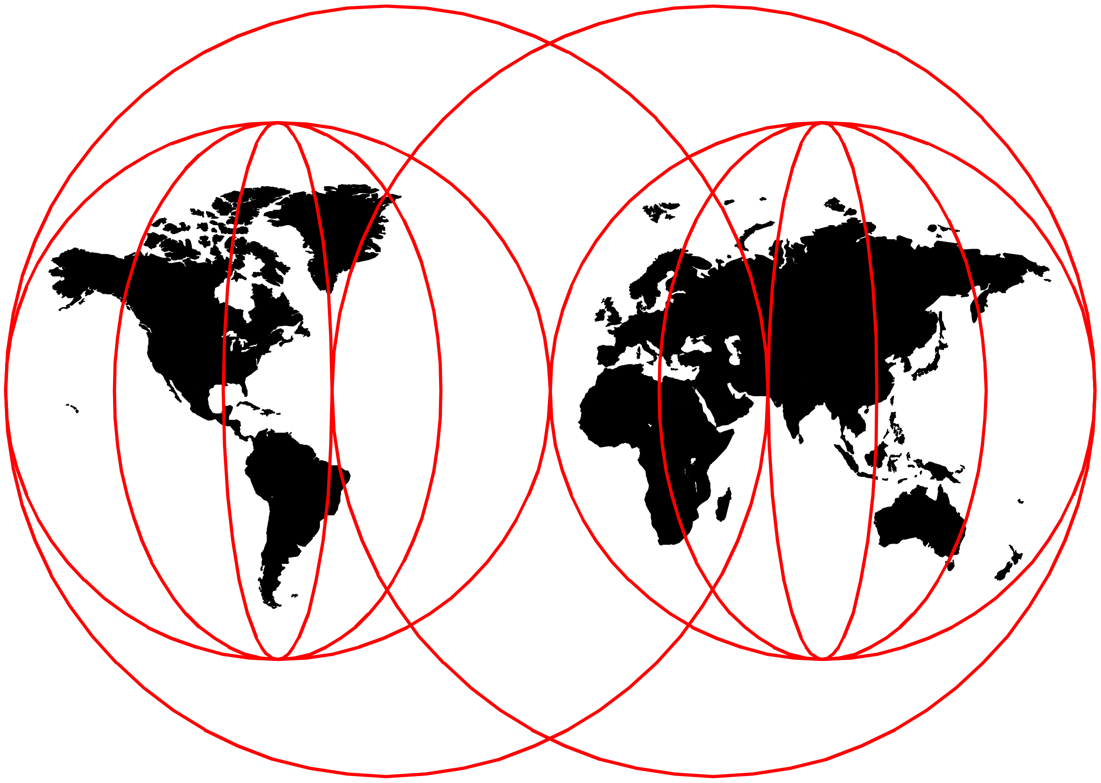
IBM WebSphere Performance Pack: Load Balancing with IBM SecureWay Network Dispatcher

Marco Pistoia, Corinne Letilley



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-5858-00

International Technical Support Organization

SG24-5858-00

IBM WebSphere Performance Pack: Load Balancing with IBM SecureWay Network Dispatcher

October 1999

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix D, “Special Notices” on page 413.

First Edition (October 1999)

This edition applies to IBM SecureWay Network Dispatcher, the Load Balancing component of WebSphere Performance Pack Version 2, for use with the AIX, Solaris, and Windows NT operating systems.

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

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Preface

IBM WebSphere Performance Pack is Web infrastructure software that addresses the scalability, reliability and performance needs of e-business applications in both local and geographically distributed environments. Its functions incorporate leading-edge and robust caching, file management and load balancing, that together compensate for the inherent weakness of the Internet to support critical business applications and expectations.

This redbook will give you a clear understanding of the features of IBM SecureWay Network Dispatcher, the Load Balancing component of IBM WebSphere Performance Pack. It shows how to plan for, install, configure, use, tune and troubleshoot this component and offers specific implementation examples. Moreover, it helps explain how to build complex scenarios that involve all the components of IBM WebSphere Performance Pack, to give you a better understanding of the technologies involved.

Note that this publication was written in conjunction with two other volumes about WebSphere: *IBM WebSphere Performance Pack: Web Content Management with IBM AFS Enterprise File System*, SG24-5857, and *IBM WebSphere Performance Pack: Caching and Filtering with IBM Web Traffic Express*, SG24-5859. To realize the most benefit, all three volumes should be obtained.

The Team That Wrote This Redbook

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

The leader of this project was Marco Pistoia.

Marco Pistoia is an Advisory Software Engineer, working as a project leader at the International Technical Support Organization Raleigh Center. He writes extensively and teaches IBM classes worldwide on all areas of the e-business Application Framework, WebSphere, Java and Internet security. Marco holds a Master of Science degree with honors in Pure Mathematics from the University of Rome. Before joining the ITSO, he was a System Engineer in IBM Italy. He received an Outstanding Technical Achievement Award in 1996.

Corinne Letilley is an Advisory Availability Systems Specialist - AIX and RS/6000 Systems with IBM Global Services Canada Ltd. She has been working with AIX and RS/6000 since the product's introduction in the early 1990s. She holds a degree in Commerce with a major in Computer Science from the University of Saskatchewan. Her areas of expertise include computer graphics as well as the IBM eBusiness product set.

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

Poh Yee Tiong
IBM Singapore

Steve Roma, Rick Schenck, Jerry Gschwind, Jeremy Noonan, Blake Corbitt, Susan Hanis, Andy Dingsor
IBM Research Triangle Park, North Carolina

Shawn Walsh, Jorge Ferrari, Tim Kearby, Margaret Ticknor, Pat Donleycott, Tate Renner, Linda Robinson, Gail Christensen
International Technical Support Organization, Raleigh Center

Vincenzo Iovine, Stefano Pischedda
IBM SEMEA Sud, Italy

Karen Gelveles
IBM Boca Raton

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 425 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`

Part 1. SecureWay Network Dispatcher Usage and Administration

Chapter 1. IBM WebSphere Performance Pack Concepts

IBM WebSphere Performance Pack is Web infrastructure software that addresses the scalability, reliability and performance needs of e-business applications in both local and geographically distributed environments. Its functions incorporate leading-edge and robust caching, file management and load balancing, that together compensate for the inherent weakness of the Internet to support critical business applications and expectations.

IBM WebSphere Performance Pack has been developed using IBM's extensive experience with very demanding Web sites.

IBM WebSphere Performance Pack is composed of three main components, which permit you to reduce Web server congestion, increase content availability and improve Web server performance:

1. **File sharing**

   The file-sharing component, known as IBM AFS Enterprise File System (AFS), is an enterprise file system that enables cooperating hosts (clients and servers) to efficiently share file system resources across both local area networks (LANs) and wide area networks (WANs). It provides non-disruptive real-time replication of information across multiple servers, which guarantees data consistency, availability, global stability and administrative efficiency, necessary by large distributed Web sites or by Web sites with volatile content requiring considerable administrative effort to maintain content links and URLs to file I/O mapping.

2. **Caching and filtering**

   The caching and filtering component, known as IBM Web Traffic Express (WTE), is a caching proxy server that provides highly scalable caching and filtering functions associated with receiving requests and serving URLs. With tunable caching capable of supporting high cache hit rates, this component can reduce bandwidth costs and provide more consistent rapid customer response times.

3. **Load balancing**

   The load balancing component, known as IBM SecureWay Network Dispatcher (ND), is a server that is able to dynamically monitor and balance TCP servers and applications in real time. The main advantage of the load balancing component is that it allows heavily accessed Web sites to increase capacity, since multiple TCP servers can be dynamically linked in a single entity that appears in the network as a single logical server.

WebSphere Performance Pack V2 offers a fourth component named Common Configuration. This new feature allows for centralized configuration of the main components and can be run remotely on a separate machine connected to the network.

The WebSphere Performance Pack components, which were previously unavailable in a single Internet software offering, can increase the scalability, availability and reliability of your Web site while reducing infrastructure costs.

Installation procedures permit selection of which components to install, and specification of on which machine(s) the selected component(s) should be located. Subject to installation needs and operating platforms, components can coexist on a single machine or can be distributed over multiple machines.

IBM WebSphere Performance Pack Version 2 is supported on the following platforms:

- Any IBM RISC/6000-based machine running IBM AIX 4.2.1 or later and Java Runtime Environment (JRE)[1] 1.1.6 or later
- Any SPARC workstation running Sun Solaris 2.6 or later and JRE 1.1.6 or later – JRE 1.1.7B or later is required to use ND
- Any Intel x86 PC running Microsoft Windows NT[2] 4.0 and JRE 1.1.6 or later – JRE 1.1.7B is required to use ND

If you are planning to use the Common Configuration, then you need the Java Development Kit (JDK)[3] 1.1.6 or higher and a Java 1.1-enabled Web browser, such as:

- Netscape Communicator V4.08 or higher
- Microsoft Internet Explorer V4.01 with the fix pack, or higher
- Sun HotJava V1.1 or higher

1.1 AFS Enterprise File System in Distributed Computing Environments

AFS has provided scalable file administration and file sharing for large enterprises for many years, based upon its use of a virtual name space to make naming and logical directory structures of files independent of their physical location. AFS clients and AFS servers are used to establish this virtual name space capability. In typical local area network (LAN) file systems, this is achieved by installing AFS clients on user workstations, communicating with an AFS server that manages the I/O operations associated with the actual files. In a Web site, the AFS clients can be installed on HTTP servers to reduce the administrative effort associated with maintaining URL to file I/O mapping relationships. In addition, HTTP servers that are simultaneously AFS clients can significantly increase the connectivity capacity to Web server content and can provide local and geographically distributed access efficiency.

AFS is a central and scalable file system:

- It is *central* because AFS brings together all of the files within the file system into a single name space. Every AFS user shares this same name space, making all AFS files easily available from any AFS machines. With AFS, the name of a file is independent of both the file's and the user's physical location, contributing to ease of file sharing and resource management.
- It is *scalable* because AFS is able to manage a very large number of files, spread across many geographical locations. When remote files, residing on AFS servers, are accessed by remote AFS clients, they are cached on the client machines to improve performance. This makes remote working across global distances feasible, since it is possible to access your own files from sites many thousands of miles away as if they were local.

[1] You can download the JRE from `http://www.javasoft.com`.
[2] IBM WebSphere Performance Pack can be installed on either Windows NT Server or Workstation V4.0. However, Windows NT Server is required for PCs running the ND functions Interactive Session Support (ISS) Nameserver and Observers.
[3] You can download the JDK from `http://www.javasoft.com`.

Both small and large-scale distributed environments benefit from AFS mechanisms to reduce server and network load:

- AFS caches data on client machines to reduce subsequent data requests directed at file servers, substantially reducing network and server loads. Servers keep track of data given to clients through *callbacks*, guaranteeing cache consistency without constant queries to the server to see if the file has changed. It is important to underline that AFS also allows disk cache, not just memory cache. This is a key advantage of AFS over other shared file systems.
- The AFS remote procedure call (RPC) reads and writes data to an RPC stream, further improving the efficiency of data transfer across a local or wide area network.

Extended security is guaranteed through Kerberos authentication and access control lists (ACLs). AFS Kerberos-based authentication requires that users prove their identities before accessing network services. Once authenticated, AFS access control lists give individual users or groups of users varying levels of permission to perform operations on the files in a directory.

The AFS component also offers replication techniques for file system reliability. Multiple copies of frequently accessed (but infrequently changed) data are replicated on multiple file servers within a cell. When accessing this information, a client will choose among the available servers that house replicas. If one server is unavailable or unreachable, the client will go to another server. Replication also reduces the load on any particular server by placing frequently accessed information on multiple servers.

Moreover, management utilities are provided to ease the load of system administrators in growing environments. Backup, reconfiguration and routine maintenance are all done without any system down time. Files remain available to users during these operations. This is done by creating online clones of volumes.

AFS commands are RPC-based. Administrative commands can be issued by any authenticated administrator from any client workstation. System databases track data location information, authentication information and protection groups. These databases are replicated on multiple servers, and are dynamically updated as information changes. Server processes accomplish many tasks automatically, such as restarting servers, tracking file locations and updating file servers with new binaries and configuration files.

1.2 Caching and Filtering to Manage Internet Traffic and Bandwidth Demand

WTE, the caching and filtering component of IBM WebSphere Performance Pack, is both a caching proxy server and a content filter. The advanced caching of this component minimizes network bandwidth and ensures that end users spend less time when retrieving the same content multiple times.

This component acts as a gateway for multiple clients and performs basic Web server duties, such as receiving requests and serving URLs.

A traditional proxy server receives a request for a URL from a client and it forwards the request to the destination content server. WTE does something more; it can save or cache the Web documents it retrieves, and serve subsequent requests for those documents from its local cache. The client gets the requested information faster and network bandwidth is reduced.

This component of IBM WebSphere Performance Pack also offers other key features of advanced caching, such as:

- The ability to handle very large caches
- An option to automatically refresh the cache with the most frequently accessed pages
- The possibility to cache even those pages where the header information says to fetch them every time
- Configurable daily garbage collection, to improve server performance and ensure cache maintenance
- Remote Cache Access (RCA), a new function – available for the first time with IBM WebSphere Performance Pack V2 – that allows multiple WTE machines to share the same cache by using a distributed file system, such as AFS, therefore reducing redundancy of cached content

Moreover, WTE allows you to set content filtering at the proxy server level, rather than or in addition to the browser level, where content filtering could be easily compromised or over-ridden. This way, offensive contents will not be displayed on the client’s browser, depending on the parameters used in the configuration. Content filtering in WTE can use:

- Platform for Internet Content Selection (PICS) rules guiding use of rating labels - such as Recreational Software Advisory Council on the Internet (RSACi) criteria for inappropriate language, nudity or violence - placed in HTML or HTTP headers or third-party content rating label distributions
- Lists of URLs/sites for which access is to be blocked
- APIs for filtering applications

1.3 Load Balancing and Server Monitoring Capabilities

The Internet has grown so rapidly over the last few years, that you are probably looking for a way to handle your company's share of that traffic. If this growth is not properly handled, users get slow response or refused connections, creating an unsatisfactory user experience which may cause the user never to visit your site again. Internet sites can become unstable or even fail under critical load conditions. What is needed is a solution that balances the load effectively and protects the user from these bad experiences.

ND, the load balancing component of IBM WebSphere Performance Pack, has been developed to address these limitations and provide customers with advanced functions to meet their site’s scalability and availability needs. It consists of three functions: the Dispatcher, Interactive Session Support (ISS) and the new Content Based Routing (CBR) function, available for the first time with IBM WebSphere Performance Pack V2. These three functions can be deployed separately or together in various configurations to suit a wide variety of customer application requirements:

- You can use the Dispatcher function to balance the load on the server within a local area network or wide area network using a number of weights and measurements that are set dynamically.
- ISS is a DNS-based load monitoring component (daemon) that can be installed on each of your servers. This group of daemons is called an *ISS cell*. One of the members of the cell becomes a spokesman for the load monitoring service.

  You can use the ISS function to balance the load on servers within a local area network or wide area network using a domain name server round-robin approach or a more advanced user-specified approach. ISS periodically monitors the level of activity on a group of servers and detects which server is the least heavily loaded.

  ISS provides an *observer* interface to enable other applications to use the load monitoring service. Observers watch the cell and initiate actions based on the load. Application servers with the ISS load monitor daemon installed can pass periodic load reports to the Dispatcher using the Dispatcher observer. The results of these reports can be factored into the load-balancing performed by Dispatcher.
- You can use the CBR function (which must be installed and configured to work together with WTE) to load balance traffic based on the content of a client's URL request. CBR also offers a Cookie Affinity feature that allows requests from a particular HTTP client session to be load balanced to the same server for a specified time period. The client session will maintain affinity for a particular server without relying on the IP address of the client.

These three components offer a high availability feature:

- The Dispatcher high availability feature involves the use of a secondary machine that monitors the main, or primary, machine and stands by to take over the task of load balancing, should the primary machine fail at any time. This feature is available on all the platforms where IBM WebSphere Performance Pack is supported, without using High Availability Cluster Multi-Processing (HACMP).
- In ISS high availability, all the nodes in a site work together to eliminate any single point of failure.
- Multiple CBR machines can be load balanced in turn using ND, therefore granting CBR high availability.

The high availability feature provided by the Dispatcher function can be successfully used even in other configurations, for example to guarantee firewall high availability.

1.4 Building Record-Breaking Web Sites

IBM WebSphere Performance Pack allows you to design several architectures to enhance the performance of your Web site. Figure 1 on page 8 offers an idea of the multiple configurations that can be obtained combining the WebSphere Performance Pack components:

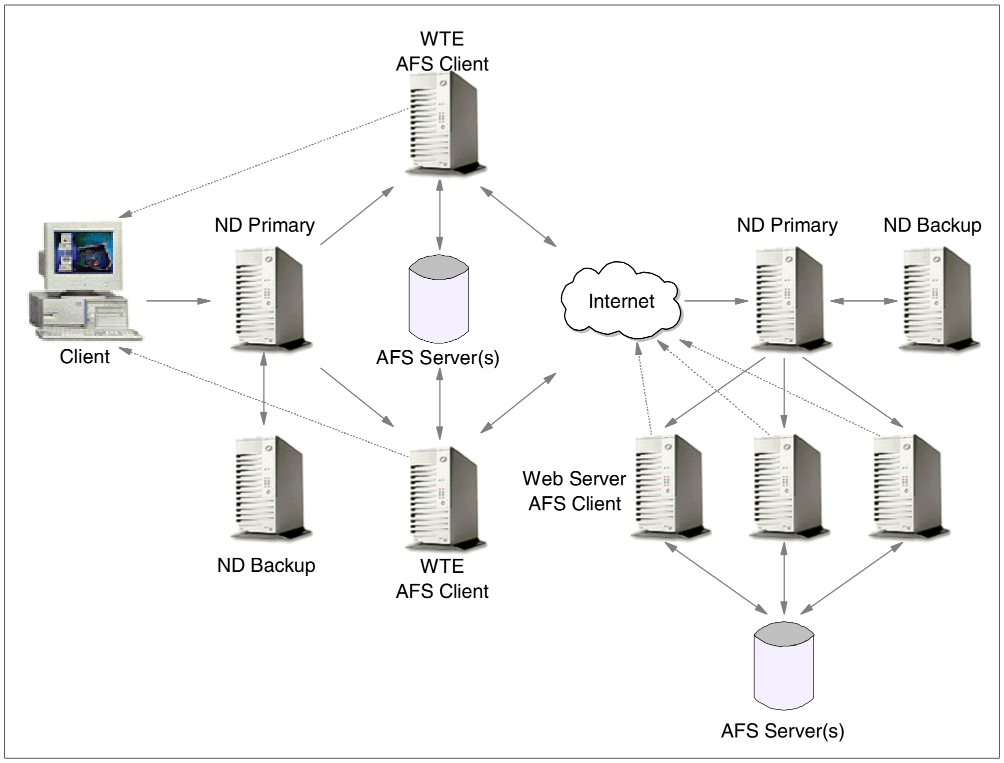


Figure 1. How to Implement a WebSphere Performance Pack Environment

- The WTE component minimizes response time and network bandwidth utilization by providing Web content caching. It also ensures reliable content filtering at the proxy server level. Multiple WTE servers can be load balanced.
- The ND component distributes the load between multiple clustered Web servers and WTE servers. As soon as a client request arrives, the load balancing machine uses sophisticated monitoring tools and then forwards the request to the least loaded server. High availability is provided by a backup Dispatcher machine, which monitors the state of the primary machine and takes over the primary machine if it fails.
- The clustered Web servers can share the same content by using AFS. This ensures scalability, high availability, reliable access to replicated data and an efficient security model for access and group management.

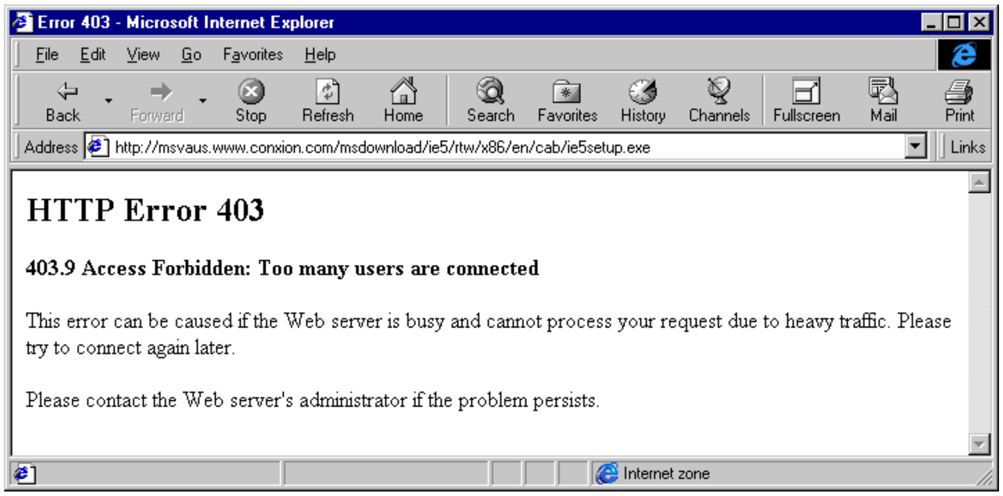
  AFS can also be integrated with RCA, so that clustered WTE servers can share the same Web cache therefore eliminating the need to fetch the same Web pages multiple times.
- The server selected by the ND machine can then respond directly to the client without any further involvement of the ND machine. Since there is no need for the server response to go back through the same physical path, a separate high-bandwidth connection can be used.

IBM used the WebSphere Performance Pack technology to create a scalable and reliable system that efficiently handled unprecedented traffic volumes. On February 17th, 1998, at 12:41 (Japan Standard Time), the official Web site of the Olympic Winter Games in Nagano made Internet history by logging a staggering 98,226 hits per minute. Less than a week later, a new all-time record was established with a peak load of more than 103,400 hits per minute, while still providing normal response time. The Internet site of the 1998 Nagano Olympic Winter Games is recognized by the Guinness Book of World Records.

In addition to the Winter Olympics site, IBM has built some of the largest Web sites in the world. For example, IBM hosted the Deep Blue chess match, the 1996 Olympic Games, the U.S. Open tennis tournament, Wimbledon, the Masters golf tournament, and the official Web site of the 1998 French Open tennis championship.

By using WebSphere Performance Pack on your Web site, you will have an efficient Web site, capable of providing fast responses. Your Web site will be able to handle very large amounts of simultaneous requests, without any major delay. Furthermore, the high availability features built into WebSphere Performance Pack will make the Web services available even when one or more of your server machines should unexpectedly fail.

The following figure shows an example of what your Web customers would not see if your Web site uses WebSphere Performance Pack:



http://msvaus.www.conxion.com/msdownload/ie5/rtw/x86/en/cab/ie5setup.exe

HTTP Error 403

403.9 Access Forbidden: Too many users are connected

This error can be caused if the Web server is busy and cannot process your request due to heavy traffic. Please try to connect again later.

Please contact the Web server's administrator if the problem persists.

Figure 2. Don't Let This Happen to You!

1.5 What Is New in Version 2

In addition to the functions already available in Version 1.0, described in the IBM redbook *IBM WebSphere Performance Pack Usage and Administration*, SG24-5233, IBM WebSphere Performance Pack enhancements in Version 2 include:

- Quality of service enhancements
- New ND functions
- New WTE functions
- New AFS functions
- New functions available by combining components
- Integrated configuration assistance

1.5.1 Quality of Service Enhancements

WebSphere Performance Pack now provides support for differentiated quality of service, as discussed in the following sections.

1.5.1.1 User Classes

User classes give you the ability to change the level of service depending on the identity of the user. Preferred customers would expect to get better service. In Version 1.0, ND provided this capability by allowing rules-based load-balancing based on the client IP address, and WTE allowed PICS filtering based on the client IP address or the user and group. In Version 2, ND and WTE add the ability for rules-based load-balancing based on HTTP headers including `Cookie`, `Referer`, and `User-Agent`.

Differentiated service based on user identity lets you determine what level of service should be provided. For example, a frequent buyer can be routed to a higher capacity server or given access to additional content (for example, sale information) not made available to an unknown customer. The information used to identify the user comes from cookies or header information in the HTTP request.

1.5.1.2 Service Classes

Server classes give you the ability to change the level of service depending on the information or application requested by the client. Services involving a purchase would get better service than requests for information. In Version 1.0, WTE provided this capability through its reverse proxy function by allowing requests to be redirected to different servers based on the contents of the URL. In Version 2, ND and WTE add the ability for rules-based load-balancing based on the protocol, host, and path portion of the URL.

Differentiated service based on the service requested lets you give preferential service to some requests. For example, you may want to give preferential service to a *proceed to checkout* request over a *search* request.

1.5.2 New ND Functions

ND has been enhanced to include several new features. These are discussed in the following sections.

1.5.2.1 Server Directed Affinity API

In previous releases, administrators could define a particular port to be configured as *sticky*, meaning all requests to that port remain with a particular physical server for a short period of time. When a client connects to a sticky port, an entry for that client’s IP address is made in the affinity table, and a time stamp is set. If a new connection from the same client arrives, and the time stamp has not expired, then the new connection is sent to the same server as before. This function is still available in Version 2, but it is no longer the only option.

Customers can now write their own software using the *Server Directed Affinity* (SDA) Application Programming Interface (API) to implement an SDA agent, which communicates with a listener in the Dispatcher. This software can then manipulate the Dispatcher affinity tables to:

- Query the contents
- Insert new records

- Remove records

1.5.2.2 Binary Logging and Statistics

Dispatcher now provides a log in binary format. The command line and the GUI have been enhanced to provide access to the binary log information. A sample Java program is provided to allow customers to access and manipulate the log.

1.5.2.3 Remote Administration

In the previous release, the Dispatcher administration GUI could only be run on the same machine where Dispatcher was installed. Now remote administration is possible, since the Dispatcher administration GUI can be run on a separate machine.

1.5.2.4 Authenticated Administration

Authentication is provided to make remote administration more secure. The communication between the ND server and an ND administration client is authenticated using a key pair.[4] These keys are generated when the ND server is started for the first time.

1.5.2.5 Wildcard Cluster

Wildcard cluster is the capability to define a cluster which will receive traffic which is not destined for this particular machine, or which is destined for this machine, but for an address which is not defined as a cluster. The traffic will be intercepted and routed to a default cluster. This feature also makes it easier to configure multiple aliases on the same Dispatcher machine to use the same port and server configuration.

1.5.2.6 Wildcard Port

Wildcard port is the capability to define default actions when no port matches a particular request. You can use this to create a load balancing configuration for traffic to any port that has not been explicitly defined in your Dispatcher configuration, for example in a firewall load balancing environment, or to discard requests for ports that have not been configured.

1.5.2.7 ISS GUI

The GUI for ISS is essentially an `isscontrol` command generator much like the Dispatcher GUI. In other words, the user makes a change in the GUI, and an isscontrol command is generated and issued. Asynchronously, the issd daemon tells the GUI that the configuration has changed and the GUI refreshes itself.

1.5.3 New WTE Functions

The enhancements to the WTE component of WebSphere Performance Pack are described in the following sections.

1.5.3.1 Transparent Proxy

Transparent proxying means that the client software is totally unaware of the existence of the intermediate proxy server. Normally, if a client browser uses a proxy server, then the browser must be configured to specify the address and port of the proxy server. This is no longer necessary with transparent proxy, in fact the client is unaware that an intermediate proxy is in the network.

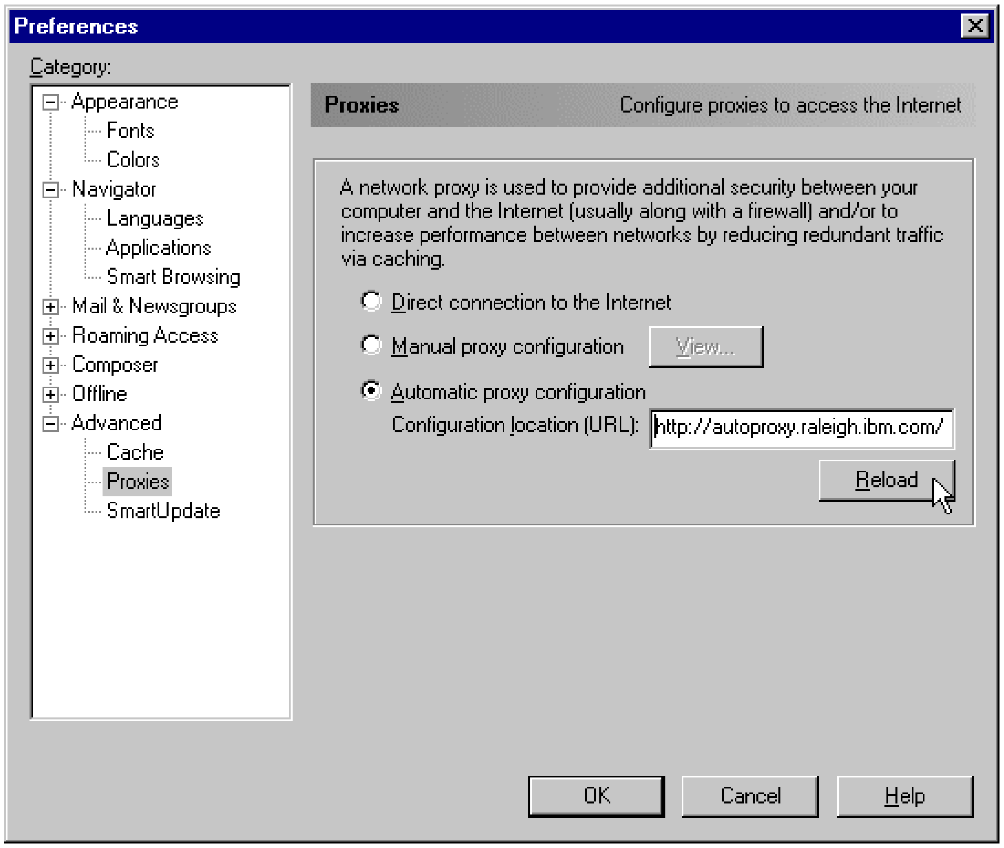
[4] A *key pair* is a matching pair of public and private keys, used for digital signatures and asymmetric encryption.

To use transparent proxy, the router, which may be a Dispatcher machine, is programmed to redirect requests to the WTE transparent proxy. WTE then intercepts all HTTP requests on port 80 that are targeted at some server out in the Internet. The request is parsed and processed, and may be satisfied from the transparent proxy's cache.

Note that in WebSphere Performance Pack V2, transparent proxy is only supported on the AIX platform and works for HTTP requests only.

1.5.3.2 Proxy Autoconfiguration Support

WTE now supports *automatic proxy configuration*, a feature of Netscape Navigator 2.0 (and later) and Microsoft Internet Explorer V 4.0 (and later). This feature provides a form of transparency, in that clients do not have to configure their browser to point to a specific proxy or SOCKS server, but to an automatic configuration file, as shown in the following figure:



A network proxy is used to provide additional security between your computer and the Internet (usually along with a firewall) and/or to increase performance between networks by reducing redundant traffic via caching.

( ) Direct connection to the Internet
( ) Manual proxy configuration
[✓ Automatic proxy configuration]
Configuration location (URL): http://autoproxy.raleigh.ibm.com/

Figure 3. Automatic Proxy Configuration in Netscape Navigator

This lets the system administrator modify the configuration with little impact to the clients, who update their automatic configuration files and are automatically directed to the new configuration. Server administrators can use this to reroute requests when servers are down, to balance workload, to send requests for specific URLs to specific proxies, or other reasons specific to their installation.

1.5.3.3 FTP Proxy Enhancements

The WTE FTP proxy code now includes FTP PUT capability, improved authentication to prompt for the user ID and password (instead of requiring it in the URL), and a configuration directive to allow the user to specify whether FTP URLs will be treated as relative URLs or absolute URLs.

1.5.3.4 SNMP Subagent and MIB Support

WTE provides a Simple Network Management Protocol (SNMP) management information base (MIB) and SNMP subagent so you can use any SNMP-capable network management system, such as Tivoli NetView or Tivoli Distributed Monitoring to monitor your proxy server's health, throughput, and activity. The MIB data describes the proxy server being managed, reflects current and recent server status, and provides server statistics.

1.5.3.5 Performance Improvements

The cache architecture has been restructured to map URLs onto the cache file system more efficiently. This speeds retrieval of cached objects, uses disk space more efficiently within the cache, and speeds cache garbage collection. The cache also uses write-behind techniques for greater throughput.

Additionally, WTE now includes caching of Domain Name System (DNS) server lookup results, which can improve response time and reduce network load.

1.5.3.6 HTTP 1.1-Compliant Proxy Server

WTE is now an HTTP 1.1-compliant proxy server. WTE identifies itself as an HTTP 1.1 server and sends HTTP 1.1 in the outbound flows to origin servers. Persistent connections are supported from the client to the proxy, and from the proxy to the origin server. HTTP 1.1 cache control headers are processed and used to determine if a Web document is able to be cached. WTE receives and processes chunked data sent by HTTP 1.1 origin servers; WTE will unchunk data before giving control to a data filter or transmogrifier plug-in.

WTE provides new directives that allow the administrator to override certain HTTP 1.1 cache control headers. For example, query strings – URLs with a question mark (`?`) in them – are not considered cacheable by the HTTP 1.1 protocol, but WTE provides a directive that allows the administrator to specify which query strings should be cached.

WTE also provides an aggressive caching directive that allows the administrator to override the `Cache-Control: no-cache` header in Web documents.

1.5.3.7 Customization Exits

Several enhancements have been made to the WTE API to simplify writing an application plug-in. New request steps (*exit points*) have been added: *transmogrifier* and *garbage collection* (GC) advisor.

The transmogrifier gives the application write access to the outgoing data stream while the GC advisor allows the plug-in to influence garbage collection decisions:

- The transmogrifier step is intended to be used by applications that wish to perform transformations on the HTTP response data stream. Examples include converting Adobe PDF files to HTML, converting high resolution images to lower resolution quality, or translating pages from one language to another. The transmogrifier step is an extension of the data filter step.

  The WTE enhancements allow the application to specify multiple transmogrifiers, thus allowing the application to have multiple plug-ins, each of them performing different transformations on the data.

WTE introduces a correlator mechanism that eases state maintenance in the plug-in; WTE now automatically determines the content length of the response data, and the application no longer has to buffer the data to determine the content length. WTE also makes HTTP header processing easier; the response headers can now be extracted and set using API variables.

- The GC Advisor step is called for each file in the cache during the garbage collection process and allows the application to influence which files are kept and which are discarded.

1.5.3.8 Tivoli Ready

WTE is Tivoli Ready, which means it can be managed through either the Tivoli Enterprise Console (TEC), or through Tivoli Global Enterprise Manager (GEM). Supported Tivoli configurations include managing IBM applications that are installed on Tivoli-managed nodes, PC-managed nodes and endpoints in a distributed environment.

With the purchase of an IBM software product that carries the Tivoli Ready logo, you have the ability to manage your IBM software products through the Tivoli Enterprise management products, allowing you to automatically discover, monitor, and inventory one or more Tivoli Ready applications.

This Tivoli Ready instrumentation, when configured, provides you with the ability to:

- Graphically view the health of WTE through Tivoli GEM 2.2, TEC 3.1 or higher consoles
- Inventory WTE using Tivoli Inventory Version 3.2

1.5.3.9 Variant Caching

Variant caching extends the capabilities of the transmogrifier and allows WTE applications to request that WTE cache another version (*variant*) of the original document retrieved from the Web. This need arises when a plug-in performs a transformation on the original page retrieved from the Web. For example, if a plug-in translates a page from English to Italian, it would be advantageous to be able to cache not only the original document but the variant, as well. In fact, transformations of Web data are very CPU-intensive and degrade the performance of the proxy server.

By caching the variant, the number of transformations required is decreased, which improves overall proxy caching performance. The WTE API provides new predefined functions, HTTPD_variant_lookup() and HTTPD_variant_insert(), to allow the application to find a variant that has already been cached, and to insert a new variant into the cache.

1.5.3.10 Enhanced Log Maintenance

To effectively manage the space requirements for the four logs (error, access, proxy access, and cache access) generated by WTE, two improved functions have been implemented. They are compression and purging:

- Compression can be specified for the logs by date. The logs are compressed and stored with no regard for storage constraints.
- Purging is done by date and size of each file. The maximum size for each log is set during configuration of WTE. When the logs are purged by date, if the

maximum size for a log is still exceeded, daily logs are removed until the maximum size is no longer exceeded.

1.5.3.11 Error Message Personalization

The error messages sent to the client's browser are kept in HTML pages, which lets you change or alter the message. You can use this to give additional information, give additional instructions, or include things such as logos.

1.5.3.12 Secure Request Filtering

Filtering for Secure Sockets Layer (SSL) requests is done by using PICS filtering on the non-secure home page of the URL and applying the receive or block decision to the secure request.

1.5.3.13 Configuration Enhancements

Configuring the server using the browser, local or remote, is improved and more user friendly. The configuration screen is divided into three areas: navigation, workspace, and header. Navigation between different areas of the configuration process is always available in the navigation portion of the screen with a click of the mouse. The forms to be filled in are displayed in the workspace. The header portion contains a Help button and a Restart Server button. The server can be restarted anytime with the Restart Server button. When the Help button is clicked, another instance of the browser that does not interfere with the form being used displays the help information. The help information is divided into three types: field definition of the form being viewed, task oriented, and server general help.

1.5.4 New AFS Enterprise File System Function

The AFS component of WebSphere Performance Pack now provides some new features.

1.5.4.1 AFS Server for Windows NT

WebSphere Performance Pack Version 1.0 included AFS client and server for AIX and Solaris, but only the client was included for Windows NT. WebSphere Performance Pack V2 includes AFS Version 3.5 for Windows NT, which offers AFS server for Windows NT. With this component, you can store AFS files and directories, and run processes that provide servers on a Windows NT machine.

1.5.4.2 AFS Control Center

The AFS Control Center is a set of Windows NT-based tools for managing AFS cells. The AFS Control Center includes a User Administration graphical user interface (GUI) for account management and a Server Manager GUI for volume management. The Control Center helps simplify AFS server and account administration by letting administrators manage entire environments from a single Windows NT workstation, for example, monitoring file server utilization, transferring collections of files across servers, performing load balancing, or managing AFS accounts.

1.5.5 New Functions Available by Combining Components

As we said, WebSphere Performance Pack is composed of three main components: ND, WTE and AFS. Although these components can be installed separately, they are not different products; especially in this new version of WebSphere Performance Pack, new functions have been provided to integrate these components together and supply services that were not previously available.

1.5.5.1 Content Based Routing

Content Based Routing (CBR) is a new ND function that provides load balancing enhancements to the reverse proxy capabilities of WTE. CBR works in conjunction with WTE, which must be installed on the same machine.

The same rules used for Dispatcher can be used with CBR to load-balance requests over different sets of servers based on the client IP address, the entire URL, the protocol portion of the URL, the host portion of the URL, the path portion of the URL, the Referer HTTP header, or the User-Agent HTTP header.

1.5.5.2 Peak Load Management

Peak load management is the ability to detect and react to sudden increases in activity. In Version 1.0, ND allowed rules-based routing to alter the load-balancing algorithm based on the current connection rate, the number of active connections, and the time of day. In Version 2, an advisor for Dispatcher enhances load management on WTE by preventing Dispatcher from sending new requests to a WTE node that is engaged in garbage collection or cache refresh.

1.5.5.3 Remote Cache Access

RCA provides a way to share cache content among proxies. In many scenarios, multiple proxies are deployed *near* (in network terms) each other. Typically, they will have a load balancer in front of them for load balancing and high availability reasons. Each proxy has its own cache, and if the cached data cannot be shared, this results in cache space being wasted as multiple copies of the same document are stored. Also, since each cache is smaller than the sum of all the disks, the cache hit rate is lower (due to the smaller cache size).

RCA is a new, powerful feature implemented in WTE, and not available in the previous release of IBM WebSphere Performance Pack. RCA allows multiple proxy servers to cooperate to form cache arrays. Using RCA, multiple proxy servers can distribute the cache contents across their combined, logical cache to improve hit rates and reduce redundancy of cached content.

WTE uses new caching algorithms and information-sharing technologies to enable an Internet service provider (ISP) to manage its servers more efficiently by storing information where it is more likely to be needed and delivering it more efficiently to customers. These enhancements reduce transmission costs and eliminate the need for ISPs to replicate information in redundant proxy servers.

Although RCA is a new feature of WTE, it is best used in conjunction with the other components of WebSphere Performance Pack. In fact, RCA enables multiple peer WTE proxy servers, load balanced by one ND machine, to share the contents of their caches utilizing a shared file system, such as AFS, DFS, NFS, or Windows NT file sharing. We recommend the use of AFS, since it offers nondisruptive real-time replication of information across multiple servers, data consistency, availability, global stability and data consistency.

1.5.5.4 Integrated Configuration Assistance

To facilitate proxy caching, the install program automatically sets up caching with WTE. WebSphere Performance Pack also provides wizards to make configuration easier. Invoked from a browser, these wizards can configure key

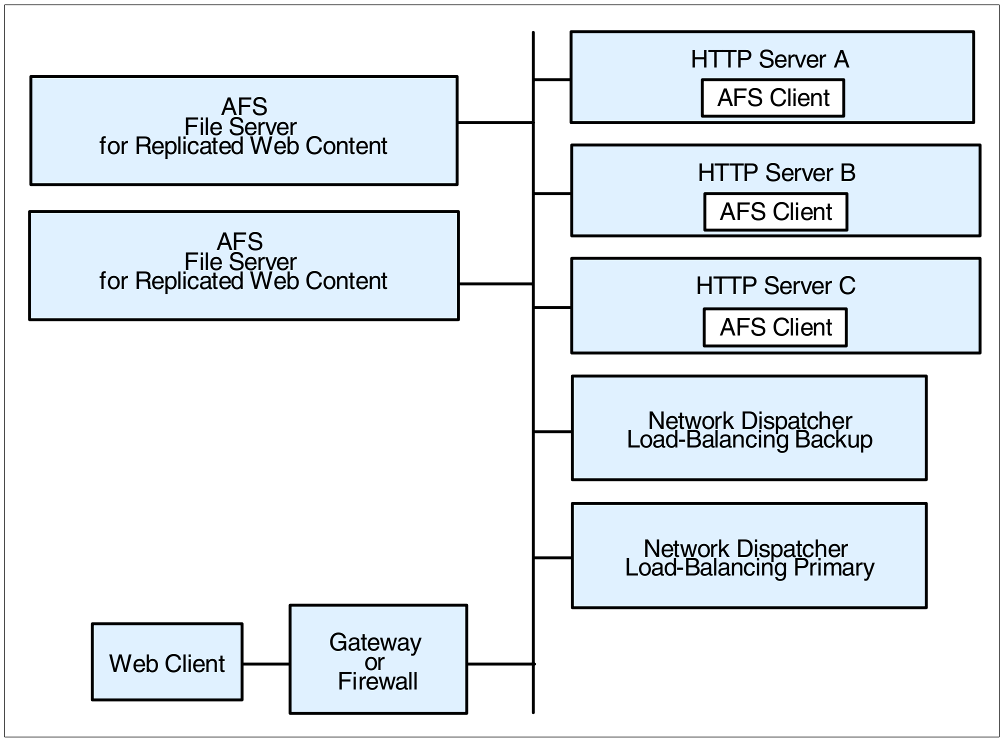
WebSphere Performance Pack scenarios that involve all of the WebSphere Performance Pack components.

1.6 Who Can Benefit

IBM WebSphere Performance Pack allows you to design and use multiple architectures. The scenarios described in this section provide specific examples of how various ISP implementations can benefit from the use of IBM WebSphere Performance Pack.

1.6.1 Content Hosting Internet Service Providers

Content hosting ISPs can use WebSphere Performance Pack to more efficiently support and distribute the content from their own Web sites, and to provide more response- and cost-effective access to other sites.



AFS File Server for Replicated Web Content
AFS File Server for Replicated Web Content
HTTP Server A
AFS Client
HTTP Server B
AFS Client
HTTP Server C
AFS Client
Network Dispatcher Load-Balancing Backup
Network Dispatcher Load-Balancing Primary
Web Client
Gateway or Firewall

Figure 4. Content Hosting Internet Service Providers

Within a content hosting *server farm*, the WebSphere Performance Pack components can be configured to provide high availability and accessibility as follows:

- Content for hosted Web sites can be distributed over multiple volumes using the AFS virtual name space to simplify administration of page content. Because all of the Web servers need to have equal access to the Web content, a shared file system is an obvious choice for manageability of the Web content. AFS is a superior file system in this environment because of its replication capabilities, which provide improved availability and scalability. The disk caching capability of the AFS client ensures that network accesses to the file server are minimized.

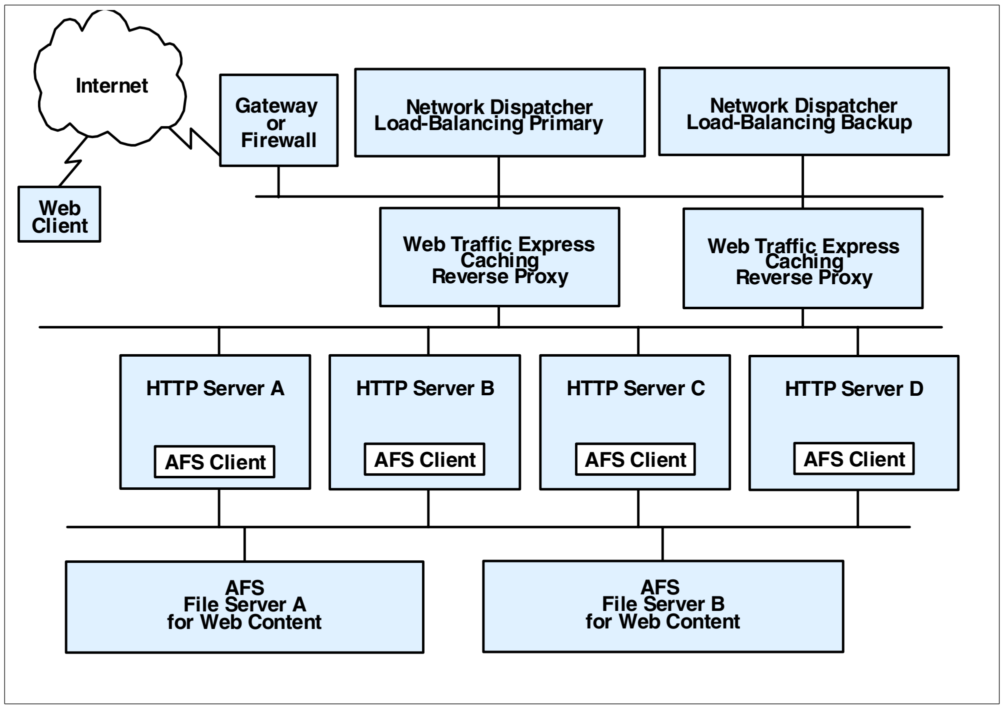
- Scalability of access to the content can be achieved by adding multiple HTTP servers that are simultaneously AFS clients to access AFS server content, and by using efficient load balancing to dispatch requests to the HTTP server with the best capacity to handle the requests. ND, the load balancing component of IBM WebSphere Performance Pack guarantees improved availability and scalability by allowing a farm of Web servers to provide a single Web site image to clients.
- High availability can be maintained by using AFS file replication capabilities, and by configuring a hot standby ND component.
- Proxy caching can be used to provide more responsive access to content from other sites, as well as to optimize backbone network traffic capacity.
- Both availability and performance may be further enhanced by geographically distributing HTTP servers with AFS clients, together with proxy caching for other Internet content closer to user access points, such as points of presence (POPs) and network access points (NAPs).

Thus, content hosting service providers or corporate webmasters can benefit from the non-disruptive replication and distribution capabilities of the file sharing functions, local and wide area load balancing, and proxy caching. Flexible configuration of these components can ensure that requests are directed to the most appropriate local or remote location, and can enable location outages or routine maintenance schedules to be handled without disrupting customers.

The IBM WebSphere Performance Pack components can be used in conjunction with firewalls and authentication gateways to provide secure access where desired, and the load balancing function of WebSphere Performance Pack can also be used to scale these capabilities.

1.6.2 Corporate Web Sites and Content Aggregators

Use of IBM WebSphere Performance Pack by corporate Web sites and content aggregators is similar to that of content hosting ISPs.



Internet
Web Client
Gateway or Firewall
Network Dispatcher Load-Balancing Primary
Network Dispatcher Load-Balancing Backup
Web Traffic Express Caching Reverse Proxy
Web Traffic Express Caching Reverse Proxy
HTTP Server A
AFS Client
HTTP Server B
AFS Client
HTTP Server C
AFS Client
HTTP Server D
AFS Client
AFS File Server A for Web Content
AFS File Server B for Web Content

Figure 5. Corporate Web Sites and Content Aggregators

In many cases, corporate Web sites and content aggregators need to maintain a demilitarized zone (DMZ) to ensure that access to Web content by employees, business partners, or customers does not expose internal computer resources to unauthorized users or hackers. However, firewalls cannot be deployed between AFS clients and their servers. In such instances, AFS replication can be used to establish read-only AFS servers within the DMZ. Here multiple AFS clients and the load balancing ND function can be used to provide the degree of scalability necessary to satisfy users. In addition, WTE caching and filtering proxy servers may be deployed on the same machines or on separate systems to filter or optimize access from within the corporation to external Web sites.

Many corporate Web sites are located at head office or regional locations, while branch offices or business partners may need frequent access to the content. Most offices of this type have relatively low line speed (somewhat less than 1.5 Mbps) network access to the regional or corporate sites. Relatively few users can, with concurrent usage, use all the available bandwidth with resulting erratic response times. At these locations HTTP servers with AFS clients combined with general purpose caching can provide more consistent user response time.

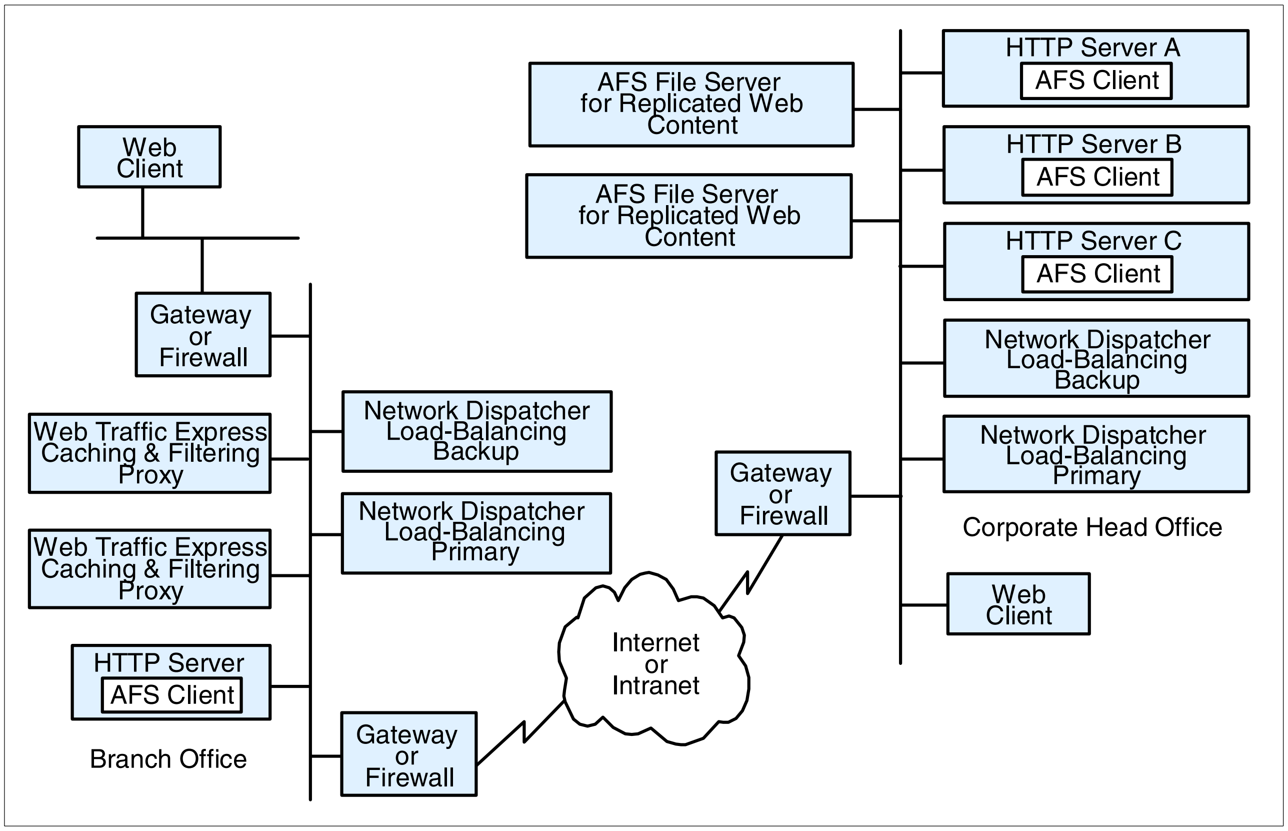

Web Client
Gateway or Firewall
Web Traffic Express Caching & Filtering Proxy
Web Traffic Express Caching & Filtering Proxy
HTTP Server
AFS Client
Branch Office
Network Dispatcher Load-Balancing Backup
Network Dispatcher Load-Balancing Primary
Gateway or Firewall
Internet or Intranet
AFS File Server for Replicated Web Content
AFS File Server for Replicated Web Content
Gateway or Firewall
HTTP Server A
AFS Client
HTTP Server B
AFS Client
HTTP Server C
AFS Client
Network Dispatcher Load-Balancing Backup
Network Dispatcher Load-Balancing Primary
Corporate Head Office
Web Client

Figure 6. Head Office - Branch Office

Some industries also have small branch offices in addition to larger branch or regional offices. A simple deployment of WTE caching and filtering on a single platform, eventually combined with a firewall, is probably sufficient to provide more consistent response time for employees in the small branch offices.[5] In the larger branches or regional offices, it may be desirable to have more than one WTE proxy server and to use the load balancing ND functionality to provide better resource management.

Two approaches can be used to reduce redundant page storage and to address the effectiveness of caching in these locations:

1. Hierarchical caching

   Optimize a primary (initial) cache for higher page hit rate by favoring more files of a smaller size, and a hierarchical cache for higher hit byte rate by favoring fewer files of a larger size.

2. RCA

   Configure RCA together with the load balancing function of ND and with AFS shared file storage to reduce the index size of each proxy and to increase the scalability of the caching storage.

[5] IBM has recently released a new product in the WebSphere family, called WebSphere Performance Pack WebSphere Cache Manager. This product is a caching and filtering proxy server obtained by the same code base as WTE. Cache Manager is available on Windows NT and Linux. Only a subset of the functions available in WTE are implemented in Cache Manager. For this reason, Cache Manager is particularly indicated for small companies or for small branch offices of large industries.

1.6.3 Corporate Headquarters Buildings or Large Campuses

On large campuses or in corporate headquarter buildings, the size of the campus or number of personnel frequently lead to the creation of smaller local area networks (LANs) interconnected by routers and a backbone LAN. Busy LAN servers combined with increasing use of Web server applications can result in congestion on the backbone LAN segments. This can be reduced by installing AFS client-enabled HTTP servers and general proxy caching on user LAN segments. The AFS clients can provide caching for corporate Web content, while the WTE proxy server can provide caching and filtering for external Internet access.

Design considerations for the smaller LAN segments are similar to those for small branch offices discussed earlier.

1.6.4 Backbone Internet Service Providers

Backbone ISPs typically provide co-location and/or peering services for other ISPs in addition to content hosting for large national or international corporations. In many cases they may provide *virtual ISP services* for other service providers such as content hosting ISPs. Backbone ISP customers are increasingly demanding both high availability and differentiated service levels, and backbone ISPs are responding by enhancing their Internet infrastructures. Features demanded by backbone ISPs include:

- Load balancing for a variety of traffic (for example, mail and FTP in addition to Web traffic)

  In addition, requests are made for traffic balancing management and high availability for authentication servers and management systems.

- Proxy caching

  To minimize the effects of *hot potato* routing and Web traffic *surges*, backbone ISPs are typically installing highly scalable caches at major peering points and network interconnections points such as network access points (NAPs).

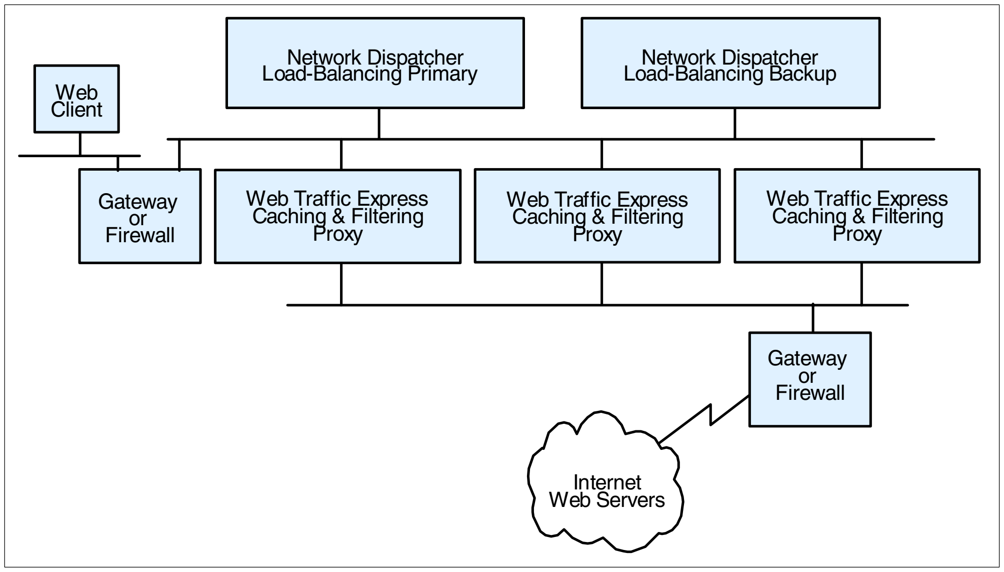

Network Dispatcher Load-Balancing Primary
Network Dispatcher Load-Balancing Backup
Web Client
Gateway or Firewall
Web Traffic Express Caching & Filtering Proxy
Web Traffic Express Caching & Filtering Proxy
Web Traffic Express Caching & Filtering Proxy
Gateway or Firewall
Internet Web Servers

Figure 7. Hierarchical Caching

International ISPs in particular need caching to reduce the costs associated with trans-oceanic links. Such installations can make very effective use of the RCA feature of WTE, a variation of the traditional caching function that when used together with load balancing and shared file storage can dramatically reduce redundancy of page storage.

Typical configurations for backbone ISPs would include load balancing for authentication gateways (such as authentication servers and subscriber management application servers) as well as for WTE caching proxy servers. Because of the amount of traffic and the desire for high availability, such caching servers would likely use the RCA feature and would thus also be configured as AFS clients. Therefore, there would also be an AFS server with the file content.

1.6.5 Access Internet Service Providers

Access ISPs need to provide both more consistent response time to their customers and to conserve their backbone network link and access charges. Thus caching at POPs would address these needs. These configurations would be similar to those for backbone ISP solutions.

Because access service providers target many of the small- to medium-size business customers, there is also an opportunity for them to create revenue producing services, by deploying smaller caching devices on customer premises but configuring and managing them as an ISP service. For this scenario, the caching would look much as it does for corporate branch offices.

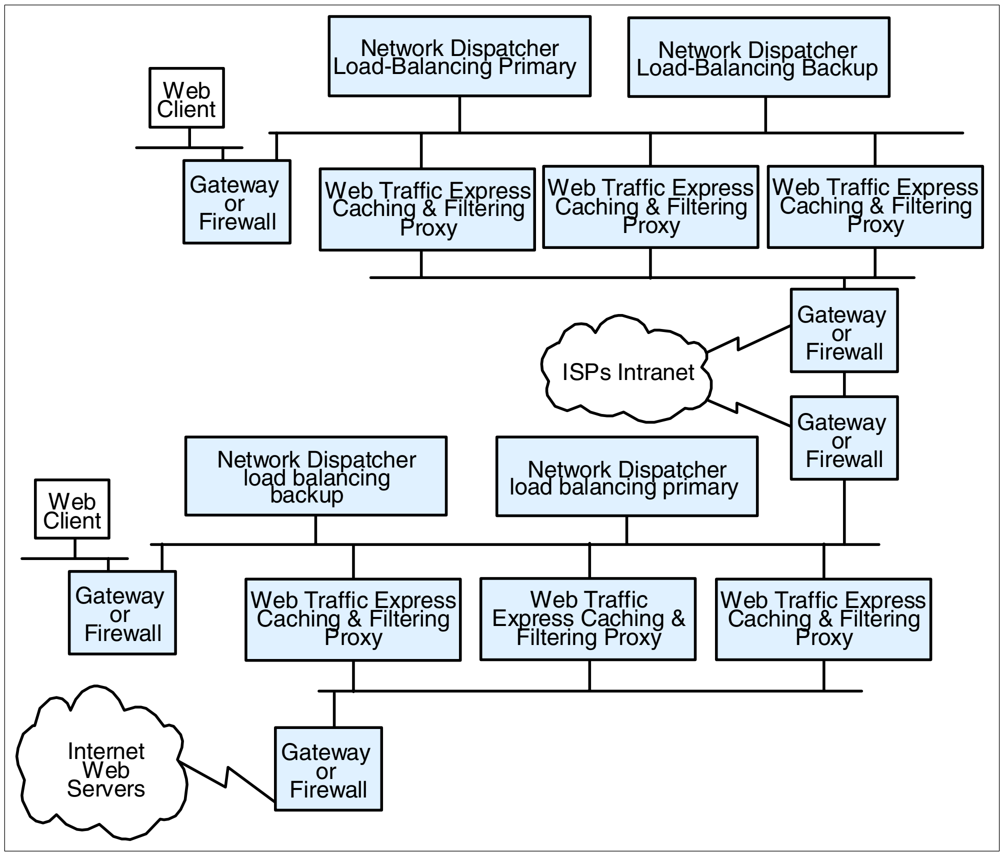

Network Dispatcher Load-Balancing Primary
Network Dispatcher Load-Balancing Backup
Web Client
Gateway or Firewall
Web Traffic Express Caching & Filtering Proxy
Web Traffic Express Caching & Filtering Proxy
Web Traffic Express Caching & Filtering Proxy
Gateway or Firewall
ISPs Intranet
Gateway or Firewall
Network Dispatcher load balancing backup
Network Dispatcher load balancing primary
Web Client
Gateway or Firewall
Web Traffic Express Caching & Filtering Proxy
Web Traffic Express Caching & Filtering Proxy
Web Traffic Express Caching & Filtering Proxy
Internet Web Servers
Gateway or Firewall

Figure 8. Access Internet Service Providers

1.6.6 Access ISPs with Subscriber Home Page Hosting

Access ISPs providing subscriber home page hosting typically do so on multi-homed servers, allowing subscribers to have their own domains. Such servers may have frequent changes, requiring the ISP to dedicate considerable time to administering the servers from a directory and backup perspective. For example, if the ISP is successful in recruiting new users, therefore having to expand the server capacity, it may involve reorganizing the various servers to spread the load and to allow room for individual sites to expand. The file management services provided by IBM WebSphere Performance Pack enable the ISP to reduce this maintenance workload, and to provide high availability replication for subscribers willing to pay for continuous availability. They also allow the ISP to scale up the site capacity with additional AFS clients, without the need to reorganize the underlying Web servers. ND load balancing can ensure that despite the growth users get rapid response time. In addition, when combined with subscriber management routings, to intelligently assign DHCP IP addresses, it becomes possible to offer differentiated classes of services for customers paying a premium.

1.7 Other IBM WebSphere Offerings

This section describes the other IBM WebSphere products that can interact with IBM WebSphere Performance Pack in an e-business Application Framework environment: IBM WebSphere Application Server and IBM WebSphere Studio.

Notice that IBM WebSphere Performance Pack can be integrated with a number of other IBM e-business products, such as IBM eNetwork Firewall, IBM HTTP Server and Lotus Domino Go Webserver. In this redbook, there are several examples and scenarios where all these e-business products are used together to create a powerful and secure Web site.

1.7.1 IBM WebSphere Application Server

IBM WebSphere Application Server lets you achieve your *Write Once, Run Anywhere* goal for Java servlet development. The product consists of a Java-based servlet engine that is independent of both your Web server and its underlying operating system.

WebSphere Application Server offers a choice of server plug-ins that are compatible with the most popular server APIs. The supported Web servers are:

- IBM HTTP Server
- Apache Server
- Domino
- Lotus Domino Go Webserver
- Netscape Enterprise Server
- Netscape FastTrack Server
- Microsoft Internet Information Server

IBM WebSphere Application Server V2.0 is available in Standard Edition and Advanced Edition.[6] In addition to the servlet engine and plug-ins, WebSphere Application Server Standard Edition provides:

- Implementation of the JavaSoft Java Servlet API, plus extensions of and additions to the API

[6] See `http://www.software.ibm.com/webservers/appserv/`.

- Sample applications demonstrating the basic classes and the extensions
- The IBM WebSphere Application Server Manager, a graphical interface making it easy to:
  - Set options for loading local and remote servlets
  - Set initialization parameters
  - Manage servlets
  - Specify servlet aliases
  - Create servlet chains and filters
  - Administer and monitor Enterprise Java Services (EJS) components
  - Enable Lightweight Directory Access Protocol (LDAP) directory support
  - Log servlet messages
  - Enable JVM debugging
  - Monitor resources used by Application Server
  - Monitor loaded servlets, active servlet sessions, and JDBC connections
  - Monitor errors, events, exceptions, and log output
  - Create dumps and data snapshots
  - Dynamically enable and disable tracing
- A connection management feature that caches and reuses connections to your JDBC-compliant databases

  When a servlet needs a database connection, it can get one from the pool of available connections, eliminating the overhead required to open a new connection for each request.
- Additional Java classes, coded to the JavaBeans specification, that allow programmers to access JDBC-compliant databases

  These data access beans provide enhanced function while hiding the complexity of using relational databases. They can be used in a visual manner in an integrated development environment.
- Support for dynamic page content called JavaServer Pages (JSP)

  JSP technology lets you produce dynamic Web pages with server-side scripting. The result is to separate your presentation logic (for example, the HTML code that defines your Web site structure and appearance) from your business logic (for example, the Java code that accesses a database for information to display on the Web site). For flexibility, JSP files can include any combination of inline Java, `<SERVLET>` tags, National Center for Supercomputing Applications (NCSA) tags, and JavaBeans.
- Enablement for LDAP supported directory services
- Modules and a command line interface for integrating Application Server and Apache Server into the Tivoli environment for distributed monitoring and operations
- eXtensible Markup Language (XML) Document Structure Services

WebSphere Application Server Advanced Edition provides all the features of Standard Edition, plus:

- Enterprise Java Services

  This function is provided to run and manage applications coded to Sun's Enterprise JavaBeans (EJB) specification.

- Common Object Request Broker Architecture (CORBA) support, enhanced to provide both bean-managed and container-managed persistence.

IBM WebSphere Application Server will also be available in an Enterprise Edition, which will include the same features as the Advanced Edition, plus:

- TXSeries support, IBM's world-class transactional application environment
- Component Broker (CB) support, with its fully distributed object and business process integration capabilities

1.7.2 IBM WebSphere Studio

IBM WebSphere Studio is a suite of tools that can be used by people involved in creating and maintaining Web sites. It allows your team to:

- Easily create Java beans, database queries, and Java servlets using the Studio wizards
- Group your Web site files into projects and folders
- Edit and update the files with your preferred tools
- Publish all or part of the Web site on any of your WebSphere Application Server systems
- Maintain the files locally on individual workstations, or in a central location using a source control system

IBM WebSphere Studio is the tool complement of IBM WebSphere Application Server. This suite of tools makes it easier to design, develop and maintain dynamic, interactive Web sites.

The tools provided by IBM WebSphere Studio are:

- Studio workbench

  The workbench helps you manage and maintain your Web site applications and files. It uses projects and folders to group and organize the files and lets you perform all other functions from this central location. The Studio supports all file types. You can create files, open them, edit them, copy them, move them around, delete them, publish them. And, you can do all this right from the workbench using the workbench menu functions, the wizards, and your preferred Web development software.

- Studio wizards

  The wizards are the fastest way to add dynamic content to your Web pages. They help you retrieve information from common databases, use server-side JavaBeans, capture information about your customers, and register Web visitors. You don't have to be expert at SQL syntax or Java programming. The wizards walk you through step-by-step and then generate sophisticated servlet code for you.

- Companion products

For added convenience the Studio comes with an integrated set of companion products. Everything is included for you to build, manage, and publish complete Web sites:

- NetObjects ScriptBuilder

  Use this text-based editor for files that contain common markup, scripting, and programming languages, such as HTML, DHTML, and JavaServer Page (JSP) extensions, JavaScript, JScript, and Java. Its features make the scripting process easy. You can quickly preview scripted Web pages, reference and add language elements, and navigate to embedded functions and objects.

- NetObjects Fusion

  Build your Web sites in a visual manner, dragging and dropping pages to create an interconnected hierarchy. You can view individual pages or the entire site structure. This powerful tool lets you include images, multi-media, and dynamic HTML, and control the visual appearance of an entire site from one central location.

- NetObjects BeanBuilder

  BeanBuilder allows you to put an applet in your Web site in a very short time. This visual authoring tool lets you quickly combine Java beans into new applets.

- VisualAge for Java, Professional Edition

  If you are familiar with Java programming, you can use this robust, full-function environment to create and customize Java components. This award-winning tool includes advanced functions, such as incremental compilation and the ability to invoke methods while debugging. You can use it to build sophisticated Java beans that you can use in the Studio basic servlet wizard. And, in turn, you can also use it to modify the Java servlets and beans generated by the Studio wizards.

- WebSphere Application Server

  What good is server-side Java logic without a server that can handle it? For your convenience, the best place to publish your Web sites is right in the package. The WebSphere Studio comes with a complimentary copy and a developer's license for the WebSphere Application Server.

- Base HTTP servers

  The WebSphere Application Server runs on several HTTP servers and more than one operating system. But, just in case you don't have one handy, the Apache Web server and a base HTTP server are included for good measure.

Chapter 2. IBM SecureWay Network Dispatcher Concepts

In this chapter, we discuss the IBM SecureWay Network Dispatcher (ND) Version 2.1 component of IBM WebSphere Performance Pack Version 2. This load balancing software improves the performance of servers by distributing TCP/IP session requests to different servers belonging to a group of servers. ND is derived from a lineage of proven products used to load balance TCP traffic on many of today's most active Web sites, as well as several high profile events over the past couple of years. Notable among these are the IBM Corporation home page `http://www.ibm.com`, IBM Software home page `http://www.ibm.com/software`, Masters Golf Tournament, Wimbledon Tennis Championship, French Open Tennis Championship, U.S. Open Tennis Tournament, the ACM Chess Challenge sites (Gary Kasparov versus IBM's chess playing supercomputer, Deep Blue), the 1996 Olympic Games in Atlanta and the 1998 Nagano Olympic Games, where a peak of almost 104,000 hits *per minut*e was achieved.

The code that would eventually evolve into this product began as the TCP Router project in Hawthorn, England. The working prototype from that project was implemented at the 1996 Olympic Games in Atlanta. In its first release as a product, this software was packaged with a component of the SP Loadleveler software known as Interactive Session Support (ISS), developed in Hursley, England. This also was the name given to the product. ISS Version 1.0 ran only on AIX at the time. ISS V1.0 for AIX was used at the 1998 Nagano Olympic games. More specifically, a variant of the TCP Router component, not the ISS component, was used in Nagano.

The first name change for the product came with the next release, known as Interactive Network Dispatcher Version 1.1. In this release, the code was ported to Solaris and Windows NT and some portions of it were written in Java to aid in the porting process. The complete product port came in the next release: Interactive Network Dispatcher Version 1.2.

The next change for the product saw not only a change in the name to IBM SecureWay Dispatcher Version 2, but also its packaging. IBM SecureWay Network Dispatcher Version 2 was included as the Load Balancing component of IBM WebSphere Performance Pack Version 1.0, along with IBM AFS Enterprise File System (AFS) – the File Sharing component – and IBM Web Traffic Express (WTE) – the Caching and Filtering component.

In the new version of WebSphere Performance Pack, Version 2, the name of the Load Balancing component has changed again. IBM SecureWay Network Dispatcher Version 2 is now known as IBM SecureWay Network Dispatcher Version 2.1.

2.1 Load Balancing Technologies

The first generation of Web traffic load balancing, called DNS round-robin, simply had a Domain Name System (DNS) name server rotating the resolution of names among a hard-coded list of IP addresses for Web application servers. Some degree of balancing is achieved with round-robin, and the multiserver hardware is used, but the balancing is hardly optimal, since the DNS round-robin treats all the requests as equal. Round-robin load balancing does not take into account the availability of the servers and the workload on them. Moreover DNS round-robin does not provide the ability to differentiate by port.

The next generation of load balancing software included a more intelligent round-robin method along with a user-specified approach to distributing TCP/IP session requests. The ND component of IBM WebSphere Performance Pack Version 2 takes the capabilities of this proven load balancing software another step forward. ND improves the performance of servers by basing its load balancing decision not only on the servers' availability, capability and workload, but also on many other new user-defined criteria as well. As a result of ND's inherent flexibility, Web sites can now take advantage of differentiated qualities of service, based on request origin, request content and overall load on the system. The entire load balancing operation is transparent to end users and other applications.

The ND component of IBM WebSphere Performance Pack is very useful for applications such as e-mail servers, World Wide Web (WWW) servers, distributed parallel database queries, and many other TCP/IP applications. When used with Web servers, it can help maximize the potential of a Web site by providing a powerful, flexible, and scalable solution to peak-demand problems. If visitors to your site can't get through at times of greatest demand, the ND component of IBM WebSphere Performance Pack can automatically find the optimal server to handle incoming requests, thus enhancing your customers' satisfaction and your profitability.

The ND component of IBM WebSphere Performance Pack consists of three subcomponents that can be used separately or together to provide superior load-balancing results: Dispatcher, ISS, and Content Based Routing (CBR).

- You can use the Dispatcher component by itself to balance the load on servers within a local area network (LAN) or wide area network (WAN) using a number of weights and measurements that are dynamically set by Dispatcher. This function provides load balancing at a level of specific services, such as HTTP, FTP, SSL, NNTP, POP3, SMTP, and Telnet. It does not use a DNS server to map domain names to IP addresses.
- You can use the ISS component by itself to balance the load on servers within a local or wide area network using a DNS round-robin approach or a more advanced user-specified approach. Load balancing is performed at the machine level. ISS can also be used to provide server load information to a Dispatcher machine.

  When used for load balancing, ISS works in conjunction with the DNS server to map DNS names of ISS services to IP addresses. When used to provide server load information, a DNS is not required.
- The CBR component works along with WTE to load balance client Web requests to specified servers; the routing is determined by comparing the content of the request to rules that have been defined in the CBR component.

2.2 Functions of the ND Component

The three components that make up the SecureWay Network Dispatcher Version 2.1 are the Dispatcher, ISS and CBR. ND gives you the flexibility of using these components separately or together, depending on your site configuration. This section gives an overview of the Dispatcher, ISS and CBR components.

2.2.1 Dispatcher

The Dispatcher component does not use a DNS for load balancing. It balances traffic among your servers through a unique combination of load balancing and management software. The Dispatcher can also detect a failed server and forward traffic around it.

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

The Dispatcher has two important features:

1. The server sends a response back to the client without any involvement of the Dispatcher.
2. No additional code is required on your servers to communicate with the Dispatcher.

The Dispatcher function is the key to stable, efficient management of a large, scalable network of servers. With the Dispatcher, you can link many individual servers into what appears to be a single, virtual server. Your site thus appears as a single IP address to the world. Dispatcher functions independently from a DNS in that all requests are sent to the IP address of the Dispatcher machine.

The Dispatcher provides distinct advantages for balancing traffic load to clustered servers, resulting in stable and efficient management of your site.

2.2.1.1 Dispatcher High Availability

The Dispatcher offers a built-in high availability feature. Dispatcher high availability involves the use of a second Dispatcher machine that monitors the main, or primary, machine and stands by to take over the task of load balancing should the primary machine fail at any time.

For more details on Dispatcher high availability, see Chapter 6, “ND High Availability Support” on page 177.

2.2.2 Interactive Session Support

You can use the ISS component with or without a DNS name server:

- If you are using ISS for load balancing, a DNS server is required. This can either be an actual DNS server or, if you set up a small, separate subdomain for a new name server, a replacement name server provided by ISS. Using this approach, ISS runs on a DNS machine. A client submits a request for resolution of the DNS name for an ISS-associated service, which has been set up by an administrator. ISS then resolves the name to the IP address of a server in the cell, and forwards this IP address to the client.
- If you are using ISS to collect server load information, a DNS is not needed. The ISS monitor collects server load information from the ISS agents running on the individual servers and forwards it to the Dispatcher. The Dispatcher uses this load information, along with other sources of information, to perform load balancing.

ISS periodically monitors the level of activity on a group of servers and detects which server is the least heavily loaded. It can also detect a failed server and forward traffic around it. Once every monitoring period, ISS ensures that the information used by the DNS server or the Dispatcher accurately reflects the load on the servers. The load is a measure of how hard each server is working. The system administrator controls both the type of measurement used to measure the load and the length of the load monitoring period. You can configure ISS to suit your environment, taking into account such factors as frequency of access, the total number of users, and types of access (for example, short queries, long-running queries, or CPU-intensive loads).

2.2.2.1 ISS High Availability

Implementing ISS high availability is very simple. All the nodes in a site work together to eliminate any single point of failure. Should the monitor machine fail, the survivors elect a new monitor to take over automatically.

For more details on ISS high availability, see 4.2, “Load Balancing Scenario Using the Dispatcher and ISS” on page 137.

2.2.3 Content-Based Routing

CBR is a new component of ND. WTE, the caching proxy server that is also a part of WebSphere Performance Pack Version 2, must be installed on the same machine as CBR to proxy client requests to specified servers. WTE allows you to manipulate caching details for faster Web document retrieval with low network bandwidth requirements. CBR along with WTE filters Web page content using specified rule types.

CBR gives you the ability to specify a set of servers that should handle a request based on regular expression matching the content of the request. Because CBR allows you to specify multiple servers for each type of request, the requests can be load balanced for optimal client response. CBR will also detect when one server in a set has failed, and stop routing requests to that server. The load balancing algorithm used by the CBR component is identical to the proven algorithm used by the Dispatcher component.

When a request is received by the WTE proxy, it is checked against the rules that have been defined in the CBR component. If a match is found, then one of the servers associated with that rule is chosen to handle the request. WTE then performs its normal processing to proxy the request to the chosen server.

CBR has the same functions as the Dispatcher with the exception of high availability, subagent, wide area network support, and some configuration commands.

CBR can only function as part of WTE. WTE must be running before any CBR configuration can be performed.

2.3 What Is New in This Version?

In addition to Content-Based Routing, this version of IBM WebSphere Performance Pack offers several new features in the Dispatcher and ISS functions.

The Dispatcher component has been enhanced to include the following:

- Configuration Wizard

    - This feature allows you to define a cluster quickly using a graphical user interface. See Chapter 3, “ND Installation and Basic Configuration Issues” on page 55 for further information on the configuration wizard’s capabilities.

- Assisted Interface Configuration
  - The assisted interface configuration generates required network interface configuration commands on the Dispatcher machine. For more information on this, see Chapter 4, “ND Basic Scenarios” on page 81.
- Server Directed Affinity (SDA) API
  - The new SDA feature provides an API that allows an external agent to influence the Dispatcher affinity behavior. For more information on this, see Chapter 9, “Server Directed Affinity API” on page 243.
- Binary Logging and Statistics
  - The Dispatcher and CBR components now provide an optional logging facility. This will capture server statistics at a user-specified interval. A sample program is provided for reading the binary logs. This program can be modified to process the binary information as appropriate for your site. For more information on this, see Chapter 13, “Binary Logging and Statistics” on page 321.
- Remote Authenticated Administration
  - The remote authenticated administration allows the ND components to be configured remotely with the ND configuration GUI, or by using a separately installable configuration component, the Common Configuration Utility. The remote configuration uses public/private keys for authentication. For more information see Chapter 3, “ND Installation and Basic Configuration Issues” on page 55 and Chapter 17, “Common Configuration” on page 365.
- Wildcard Cluster and Wildcard Port
  - The Wildcard Cluster acts as a catch-all cluster. There are two ways to use this feature:
    - You can configure multiple aliases on the Dispatcher machine, and have them all use the same port and server configuration.
    - You can make the Dispatcher machine the default route for some traffic, and have it load balance all the IP traffic for ports configured on the wildcard cluster.

    The wildcard port can be used to create a load balancing configuration for traffic to any port that has not been explicitly defined on your configuration.

    We will discuss wildcard card and wildcard port in more detail in Chapter 11, “Firewall Load Balancing and High Availability” on page 273, where we will also see how these two features of Dispatcher can be successfully used to load balance firewalls.
- Web Traffic Express Advisor

  This new Advisor specifically monitors the state of WTE servers being load-balanced. This feature will be discussed in .
- Workload Manager Advisor

The workload manager (WLM) Advisor is designed to work in conjunction with servers on OS/390 mainframes running the MVS Workload Manager component. For more information, see Chapter 10, “Custom Advisors” on page 263.

2.3.1 ISS

There are also some new features and fields for ISS as listed below:

- ISS GUI
  - The ND GUI can now also be used to define cells, resources, services, and observers for a given host. For more information see Chapter 4, “ND Basic Scenarios” on page 81.
- NotISSAgent
  - This feature is needed if you have a mixed group of servers that are not all AIX, Solaris, or Windows NT. For more information see 2.6.4, “Configuring Nodes” on page 45.
- Statistical RoundRobin
  - This feature allows the server to handle extremely heavy traffic. For more information, see 2.6.9, “ISS Selection Methods” on page 50.

2.3.2 CBR

CBR, which we introduced in 2.2.3, “Content-Based Routing” on page 30, is an entirely new component with this release of ND. The CBR component works with IBM’s WTE proxy server to load balance traffic based on the content of a client’s URL, FTP, and SSL requests.

Local content for a Web site can be placed behind a WTE proxy server, and the content can be served from different sets of workstations based on the type of request. For example, one or more Web servers can be set up to handle CGI-BIN requests, while other local requests are sent to a different set of Web servers. The CBR component provides load balancing and failure detection for all the Web servers.

Configuring CBR is very similar to configuring the Dispatcher component. For more information, see Chapter 14, “Content Based Routing” on page 329.

2.3.2.1 Client IP and Cookie Affinity

The *Client IP Affinity* is a feature of ND that can be used in both the Dispatcher and CBR components. With this support, the ND server maintains affinity between a client and one of the TCP servers based on the IP address of the client. Though a powerful concept, Client IP Affinity is very useful in an intranet environment, but cannot help in a real-life Internet environment, where all the clients are masked by the single IP address of a firewall machine.

To solve the problem just described, CBR offers a feature called *Cookie Affinity*. The Cookie Affinity support allows requests from a particular HTTP client to be load balanced to the same server for a specified time period. The client session will maintain affinity for a particular server without relying on the IP address of the client. To achieve this result, the CBR machine sets a cookie in the client machine.

2.4 Why Do I Need IBM SecureWay Network Dispatcher?

The number of users and networks connected to the global Internet is growing exponentially. This growth is causing problems of scale that can limit users' access to popular sites. Currently, network administrators are using numerous methods to try to maximize access. Some of these methods allow users to choose a different server at random if an earlier choice is slow or not responding. This approach is cumbersome, annoying, and inefficient.

Another method is standard round-robin, in which the DNS server selects TCP servers in turn to handle requests. This approach is better, but still inefficient because it blindly forwards traffic without any consideration of the server workload. In addition, even if a server fails, requests will continue to be sent to it, and end users on the client machines will experience serious problems.

The need for a more powerful solution has resulted in the development of ND. It offers numerous benefits over earlier and competing solutions:

- Scalability

  As the number of client requests increases, you can add servers dynamically, providing support for tens of millions of requests per day, on tens or even hundreds of servers.

- Efficient use of equipment

  Load balancing ensures that each group of servers makes optimum use of its hardware by minimizing the hot-spots that frequently occur with a standard round-robin method.

- Easy integration

  ND uses standard TCP/IP protocols. You can add it to your existing network without making any physical changes to the network. It is simple to install and configure.

- Low overhead

  ND needs only to look at the inbound client-to-server flows. It does not need to see the outbound server-to-client flows. This significantly reduces its impact on the application compared with other approaches and can result in improved network performance.

- Non-invasive technology

  ND does not modify any packets, nor does it require any modifications to the operating system on which it runs.

- Content Based Routing

  CBR gives a WTE administrator the ability to proxy requests to specific servers based on the content requested. For example, if a request contains the string `/cgi-bin/` in the directory portion of the URL, and the server name is a local server, CBR can direct the request to the best server in a set of servers specifically allocated to handle CGI-BIN requests.

- High Availability

  The Dispatcher component of ND offers built-in high availability, utilizing a standby machine that remains ready at all times to take over load balancing should the primary Dispatcher machine fail.

ISS is intrinsically highly available. All the nodes in an ISS configuration work together to eliminate any single point of failure within ISS.

CBR does not offer a high availability feature, but multiple CBR machines can be load balanced by an additional ND machine; if a CBR machine should unexpectedly fail, the ND server realizes it and directs traffic around the failing machine. This way, CBR high availability can be achieved as well.

- Colocation option

  The Dispatcher component can be installed on the same machine where one of the application servers reside. This option is particularly useful if you want your Web site to benefit from the high availability and scalability options of the load balancing component with a minimal investment.

  The colocation option is currently available only on AIX and Solaris.

2.5 How the Dispatcher Function Works

The Dispatcher creates the illusion of having just one server by grouping systems together into a *cluster* that behaves as a single, virtual server. The service provided is no longer tied to a specific server system, so you can add or remove systems from the cluster, or shut down systems for maintenance, while maintaining continuous service for your clients. For the clients, the balanced traffic among servers seems to be a single, virtual server, and the site appears as a single IP address to the world. All requests are sent to the IP address of the Dispatcher machine, which decides for each client request which server is the best one to accept requests, according to certain dynamically set weights. The Dispatcher routes the client's request to the selected server, and then the server responds directly to the client without any further involvement of the Dispatcher. The Dispatcher can also detect a failed server and route traffic around it.

The Dispatcher receives the packets sent to the cluster. These packets have a source and a destination address; the destination address is the IP address of the cluster. All servers in the cluster and in the Dispatcher system have their own IP address and an alias for the IP address of the cluster; the Dispatcher system has the cluster address aliased on the network interface, while all the TCP servers that will be load balanced by this ND machine have the cluster address aliased on the loopback adapter. The Dispatcher system checks which server is the next best server to handle the load and routes the packet to that server. The Dispatcher routes this request based on the hardware address of the network adapter (MAC address) of the chosen server. It changes the hardware address of the packet to the hardware address of the selected server and sends the packet to the server. However, the Dispatcher does not change the source and destination IP addresses in the packet. The server receives the packet and accepts it because all servers in the cluster have an alias for the cluster's IP address on the loopback interface. Then, the server sends a response back to the client by inverting the source and destination IP addresses from the original packet received. This way, the server can respond directly to the client. We see a more detailed explanation of this process in Chapter 4, "ND Basic Scenarios" on page 81.

The fact that the server can respond directly to the client makes it possible to have a small bandwidth network for incoming traffic, such as Ethernet or token-ring, and a large bandwidth network for outgoing traffic, such as Asynchronous Transfer Mode (ATM) or Fiber Distributed Data Interface (FDDI).

The server machines in the cluster could be a mixture of heterogeneous servers of different sizes and types, such as UNIX, Windows NT or OS/2 machines. We will see more details on this in Chapter 4, “ND Basic Scenarios” on page 81, and Chapter 6, “ND High Availability Support” on page 177.

2.5.1 Dispatcher Components

The Dispatcher consists of three main functions:

1. Executor

   This function supports port-based routing of TCP and UDP connections to servers based on the type of request received (for example HTTP, FTP or SSL). This module always runs when the Dispatcher function is being used.

   For further details, see Chapter 4, “ND Basic Scenarios” on page 81.

2. Manager

   This function sets weights used by the Executor based on internal counters in the Executor itself and feedback from the Advisors and ISS monitoring (if ISS is used as a monitoring tool). Each unit of information given to the Manager by the Advisors, ISS and Executor factor has a relative importance, so you can give more importance to one unit of information over the others, or totally ignore one or more units of information. Using the Manager is optional, but if the Manager is not used, load balancing is performed using weighted, round-robin scheduling based on the current server weights.

   For further details, see Chapter 4, “ND Basic Scenarios” on page 81.

3. Advisors

   Advisors send requests to the TCP servers to measure actual client response time for a particular protocol. These results are then fed to the Manager to adjust the load balancing weights. Currently, there are Advisors available for HTTP, FTP, SSL, SMTP, NNTP, POP3 and Telnet. Three new Advisors have been added in this version: ping, WTE, and WLM. Using the Advisors is optional, but recommended. You also have the option of writing your own Advisors. A custom Advisor will provide the precise information about servers that you need. Advisors, like the rest of the Dispatcher, must be compiled with Java 1.1. To ensure access to Dispatcher classes, make sure that the ibmnd.jar file (located in the lib subdirectory of the base directory) is included in the classpath system environment variable. Only Advisors written to the correct level of Java will be supported.

   For further details, see Chapter 4, “ND Basic Scenarios” on page 81 and Chapter 10, “Custom Advisors” on page 263.

There is also an SNMP subagent function that allows an SNMP-based management application to monitor the status of the Dispatcher.

2.5.2 Proportions of Importance

The Manager decides which is the least-loaded server on a particular port in the cluster by looking at the weight of each server. The Manager will periodically update the weight of each of the server machines, basing its decision on four parameters or policies:

1. The number of active connections on each TCP server
2. The number of new connections for each TCP server
3. Input from TCP server Advisors
4. Information from system monitoring tools, such as ISS

Setting the servers' weights in the load-balancing process is performed by using the so-called *proportions of importance*. Each of the above factors is attributed a number, from 0 to 100, that acts as a percentage. 0 means that the policy is not used, while 100 means that only that factor will be used. It is necessary that those proportions add up to 100. The default settings at the startup appears as `50 50 0 0`.

2.5.2.1 Guidelines on Proportions of Importance Settings

Although there are generally not fixed rules on how to set the proportion value, it is still possible to provide some useful guidelines, described in the following.

The first two proportions are related to active and new connections respectively (see points 1 and 2 above). Typically in an out-of-the-box configuration, this is all you will be able to use. That is why the default proportions at startup of Dispatcher appear as `50 50 0 0`.

Anyway, the load on a classical Web server depends mainly on the number of connections, both active and new. Just consider the following. If the client connections to the services provided by the TCP server machines are quick (such as small Web pages served using the HTTP GET method), then the number of active connections will be expected to be fairly low. On the contrary, if the client connections are slower (such as database queries), then the number of active connections will be higher.

If you then start the Manager, it makes sense now to use a non-zero value for the Advisor proportion (see point 3 above). The standard Advisors shipped with Dispatcher execute a trivial transaction on each TCP server. Normally you would expect a trivial response time to this transaction. Experience shows us that it is not a good idea to set the Advisor proportion to a high value. We recommend `49 49 2 0` for this setup.

Things change if you start using a custom Advisor. Your custom Advisor can, for example, execute a Java servlet (or other application transaction) on the server side, and this servlet can gather very precise values about performance, throughput, and response time via the server's API and feed it to the Advisor when asked. Thus, you may find it appropriate to put the Advisor proportion at a higher value. However, we recommend that you do not overcompensate, and do not set active and new connection proportions too low; otherwise you begin to restrict Dispatcher's adaptive and smoothing capabilities. Moreover, you may end up with a load balancer that follows your Advisor too closely, resulting in a choppy profile that is probably less than ideal. In the final analysis, what works for you is best; you need to experiment with your custom Advisor to see what results you get in the real world.

Finally, if you install the ISS daemon on your servers, it makes sense to add the ISS proportion to your mix (see item 4 on page). If you use a custom metric in ISS as well as a custom Advisor, be sure to have them go after similar objectives;

otherwise you may come to set one against the other with undesirable consequences.

2.5.3 Information Flow

The typical flow of information used in the Dispatcher is shown in the Figure 9:

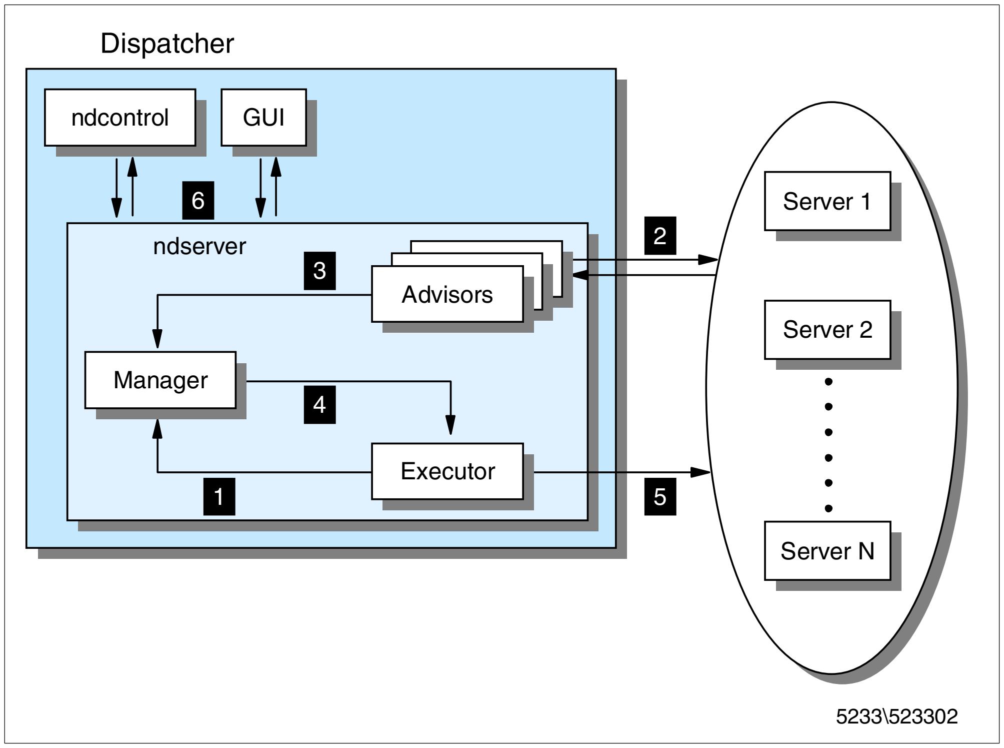


Figure 9. Information Flow Used in the Dispatcher

The first step that must be done after installing and configuring the Dispatcher is to run it using the following command:

```
ndserver
```

The logical information flow, represented in the above diagram, is described in the following list:

1. The Executor supplies information about new and active connections based on its internal counters to the Manager.
2. The Advisors collect information about response time and availability from the servers for each service and each port.
3. After processing this information, they send it to the Manager.
4. The Manager uses all the information it receives (including information sent by the ISS component, if active, about the servers' load balancing according to a specific metric), calculates the new weights to be used in connection routing. and sends the results to the Executor.
5. The Executor uses these new weights for TCP and UDP routing. If the Manager and the Advisors are not running, the Executor does the routing based on its internal counters.

6. A command line interface, represented by the `ndcontrol` command, and the graphical user interface (ND GUI) are provided to configure and manage the Executor, Advisors and the Manager.

2.5.4 TCP Ports Used by the Dispatcher

The Dispatcher uses three TCP ports for its communications:

1. Port 10099

   This port is used for receiving commands from the ndcontrol program.

2. Port 10005

   This port is used to receive information from an SDA agent.

3. Port 10004

   This port is used to receive metric response from ISS.

If another application is already using port 10099 or port 10005 for communication, then you will need to change the ndserver script to use different ports:

- On AIX, ndserver is located by default under the /usr/bin directory.
- On Solaris, ndserver is located by default under the /bin directory.
- On Windows NT, the script is called ndserver.cmd and is located by default in C:\WINNT\system32.

To change the port used for receiving `ndcontrol` commands, change the ND_RMIPORT variable to the new value. Similarly, to change the port used to communicate to an SDA agent, change the ND_AFFINITY_PORT variable to the new port number.

To change the port used to receive metric reports from ISS, use the *metric_port* option when starting the manager:

```
ndcontrol manager start log_file metric_port
```

Notice that if you specify a *metric_port* you must also specify a *log_file*. The above command has a corresponding version in the GUI, as we see in Chapter 4, “ND Basic Scenarios” on page 81.

To inform ISS of a change in the metric port number, when you define the Dispatcher Observer in the ISS configuration file, you should indicate the new port number:

```
Dispatcher hostname metric_port
```

This is shown in Figure 13 on page 43.

2.6 How the ISS Function Works

This section describes the ISS function and its concepts.

2.6.1 ISS Cells and Services

ISS logical architecture is based on the concept of a *cell*. A cell is a group of servers administered as a single, logical unit. Each server in a cell is also called a *node*.

A group of servers (or nodes) that perform the same function is a *service.* Think of a service as a group of nodes in the cell that will serve the request only for a particular protocol (for example HTTP, FTP, or telnet).

It is possible in a cell to define different services (corresponding, for example, to HTTP, FTP, or Telnet). Moreover a node can belong to one or more services. The following figure offers a graphical representation of a possible situation:

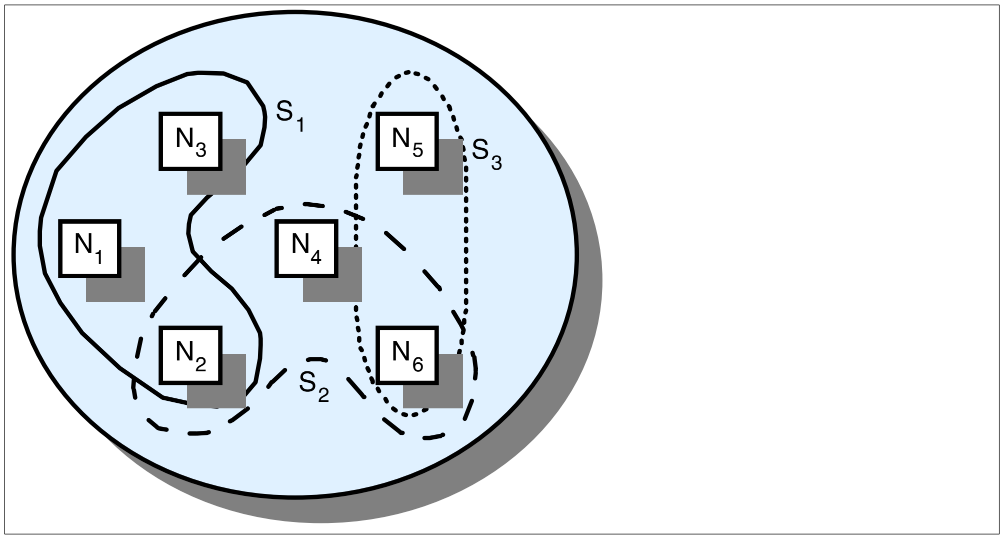


Figure 10. Cell, Services and Nodes

In this example, we have a cell with six nodes, named N1, N2, N3, N4, N5 and N6, and three services, S1, S2, S3. You can see that:

- The service S1 contains the nodes N1, N2 and N3.
- The service S2 contains the nodes N2, N4 and N6.
- The service S3 contains the nodes N5 and N6.

Note that some nodes belong to only one service, and other nodes belong to more services. In particular:

- N1 and N3 run only service S1.
- N4 runs only service S2.
- N5 runs only service S3.
- N2 runs services S1 and S2.
- N6 runs services S2 and S3.

On each node of the cell the ISS component must be installed, configured, and activated. The ISS will run as a daemon, called the issd daemon.

2.6.2 ISS Configuration

A significant enhancement has been made to the ISS component of IBM eNetwork Dispatcher Version 2.1 (shipped with WebSphere Performance Pack Version 2) in regards to ISS configuration. Configuration of the ISS cell environment has been incorporated into the IBM eNetwork Dispatcher configuration GUI. ISS cells, hosts, resources, services and observers for a given ISS environment can be defined by using the GUI. Prior to using the GUI for ISS configuration, a minimal ISS configuration file must be defined to instruct the issd daemon on how to act. A sample basic configuration file that is shipped for this purpose can be modified by you before starting the GUI. See Chapter 4, “ND Basic Scenarios” on page 81 for an example of starting and using the GUI for ISS configuration.

In total, five sample configuration files are shipped with the product. These five ISS sample configuration files are very useful for understanding how to configure ISS in your environment:

1. The Basic.sample file contains only the minimum two entries (one local cell and one host) necessary to start using the GUI to perform further configuration.
2. The Dispatcher.sample file illustrates how ISS can be used to supply load data to two Dispatcher machines in a highly availability configuration.
3. The ReplaceDNS.sample file illustrates how ISS can be used as a limited DNS name server to perform load balancing for two clusters of servers.
4. The TwoTier.sample file illustrates how ISS can be used with the Dispatcher in a two-tiered architecture, as shown in Figure 11 on page 41. The bottom tier consists of two clusters of servers. Each cluster itself has two servers. Two Dispatcher machines are used to balance these clusters. The top tier consists of two Dispatcher nodes. In this configuration, ISS performs two functions:
    1. It performs DNS load balancing for the top tier.
    2. It provides the Dispatchers with additional server load information, so that they can load balance the bottom tier more accurately.

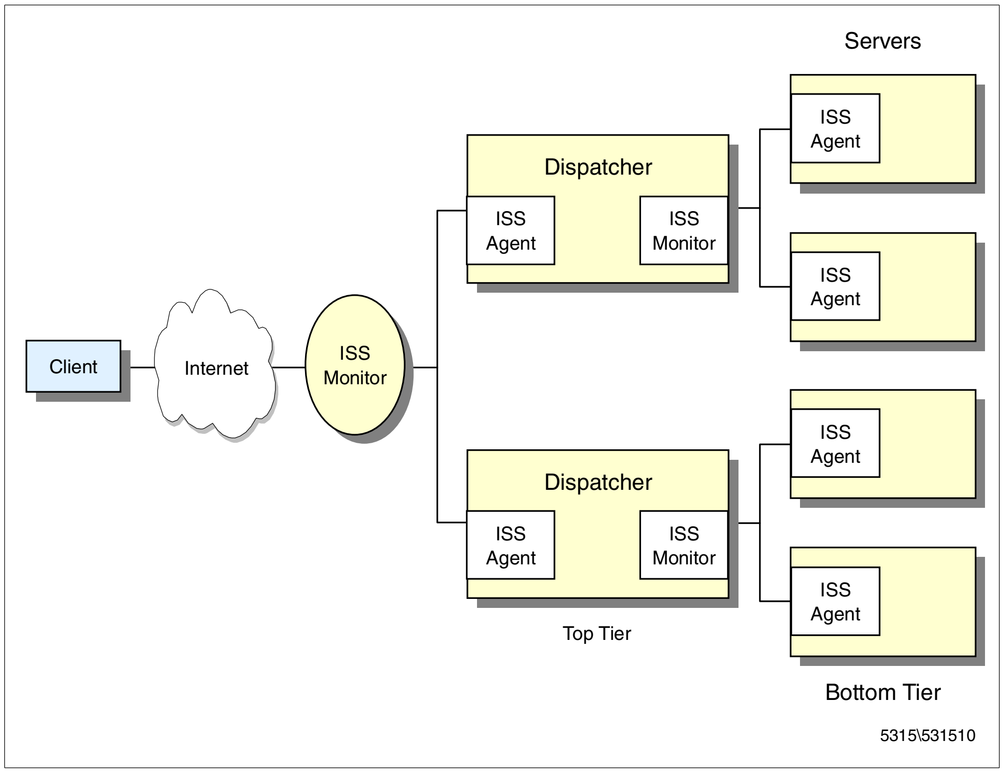


Figure 11. Two-Tiered ISS and Dispatcher Configuration

5. The file PingTri.sample file actually contains two sample configuration files. Each sample configuration file is for use in a separate cell. One is for use in a cell called Hursley, and the other for use in a cell called Raleigh. Each file defines the other cell as a global cell. With this configuration, each machine can communicate with all the other nodes in its cell, and knows enough about the remote cell to exchange ping information and redirect traffic to it. This is called ping triangulation and is one of the possible methods used by ISS to determine which server should respond to client requests. *Ping triangulation* is a technology that allows you to find out which server site is the closest to a given client.

When ISS is installed, these configuration files are placed in *installbase/*samples, where *installbase* is the installation directory for the ISS component and varies by operating system. See Table 1 on page 69 for a list of the installation directory locations.

For the purpose of explaining how ISS is configured, we will list the contents of the Dispatcher.sample file:

```
# ----------------------------------------------------------------------------
#
# Dispatcher Sample configuration file for ISS
#
# ----------------------------------------------------------------------------
#
# This is a simple configuration file used to supply an additional
# load value to Network Dispatcher. There is only one (local)
# cell, and one service running.  Two Metrics are combined to
# create the load value for the service.  Since there are two
# Dispatcher machines running in a Highly Available configuration,
# both dispatchers are listed as observers.

# Parameters for the whole cell
# In this cell, ITL, the Monitor will attempt to reach each node
# every 5 seconds (HeartbeatInterval).  After 3 Intervals, or
# 15 seconds, each service will be executed.
Cell      ITL             local
AuthKey     10043572 ADE4F354 7298FAE3 1928DF54 12345678
LogLevel                        INFO
HeartbeatInterval               5
HeartbeatsPerUpdate             3
PortNumber                      7139

# Individual node data

# The node dispatch1 has ISS monitor priority 1 and will run the issd
# process in ISS monitor mode. The node dispatch2 has monitor priority 2
# and can run as a backup ISS monitor when dispatch1 is down. The last
# node, server2, is declared NotMonitor. The issd processes on this node
# will run as ISS agents, collecting load statistics, but cannot assume
# the role of an ISS monitor.  The nodes dispatch2 and server1 are equipped
# with additional resources and are considered to have an advantage over
# the other servers and should be recommended more often than the others.
# Therefore they are given a higher weight.

Node    dispatch1.raleigh.ibm.com   001
Node    dispatch2.raleigh.ibm.com   002
NodeWeight      1.5
Node    server1.raleigh.ibm.com     003
NodeWeight      1.5
Node    server2.raleigh.ibm.com     098
NotMonitor
NodeWeight      1.0

# The service is configured to depend on two resources -- CPU availability
# and the process count. Load balancing is therefore performed based on CPU
# utilization and the number of processes currently running. However, ISS
# will not schedule work for nodes that are unreachable on the network.
#
# In this configuration, CPU utilization has more value than the process
# count for determining a better performance from a server. Therefore, the
# CPU value will be considered 2.0 times more valuable than the ProcessCount.
#
# Definition of "CPU" ResourceType
#
# The specified MetricLimits indicate that a node will not be used if its
# CPU usage goes over 95% and will not be put back in the list until CPU usage
# goes back down to 80%.
```

Figure 12. (Part 1 of 2). Dispatcher.sample Configuration File

```
ResourceType            CPU
Metric                  Internal     CPULoad
MetricNormalization     0     100
MetricLimits            80    95
Policy                  Min
MetricWeight            2.0

# Definition of "ProcessCount" ResourceType

# The metric associated with ProcessCount is assigned a range of (0-250).
# If "ps -fe | wc -l" returns a value greater than 250, the metric is
# assigned the upper limit of 250. When the value of this metric on a node
# is greater than 225, the node is removed from active participation
# in the Service that it is associated with. The node is returned to active
# participation when the metric has a value less than or equal to 200.

ResourceType            ProcessCount
Metric                  External     ps -ef | wc -l
MetricNormalization     0     250
MetricLimits            200   225
Policy                  Min
MetricWeight            1.0

# The one configured service, NDService, is an identifying string for the
# service, and is used when balancing between cells. In this configuration,
# it is used simply to identify what the service is for and define which
# servers to collect the selected metric values from.
#
# The name PlaceHolder is not needed when you are using ISS to update the
# Dispatcher. The "cluster address", Dispatch_cluster, and port 80, are used
# to signify which set of Dispatcher servers are relevant to this service,
# and where ISS should send the load information to. ISS itself does no load
# balancing in this scenario.

Service     NDService           PlaceHolder Dispatch_cluster 80
NodeList                        dispatch2.raleigh.ibm.com server1.raleigh.ibm.co
m server2.raleigh.ibm.com
ResourceList                    CPU   ProcessCount
SelectionMethod                 Best

# The machine dispatch1.raleigh.ibm.com and dispatch2.raleigh.ibm.com have an
# Network Dispatcher configured to listen for load values on port 10004. Since
# Dispatcher is running in a High Availability configuration, both Dispatchers
# need to be listed.

Dispatcher                      dispatch1.raleigh.ibm.com 10004
ServiceList NDService

Dispatcher                      dispatch2.raleigh.ibm.com 10004
ServiceList NDService
# End of ISS configuration file
```

Figure 13. (Part 2 of 2). Dispatcher.sample Configuration File

In the following sections, we see how the ISS function works to understand the settings shown in the above configuration file better. We recommend that you read the information provided in the following sections in parallel with the above configuration file.

2.6.3 ISS Cell and Its Attributes

The first thing that an ISS configuration file does is to define the cell and declare some cell attributes.

A cell can be *local* or *global*, and this is specified in the ISS configuration file through the keywords `local` or `global` respectively. A local cell is the one that the node will be a member of. There must be one and only one local cell defined in the ISS configuration file. A global cell is a separate, remote cell that you want your node to communicate with. You must have a global cell if you have widely separated mirror sites and want to perform ping triangulation.

A cell can have the following attributes, specified in the ISS configuration file:

- `Authkey`

  This key is used to provide authentication during the ISS internal traffic among the nodes of the cell. If this keyword is not specified in the configuration file, a default key will be used. A node can have its own Authkey attribute, which has to be specified after the node is defined.

- `LogLevel`

  You can have five different log levels, each one providing more information than the previous one:

  ```
  [ None | Error | Info | Trace | Debug ]
  ```

  The above values for the `LogLevel` attribute are self-explanatory.

- `PortNumber`

  This option specifies the port number used for ISS internal traffic:

  ```
  PortNumber [number]
  ```

- `HeartbeatInterval`

  ISS will periodically contact all servers that provide a specific service to ensure that they are still alive. ISS does this using the equivalent of the `ping` command. This does not verify that the service is running on the servers, but at least ensures that the servers can be contacted at the IP level. ISS maintains a ranking that lists the servers in their current priority order according to the configured load measurement metric. If the currently top-ranked server fails to respond, the next server will be selected. This check is referred to as a *heartbeat*. The time between heartbeats is determined by the `HeartbeatInterval` setting. The syntax is:

  ```
  HeartbeatInterval [seconds]
  ```

- `HeartbeatsPerUpdate`

  The syntax of this keyword is:

  ```
  HeartbeatsPerUpdate count
  ```

  Where *`count`* is a non-negative integer.

  The `HeartbeatsPerUpdate` value is multiplied by the value of `HeartbeatInterval` to give a time called the *update interval*. The value assigned to the update interval parameter indicates the number of heartbeats after which ISS will determine whether a domain name server update or a Dispatcher update is required.

Note that the value of `HeartbeatsPerUpdate` will be ignored in the situation where the top ranked server fails to respond to a heartbeat. The server will immediately be removed from its top-ranked position.

2.6.4 Configuring Nodes

After defining the cell, you must list all the nodes that will be part of it using the `Node` keyword. The syntax is:

```
Node [ hostname | IP address ] priority
```

In your cell configuration, when using ISS, you should elect one of the nodes as the cell leader, or *monitor*, while all the other nodes act as *agents*. In other words, if ISS is used to collect server load information, there is an ISS monitor that collects this information from the ISS agents that run on the individual servers. The ISS monitor can then be used to send the ISS agent information to the Dispatcher. The ISS monitor can be installed on one of the servers, a Dispatcher machine, or on a different machine.

Moreover, you can configure one or more other nodes to back up the monitor node, by defining a different priority number for each one. In a backup node, the issd daemon usually runs in agent mode, and the node can only be a server that provides a service. If the monitor fails, the first (in the priority scale) backup node in the list takes over, and on this node the issd process switches from agent mode to monitor mode. As soon as the previous node is available again, it will become the monitor because of its higher priority value. This functionality provides ISS *high availability*.

If you don't want a particular machine to run as an issd monitor, you have to declare this using the `NotMonitor` keyword immediately after the node declaration. In this case the priority field is only a numeric identifier for that node.

New Node Attribute: NotISSAgent

A new attribute has been added for the node declaration in this version. If the node is not running AIX, Solaris or Windows NT, then you need to indicate this by using the `NotISSAgent` keyword immediately after the node declaration.

2.6.5 Services

Before explaining how to define the resources, we prefer first to focus on the section of the configuration file where the services are defined, along with their parameters:

- `Service`

  You declare a service with the `Service` keyword, followed by the service name, the fully qualified service DNS name (which is the DNS name of the pool of machines that provide the same service), the cluster IP address, and the port number. The correct syntax is:

  ```
  Service name DNSName [ cluster address ] [ port number ]
  ```

- `NodeList`

After defining the service, you need to use the `NodeList` keyword, which allows you to specify the list of nodes and members of your cell providing the service. `NodeList` must be followed by the host names or IP addresses of those servers:

```
NodeList DNSName [ DNSName ... ]
```

- `ResourceList`

  The `ResourceList` parameter allows you to declare which resource will be used to determine the appropriate node that will provide that service. A resource can be CPU, disk, process, and so forth.

- `Overflow`

  This keyword identifies a server on which to fall back if all the other nodes specified in `NodeList` are unable to provide the service. The syntax is:

  ```
  Overflow DNSName
  ```

2.6.6 Resources

When you define a service in the configuration file, you specify the name of one or more resources as arguments of the `ResourceList` parameter. A single resource can be used as the selection criterion in different services. Consequently, the resources should be defined in the configuration file before the service definitions.

You define resources using the `ResourceType` parameter and its own keywords `Metric`, `MetricNormalization`, `MetricLimits`, and `Policy`.

The `ResourceType` keyword must be followed by the symbolic name of the resource you are defining. It specifies the criteria you have decided to use for selecting the best server in a service. The syntax is:

```
ResourceTypes Name
```

The next section explains how to use these parameters.

2.6.7 Metrics

A resource type is characterized by several parameters that define the metric that will be used to measure the load among the servers.

A metric defines how ISS will measure the load on the server for each service. This measurement should be appropriate to the particular service. For example, if the service provided is CPU-intensive, the metric could be defined as the percentage of time that the CPU is busy. If the service is disk- or I/O-intensive, the metric could be based on the amount of time that the disks are busy or the time spent waiting for I/O to complete.

The metric keywords are the following:

- `Metric`

  The keyword `Metric` in the configuration file specifies the type of measurement ISS will use to provide a specific service. The syntax is:

  ```
  Metric [ Internal | External ] string
  ```

  where the *`string`* field indicates a command or a program that gives a numeric output.

The sample configuration file Dispatcher.sample that we have shown uses the line:

```
Metric Internal     CPULoad
```

The keyword `Internal` identifies a given measurement method as being provided by the system. The parameter `CPULoad` forces ISS to measure the percentage of the CPU that is being utilized. The other system-provided measurement system is `FreeMem`, which returns the amount of free physical memory as a percentage.

When you set `Metric` to `External`, you define how to measure the load on the servers. This metric is anything that can be executed on the server and returns a numeric value as a result. In this case the *`string`* field defines a command or program that when executed, returns a numeric value that can be used to measure of the load on the server.

The Dispatcher.sample file shown defines a metric based on the number of processes running on the server:

```
Metric External ps -ef | wc -l
```

- `MetricNormalization`

  This is a range between the lower and upper limits of measurement, indicated as integer numbers. The syntax is:

  ```
  MetricNormalization LowerLimit UpperLimit
  ```

  The limits referred to depend upon the metric you are using. In case you want to measure the CPU utilization, as shown in the Iss_config_1 sample configuration file, the two limits are percentage points. The range is therefore 0 to 100. Any metrics coming into the monitor outside of this range would be corrected by being clipped to fit within these limits.

- `MetricLimits`

  The syntax for this entry is:

  ```
  MetricLimits RecoverLimit FailLimit
  ```

  You can set these limits to prevent a server from failing totally by having too many requests assigned to it. ISS measures the loads on servers periodically. If the loads on a certain server are within predefined limits, ISS continues to keep the server in the list of the available nodes. If the loads are outside the limits, ISS removes the server from the ranks of that service. If the server is handling more than one service, it may still be included in the ranks of the other service or services. There are two levels you define:

  1. The *`FailLimit`* level represents the level beyond which the metric should not go. When the *`FailLimit`* level is reached, ISS removes the server from the ranking. In the Dispatcher.sample configuration file that we have shown, the specified value for *`FailLimit`* indicates that a node will not be used if its CPU usage goes over 95%.
  2. The *`RecoverLimit`* level sets the point at which a node that had been removed from the ranking should be returned to active participation. In the Iss_config_1 ISS sample configuration file, the *`RecoverLimit`* parameter is set to 80%.

  So, defining the limits of the CPU load metric with the following entry causes the resource to be removed at 95% of utilization, and returned only when it is back down to 80%:

```
MetricLimits 80 95
```

Specifying a significant difference between the two above levels prevents the phenomenon wherein resources come into service, fail, then come back after minimal recovery only to quickly fail again.

- `Policy`

  The parameter `Policy` can be assigned either the value `Max` or `Min`. The correct syntax is:

  ```
  Policy { Max | Min }
  ```

  It indicates whether a metric function is to be minimized or maximized. The default value is `Max`.

  For example, a metric based on the number of active users would regard the smallest value as the best and would have the following entry:

  ```
  Policy Min
  ```

  On the other hand, a metric function based on CPU idle time would regard the maximum value as best and, therefore, the policy would be specified as the following:

  ```
  Policy Max
  ```

2.6.8 ISS Observers

After configuring nodes and services, and after defining the resources, you should define who will use the information about the status of nodes. In other words, you should configure the observers.

An *observer* is a network process (such as a Dispatcher) that uses information about the status of nodes. ISS provides three observers, which perform different actions but have the same purpose: load balancing. The three observers are named NameServer, ISSNameServer and Dispatcher. These names correspond to the keywords `NameServer`, `ISSNameServer` and `Dispatcher` that you should use in your ISS configuration file.

Depending on how you decide to configure your environment, you can configure one or more observers in your cell. This is very important as it determines how ISS will be implemented (statistics gathering versus load balancing).

2.6.8.1 NameServer

When using the NameServer observer, ISS runs in conjunction with a DNS name server. It uses the DNS name server to map DNS names of ISS services to IP addresses of the most appropriate server. ISS assists the DNS server in making the balancing decision. ISS monitors the load on each server of the cell and ensures that the server currently used for a particular service is the one with the lightest load.

It is not mandatory that the machine where the issd monitor process running as a monitor is the same machine where the DNS daemon named runs. For this reason, you can configure one or more servers on your cell as backup issd monitor nodes.

The load-balancing decisions are based on how you have configured ISS. In this case, ISS makes load-balancing decisions by itself and instructs the DNS name server about the server that the incoming request are to be routed to.

The NameServer observer involves some load on the DNS server machine. Every time a new server is selected as the top-ranked server, the DNS configuration files are updated. A signal is then sent to the named daemon to reload the name resolution data files. If these files are large, it can take a significant amount of time and processing for the named daemon to process them. During this processing, the named daemon is unable to respond to name resolution requests.

The syntax to define a NameServer observer is the following:

```
NameServer DNSName [ PortNumber ]
```

2.6.8.2 ISSNameServer

When using the ISSNameServer observer, ISS works as a DNS name server. In this case, ISS runs on a DNS domain name server machine, where the DNS daemon named is not running.

ISS replaces the named daemon and makes use of the DNS configuration file of the DNS name server. In this case ISS provides the name serving function by providing minimal name server implementation. In this way, besides taking over the DNS, ISS makes load-balancing decisions by itself.

Using the ISSNameServer observer does not increase the load on the DNS machine, so it is a good option to use the ISSNameServer observer with a new subdomain for your pool of servers.

The syntax to define an ISSNameServer observer is the following:

```
ISSNameServer DNSName [ PortNumber ]
```

For example, the following is the entry used in the Iss_config_1 sample configuration file:

```
ISSNameServer delta 53
```

2.6.8.3 Dispatcher

When using a Dispatcher observer, you should have defined in your cell a Dispatcher machine, which ISS strictly cooperates with to perform load balancing. Different from the previously mentioned two observer types, a DNS server is not required (neither a DNS server nor an ISSNameServer observer working as DNS).

The ISS configuration file will reflect your cell configuration, and the Dispatcher machine you selected should be specified immediately after the `Dispatcher` keyword, according to the following syntax:

```
Dispatcher DNSName [ PortNumber ]
```

The Dispatcher.sample configuration file shown contains two Dispatcher observer entries. There is one entry for each Dispatcher machine that is to receive load information from ISS.

With this type of Observer, you can view ISS as a monitoring tool on the TCP server machines. ISS provides the Dispatcher with load server information, but ISS does not make any load-balancing decision.

The ISS monitor collects specific server information, in this case CPU usage and process count information, but could also include memory usage and disk activity, from the ISS agents running on the individual servers, and forwards it to the Dispatcher. The Dispatcher uses this load information, along with other sources of information, to determine which is the least-loaded server of the cluster and then routes the request.

2.6.9 ISS Selection Methods

For each observer you plan to use in your cell, you should specify a selection method, which defines how the ISS monitor selects the best node to perform that service. A third selection method has been added to ISS in this version of ND. The three selection methods- RoundRobin, Best and Statistical RoundRobin - are identified with the keywords `RoundRobin` , `Best` and `Statistical RoundRobin` respectively.

1. `RoundRobin`

   The load among servers is distributed on a round-robin basis; the load on each server is ignored, although a server will not be recommended if it appears to be down.

   The main advantage of using round-robin is that round-robin avoids overhead in the server associated with other measurement types. The main disadvantage of using round-robin is that ISS does not notice whether a particular service is getting very busy, and ISS could continue to use a busy server as the server to which new users connect.

2. `Best`

   For each service, the ISS monitor maintains a ranking of the best nodes to perform that service. Periodically, the monitor calculates the best server for every defined service and updates the ranking of the nodes. Before the next updating (that is, for the update period), ISS will use (or recommend to use) the highest server in the ranking.

   In order to establish the ranking, ISS monitors the load on each server and for a particular service selects the node that best performs that service. In this case, the node selection performed by ISS truly depends on the resources defined for the service.

3. `Statistical RoundRobin`

   This selection method is a new feature of ISS in WebSphere Performance Pack V2. It allows the server to handle extremely heavy traffic. For each update period, the monitor calculates the load on each server and assigns a proportion of tokens to that server. Each incoming request is assigned a token. If the number of tokens runs out before the next update, then the token levels are reset based on the latest available figures.

Server Failure

For all of the selection methods, ISS detects if a server fails and does not schedule requests for that service to that server.

Each service may have a different selection method without affecting any other services. A node may belong to more than one service and each of those services may have different selection methods without affecting any other.

2.6.10 Ports used by ISS

ISS uses four TCP ports for its communications, which we discuss in this section.

1. Port 7139

   As shown in Figure 12 on page 42 and Figure 13 on page 43, port 7139 is used for communication between the ISS monitor and agents. This can be changed with the `PortNumber` cell attribute in the configuration file or by using the -p option when starting ISS.

2. Port 12099

   This port is used to receive commands from the GUI. Use the `-g` option when starting ISS to change this port number.

Each of the defined Observers also must have a port specified to communicate with the ISS monitor. The three Observer types have these default ports used for this communication:

3. Port 53

   NameServer and ISSNameServer- DNS protocol communications

4. Port 10004

   ISS metric updates are communicated to the Dispatcher on this port.

These port numbers can be changed by modifying the ISS configuration file or by specifying a different port number value when the Observer is added to the configuration through the GUI. Additionally, in the case of the Dispatcher, see 2.5.4, “TCP Ports Used by the Dispatcher” on page 38 for instructions on how to inform the Dispatcher Manager of the new port number.

It is important to note that the initial transfer of the ISS configuration file to the other nodes in the cell, before ISS has been started for the first time on the other nodes, must be done by you. Once ISS has been started on the other nodes in the cell, changes made to the ISS configuration file through the configuration GUI or with the `isscontrol` command are automatically propagated to all the other machines participating in the cell. A demonstration of this feature is shown in Chapter 4, “ND Basic Scenarios” on page 81.

2.7 How the CBR Function Works

CBR works in conjunction with Web Traffic Express (WTE), whereby a client sends a request to a WTE proxy server that has been configured to make use of the CBR functionality; CBR must be installed and configured on the same machine as the WTE server. The WTE server queries the CBR component to see if it knows which server should serve the request. When CBR receives the request, it tries to match the request to a set of prioritized rules. If a match is found, CBR then selects the best server from a preconfigured set of servers, by load balancing the request. The *incoming* URL is then changed to point to the selected server.

If WTE caching is enabled, and a cache hit is found for the requested page, WTE will use the cached copy to fulfill the request. Otherwise, WTE proxies the request to the CBR-recommended server by using the translated *outgoing* URL. When the server responds back to the proxy server, the page can then be cached by its incoming or outgoing URL and sent back to the client.

CBR is very similar to the Dispatcher in its component structure. The three key functions of CBR (Executor, Manager, and Advisors) interact to balance and dispatch the incoming requests between servers. Along with load-balancing requests, the Executor monitors the number of new connections and active connections and supplies this information to the Manager. Notice that, unlike Dispatcher and ISS, CBR does not offer a high availability function. Nonetheless, multiple CBR servers can be load balanced by an ND server, which would detect if one of the CBR servers has failed and stop routing client requests to that CBR server, therefore allowing CBR high availability.

See Chapter 14, “Content Based Routing” on page 329 for details on how to install and configure a CBR environment.

2.7.1 Why Do I Need CBR?

The CBR component allows you to partition your site so that different content or application services can be served by different sets of servers. This partitioning will be transparent to clients accessing your site. By allowing multiple servers to be assigned to each type of content, you are protected if one server fails. CBR will recognize the failure and continue to load balance client requests to the other servers in the set.

CBR is a very powerful function of ND. Using CBR, local content for a Web site can be placed behind a WTE proxy server and the content can be served from different sets of servers based on the type of requests. For example, one or more Web servers could be set up to handle servlet requests, while other requests would be sent to a different set of Web servers. This allows you, for example, to choose a set of more powerful Web servers to serve applications that require a fast response, while other, less powerful Web servers in the same Web site can be used to serve static Web pages.

The set of servers that should handle particular requests can be specified based on regular expression matching of the content of the request. Because CBR allows you to specify multiple servers for each type of request, the requests can be load balanced for optimal client response.

2.7.2 Client Affinity with CBR

Some applications have been designed assuming a single-server environment. They keep state information in memory or on local disk across multiple URLs. When you put these applications on a cluster of servers with load balancing, subsequent connections go to a different server and the URL application cannot locate the state information. ND sticky port support is an attempt to help address this problem without correcting the application. Sticky port support overrides load balancing and sends all connections from the same user on a specific port back to the same server for a configured period of time. ND must assume that all connections from the same source IP address are potentially from the same user.

This works fine on an intranet where each source IP address represents a single-user machine. This assumption breaks down on intranets with multi-user machines (UNIX, DEC VMS, IBM OS/400, IBM VM, IBM MVS) but these are unusual clients in the Web workload.

More importantly, it breaks down the majority of the time on the Internet. For security and network cost reasons, most large ISPs and corporate networks go through either a proxy server or a SOCKS server before reaching the Internet. These both terminate the real clients connection and open a new connection to the target server. This concentrates the users behind them onto the single IP address of the proxy or SOCKS server. The result is that ND cannot distinguish whether two connections from the same source IP are independent users or the same user, so it must assume that all connections from that source are from a single user. For this reason, sticky ports largely defeat load balancing in an Internet environment. Note that this is a load issue, not a functional issue. The application works, but you wind up with severe hot spots because of the client IP address concentration.

CBR has two ways to maintain client affinity:

- *Client IP affinity*, based on the IP address of the client requesting a Web page
- *Cookie affinity*, obtained by setting a cookie on the client system

Both these kinds of affinity require a sticky time to be set; this indicates the time after which the affinity expires.

2.7.2.1 Client IP Affinity

Client IP affinity is based on the client IP address. This affinity works very well in an intranet environment, but it fails if client requests pass through a proxy or a SOCKS server. In fact, in this case client IP addresses are all masked by the proxy or firewall IP address; multiple requests coming from different clients would appear as coming from the same client, and would not be load balanced among the available Web servers. On the contrary, these requests would be directed to the same Web server.

Another limit of client IP affinity is evident in the case where multiple browser sessions are running on the same client machine. In this case, the requests coming from the browser sessions appear as coming from the same client, because the IP address is the same, and CBR would redirect them to the same Web server if client IP affinity is configured.

This demonstrates that CBR client IP affinity has the same disadvantages as the sticky port support provided by ND. However, while a sticky port, once configured, would be activated at each client’s request on that specific port, the advantage of CBR is that you can configure it to invoke client IP affinity only when it is really necessary. For example, you can configure CBR so that client IP affinity is activated every time the URL contains the string `purchase`.

2.7.2.2 Cookie Affinity

Cookie-based affinity solves the problems of client IP affinity and is particularly indicated for an Internet environment. The cookie-based affinity feature applies only to the CBR component and provides a new way to make clients *sticky* to a particular server. This function is enabled by setting the sticky time of a rule to a positive number and setting the affinity to **Cookie**. This can be done when the rule is added, or using the command:

```
cbrcontrol rule set
```

Once a rule has been enabled for cookie affinity, new client requests will be load-balanced using standard CBR algorithms, while succeeding requests from the same client will be sent to the initially chosen server. The chosen server is stored as a cookie in the response to the client. As long as the client’s future requests contain the cookie, and each request arrives within the sticky time interval, the client will maintain affinity with the initial server.

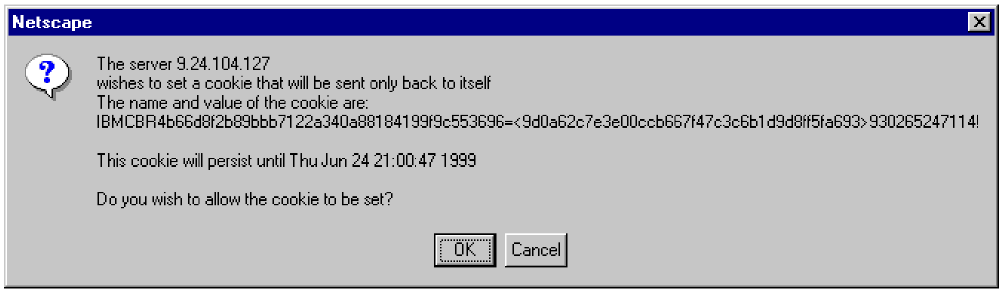
Cookie affinity is used to ensure that a client continues to be load balanced to the same server for some period of time. This is accomplished by sending a cookie to be stored by the client's browser. The cookie contains the *cluster*:port:*rule* information that was used to make the decision, the server that was load balanced to, and a timeout timestamp for when the affinity is no longer valid. Whenever a rule is found that has cookie affinity turned on, the cookies sent by the client are examined. If a cookie is found that contains the identifier for the *cluster*:*port*:*rule* information, then the server load balanced to and the timeout timestamp are extracted from the cookie. If the server is still in the set used by the rule, its weight is greater than zero, and the timeout timestamp is greater than now, then the server in the cookie is chosen to load balance to. If any of the preceding three conditions are not met, a server is chosen using the normal algorithm. Once a server has been chosen (using either of the two methods) a new cookie is constructed with the five pieces of information.

A cookie set on the client machine always starts with the string `IBMCBR`. The *cluster*:*port*:*rule* information and the server chosen to load balance to are encoded so that no information about the CBR configuration is revealed.

An `Expires` parameter is also inserted in the cookie. This parameter is in a format the browser can understand, and causes the cookie to become invalid two hours after the timeout timestamp. This is so the clients' cookie database is not cluttered up.

This new cookie is then inserted in the headers that go back to the client, and if the client's browser is configured to accept cookies, it will send it back in subsequent requests.

The following figure shows what a user in the client system would see when CBR sends a cookie to the client and the browser is configured to warn the user before accepting cookies:



The server 9.24.104.127
wishes to set a cookie that will be sent only back to itself
The name and value of the cookie are:
IBMCBR4b66d8f2b89bbb7122a340a88184199f9c553696=<9d0a62c7e3e00ccb667f47c3c6b1d9d8ff5fa693>930265247114!

This cookie will persist until Thu Jun 24 21:00:47 1999

Do you wish to allow the cookie to be set?

Figure 14. CBR Cookie Warning

Cookie affinity does not work if cookie support is absent or has been disabled in the browser.

Chapter 3. ND Installation and Basic Configuration Issues

This chapter shows how to install IBM SecureWay Network Dispatcher (ND), the Load Balancing component of IBM WebSphere Performance Pack V2. In addition to the installation procedures, this chapter also describes the configuration methods offered by this new version of ND.

3.1 Installation of ND

ND is supported on three operating systems: IBM AIX 4.2.1 or later, Microsoft Windows NT 4.0 Server or Workstation, and Sun Solaris 2.6 or later.

In this section we show you step by step how to perform the installation on AIX, Solaris and Windows NT using the WebSphere Performance Pack Version 2 Java InstallShield.

3.1.1 Installation on UNIX Systems

In this section we describe the installation of ND on UNIX systems. The installation of WebSphere Performance Pack V2 can be performed through a Java program, and there are no particular differences when installing on AIX and Solaris systems. For this reason, we will focus only on AIX systems in this section.

For the AIX installation, we used a uniprocessor IBM RS/6000 43P with 192 MB of RAM, 2.2 GB of hard disk and one token-ring interface. AIX Version 4.3.1 was installed on the machine.

The AIX installation program for WebSphere Performance Pack Version 2 makes use of the Java InstallShield’s setup class. For this reason you are required to pre-install the Java Runtime Environment (JRE) Version 1.1.6 or later. The JRE is part of the Java Development Kit (JDK) Version 1.1.6. In effect, you don’t need the full JDK, but only its subset known as JRE, which contains just the Java Virtual Machine (JVM), the Java platform core classes, and supporting files. In other words, the JRE is the smallest set of executables and files that constitute the standard Java platform and it contains only the run-time part of the JDK: no compiler, no debugger, and no tools.

Use the following command to determine whether or not Java is installed on your machine:

```
java -version
```

If Java is not installed on your machine or installed at a lower level than 1.1.6, you can find the install image for JDK 1.1.6 for AIX at `http://www.ibm.com/java/jdk/download/index.html` or on the IBM WebSphere Performance Pack Version 2 CD (in the jdk/aix directory). The installation of JDK on AIX is described in the IBM redbook *Network Computing Framework Component Guide,* SG24-2119. Notice that if the level of Java on your AIX system is 1.1.6, it is necessary to install the patch IX83246, or the Dispatcher Manager will not start. This additional requirement does not apply with later versions of Java.

In our case, we used just the JRE provided by the filesets:

- Java.rte.bin
- Java.rte.classes
- Java.rte.lib

To prepare for the installation of ND on AIX, follow the steps listed below:

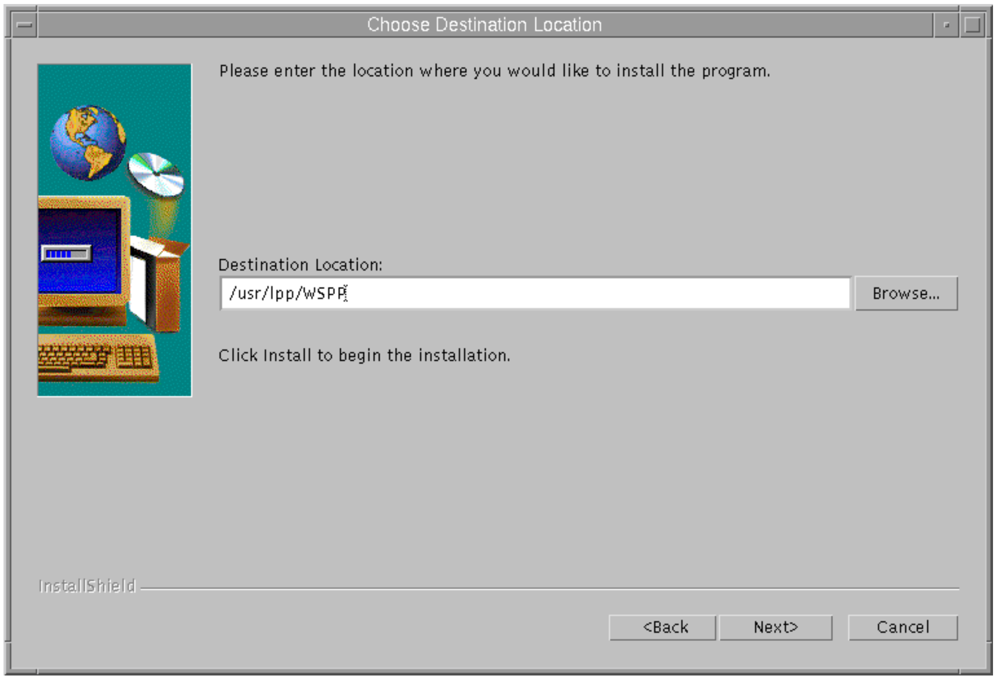
1. Insert the IBM WebSphere Performance Pack Version 2 CD in the CD-ROM drive
2. From a command line, enter the following commands:

   ```
   mkdir /cdrom
   mount -rv cdrfs /dev/cd0 /cdrom
   cd /cdrom/aix
   ```

3. To start the Java InstallShield installation program, enter:

   ```
   java setup
   ```

4. The first screen you will see is the Welcome window. After clicking the **Next** button, we saw another window displaying the IBM WebSphere Performance Pack Version 2 Readme file. We suggest that you take a look at the file, since it contains interesting information about the product. After clicking the **Next** button, we were prompted to enter the destination location or accept the default. Note that this is a working directory for WebSphere Performance Pack. We accepted the default location of /usr/lpp/WSPP.


Choose Destination Location

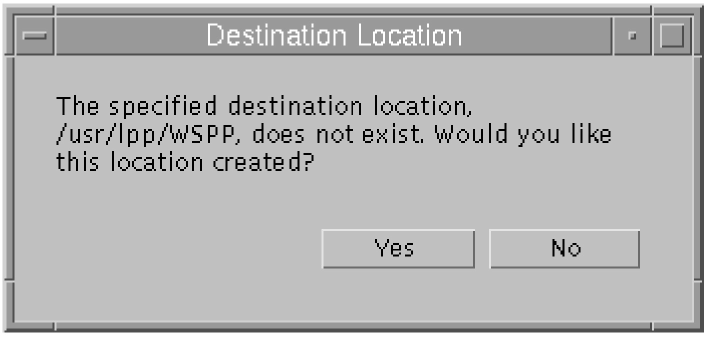
Please enter the location where you would like to install the program.

Destination Location: /usr/lpp/WSPP

Click Install to begin the installation.

Figure 15. Choose Destination Location (aNDinst01)

5. In the above window, you are prompted to click **Install** to proceed. Such a button does not exist, due to a known InstallShield for Java problem. Click the **Next** button to continue instead. If the destination directory does not exist on your system, you will be presented with a question dialog asking you if you would like the location to be created.


Destination Location

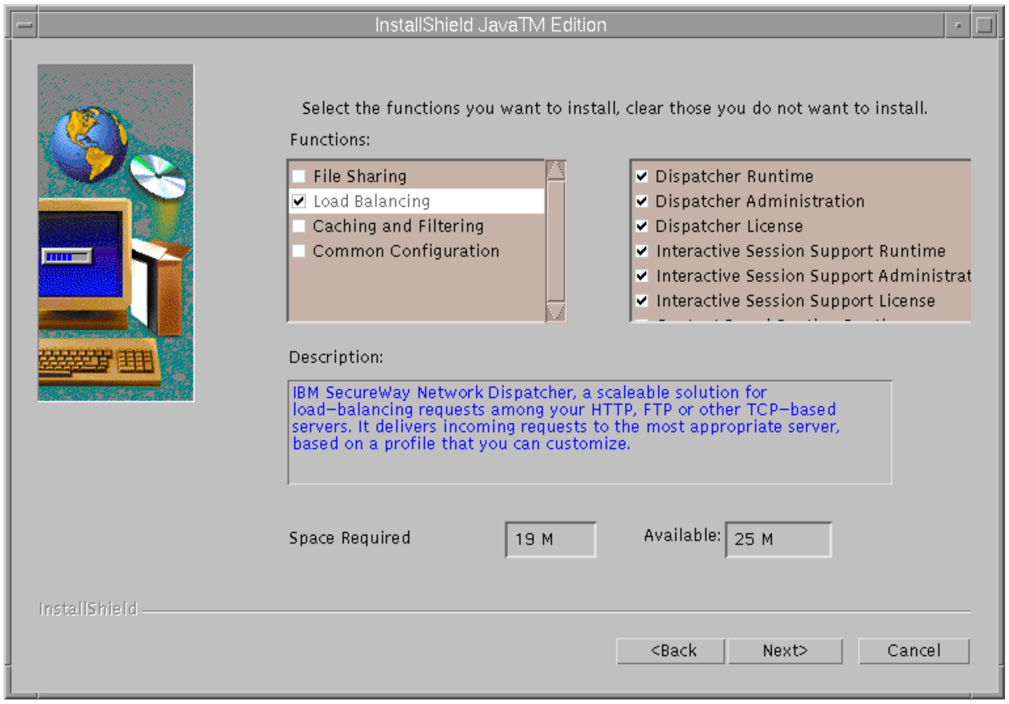
The specified destination location, /usr/lpp/WSPP, does not exist. Would you like this location created?

Figure 16. Location Creation Confirmation Dialog (aNDinst01a)

6. Then you will be presented with a window allowing you to choose which IBM WebSphere Performance Pack V2 components you want to install:

- File Sharing
- Load Balancing
- Caching and Filtering
- Common Configuration

We selected **Load Balancing**, and a scrolled list of ND components was added to the window, as shown in the following figure:


InstallShield JavaTM Edition

Select the functions you want to install, clear those you do not want to install.

Functions:

- [ ] File Sharing
- [✓] Load Balancing
- [ ] Caching and Filtering
- [ ] Common Configuration

- [✓] Dispatcher Runtime
- [✓] Dispatcher Administration
- [✓] Dispatcher License
- [✓] Interactive Session Support Runtime
- [✓] Interactive Session Support Administrat
- [✓] Interactive Session Support License

Description:

IBM SecureWay Network Dispatcher, a scaleable solution for load-balancing requests among your HTTP, FTP or other TCP-based servers. It delivers incoming requests to the most appropriate server, based on a profile that you can customize.

Space Required 19 M Available: 25 M

Figure 17. Choose Components to Install

Note that as you select each subcomponent in the selection list, a description of that component is displayed as well as the space required to install all the selected components. This window also shows you how much space is available in the destination location file system.

The ND components that appear on that list are:

- Dispatcher Runtime
- Dispatcher Administration
- Dispatcher License
- Interactive Session Support Runtime
- Interactive Session Support Administration
- Interactive Session Support License
- Content Based Routing Runtime
- Content Based Routing Administration
- Content Based Routing Runtime
- User's Guide

ND Documentation on AIX

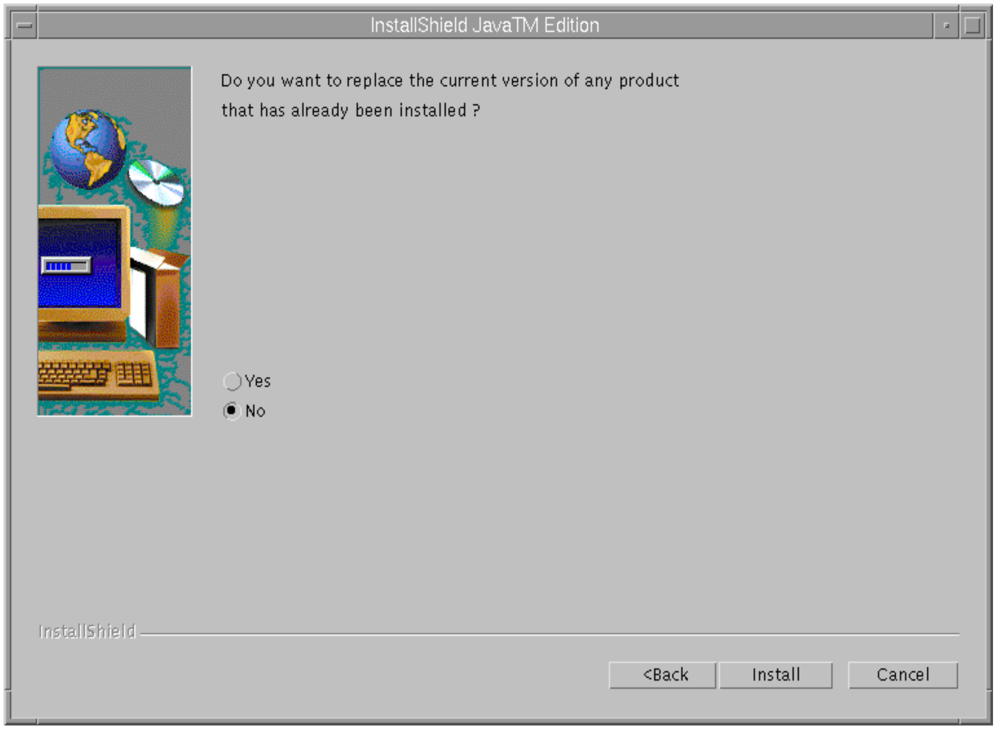
Figure 17 on page 57 shows that we selected the three subcomponents that make up the Dispatcher and the three subcomponents that make up the ISS. Not shown is that we also selected the User's Guide for installation. The User's Guide subcomponent contains the *SecureWay Network Dispatcher User's Guide v2.1 for AIX, Solaris and Windows NT*, GC31-8496, in HTML and PDF formats. After installing the User's Guide, we were able to access this book with a local browser at file://usr/lpp/nd/documentation/ndugv2r1.htm or file://usr/lpp/nd/documentation/ndugv2r1.pdf.

Content Based Routing

If you select the **Content Based Routing Runtime** ND component, the **Caching and Filtering** WebSphere Performance Pack Version 2 component is automatically selected to be installed, as Content Based Routing (CBR) cannot be installed without it. For more details on CBR, see 2.7, "How the CBR Function Works" on page 51 and Chapter 14, "Content Based Routing" on page 329.

The ability to select individual ND components for installation via the Java InstallShield is a new feature in WebSphere Performance Pack Version 2. In the previous version of WebSphere Performance Pack, `smitty` or `installp` were required to install individual ND components. However, for those of you who prefer to use the command line, the option to use `smitty` or `installp` is still available.

7. After you click **Next**, you will see a window that asks if you want the installation program to replace other programs already installed on your system. We selected **No** in this case, since we had not installed any programs on our system yet.



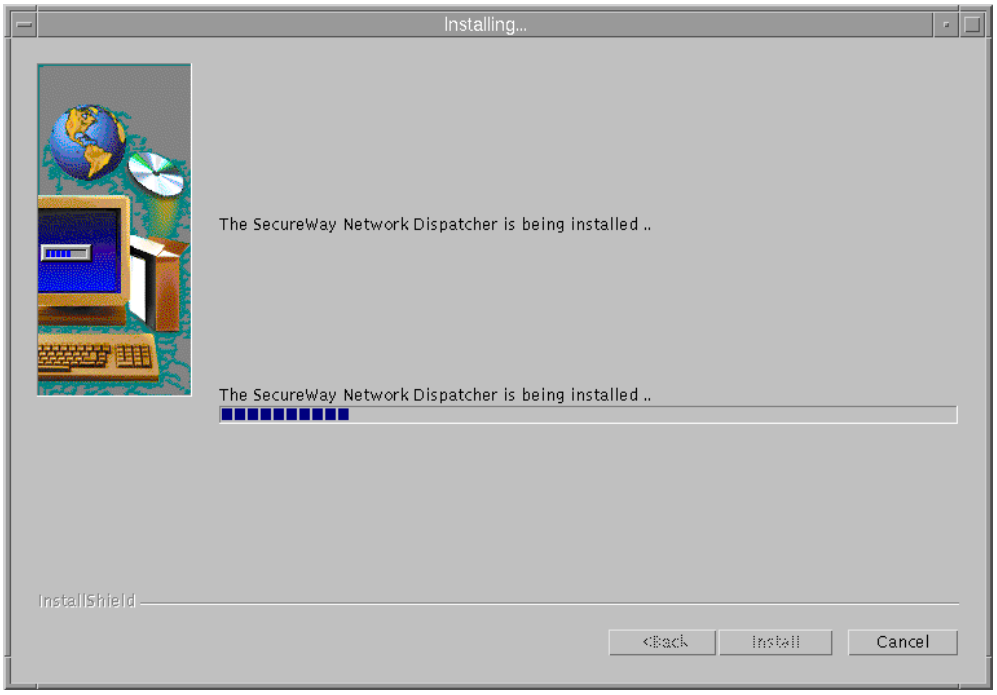
InstallShield JavaTM Edition

Do you want to replace the current version of any product that has already been installed ?

Yes [✓ No]

Figure 18. Choose to Replace Version (aNDinst05)

8. When we clicked **Install** the following window was displayed:



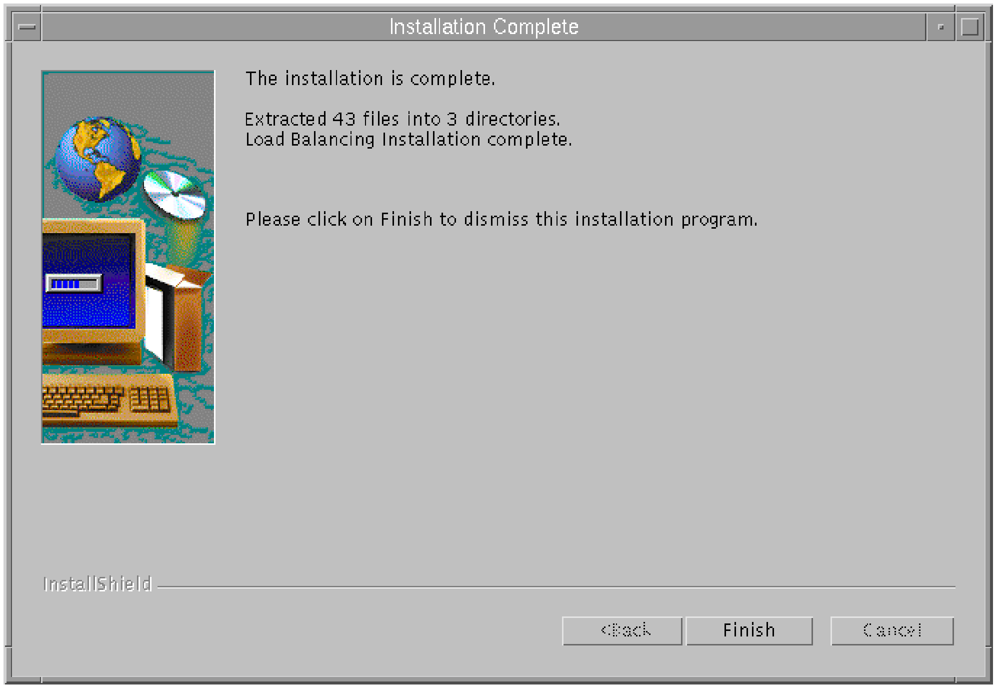
Installing...

The SecureWay Network Dispatcher is being installed ..

The SecureWay Network Dispatcher is being installed ..

Figure 19. SecureWay Network Dispatcher Installation Screen

9. At this point, the Java install terminated with this successful completion message window:


Installation Complete

The installation is complete.

Extracted 43 files into 3 directories.
Load Balancing Installation complete.

Please click on Finish to dismiss this installation program.

Figure 20. Installation Completed Successfully

10.The Java InstallShield uses `installp` to install the selected components. Because of this, you are able to use `lslpp` to verify that the filesets have been successfully installed. We verified that the product had been installed by entering the following command:

```
lslpp -h | grep intnd
```

By entering the command above, the list of ND filesets installed when all of the ND components are selected is displayed, and you should see the following output:

```
intnd.admin.rte
intnd.cbr.license
intnd.cbr.rte
intnd.cbradmin.rte
intnd.doc.en_US
intnd.iss.license
intnd.iss.rte
intnd.issadmin.rte
intnd.msg.en_US.cbr.rte
intnd.msg.en_US.cbradmin.rte
intnd.msg.en_US.iss.rte
intnd.msg.en_US.issadmin.rte
intnd.msg.en_US.nd.rte
intnd.msg.en_US.ndadmin.rte
intnd.nd.driver
intnd.nd.license
intnd.nd.rte
intnd.ndadmin.rte
```

Each of the subcomponent's message filesets are installed in the language that the locale of the machine was set to when the `java setup` command was run.

At this point you are ready to configure ND to load balance your servers.

3.1.2 Installation on Windows NT

In this section we describe the steps that were necessary to install the ND component of IBM WebSphere Performance Pack Version 2 on our Windows NT platform.

The machine where we performed this installation was an IBM PC 750 with 166 MHz of CPU, 96 MB of RAM, 1.5 GB of hard disk and one token-ring adapter. This machine had been installed with Windows NT Server 4.0 and Service Pack 3. Service Pack 4 and 5 are supported as well.

The Windows NT installation program for each component of IBM WebSphere Performance Pack makes use of Java InstallShield's setup class. For this reason you are required to pre-install the JRE, which is a subset of the JDK, as we explained in 3.1.1, "Installation on UNIX Systems" on page 55. Unlike our experience with ND on AIX, we found that on our Windows NT system we needed the full JDK to be installed in order to install ND.

The IBM WebSphere Performance Pack Version 2 CD contains JDK 1.1.6 for Windows NT in the nt\jdk directory. The latest version of the JDK can also be downloaded for free from the JavaSoft Web site `http://www.javasoft.com`. On our platform we used JDK 1.1.6.

For further details on installing the JDK for Windows NT, see the IBM redbook *Internet Security in the Network Computing Framework*, SG24-5220.

In order to perform the following steps, you must be a Windows NT system administrator. To install ND, we followed the steps listed below:

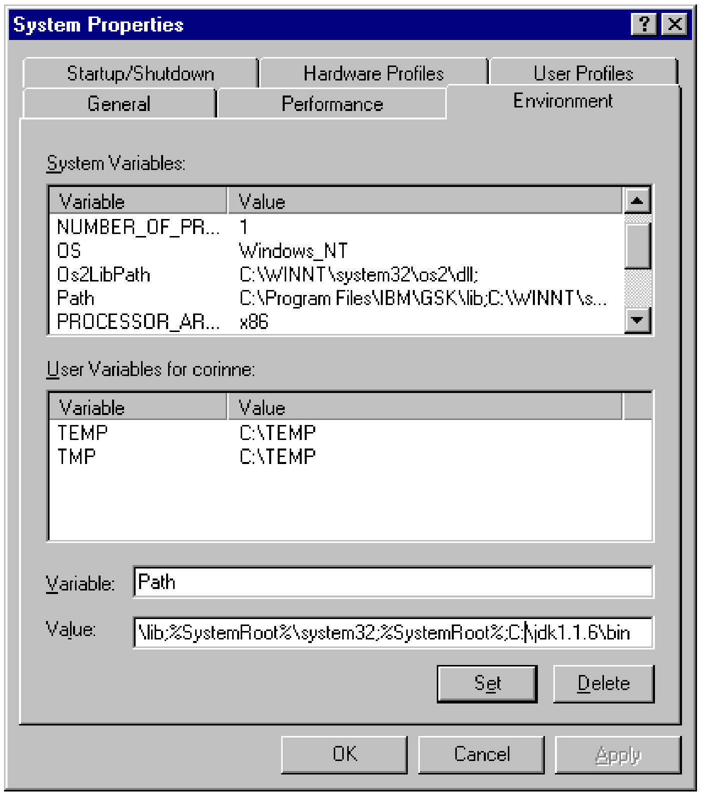
1. We inserted the IBM WebSphere Performance Pack Version 2 CD in the CD-ROM drive.
2. To start the Java InstallShield installation program, we entered the following command from a Command Prompt window:

```
E:\ND\java setup
```

1. E in our case was the letter assigned to the CD-ROM drive. We saw this message:

```
The name specified is not recognized as an
internal or external command, operable program or batch file.
```

2. This error message is displayed when you launch the `java` command, but the Java bin directory containing all the Java executable files was not in our Path system environment variable. JDK 1.1.6 is installed by default in the directory C:\jdk1.1.6, and this was also the location where it was installed on our system, so we appended the location of the java.exe executable (in this case `C:\jdk1.1.6\bin`) to the value of the Path system environment variable, accessible through the System dialog box of the Control Panel folder:



System Variables:

Variable	Value
NUMBER_OF_PR...	1
OS	Windows_NT
Os2LibPath	C:\WINNT\system32\os2\dll;
Path	C:\Program Files\IBM\GSK\lib;C:\WINNT\s...
PROCESSOR_AR...	x86

User Variables for corinne:

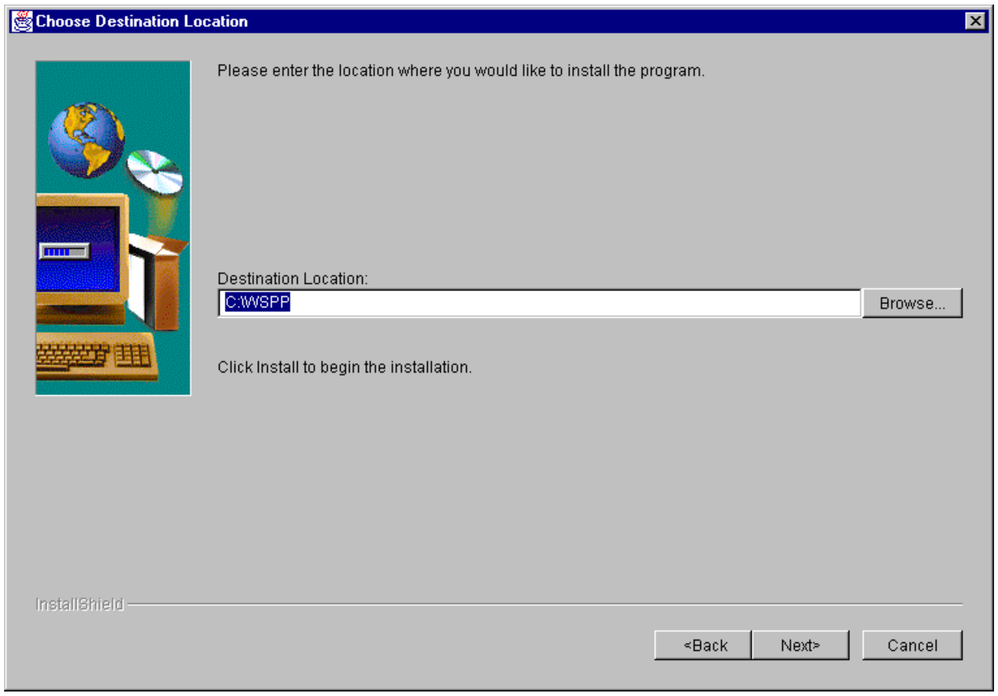
Variable	Value
TEMP	C:\TEMP
TMP	C:\TEMP

Variable: Path

Value: \lib;%SystemRoot%\system32;%SystemRoot%;C:\jdk1.1.6\bin

Figure 21. Adding the Java bin Directory to the Java Environment Variable

3. After setting the Path system environment variable correctly, we launched the ND install again and saw the Welcome window. After clicking the **Next** button, we saw another window displaying the IBM WebSphere Performance Pack Version 2 Readme file. We suggest that you take a look at the file as it contains interesting information about the product.
4. After clicking the **Next** button, we were prompted to enter the destination location or accept the default C:\WSPP. We accepted the default location, as shown in the following figure.


Choose Destination Location

Please enter the location where you would like to install the program.

Destination Location:

C:\WSPP

Click Install to begin the installation.

Figure 22. Choose Destination Location

5. In the above window, you are prompted to click **Install** to proceed. However, the install button does not exist. As we already said in 3.1.1, “Installation on UNIX Systems” on page 55, this is a known InstallShield for Java problem, and so we clicked the button labeled Next to continue instead. If the destination directory does not exist on your machine, you will receive a window asking you if you would like it to be created. Then we were presented with a window allowing us to choose which IBM WebSphere Performance Pack V2 components we wanted to install.

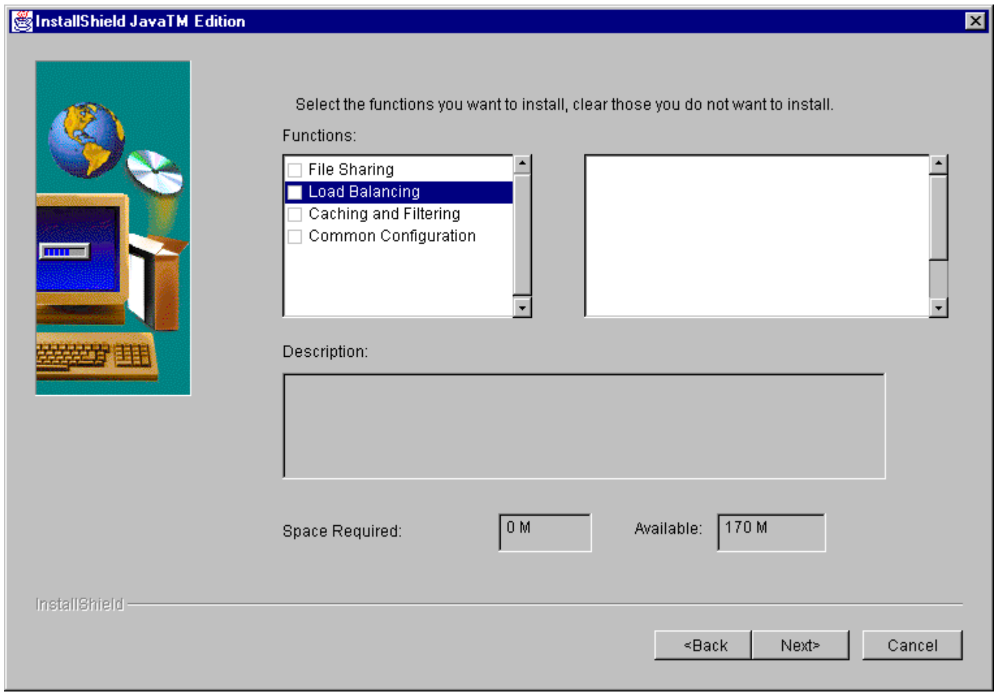
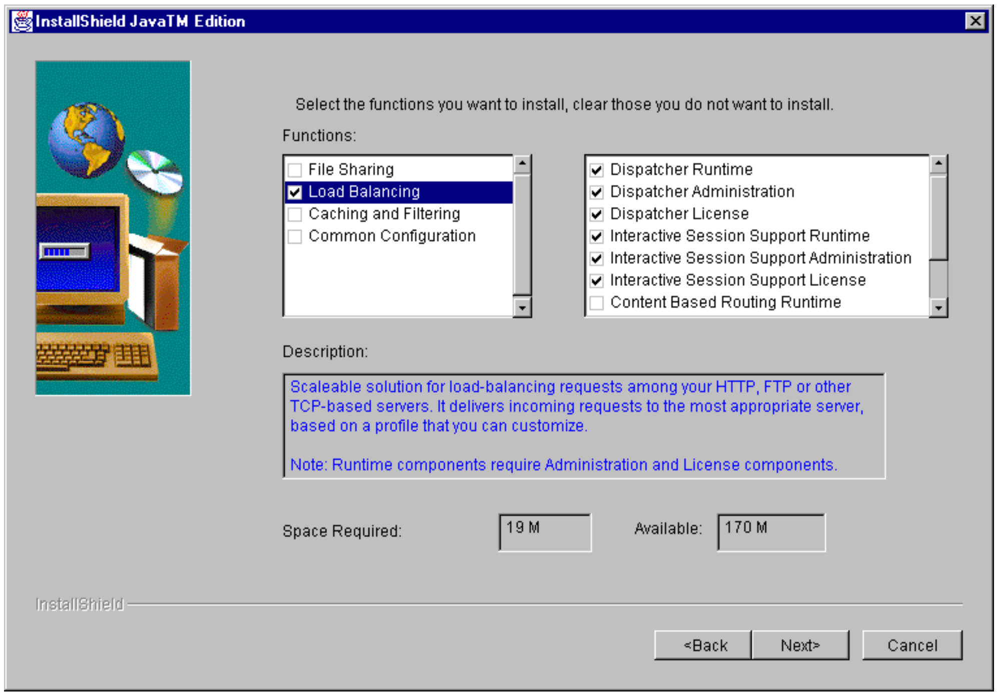


Figure 23. Select the Load Balancing Component to Install

6. We selected **Load Balancing**, and a scrolled list of ND subcomponents was added to the window:


InstallShield JavaTM Edition
Select the functions you want to install, clear those you do not want to install.
Functions:

- [ ] File Sharing
- [✓] Load Balancing
- [ ] Caching and Filtering
- [ ] Common Configuration

- [✓] Dispatcher Runtime
- [✓] Dispatcher Administration
- [✓] Dispatcher License
- [✓] Interactive Session Support Runtime
- [✓] Interactive Session Support Administration
- [✓] Interactive Session Support License
- [ ] Content Based Routing Runtime

Description:

Scaleable solution for load-balancing requests among your HTTP, FTP or other TCP-based servers. It delivers incoming requests to the most appropriate server, based on a profile that you can customize.

Note: Runtime components require Administration and License components.

Space Required: 19 M Available: 170 M

Figure 24. ND Component Selection List

Note that as you select each subcomponent in the selection list, a description of that component is displayed as well as the space required to install all the selected components. This window also shows you how much space is available in the destination location file system.

The ND components that appear on that list are:

- Dispatcher Runtime
- Dispatcher Administration
- Dispatcher License
- Interactive Session Support Runtime
- Interactive Session Support Administration
- Interactive Session Support License
- Content Based Routing Runtime
- Content Based Routing Administration
- Content Based Routing Runtime
- User's Guide

ND Documentation on NT

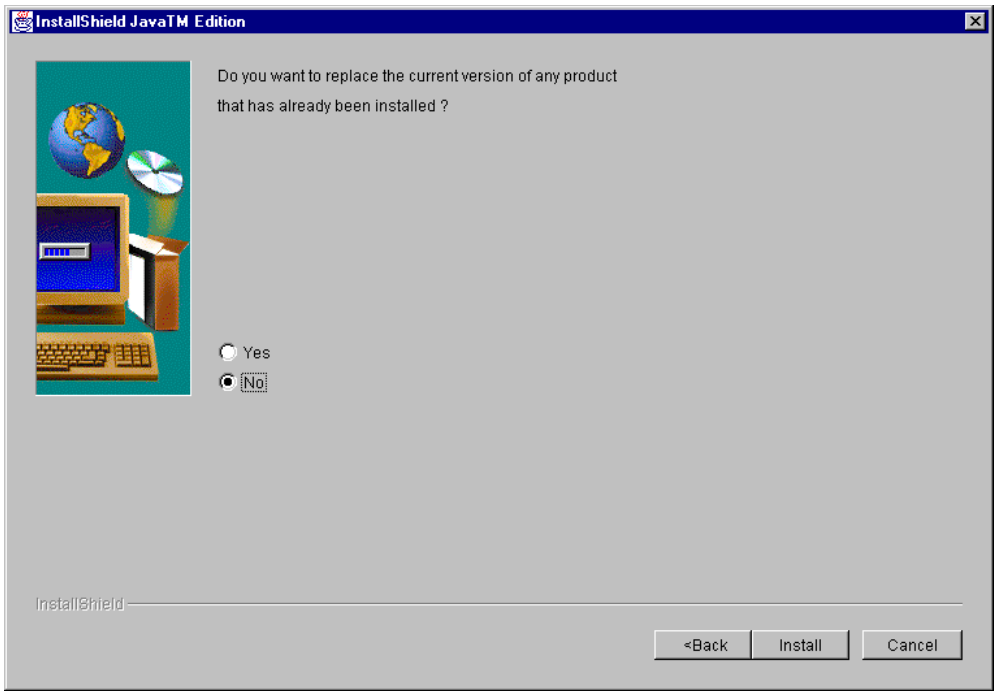
Figure 24 on page 64 shows that we selected the three subcomponents that make up the Dispatcher and the three subcomponents that make up the ISS. Not shown is that we also selected the User’s Guide for installation. The User’s Guide subcomponent contains the *SecureWay Network Dispatcher User’s Guide v2.1 for AIX, Solaris and Windows NT*, GC31-8496 in HTML and PDF formats. After installing the User’s Guide, we were able to accessed this book with a local browser at file://C:/WSPP/nd/documentation/ndugv2r1.htm or file://C:/WSPP/nd/documentation/ndugv2r1.pdf.

Content Based Routing

If you select the **Content Based Routing Runtime** ND components, the **Caching and Filtering** WebSphere Performance Pack Version 2 component is automatically selected to be installed, as CBR cannot be installed without it. For more information about CBR, see 2.7, “How the CBR Function Works” on page 51 and Chapter 14, “Content Based Routing” on page 329.

The ability to select individual ND components for installation via the Java InstallShield is a new feature in WebSphere Performance Pack Version 2. In the previous version of WebSphere Performance Pack, a non-Java custom install was required to install individual ND components.

7. We selected the three Dispatcher and the three Interactive Session Support (ISS) components to continue with the installation. After we clicked **Next**, we were asked if the installation program should replace the current version of any product already installed on the system. We selected **No** in this case, since we had not installed any programs on our system yet:

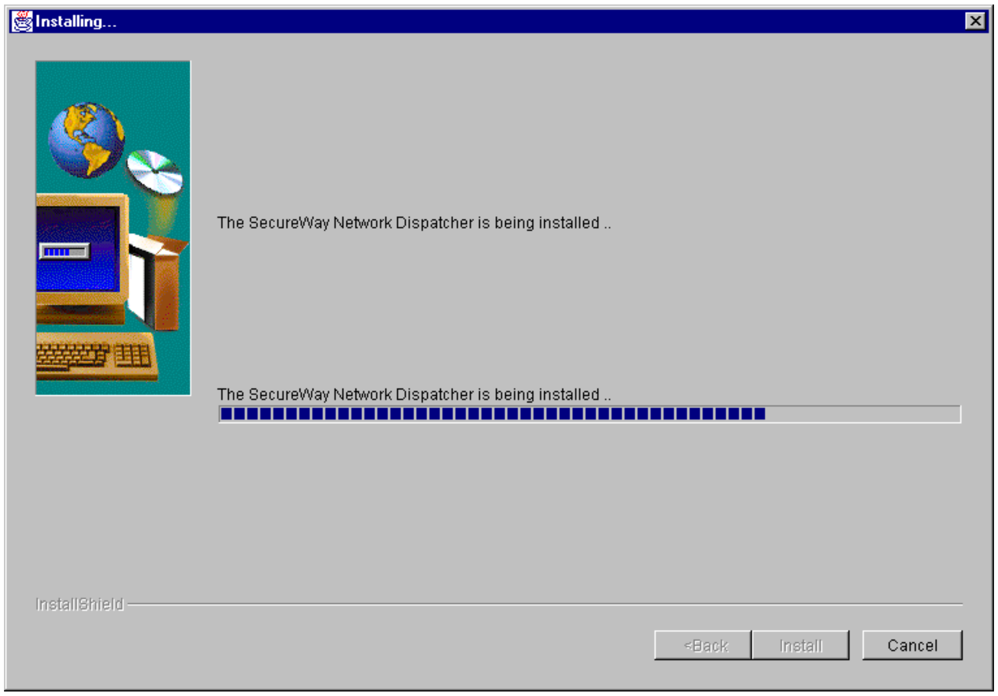

InstallShield JavaTM Edition

Do you want to replace the current version of any product that has already been installed ?

Yes [✓ No]

Figure 25. Choose to Replace Version

8. When we clicked the **Install** button, we were notified of the progress of the installation as shown in the following screen:

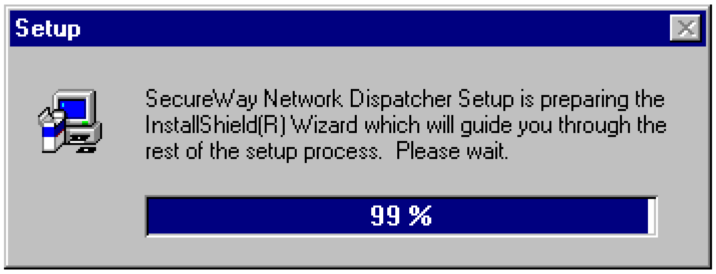

Installing...

The SecureWay Network Dispatcher is being installed ..

The SecureWay Network Dispatcher is being installed ..

Figure 26. IBM SecureWay Network Dispatcher Installation Screen

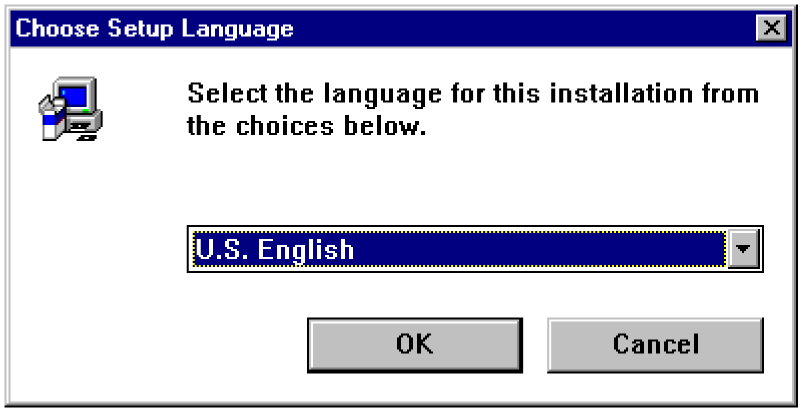
9. At the same time that the progress indicator window was visible, we also saw the InstallShield Setup Notification (see Figure 27) and the Language Selection Window (see Figure 28).


Setup

SecureWay Network Dispatcher Setup is preparing the InstallShield(R) Wizard which will guide you through the rest of the setup process. Please wait.

99 %

Figure 27. InstallShield Setup Notification Window

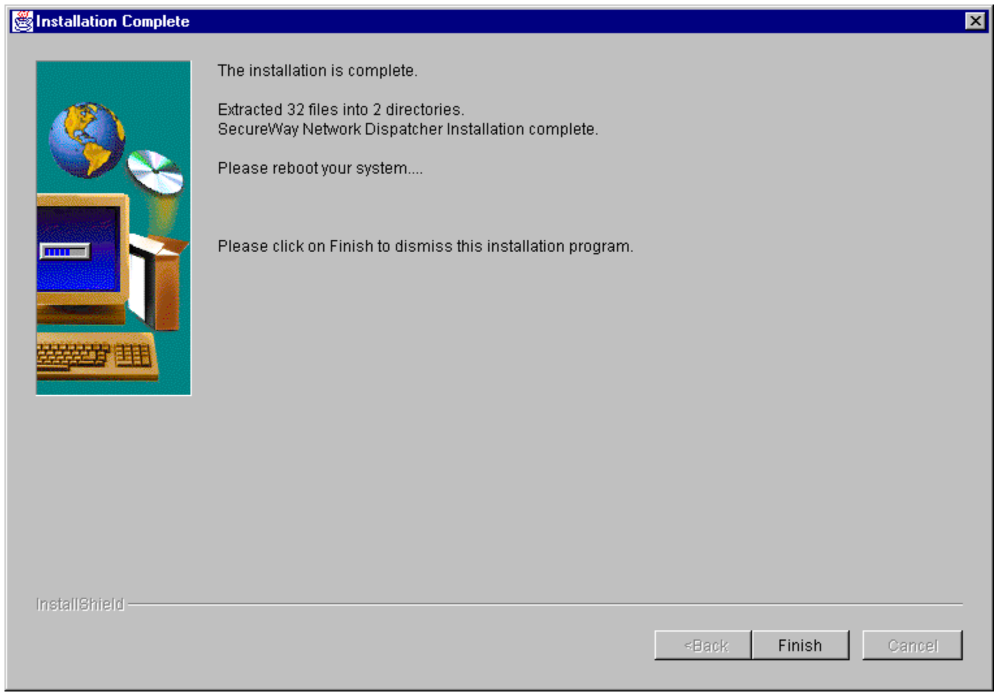

Choose Setup Language

Select the language for this installation from the choices below.

U.S. English

Figure 28. Language Selection Window

10.We selected **U.S.English** and clicked the **OK** button. Shortly, this successful installation notification window was displayed along with a Reboot Message window (see Figure 30):


Installation Complete

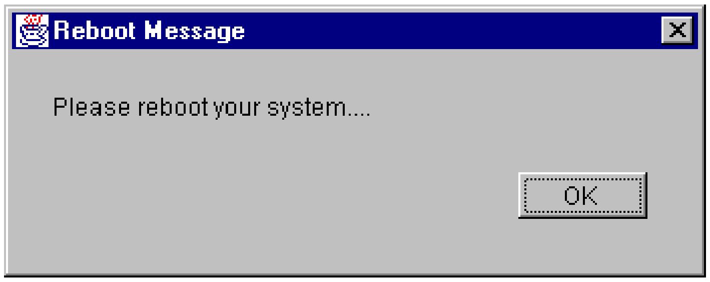
The installation is complete.

Extracted 32 files into 2 directories.
SecureWay Network Dispatcher Installation complete.

Please reboot your system....

Please click on Finish to dismiss this installation program.

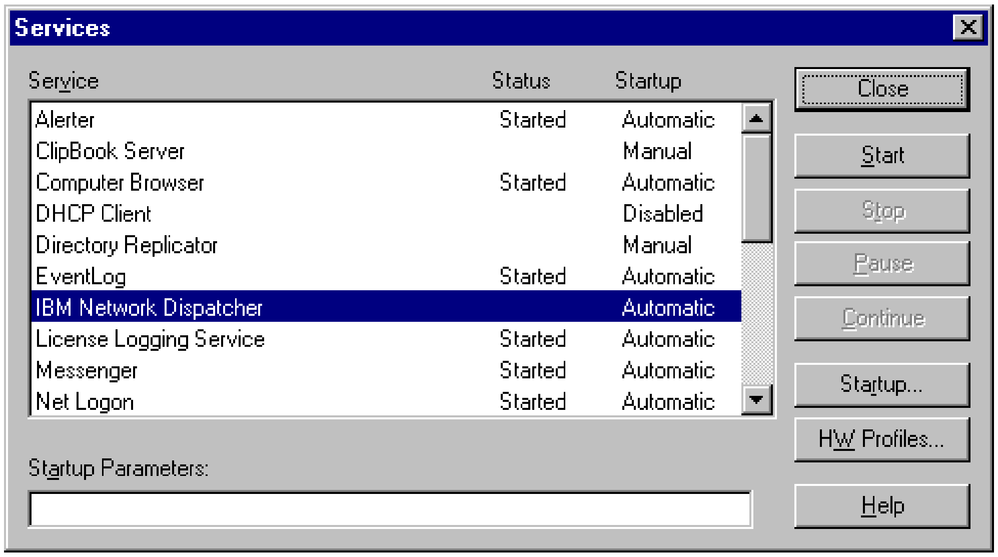
Figure 29. ND Successful Installation Notification


Reboot Message

Please reboot your system....

Figure 30. Reboot Notification

The IBM Network Dispatcher process should automatically start when the machine is rebooted. After the system restarts, check the Services folder of the Control Panel to verify that the ND has been started. Our Services window appeared as follows:


Services

Service	Status	Startup
Alerter	Started	Automatic
ClipBook Server		Manual
Computer Browser	Started	Automatic
DHCP Client		Disabled
Directory Replicator		Manual
EventLog	Started	Automatic
IBM Network Dispatcher		Automatic
License Logging Service	Started	Automatic
Messenger	Started	Automatic
Net Logon	Started	Automatic

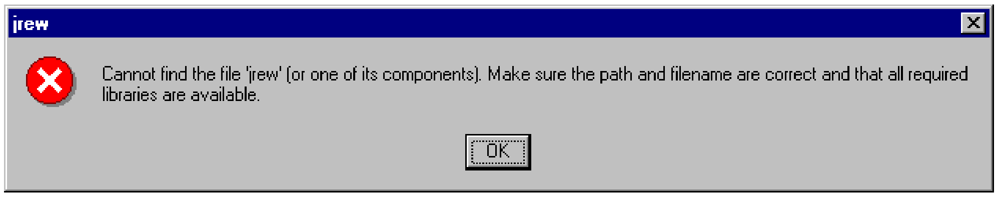
Startup Parameters:

Figure 31. Services Panel Showing that the IBM Network Dispatcher Has Not Been Started

In our case, the status of the IBM Network Dispatcher is not started. This was due to the fact that we were not logged on as a Windows NT administrator. In order to launch the IBM Network Dispatcher process, we needed to log on as a user that had administrator authority.

While we were still logged on without administrator authority, we could not launch the SecureWay Network Dispatcher configuration graphical user interface (GUI).[1] We tried to start the ND GUI from the Start menu by selecting **Programs**, **IBM WebSphere**, **Performance Pack** and finally **SecureWay Network Dispatcher**, but received the following error:

[1] Notice that in order to start the ND GUI, the ND server must be running. Therefore, even if you were logged on as Administrator (or a user with Administrator authority), you would not have been able to start the GUI if IBM Network Dispatcher was set to start manually.


jrew

Cannot find the file 'jrew' (or one of its components). Make sure the path and filename are correct and that all required libraries are available.

Figure 32. The Error Seen When Trying to Start ND without the Correct Authorization

11. After we logged off and back on with a user ID that had administrator authority, the IBM Network Dispatcher was automatically started. We were then able to launch the ND GUI and start to configure the Dispatcher and ISS components.

3.1.3 SecureWay Network Dispatcher Default Installation Directories

The following table shows the default installation directory for each of the ND components on each of the supported operating system platforms. The entries in the table are valid when the WebSphere Performance Pack Version 2 Java InstallShield method of installing the components is used (as shown in 3.1, "Installation of ND" on page 55).

Table 1. ND Default Installation Base Directories

Platform	**SecureWay Network Dispatcher Components**			
	Dispatcher	*ISS*	*CBR*	*Admin*
AIX	/usr/lpp/nd/dispatcher	/usr/lpp/nd/iss	/usr/lpp/nd/cbr	/usr/lpp/nd/admin
Solaris	/opt/nd/dispatcher	/opt/nd/iss	/opt/nd/cbr	/opt/nd/admin
Windows NT	C:\WSPP\IBM\nd\dispatcher	C:\WSPP\IBM\nd\iss	C:WSPP\IBM\nd\cbr	C:\WSPP\IBM\nd\admin

Note that if instead of using the WebSphere Performance Pack setup program (the Java InstallShield method as shown in 3.1, "Installation of ND" on page 55) to install the components, you used the SecureWay Network Dispatcher setup program in the \nt\nd subdirectory of the WebSphere Performance Pack Version 2 installation CD on Windows NT, then the install base directories will be different from what is shown in the above table. In that case, Program Files is substituted for the WSPP element of the path. AIX and Solaris install directories are the same regardless of the installation method used.

3.2 Deinstallation of IBM SecureWay Network Dispatcher

At this time the Java InstallShield cannot be used to deinstall ND or any of its subcomponents. The methods used to deinstall ND or any of its subcomponents vary on each of the three platforms. We will discuss how to perform a deinstallation on AIX, Solaris and Windows NT.

3.2.1 Deinstallation on UNIX Systems

For the reasons we explained in 3.1.1, "Installation on UNIX Systems" on page 55, we will focus only on AIX systems.

To uninstall ND on AIX is as simple as typing:

```
installp -u intnd
```

In order for you to deinstall ND or any of its subcomponents with `smitty`, it is necessary for you to know the names of the filesets that make up each of the ND subcomponents. For demonstration purposes we installed all of the ND components. The following table shows the ND AIX filesets that could be deinstalled and which function they are associated with:

Table 2. ND Function Fileset Names on AIX

Function	**Fileset**
Dispatcher	intnd.nd.license intnd.nd.rte intnd.nd.driver intnd.ndadmin.rte intnd.msg.en_US.nd.rte intnd.msg.en_US.ndadmin.rte
ISS	intnd.iss.license intnd.iss.rte intnd.issadmin.rte intnd.msg.en_US.iss.rte intnd.msg.en_US.issadmin.rte
CBR	intnd.cbr.license intnd.cbr.rte intnd.cbradmin.rte intnd.msg.en_US.cbr.rte intnd.msg.en_US.cbradmin.rte
Base Administration	intnd.admin.rte intnd.msg.en_US.admin.rte
User's Guide	intnd.doc.en_US

Base Administration Filesets

The Base Administration fileset provides the ND admin GUI. Both it and its associated message fileset are prerequisites of the three component admin filesets:

- intnd.ndadmin.rte
- intnd.issadmin.rte
- intnd.cbradmin.rte

You can verify the dependency on your system with the command:

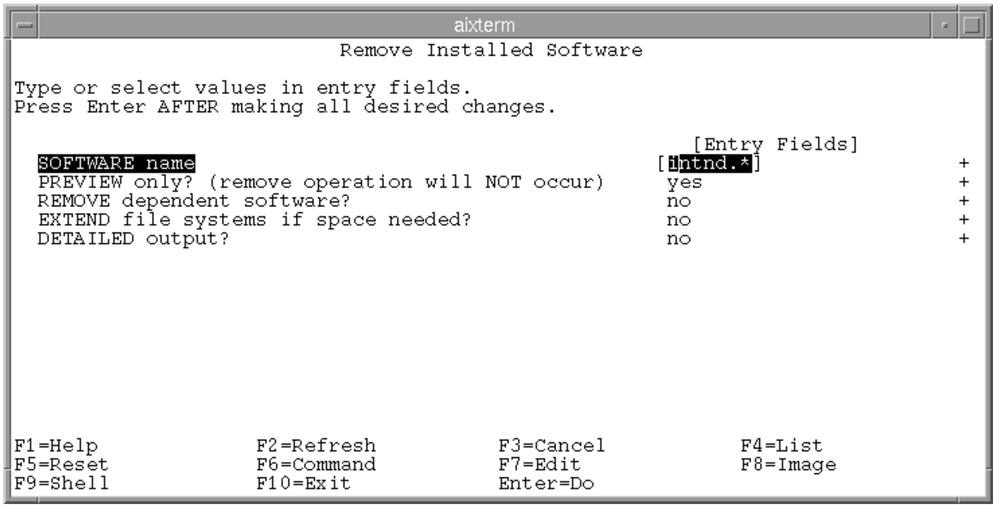
```
lslpp -d intnd.admin.rte
```

The filesets installed on your machine may differ from this list if not all the components are installed. The language component of the message filesets may also not be en_US on your machine. To deinstall any or all of the ND filesets, use the command:

```
smitty remove
```

In the `SOFTWARE name` field of the Remove Installed Software panel, you can type the names of the filesets you want to deinstall or press the F4 key to display a list of installed filesets for you to select from.

As you can see in Figure 33 on page 71, you can use the wildcard character (*) in the `SOFTWARE name` field:



```
                       Remove Installed Software

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                    [Entry Fields]
  SOFTWARE name                                    [intnd.*]                +
  PREVIEW only? (remove operation will NOT occur)   yes                     +
  REMOVE dependent software?                        no                      +
  EXTEND file systems if space needed?              no                      +
  DETAILED output?                                  no                      +









F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

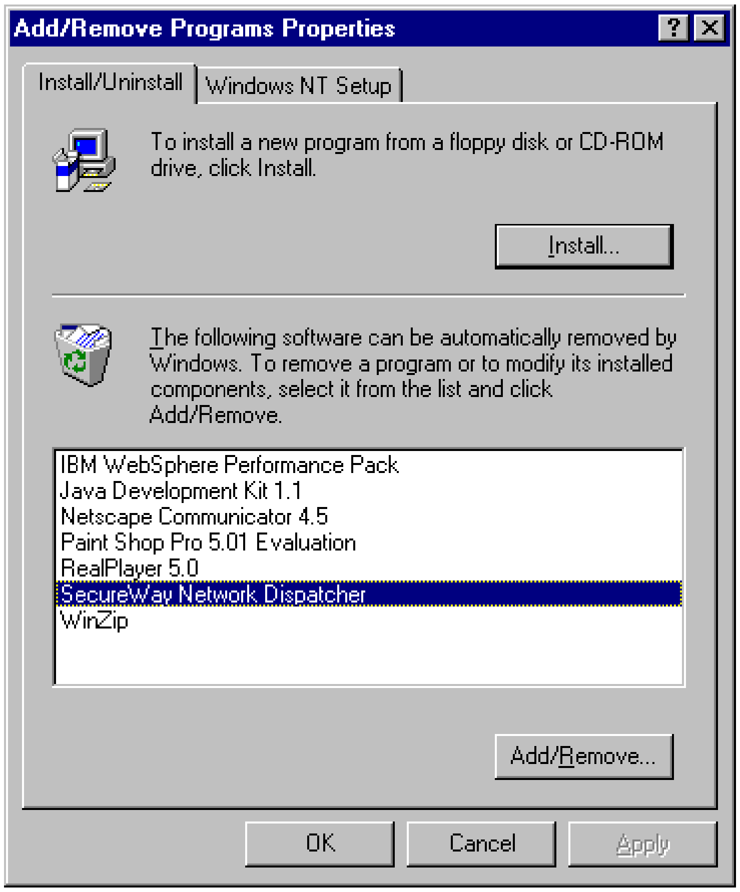
Figure 33. The smitty Used to Deinstall ND (aNDdinst00)

In order to perform the deinstall, you will have to change the `PREVIEW only?` flag from its default value of **yes** to **no** (you can use the Tab key to change this value). Deinstalling the ND components does remove the filesets, but does not remove the destination directory (or its contents) that was used to perform the installation (by default, /usr/lpp/WSPP).

Note that some of the filesets cannot be removed unless other filesets that require them as prerequisites are also removed. For example, you cannot remove intnd.iss.rte without also removing intnd.iss.license and intnd.msg.en_US.iss.rte.

3.2.2 Deinstallation on Windows NT

In order to perform the deinstall, select the **SecureWay Network Dispatcher** entry on the Install/Uninstall folder accessible through the Add/Remove Programs dialog box of the Control Panel:


Add/Remove Programs Properties

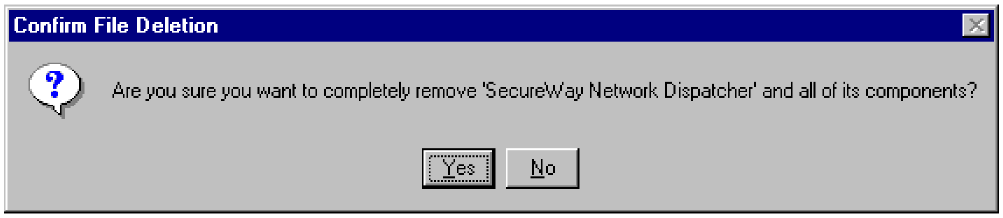
To install a new program from a floppy disk or CD-ROM drive, click Install.

The following software can be automatically removed by Windows. To remove a program or to modify its installed components, select it from the list and click Add/Remove.

IBM WebSphere Performance Pack
Java Development Kit 1.1
Netscape Communicator 4.5
Paint Shop Pro 5.01 Evaluation
RealPlayer 5.0
SecureWay Network Dispatcher
WinZip

Figure 34. Add/Remove Program Window

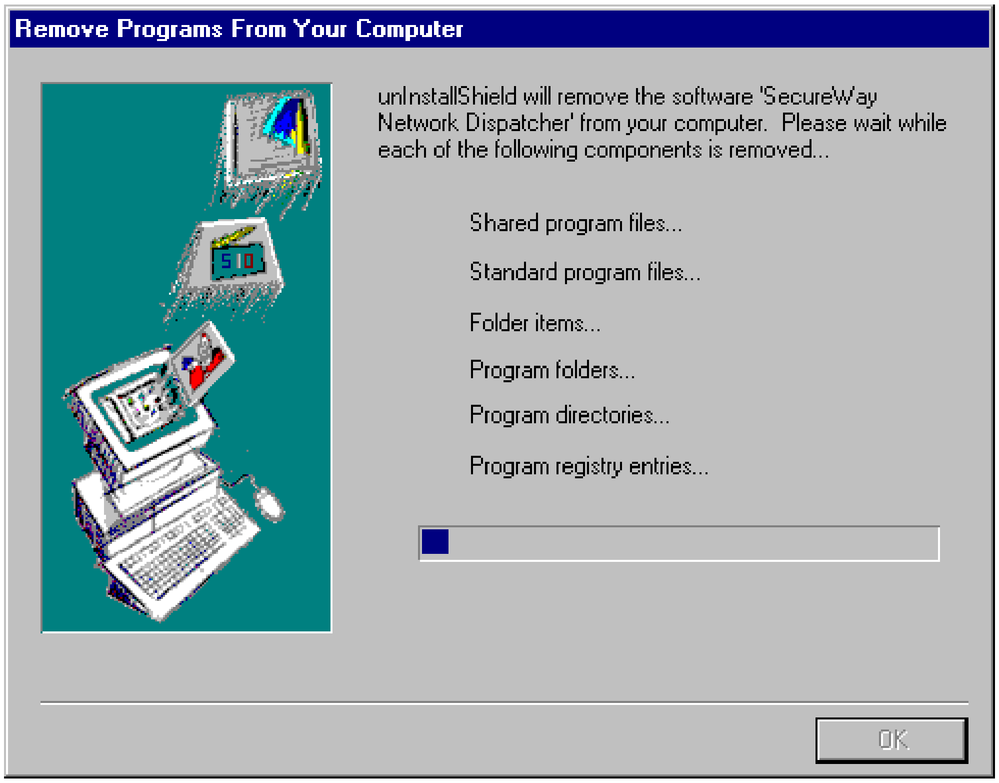
After clicking the **Add/Remove** button, you will be asked to confirm that you want to remove SecureWay Network Dispatcher and *all* of its components.


Confirm File Deletion

Are you sure you want to completely remove 'SecureWay Network Dispatcher' and all of its components?

Figure 35. DeInstallation Confirmation Panel

After clicking the **Yes** button, you will be shown the following progress indicator window:


Remove Programs From Your Computer

unInstallShield will remove the software 'SecureWay Network Dispatcher' from your computer. Please wait while each of the following components is removed...

Shared program files...

Standard program files...

Folder items...

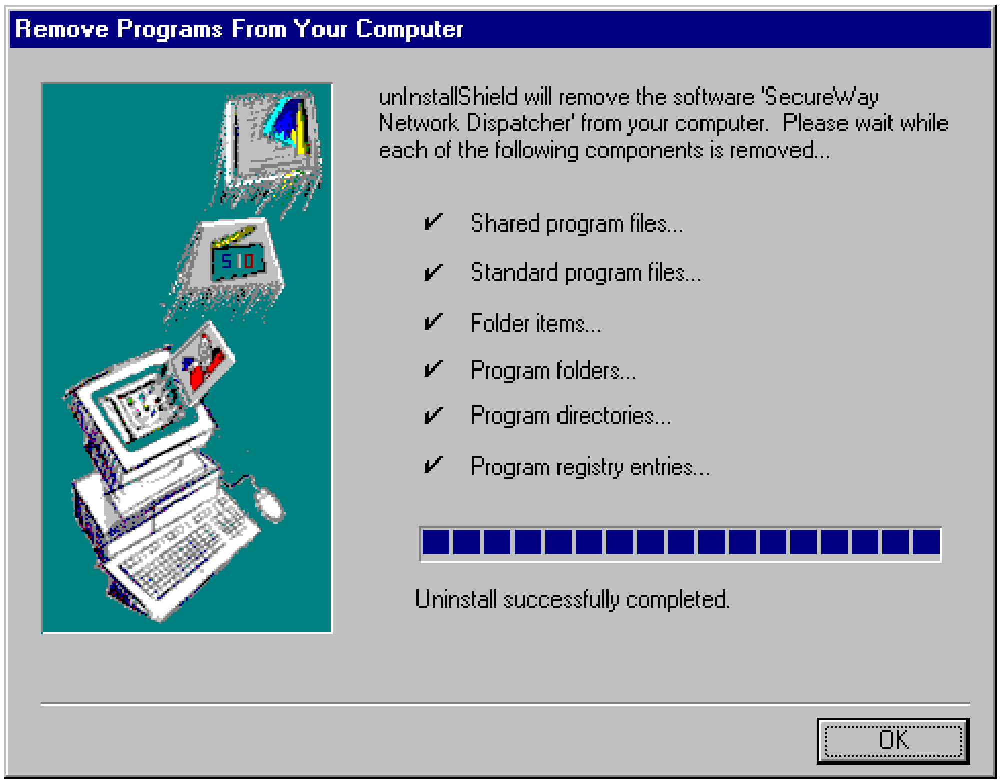
Program folders...

Program directories...

Program registry entries...

Figure 36. Deinstallation Progress Indicator

The successful completion window will appear at the end, as shown in the following figure:


Remove Programs From Your Computer

unInstallShield will remove the software 'SecureWay Network Dispatcher' from your computer. Please wait while each of the following components is removed...

✔ Shared program files...

✔ Standard program files...

✔ Folder items...

✔ Program folders...

✔ Program directories...

✔ Program registry entries...

Uninstall successfully completed.

Figure 37. Deinstallation Successful Completion

Notice that this deinstallation method removes all of the ND components. You do not have the option to remove only selected components.

3.3 Configuration Methods

In order to perform the configuration, you must be the root user on AIX and Solaris or, if the SecureWay Network Dispatcher server is installed on Windows NT, a member of the Administrators group.

Several methods can be used to configure the Dispatcher, ISS and CBR components of ND:

- Command Line

  You can enter the `ndcontrol`, `isscontrol`, and `cbrcontrol` commands from a command prompt.

- Scripts

  You can put the commands into a configuration file script that acts as a batch file. When the script is run, the commands are executed in sequence.

- GUI

  With the ND GUI, the user is able to perform all the configuration steps that can be performed with the `ndcontrol`, `isscontrol`, and `cbrcontrol` commands. Also new to this version are the capabilities of the GUI and the `ndcontrol` commands to implement the basic system TCP/IP commands that previously had to be performed outside the GUI manually.

Another enhancement to the ND GUI in this version is that the user is no longer restricted to performing the configuration on the machine that is running the ND server. This new feature is referred to as Remote Authenticated Administration and is explained further in 3.3.1, "Remote Authenticated Administration" on page 75.

- WebSphere Performance Pack Common Configuration Utility

  This is a new separately installable component of WebSphere Performance Pack Version 2. With this utility, configuration can be performed from a Web browser accessing configuration servlets that reside on a configuration server. Currently the Common Configuration utility provides access to seven browser-based wizards, one of which can be used to add a ND cluster. It does not provide access to all the functions available with the ND GUI. For further details on the Common Configuration component, refer to Chapter 17, "Common Configuration" on page 365.

- SecureWay Network Dispatcher Configuration Wizard

  This Java-based wizard guides you step by step through the process of creating a basic configuration for the Dispatcher component. After launching it from the command line with the `ndwizard` command or from the ND GUI, you will be asked questions about your network and guided through the setup of a cluster for which traffic is to be load balanced. The wizard does not provide access to all the functions of the ND GUI.

The functionality provided by the Common Configuration Utility and the Configuration Wizard are very similar, although their implementation is different. The Add Cluster wizard provided by the Common Configuration Utility was one of many wizards implemented as servlets and intended to be used on all of the WebSphere Performance Pack components. The SecureWay Network Dispatcher Configuration Wizard has been developed as an extension to the ND GUI, which is used exclusively by the SecureWay Network Dispatcher components.

3.3.1 Remote Authenticated Administration

ND V2.1 introduces a new method of accessing the configuration and administration programs for the three ND server programs: ndserver, issd, and the CBR subprocess of the WTE server. We now have the option of running the configuration and administration programs on a machine other than the one running these servers. This machine is sometimes referred to as the *administration client* machine.

The following table shows which configuration methods have been enhanced to use Remote Authenticated Administration for each of the ND components:

Table 3. ND Remote Authenticated Administration Methods by Component

Dispatcher	**ISS**	**CBR**
	ND GUI	
`ndcontrol`	`isscontrol`	`cbrcontrol`
Common Configuration – Add Cluster wizard		
Configuration Wizard – Add Cluster wizard		

Communication between the configuration and administration programs running on the administration client machine and the ND servers is performed using Java Remote Method Invocation (RMI) calls. If the RMI call comes from a machine other than the local machine, a public/private key authentication sequence must take place before the configuration command will be performed on the ND server. This means that if you want to perform configuration on any of the ND servers from a machine (the administration client) other than where the ND server is running, the steps described in the following sections must be completed before the remote configuration will work.

3.3.1.1 Generating the Key Pairs

You must manually create the public/private key pairs on the machine that is running the ND server. The public/private key pairs, for Dispatcher, CBR, and ISS respectively, can be generated or deleted with these commands:

```
ndkeys [create|delete]
cbrkeys [create|delete]
isskeys [create|delete]
```

The `create` option causes two files (each one contains a key) to be created on the ND server machine.

One key is placed in the component's key directory and the other key is placed in the component's administration keys' directory. The file name of this second key is based on the IP address of the server and RMI port it is listening on, and it contains the public key that must be copied over to the administration client machine.

For example, if the ndserver process is running on an AIX machine with address 10.0.0.25 and is listening on RMI port 10099, the `ndkeys create` command would generate these files:

- /usr/lpp/nd/dispatcher/key/authorization.key as the private key
- /usr/lpp/nd/admin/keys/dispatcher/10.0.0.25-10099.key as the public key

The public key file (10.0.0.25-10099.key) must then be placed in the /usr/lpp/nd/admin/keys/dispatcher/ directory on the remote machine. When this is done, the remote client is authorized to configure the ndserver process on 10.0.0.25. This same key must be used on all remote clients that you want to authorize to configure the ndserver process on 10.0.0.25.

Key Naming Conventions

In this book we are referring to the two keys that are generated differently from how they are referred to in the *SecureWay Network Dispatcher User's Guide v2.1 for AIX, Solaris and Windows NT*, GC31-8496. We have chosen to refer to the key that is distributed as the public key, and the key that is kept on the ND server machine as the private key, in accordance with the current asymmetric key cryptography conventions.

If you were to run the command `ndkeys create` again, a new set of public/private keys would be generated. This would mean that all remote clients who tried to connect using the previous keys would not be authorized. The new key would have to be placed in the correct directory on those clients you want to reauthorize.

The command `ndkeys delete` deletes the public and private keys on the server machine. If these keys are deleted, no remote clients will be authorized to connect the servers.

The two key files must be protected from any unauthorized access. Only the ND administrator should be allowed to handle these files.

The following figure summarizes the keys that are involved in the ND administration and the context in which they are used:

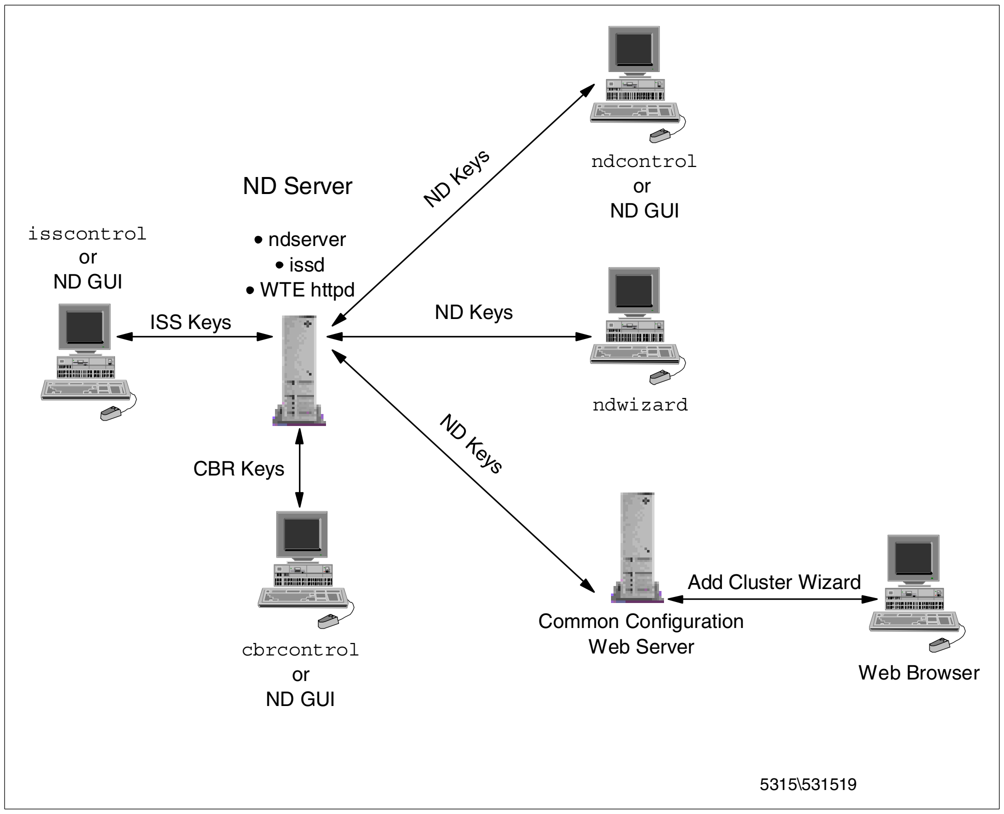


Figure 38. Remote Configuration Possibilities

Remote Authenticated Administration Limitations

The level of security built into this feature as it is implemented does not contain the *robustness* required for remote configuration to be performed on a machine outside the local network for the reasons listed below. For network administrators who are considering remote configuration from a client machine that resides outside the local network, please consider these issues:

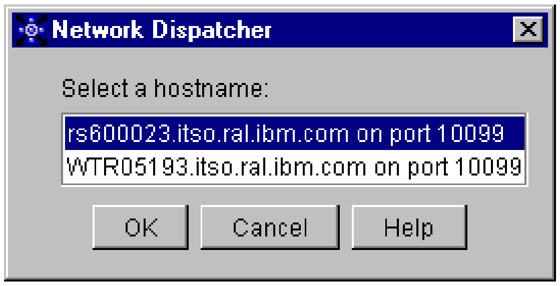
- Transmission of the public key file to the remote client machine should be done in a secure fashion to prevent unwanted listeners.
- Once the public key resides on the remote client machine, subsequent communication between the client and the SecureWay Network Dispatcher server machine is not encrypted. Keys are used only for authentication purposes. The encryption would have to be performed by you.

The intention of the Remote Authenticated Administration in this release is to allow for an in-shop means for an administrator to be able to configure the SecureWay Network Dispatcher server without physically having to be at the machine.

3.3.1.2 Initiating Remote Configuration

The Dispatcher Administration component, ISS Administration component or the Content Based Routing Administration component must be installed on the remote machine to perform remote administration. These components can be installed during the installation process (see 3.1, “Installation of ND” on page 55) by selecting the related check boxes when prompted (see Figure 17 on page 57 and Figure 24 on page 64).

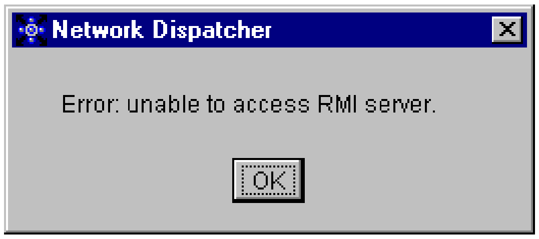
After the public key is transported to the administration client machine, subsequent invocations of the ND GUI on the administration client machine will automatically add the ND server machine from where the key originated, to its list of hosts to connect to. From within the ND GUI, a Dispatcher host connection list may appear as follows:



rs600023.itso.ral.ibm.com on port 10099
WTR05193.itso.ral.ibm.com on port 10099

Figure 39. Host Connection Selection Screen

When we clicked the **OK** button, we saw the following message:



Error: unable to access RMI server.

Figure 40. Unable to Access the Server Machine

This will occur if the ndserver program (in this case the ndserver program for the Dispatcher component) is not running. When we started the ndserver on rs60023 we were able to successfully connect.

Once the host connection is successful, you can proceed to configure the server process (ndserver, issd, or the CBR subprocess of the WTE server) as if you were performing the configuration on the same machine as the server.

Chapter 4. ND Basic Scenarios

In this chapter we show you how to configure basic scenario making use of IBM SecureWay Network Dispatcher (ND), the Load Balancing component of IBM WebSphere Performance Pack V2.

The scenarios described in this chapter show you how to configure Dispatcher and Interactive Session Support (ISS). For details on the Content Based Routing (CBR) component, see Chapter 14, “Content Based Routing” on page 329.

4.1 Load Balancing Basic Scenario Using the Dispatcher

In this section we show you how to create a basic configuration for the Dispatcher. We built up a network environment with five workstations. On one of them we installed the Dispatcher component of ND, which worked to load balance requests to three TCP Web servers. Another machine played the role of the Web client.

4.1.1 Installation of Dispatcher

You do not need to install all the three ND components to run a scenario that makes use of the Dispatcher, but does not make use of ISS and CBR. In this section we give you directions to install only the Dispatcher component.

Dispatcher is supported on three operating systems: IBM AIX 4.2.1 or later, Microsoft Windows NT 4.0, and Sun Solaris 2.6 or later. Refer to the appropriate platform section of 3.1, “Installation of ND” on page 55 for details on how to use the Java InstallShield on your respective platform. When you reach the point where you choose which ND component to install (see Figure 17 on page 57 and Figure 24 on page 64), select these three components to install Dispatcher on your machine:

- Dispatcher Runtime
- Dispatcher Administration
- Dispatcher License

Then, follow the steps indicated in 3.1.1, “Installation on UNIX Systems” on page 55 if the platform where you are installing is AIX and Solaris, and 3.1.2, “Installation on Windows NT” on page 61 if the platform where you are installing is Windows NT.

4.1.2 Network Environment

A summary of the hardware, software and network configuration of the environment where we performed our test is reported in the following table:

Table 4. Basic Scenario - Hardware, Software, and Network Configuration

Workstation	Host Name	IP Address	Operating System	Service
IBM PC 365	wtr05212	9.24.104.218	Windows NT Server 4.0	Web Client
IBM RS/6000 43P	rs600023	9.24.104.128	AIX 4.3.1	Dispatcher
	clusterend	9.24.104.105		
IBM RS/6000 43P	aixncf157	9.24.104.157	AIX 4.3.1	Web Server

Workstation	Host Name	IP Address	Operating System	Service
IBM RS/6000 43P	aixafs	9.24.104.158	AIX 4.3.1	Web Server
IBM PC 365	wtr05193	9.24.104.239	Windows NT Server 4.0	Web Server

All the above machines were provided with a token-ring interface and connected to the same local area network (LAN).

Notice that:

1. The load balancing function was provided by the Dispatcher component of ND Version 2.1.
2. Netscape Navigator 4.5 was the Web browser running on the Web client machine.
3. The Web server function on the three clustered Web servers was provided by the IBM HTTP Server Version 1.3.3.

The following figure offers a graphical representation of the network environment where we performed this scenario:

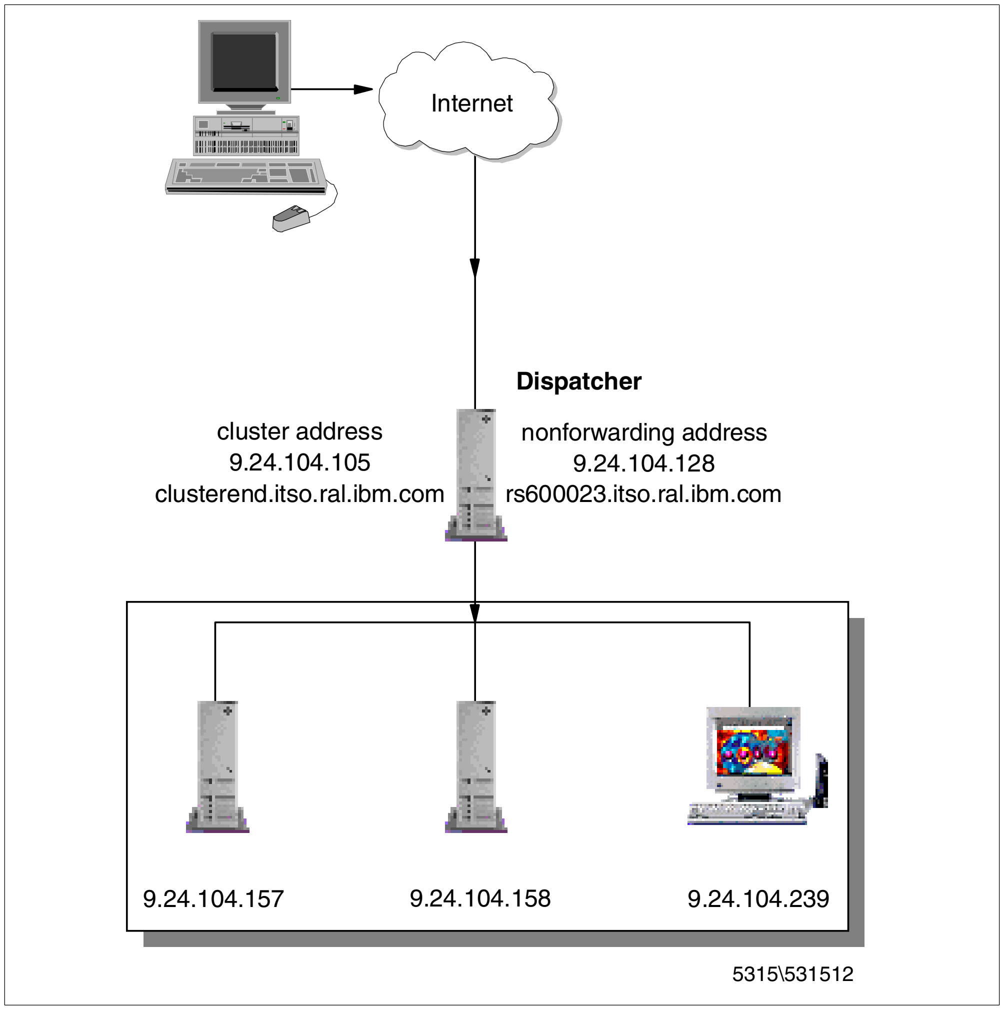


Figure 41. Graphical Representation of the Basic Dispatcher Scenario

4.1.3 Cluster Address and Nonforwarding Address

Note that for the Dispatcher machine two addresses are needed:

1. The primary IP address for the Dispatcher machine is the IP address that is returned by the command:

   ```
   host hostname
   ```

   It is also called a *nonforwarding address*.

   Through the nonforwarding address, you can connect to the machine for management purposes (using FTP or Telnet, for example). In our configuration, the nonforwarding address was 9.24.104.128. In fact, by issuing the `host rs600023` command, the output produced was:

   ```
   rs600023.itso.ral.ibm.com is 9.24.104.128
   ```

2. The *cluster address* is the IP address that will be used by clients to access the entire site. The Dispatcher's work will be dedicated to load balancing requests that are sent to this address.

   A host name can be associated to the cluster address. In our configuration, the cluster address of the Dispatcher machine was 9.24.104.105 and the associated host name was clusterend.itso.ral.ibm.com.

   Notice that you need to define a cluster address for each cluster you are going to define in your environment.

Cluster Address and Network Administration

The cluster address is the unique IP address by which client requests access your cluster. It is not a virtual address that is valid only locally. No other machine in the network should be given that address. For this reason, you must contact the network administrator before assigning a cluster address and the associated host name to the Dispatcher.

All of our workstations were located in the same LAN, and each of them was provided with only one token-ring network interface card. Notice that this configuration involved only the Dispatcher component of the ND. We also ensured that the workstations could ping each other.

Moreover, we made sure that the three Web servers hosted the same Web content. We did this by simply duplicating the same Web page on the three machines. In this case we did not use IBM AFS Enterprise File System (AFS), the File Sharing component of IBM WebSphere Performance Pack Version 2, to make the three Web servers share the same content.

The configuration process can be divided into two logical phases: the setup of the Dispatcher machine and the setup of the TCP/IP server machines. For detailed TCP/IP information, please refer to the IBM redbook *TCP/IP Tutorial and Technical Overview*, GG24-3376.

4.1.4 Dispatcher Configuration

In the following, we refer to the configuration of the Dispatcher on the AIX platform. The configuration process on Solaris is very similar. Also for Windows NT there are no particular differences, and if something differs, we explicitly mention it.

4.1.4.1 Configuration Methods

Consult 3.3, “Configuration Methods” on page 74 for a list of all the possible methods that can be used to configure the Dispatcher.

We will demonstrate how the GUI can be used to achieve the same result as performing `ndcontrol` from the command line. In our configuration process we used a combination of the GUI and the command line to configure the Dispatcher. It would be possible to use one or the other, provided that you have no special TCP configuration requirements. For each of the configuration steps we show the corresponding command line method.

In order to perform the configuration steps listed below, you must be the root user on AIX and Solaris or, if the Dispatcher is installed on Windows NT, a member of the system Administrator group.

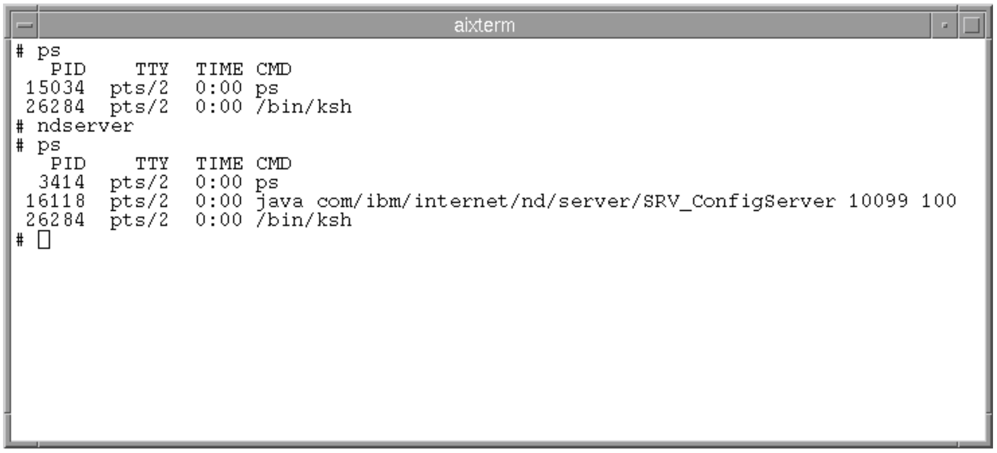
4.1.4.2 Starting the Server Component

To start the server component of the Dispatcher, if on AIX and Solaris, from a command line, enter the `ndserver` command.

Note that on Windows NT the server component is a Windows NT service that starts automatically by default. The name of this service is IBM Network Dispatcher. If you want to stop or restart the IBM Network Dispatcher service, you should open the Services window of the Windows NT Control Panel, highlight **IBM Network Dispatcher** and then click **Stop** or **Start** as required.

If you prefer that the IBM Network Dispatcher service does not start by default at each system reboot, select **IBM Network Dispatcher**, click **Startup...**, and then set the Startup Type to **Manual**.

On AIX, if you enter the `ps` command after launching the command `ndserver`, you can see that the server component is implemented as a Java class, running through the JVM `java` executable file:



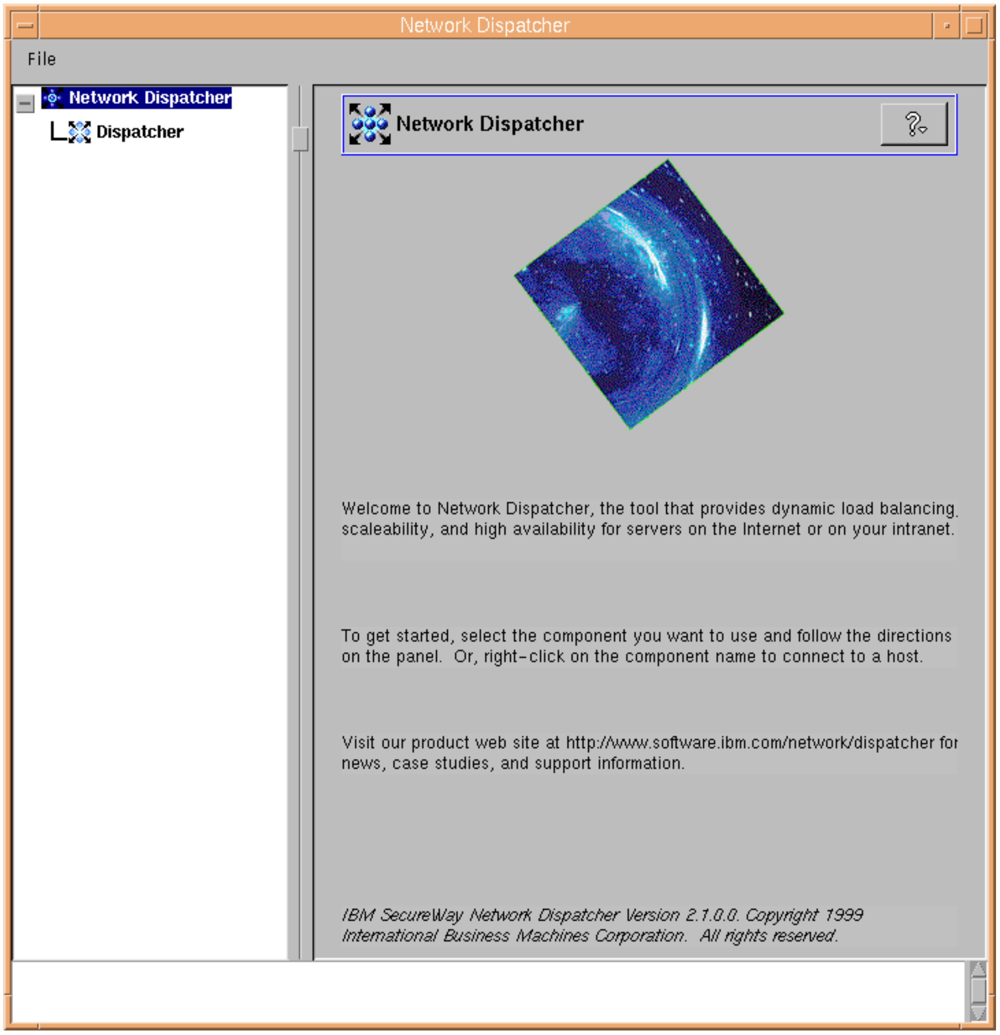
```
# ps
   PID    TTY  TIME CMD
 15034  pts/2  0:00 ps
 26284  pts/2  0:00 /bin/ksh
# ndserver
# ps
   PID    TTY  TIME CMD
  3414  pts/2  0:00 ps
 16118  pts/2  0:00 java com/ibm/internet/nd/server/SRV_ConfigServer 10099 100
 26284  pts/2  0:00 /bin/ksh
# 
```

Figure 42. ndserver Starts a Java Process

4.1.4.3 Starting the Configuration GUI

Now you can start the GUI. To do so, enter the command `ndadmin`.

If your platform is Windows NT, you can also click **Start**, select **Programs** and then click **Network Dispatcher**. The Network Dispatcher item in the Programs menu was created during the installation and is a method of launching the GUI. You will see this window:



Welcome to Network Dispatcher, the tool that provides dynamic load balancing, scaleability, and high availability for servers on the Internet or on your intranet.

To get started, select the component you want to use and follow the directions on the panel. Or, right-click on the component name to connect to a host.

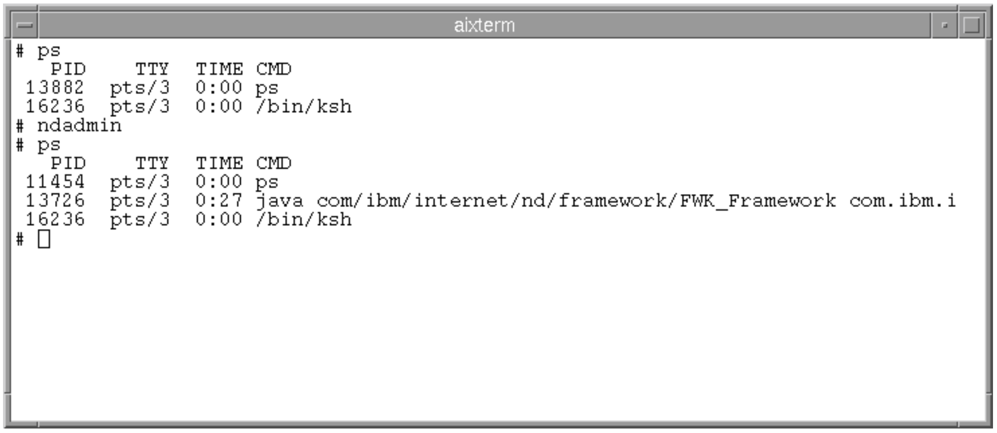
Visit our product web site at http://www.software.ibm.com/network/dispatcher for news, case studies, and support information.

IBM SecureWay Network Dispatcher Version 2.1.0.0. Copyright 1999 International Business Machines Corporation. All rights reserved.

Figure 43. Network Dispatcher Configuration GUI

The left side of the window displays a tree structure, with Network Dispatcher at the top level, and Dispatcher, Interactive Session Support and Content Based Routing as components if they are installed. You can select elements in the tree structure by clicking mouse button one (typically the left button) and display pop-up menus by clicking mouse button two (typically the right button). The pop-up menus for the tree elements are also accessible from the menu bar located at the top of the window. Each item in the tree is marked with a plus sign (+) or a minus sign (-). Click on the plus sign (**+**) to expand the items within it and the minus (**-**) to compact the items.

The server component is implemented as a Java class, running through the JVM `java` executable file. You can see this for example on AIX, if you enter the `ps` command after launching the command `ndadmin`:



```
# ps
   PID    TTY  TIME CMD
 13882  pts/3  0:00 ps
 16236  pts/3  0:00 /bin/ksh
# ndadmin
# ps
   PID    TTY  TIME CMD
 11454  pts/3  0:00 ps
 13726  pts/3  0:27 java com/ibm/internet/nd/framework/FWK_Framework com.ibm.i
 16236  pts/3  0:00 /bin/ksh
# 
```

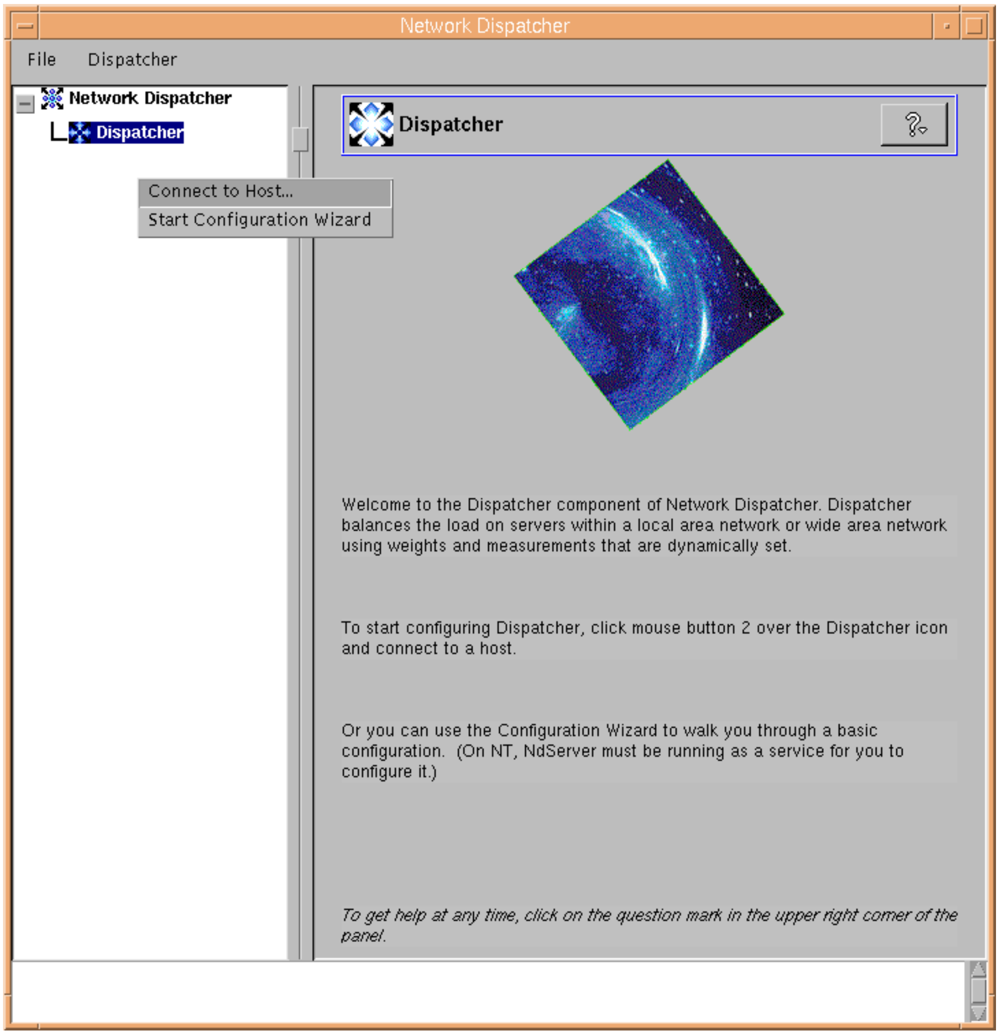
Figure 44. ndadmin Starts a Java Process

4.1.4.4 Connecting to a Host

The first step in configuring the Dispatcher is to make a connection to a host where the Dispatcher is running. This is an effect of the new Remote Authenticated Administration feature, which enables ND component configuration to be done on a remote client machine. See 3.3.1, “Remote Authenticated Administration” on page 75 for further details on remote configuration.

In this case, we were performing this configuration on the Dispatcher machine itself; however, the connection still must take place.

To make the connection, we selected the **Dispatcher** in the tree structure and right-clicked with the mouse. In the pop-up menu that appeared, we selected **Connect to Host...**.



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

To start configuring Dispatcher, click mouse button 2 over the Dispatcher icon and connect to a host.

Or you can use the Configuration Wizard to walk you through a basic configuration. (On NT, NdServer must be running as a service for you to configure it.)

To get help at any time, click on the question mark in the upper right corner of the panel.

Figure 45. Dispatcher Selection Menu

We selected the host name of our Dispatcher machine (see Table 4 on page 81) in the following window:

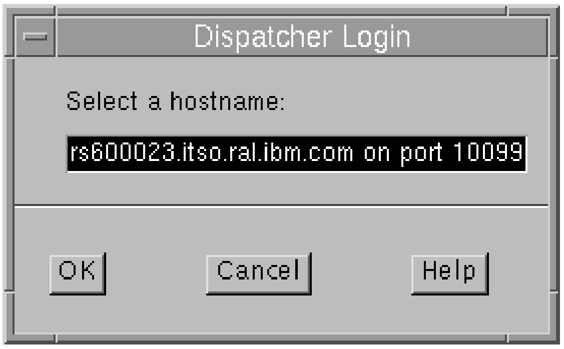
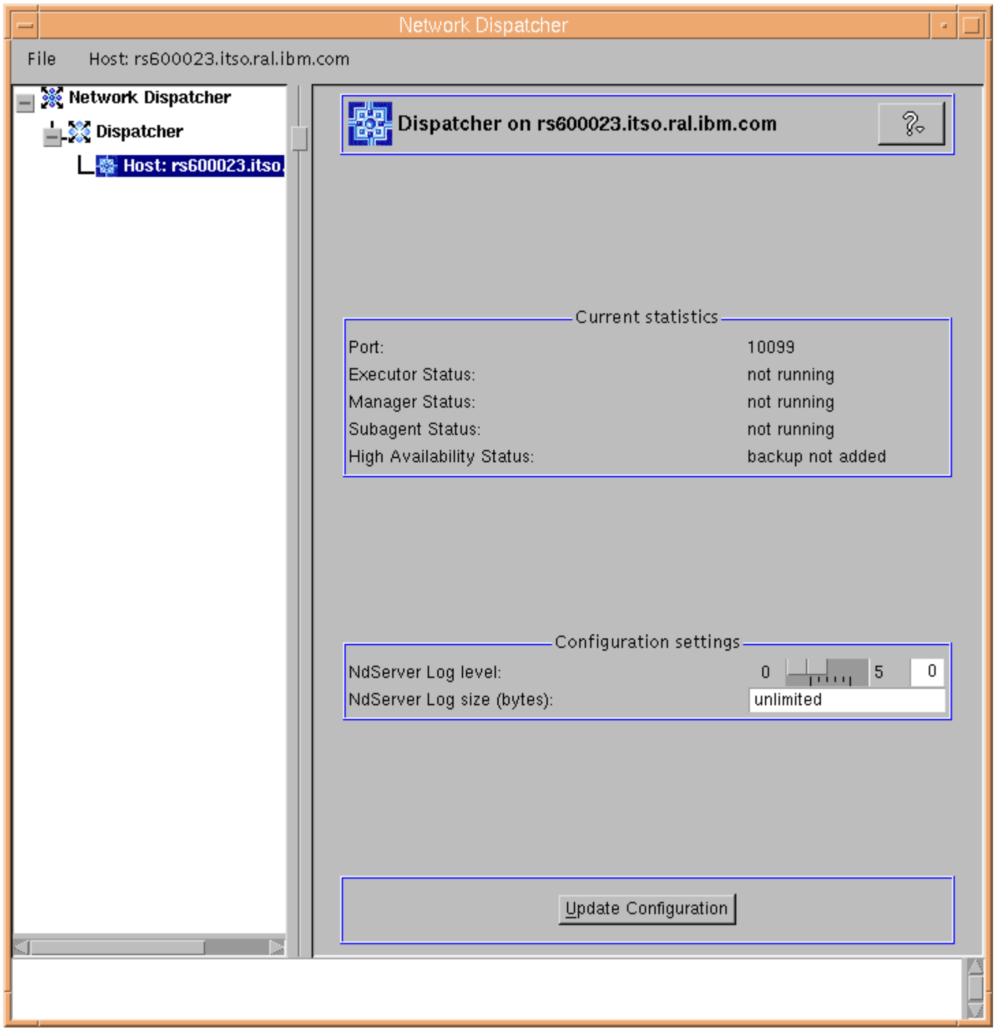


Figure 46. Dispatcher Login Window (andcfg04)

After clicking **OK**, we were presented with this host statistics screen:



Dispatcher on rs600023.itso.ral.ibm.com

Current statistics

Port:	10099
Executor Status:	not running
Manager Status:	not running
Subagent Status:	not running
High Availability Status:	backup not added

Configuration settings

NdServer Log level:	0 5 0
NdServer Log size (bytes):	unlimited

Figure 47. Dispatcher Host Statistics Display

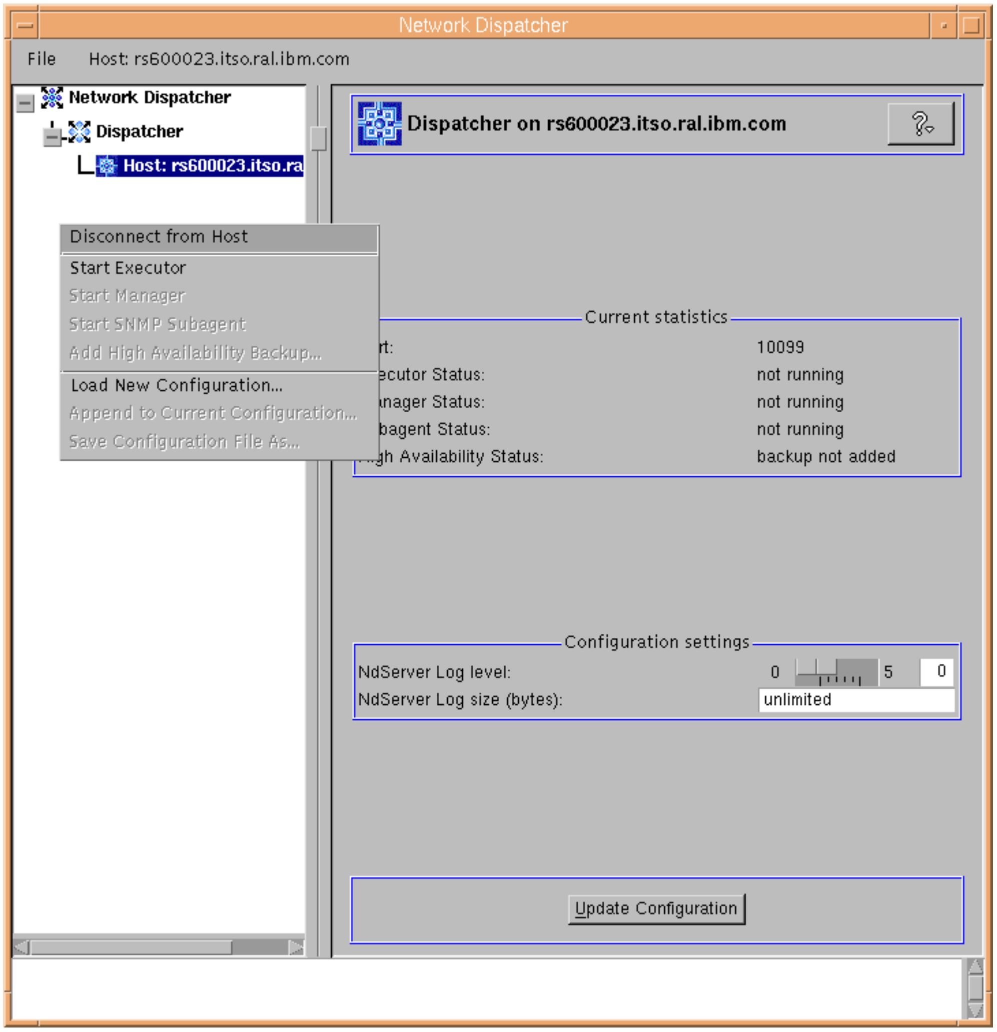
On the right side of the window are two lists of status indicators for the element currently selected. **Current Statistics** are at the top of the panel (you cannot change these). **Configurable Settings** are at the bottom of the panel (these can be changed by performing subsequent configuration). The **Update Configuration** button at the bottom of the panel can be used to apply the latest changes to the configuration currently running.

Note that if you connect to a host where the Executor is already running, then the GUI will be updated to reflect the current execution environment running on the machine. In our case the Executor was not already running, so we had to start it as described in the next section.

4.1.4.5 Starting the Executor

The next step is to start the Executor function of the Dispatcher. The Executor is the Dispatcher component that routes requests to the TCP and UDP servers. It also monitors the number of new, active, and finished connections and performs garbage collection of complete or reset connections. The Executor supplies information about the new and active connections to the Manager component.

To start the Executor, we right-clicked the **Host** entry in the tree structure, and in the pop-up menu that appeared selected **Start Executor**, as shown in the next figure:


Network Dispatcher
File
Host: rs600023.itso.ral.ibm.com
Network Dispatcher
Dispatcher
Host: rs600023.itso.ra
Disconnect from Host
Start Executor
Start Manager
Start SNMP Subagent
Add High Availability Backup...
Load New Configuration...
Append to Current Configuration...
Save Configuration File As...
Dispatcher on rs600023.itso.ral.ibm.com
Update Configuration

Current statistics

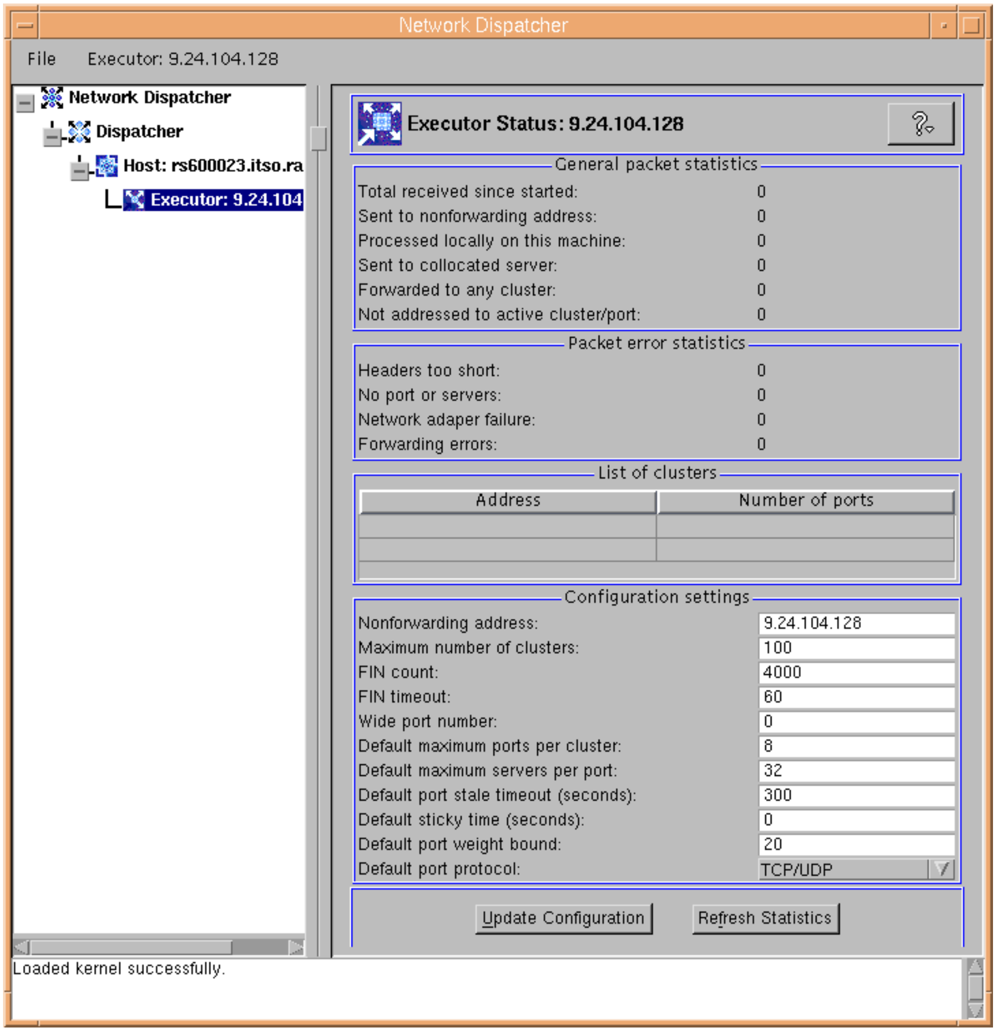
rt:	10099
ecutor Status:	not running
nager Status:	not running
bagent Status:	not running
gh Availability Status:	backup not added

Configuration settings

NdServer Log level:	0 5 0
NdServer Log size (bytes):	unlimited

Figure 48. Start Executor

We then saw the following Executor Statistics window for our host:



Executor Status: 9.24.104.128

General packet statistics

Total received since started:	0
Sent to nonforwarding address:	0
Processed locally on this machine:	0
Sent to collocated server:	0
Forwarded to any cluster:	0
Not addressed to active cluster/port:	0

Packet error statistics

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

List of clusters

Address	Number of ports

Configuration settings

Nonforwarding address:	9.24.104.128
Maximum number of clusters:	100
FIN count:	4000
FIN timeout:	60
Wide port number:	0
Default maximum ports per cluster:	8
Default maximum servers per port:	32
Default port stale timeout (seconds):	300
Default sticky time (seconds):	0
Default port weight bound:	20
Default port protocol:	TCP/UDP

```
Loaded kernel successfully.
```

Figure 49. Executor Status Window

Alternatively, you can start the Executor component from a command line by entering the command:

```
ndcontrol executor start
```

Similarly, you can stop the executor with the command:

```
ndcontrol executor stop
```

Notice, however, that the Executor cannot be stopped on the Windows NT platform.

4.1.4.6 Configuring the Executor

At this point, you might want to redefine the nonforwarding address or the host name of the Load Balancing machine. In our configuration the nonforwarding address appeared by default in the Configuration settings section of the Executor Status window as 9.24.104.128 (which was the IP address assigned to the network interface card of our machine), and we did not need to change that value (see also Table 4 on page 81). However, there are cases where machines have more than one network interface card and administrators may want to use one for cluster traffic and the other for administrative purposes. In those cases, the nonforwarding address displayed by default might not be the right one, and then it becomes necessary to modify it by entering the proper value in the Configuration settings section of the Executor Status window.

Using the Command Line

To set the nonforwarding address from the command line, you can enter the following command:

```
ndcontrol executor set nfa IP_address
```

Put the nonforwarding address in place of *IP_address*. For example, on our platform, the full command should have been:

```
ndcontrol executor set nfa 9.24.104.128
```

However, as we said, we did not need to perform this step, since the correct nonforwarding address by default appeared in the Executor Status window.

More About the Nonforwarding Address

The nonforwarding address has a special meaning inside the kernel code. Any packets that have the non-forwarding address as the destination address are immediately given back to the local operating system for processing. No Dispatcher logic is used to determine where the packet should go. This check is done immediately after receiving the packet.

Notice that no error checking is done to see that the nonforwarding address is an existing IP address on that machine, so you could even set the nonforwarding address to an IP address that has not been defined on that machine, and your command would be accepted, even if you could not administer the Dispatcher machine using such an address.

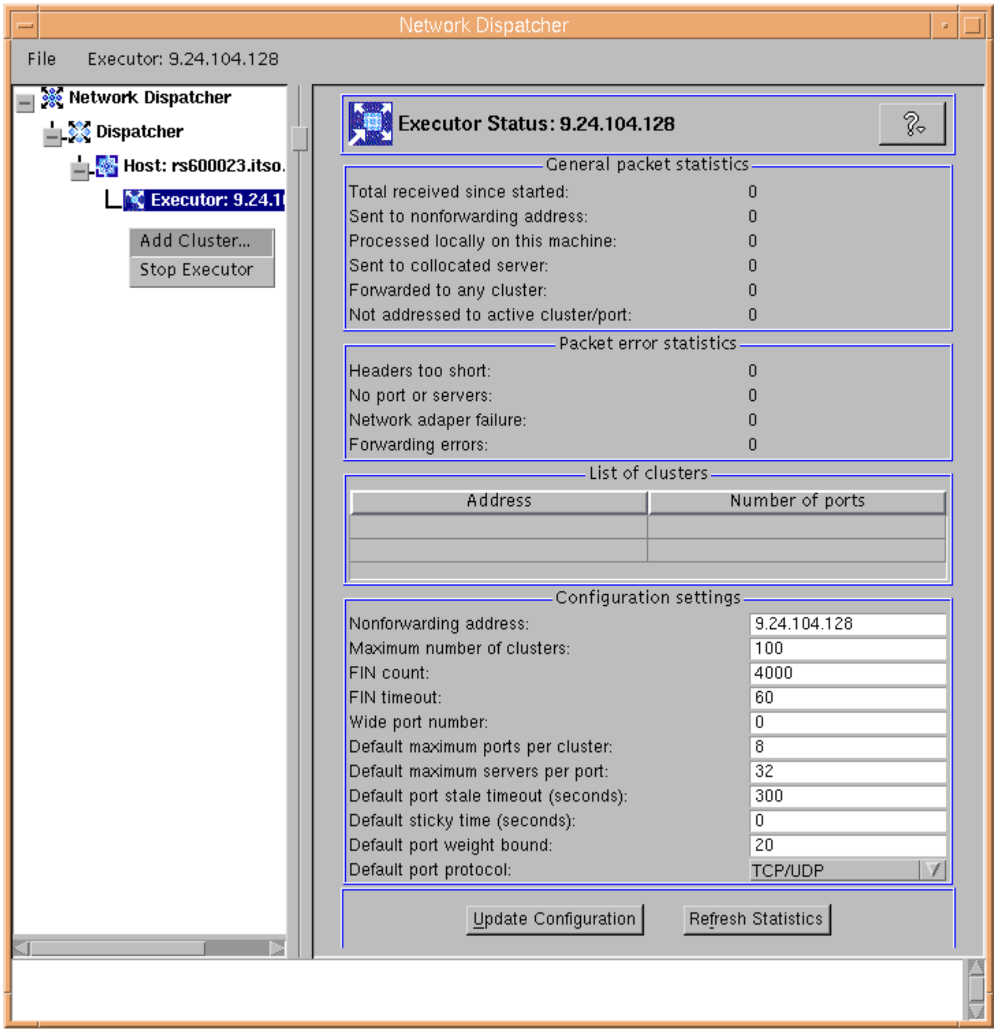
Another use of the nonforwarding address is for colocated servers. If the Dispatcher logic determines that the destination server has the nonforwarding address as its address, the packet is given back to the underlying operating system, not routed out to the network interface. Therefore, in order for colocation to work, the server on the local machine must be added with the nonforwarding address.

The Configuration settings section lists a set of parameters for the Executor that you can change. Some of them have immediate meaning, such as the Maximum number of clusters (default is 100), Default maximum ports per cluster (default is 8), and the Default maximum servers per port (default is 32). Others are more difficult to understand, such as the FIN count and the FIN timeout. FIN is a control bit occupying one sequence number, which indicates that the sender will send no more data. A client sends a FIN after it has sent all its packets, so that the server will know that the transaction is finished. When the Dispatcher receives a FIN, it marks the transaction from active state to FIN state, and at this point the Dispatcher garbage collector can clear the memory reserved for those connections.

For a stable running configuration, you can safely leave the parameter values that appear by default in the Configuration settings section of the Executor Status window unchanged. For detailed information about the parameters, you can consult the SecureWay Network Dispatcher help, which can be accessed by selecting the **Help** button on the top menu bar, or see the document *SecureWay Network Dispatcher for Solaris, Windows NT and AIX User's Guide*, GC31-8496, shipped with the CD-ROM of IBM WebSphere Performance Pack.

4.1.4.7 Adding a Cluster

Next, we needed to define a cluster. From the GUI, we right-clicked the left panel (on the **Executor** item or below it), and in the pop-up menu that appeared selected **Add Cluster...**, as shown in the next figure:



General packet statistics

Total received since started:	0
Sent to nonforwarding address:	0
Processed locally on this machine:	0
Sent to collocated server:	0
Forwarded to any cluster:	0
Not addressed to active cluster/port:	0

Packet error statistics

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

List of clusters

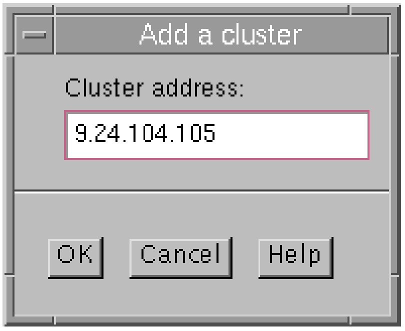
Address	Number of ports

Configuration settings

Nonforwarding address:	9.24.104.128
Maximum number of clusters:	100
FIN count:	4000
FIN timeout:	60
Wide port number:	0
Default maximum ports per cluster:	8
Default maximum servers per port:	32
Default port stale timeout (seconds):	300
Default sticky time (seconds):	0
Default port weight bound:	20
Default port protocol:	TCP/UDP

Figure 50. Adding a Cluster

Then we typed the cluster address in the panel that was brought up:



Cluster address: 9.24.104.105

Figure 51. Enter the Cluster Address

Using the Command Line

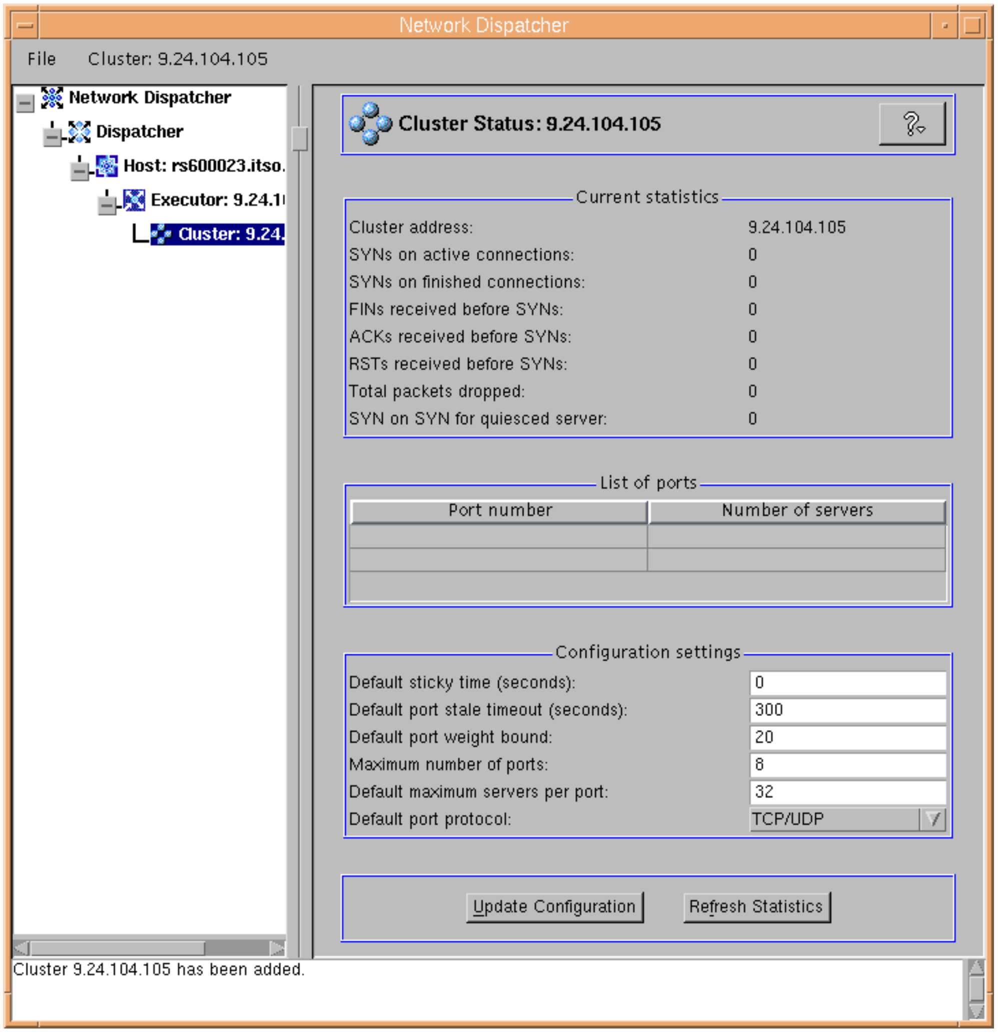
If instead of the GUI you elect to use the command line, then you should enter:

```
ndcontrol cluster add cluster_address
```

In our case the command would have been:

```
ndcontrol cluster add 9.24.104.105
```

The Cluster status window was displayed confirming that the cluster was added to the configuration:



Cluster Status: 9.24.104.105

Current statistics

Cluster address:	9.24.104.105
SYNs on active connections:	0
SYNs on finished connections:	0
FINs received before SYNs:	0
ACKs received before SYNs:	0
RSTs received before SYNs:	0
Total packets dropped:	0
SYN on SYN for quiesced server:	0

List of ports

Port number	Number of servers

Configuration settings

Default sticky time (seconds):	0
Default port stale timeout (seconds):	300
Default port weight bound:	20
Maximum number of ports:	8
Default maximum servers per port:	32
Default port protocol:	TCP/UDP

```
Cluster 9.24.104.105 has been added.
```

Figure 52. Cluster Added

4.1.4.8 Methods of Aliasing the Cluster to the Network Interface

In order for the Dispatcher to route packets, each cluster address must be aliased to a network interface card on the Dispatcher machine.

The next three sections describe different methods available for creating an alias on the Dispatcher machine's network interface card to the cluster address. Although you will only need to use one of these methods, we describe details of each of the available methods as well as when it may be more appropriate to use one method over another. The three methods are:

- Using system commands `ifconfig` or `ndconfig`. This method is explained in 4.1.4.9, "Aliasing the Network Interface Using the Command Line" on page 95.
- Using configuration scripts to create this alias. Refer to 4.1.4.10, "Aliasing the Network Interface Using Scripts" on page 97.
- New to SecureWay Network Dispatcher Version 2.1 is the ability to create this alias with the ND GUI or the underlying `ndcontrol` commands. For further information on using the GUI to alias the cluster, see 4.1.4.11, "Aliasing the Network Interface Using the GUI" on page 99.

4.1.4.9 Aliasing the Network Interface Using the Command Line

On AIX and Solaris, the TCP/IP command `ifconfig` can be used to assign the cluster address as an alias to the existing TCP network interface. The full command syntax is as follows:

```
ifconfig interface_name alias cluster_address netmask netmask
```

where:

- *`interface_name`* is the name assigned to the network interface.

  On our platform, the network interface name was tr0, since our machine was equipped with a token-ring network adapter.

- *`cluster_address`* is the cluster address assigned to the Dispatcher machine.

  In our case we entered 9.24.104.105, according to Table 4 on page 81.

- *`netmask`* is the netmask value you are using in the TCP/IP configuration of the Dispatcher machine. It can be specified either in dotted-decimal or hexadecimal form.

  In our case we could have entered either `255.255.255.0` or `0xffffff00`.

On Windows NT, you should use the command `ndconfig` instead. Actually, this command does not belong to the set of system commands related to the TCP/IP protocol. However, while installing the ND component of IBM WebSphere Performance Pack, the installation process installs the executable for `ndconfig` and the same installation process updates the value of the Path system environment variable, adding the complete path to `ndconfig`. The syntax of the `ndconfig` command is the same as the AIX and Solaris command `ifconfig`.

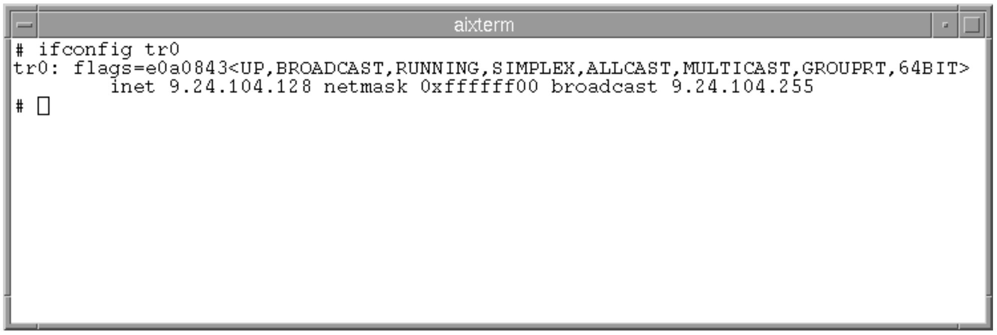
It is recommended that you know exactly the value of the netmask parameter as it was set on your Dispatcher machine. If you fail or omit it, an unwanted route to the IP address might be created in the routing table.

In order to know what the value of the netmask parameter is, on AIX and Solaris you can enter the command `ifconfig` followed by the name of the network interface. If the platform is Windows NT, `ifconfig` should be replaced by `ndconfig`.

For example, on the Dispatcher machine that we installed on AIX, we entered the command:

```
ifconfig tr0
```

The output we received is displayed in the following screen:



```
# ifconfig tr0
tr0: flags=e0a0843<UP,BROADCAST,RUNNING,SIMPLEX,ALLCAST,MULTICAST,GROUPRT,64BIT>
        inet 9.24.104.128 netmask 0xffffff00 broadcast 9.24.104.255
#
```

Figure 53. ifconfig Command Entered to Discover the netmask Value

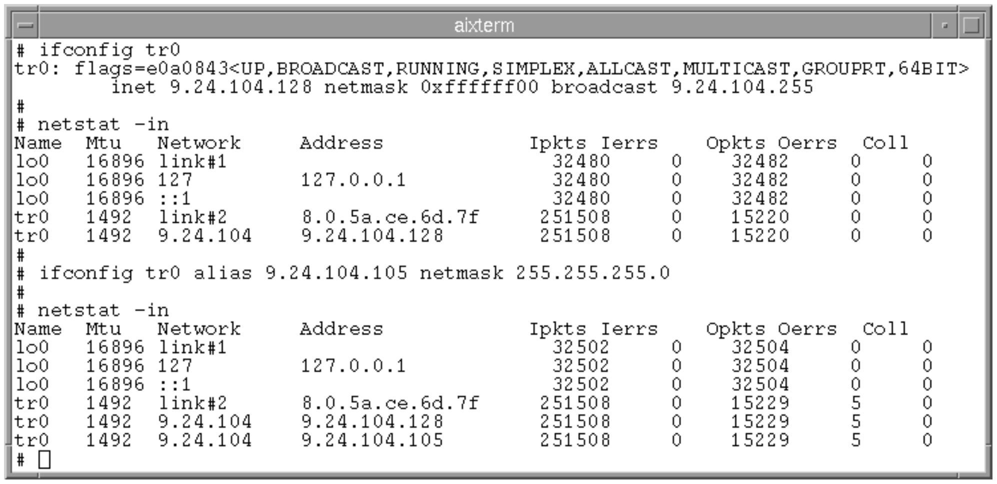
So, on our platform, the full command to create the alias on the network interface tr0 to the cluster address 9.24.104.105 was:

```
ifconfig tr0 alias 9.24.104.105 netmask 255.255.255.0
```

To ensure that the alias has been created correctly, enter the command:

```
netstat -in
```

You should get two IP addresses for your network interface. The following figure shows the output we got from the command above before and after we created an alias on the Dispatcher network interface using the `ifconfig` command:



```
# ifconfig tr0
tr0: flags=e0a0843<UP,BROADCAST,RUNNING,SIMPLEX,ALLCAST,MULTICAST,GROUPRT,64BIT>
        inet 9.24.104.128 netmask 0xffffff00 broadcast 9.24.104.255
#
# netstat -in
Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
lo0   16896 link#1                          32480     0    32482     0     0
lo0   16896 127         127.0.0.1           32480     0    32482     0     0
lo0   16896 ::1                             32480     0    32482     0     0
tr0   1492  link#2      8.0.5a.ce.6d.7f    251508     0    15220     0     0
tr0   1492  9.24.104    9.24.104.128       251508     0    15220     0     0
#
# ifconfig tr0 alias 9.24.104.105 netmask 255.255.255.0
#
# netstat -in
Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
lo0   16896 link#1                          32502     0    32504     0     0
lo0   16896 127         127.0.0.1           32502     0    32504     0     0
lo0   16896 ::1                             32502     0    32504     0     0
tr0   1492  link#2      8.0.5a.ce.6d.7f    251508     0    15229     5     0
tr0   1492  9.24.104    9.24.104.128       251508     0    15229     5     0
tr0   1492  9.24.104    9.24.104.105       251508     0    15229     5     0
# 
```

Figure 54. How to Verify that the Alias Has Been Created Correctly

Cluster Addresses and Network Interfaces

Note that it is possible to configure the Dispatcher to manage more than one cluster, whether your Dispatcher machine is shipped with one network interface card or more than one. Following are some examples:

- If your Dispatcher machine has only one network interface adapter, you can configure as many aliases as the number of clusters you want to define, on the same network interface card.
- If your Dispatcher machine is shipped with two network interface cards (for example, one Ethernet card, en0, and one token-ring card, tr0), and you want to manage two clusters, one on en0 and the second on tr0, then each cluster address must be aliased on the corresponding network interface card.
- One more example is if your Dispatcher machine is shipped with two network interface cards (one Ethernet card, en0, and one token-ring card, tr0), and you want to manage five clusters, say three on en0 and two on tr0. The first three cluster addresses should be aliased on en0, and the other two cluster addresses on tr0.

No limits are known to the number of cluster addresses that can be aliased on the same network interface card.

4.1.4.10 Aliasing the Network Interface Using Scripts

For the Dispatcher to route packets, each cluster address must be aliased to a network interface card.

Each time the Executor changes state, you will have to again configure all the aliases for the network interfaces of the Dispatcher machine. To avoid manually configuring the network interface every time, you can use one or more script files, which are automatically launched by the Dispatcher as different conditions occur, or as the Dispatcher changes its own *state*.

For example, when using the High Availability feature of the Dispatcher (as described in Chapter 6, “ND High Availability Support” on page 177), when one Dispatcher starts to monitor the condition of a currently *active* Dispatcher, and does not do any load balancing itself, it is said that this Dispatcher has gone into the *standby* state. Alternatively, when this standby Dispatcher finishes monitoring the health of the active Dispatcher and it changes its state to begin load balancing and routing packets, it is said that this Dispatcher has gone into the *active* state.

Every time the Dispatcher changes state, a corresponding script is launched. For example, when the Dispatcher goes into the idle state, the related script that is launched is `goIdle`. The Dispatcher goes into the *idle* state when it begins routing packets in a stand-alone configuration. Use of the goIdle.sample script file that comes with the installation of the product is appropriate if you are not implementing the High Availability feature in your Dispatcher configuration. Sample versions of script files that accomplish these functions are located in the directory *installbase*/samples, where *installbase* is for the Dispatcher component and varies by operating system. See Table 1 on page 69 for a list of the installbase locations.

We made a backup copy of goIdle.sample and then modified it by changing the IP addresses and netmask to match our environment. When run, the script would create an alias of the cluster address on the desired network interface card in our Dispatcher machine, as shown in the following figure:

```
#!/bin/ksh
#
# goIdle script
#
# Configure this script only if you are not using the high
# availability feature of Network Dispatcher.
# Configuration of this script is optional in a stand-alone
# configuration.
#
# This script is executed when Network Dispatcher goes into the
# 'Idle' state and begins routing packets. This occurs when
# the high availability feature has not been added. It can
# also occur in a high availability configuration before the
# high availability feature has been added or after it has been
# removed. goIdle should NOT be used in conjunction with high
# availability. Its intended purpose is for a stand-alone
# environment.
#
# This script must be placed in Network Dispatcher's bin
# directory (by default this is /usr/lpp/nd/dispatcher)
# and it needs to have root execute permission.
#
# Modify NETWORK, CLUSTER, INTERFACE and NETMASK to match your
# environment. en0=first Ethernet adapter, tr0=first Token ring
# adapter, fi0=first FDDI adapter. NETMASK must be the netmask
# of your LAN. It may be hexadecimal or octal notation.
#
NETWORK=9.24.104
INTERFACE=tr0
NETMASK=0xffffff00
#
echo "Adding cluster alias(es)"
  for CLUSTER in 105; do
     ifconfig $INTERFACE alias $NETWORK.$CLUSTER netmask $NETMASK
  done
```

Figure 55. Modified goIdle.sample Script File on AIX

In order for the above script to run, you are required to name the file goIdle (without an extension) and place it in the Dispatcher bin directory, which by default is *installbase*/bin, where *installbase* is for the Dispatcher component and varies by operating system. See Table 1 on page 69 for a list of the installbase locations. You are also required to ensure that the script has execute permission.

Notice that the scripts are slightly different depending on the platform they are to run, because the syntax depends on the operating system. Sample scripts are provided in the *installbase*/samples directory on each platform, so it is not necessary to show them here. The following box summarizes the syntax modifications on Windows NT:

Script Modifications on Windows NT

The script files performing the Dispatcher configuration need to be modified if they have to be run on the Windows NT platform. The reason for this is that, on the Windows NT platform, scripts have to follow a different syntax:

- The first line in the AIX script

  ```
  #!/bin/ksh
  ```

  is a special comment, which forces the script to be interpreted by the ksh Korn shell interpreter, located in the /bin directory. That line should be removed in the Windows NT script. On Windows NT, the first line is usually:

  ```
  @echo off
  ```

  With this instruction in place, the commands are not echoed on the screen.

- Comments on Windows NT must be preceded by the `rem` keyword, rather than the pound sign (`#`). So, for example, the AIX script comment:

  ```
  # This script must be placed in Network Dispatcher's bin directory
  ```

  on Windows NT becomes:

  ```
  rem This script must be placed in Network Dispatcher's bin directory
  ```

- Settings must be preceded by the `set` keyword. So, for instance, the AIX statement:

  ```
  NETWORK=9.67.123
  ```

  is equivalent on Windows NT to:

  ```
  set NETWORK=9.67.123
  ```

- The value of a variable is obtained by typing the percent sign (`%`) before and after the variable name, rather than using the dollar sign (`$`) before the variable name. So, for instance, `$CLUSTER` on WIndows NT becomes `%CLUSTER%`.

- There is no native `ifconfig` command on Windows NT. So all the occurrences of `ifconfig` need to be replaced by calls to the `ndconfig` command, which comes with the installation of ND.

- Commands must be preceded by the `call` keyword. So, for instance, the AIX command:

  ```
  ifconfig $SECINTERFACE delete $SECNETWORK.$SECCLUSTER
  ```

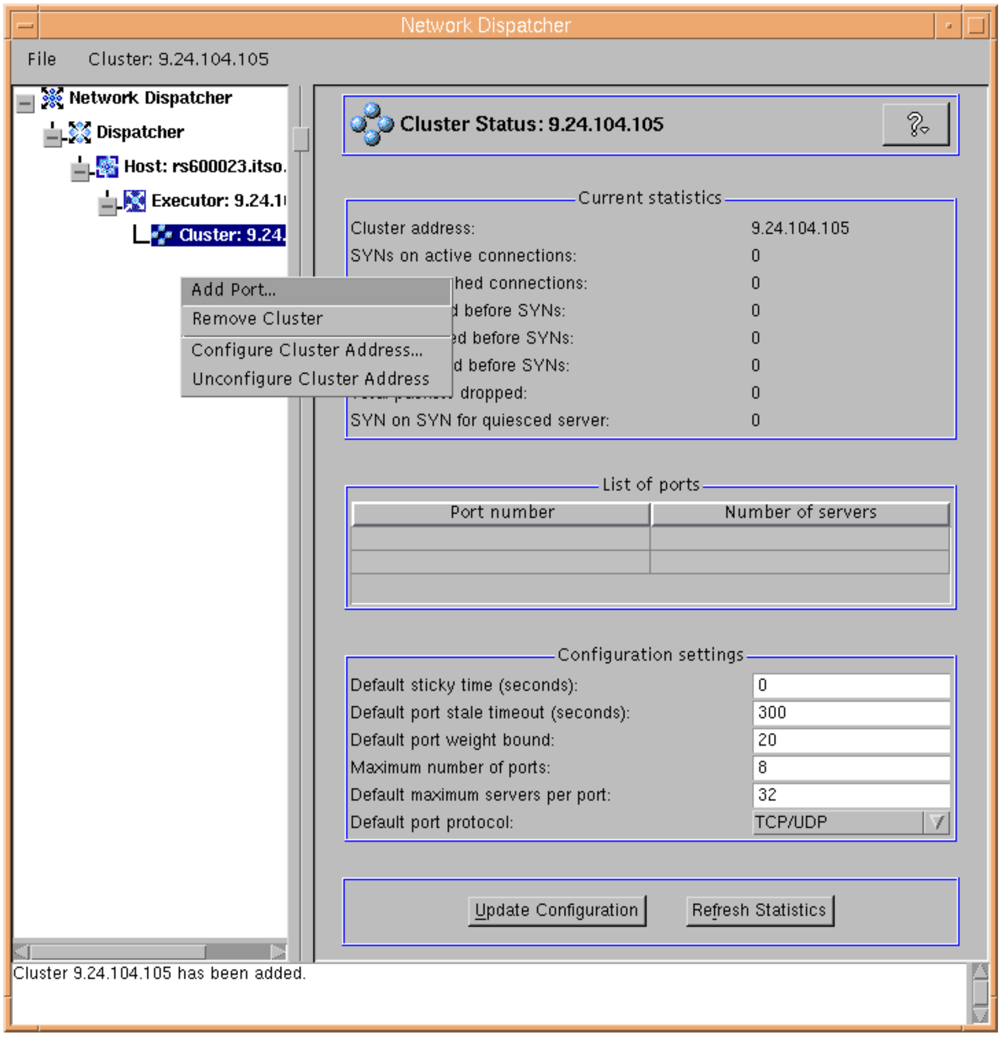
  is equivalent on Windows NT to:

  ```
  call ndconfig %SECINTERFACE% delete %SECNETWORK%.%SECCLUSTER%
  ```

4.1.4.11 Aliasing the Network Interface Using the GUI

As mentioned in 4.1.4.9, “Aliasing the Network Interface Using the Command Line” on page 95, new to ND V2.1 is the ability to alias the network interface to the cluster address with the GUI or the underlying `ndcontrol` commands.

To create an alias on the Dispatcher machine’s network interface card to the cluster address with the GUI, once the cluster had been added, we right-clicked the **Cluster** and in the pop-up menu, selected **Configure Cluster Address...**, as shown in the following figure:



Current statistics

Cluster address:	9.24.104.105
SYNs on active connections:	0
…hed connections:	0
…d before SYNs:	0
…ed before SYNs:	0
…d before SYNs:	0
… dropped:	0
SYN on SYN for quiesced server:	0

List of ports

Port number	Number of servers

Configuration settings

Default sticky time (seconds):	0
Default port stale timeout (seconds):	300
Default port weight bound:	20
Maximum number of ports:	8
Default maximum servers per port:	32
Default port protocol:	TCP/UDP

```
Cluster 9.24.104.105 has been added.
```

Figure 56. Configure Cluster Address with the GUI

We were prompted for an Interface name and Netmask. Filling in these values is optional and should be used if your cluster address does not match any subnet for existing addresses.

Using the Command Line

If instead of the GUI, you elect to use the command line, then you should enter:

```
ndcontrol cluster configure cluster_address
```

In our case, the command would have been:

```
ndcontrol cluster configure 9.24.104.105
```

Similarly, if your cluster address does not match any subnet for existing addresses, you can extend the above command to include the interface name and subnet mask.

When you configure the cluster address by either of these means, the system's adapter configuration command is used to alias the cluster.

As mentioned earlier, there are circumstances where you would not want to configure the alias with this method:

- If the cluster is added to a standby server in high-availability mode
- If you use the goIdle script in standby mode

For these situations, refer to 4.1.4.9, "Aliasing the Network Interface Using the Command Line" on page 95 and 4.1.4.10, "Aliasing the Network Interface Using Scripts" on page 97.

4.1.4.12 Adding Ports

The Dispatcher can load balance any TCP or stateless UDP application. You need to define as many ports as the protocols you want the Dispatcher machine to communicate through. Table 5 lists some of the available protocols and assigned ports:

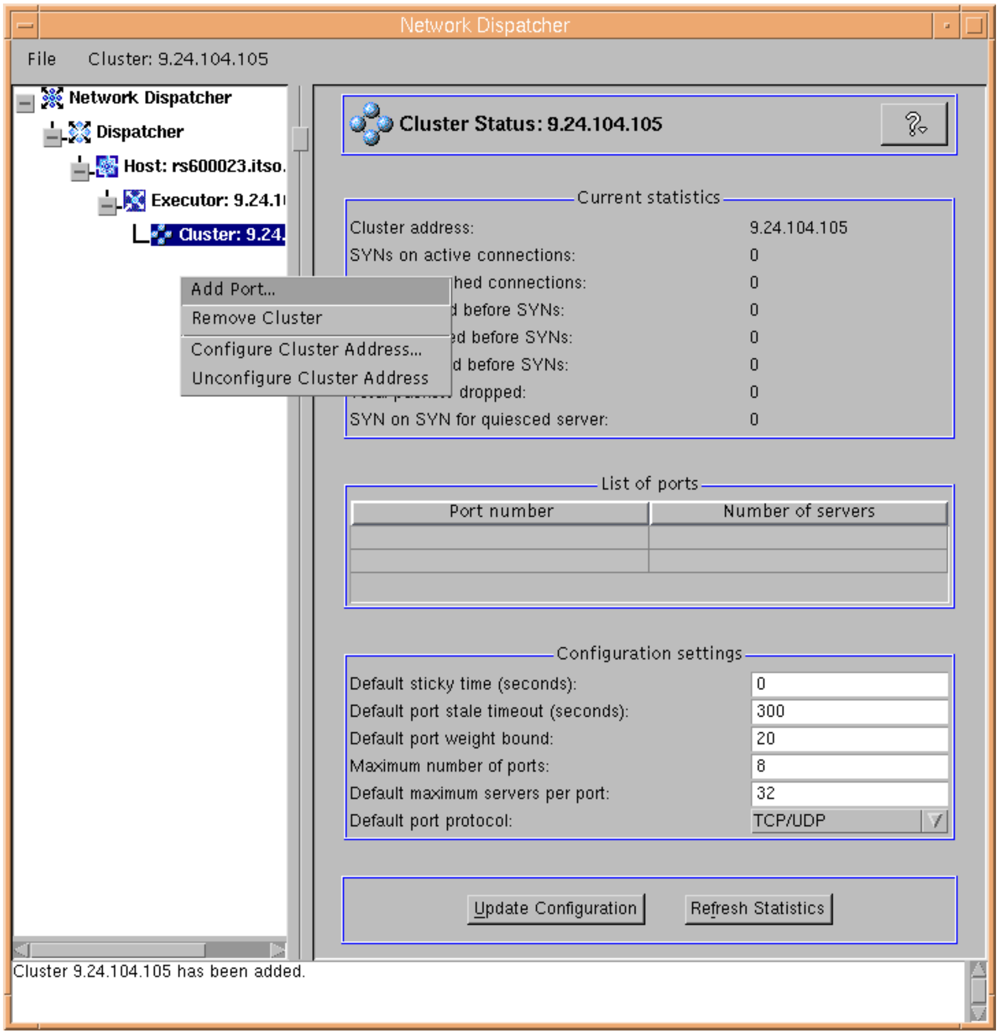
Table 5. Protocol and Port Used by the Dispatcher

Protocol	Port
FTP	20, 21
Telnet	23
SMTP	25
HTTP	80
POP3	110
NNTP	119
SSL	443

The port values shown in Table 5 reflect the assigned port numbers provided by the Networking Working Group in the Request For Comments (RFC) 1700. See `http://info.internet.isi.edu/in-notes/rfc/files/rfc1700.txt` for a complete list.

Note that if you want to use the FTP protocol, you should add port 21 (also known as the FTP *control port*), which is the port through which an FTP server is listening for the commands, and port 20 (also known as the FTP *data port*), which is the port through which the data is transmitted during an FTP connection.

To add ports, from the GUI right-click the **Cluster** item in the left panel and in the pop-up window that appears select **Add Port...**, as shown in the next figure:



Cluster: 9.24.104.105

Cluster Status: 9.24.104.105

Current statistics

Cluster address:	9.24.104.105
SYNs on active connections:	0
hed connections:	0
d before SYNs:	0
ed before SYNs:	0
d before SYNs:	0
dropped:	0
SYN on SYN for quiesced server:	0

List of ports

Port number	Number of servers

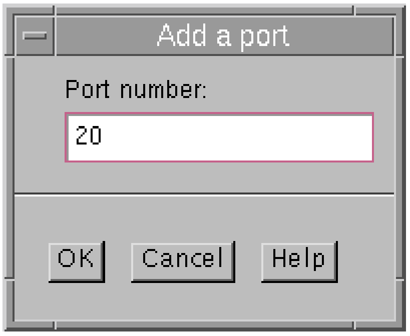
Configuration settings

Default sticky time (seconds):	0
Default port stale timeout (seconds):	300
Default port weight bound:	20
Maximum number of ports:	8
Default maximum servers per port:	32
Default port protocol:	TCP/UDP

Cluster 9.24.104.105 has been added.

Figure 57. Adding a Port

Then type the port number you want and click **OK** in the panel that appears:



Port number:

20

Figure 58. Enter the Port Number

First of all, we entered `20`, as shown in Figure 58 on page 102. Following the same process, we also added ports 21 and 80 to the same cluster using the GUI. We also added port 443, assigned to the SSL protocol, but in this case we wanted to experiment with the command line, rather than using the GUI again, and so we entered:

```
ndcontrol port add 9.24.104.105:443
```

Coming back to the GUI, and clicking the **Refresh** button in the top menu bar, we got the updated configuration with the four ports added.

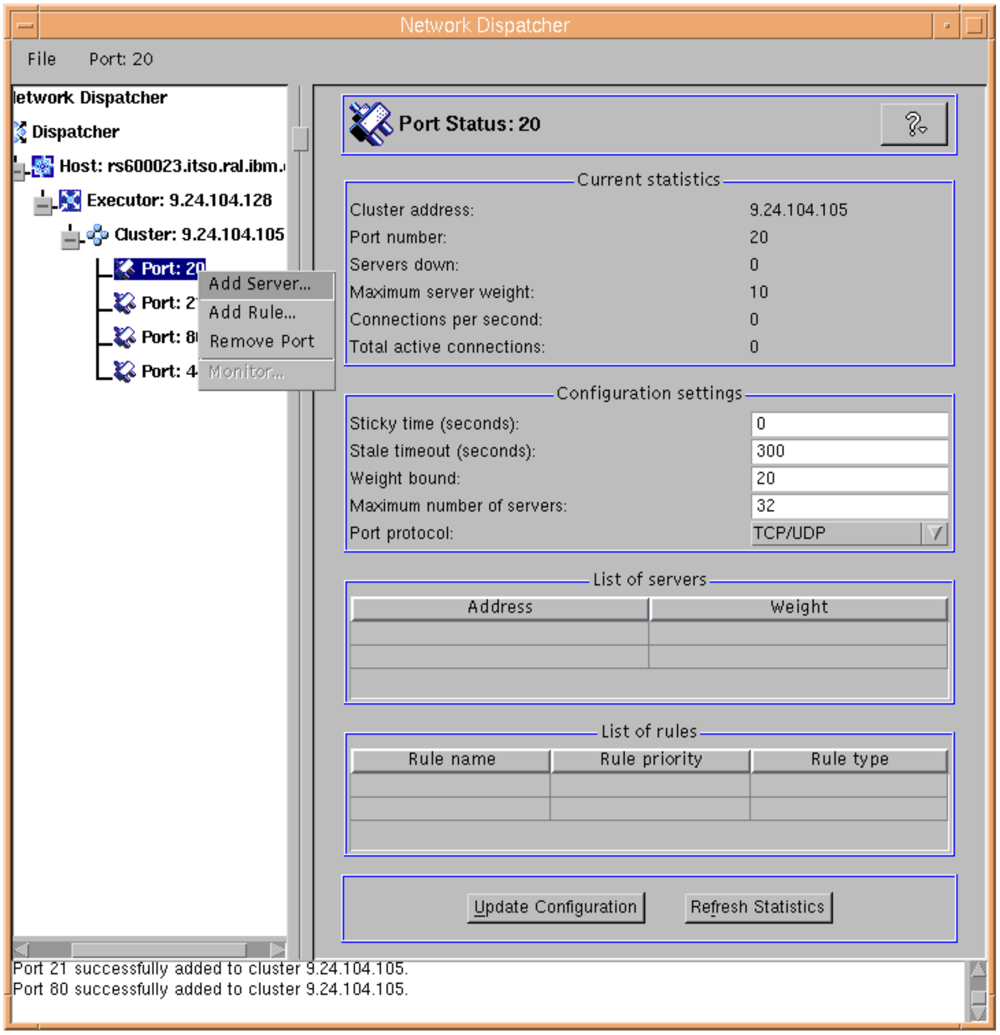
Multiple Ports Using the Command Line

Note that, using only the command line, you could add multiple ports entering just one command. In our configuration, in order to add all four ports to the cluster, we could have entered:

```
ndcontrol port add 9.24.104.105:20+21+80+443
```

4.1.4.13 Adding Servers

At this point of the configuration process, you need to define the TCP server machines that you want in the cluster for the ports you have just configured. To do so, we right-clicked the **Port 20** item and chose **Add Server...**, as shown in the following figure:



Current statistics

Cluster address:	9.24.104.105
Port number:	20
Servers down:	0
Maximum server weight:	10
Connections per second:	0
Total active connections:	0

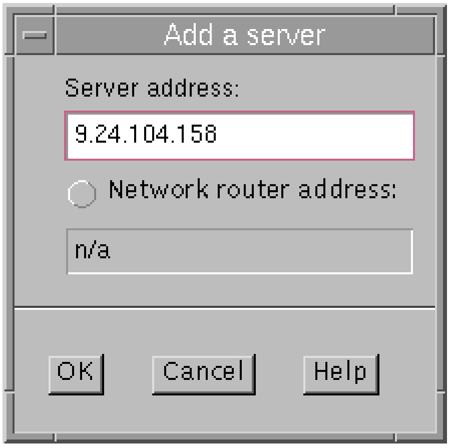
Configuration settings

Sticky time (seconds):	0
Stale timeout (seconds):	300
Weight bound:	20
Maximum number of servers:	32
Port protocol:	TCP/UDP

```
Port 21 successfully added to cluster 9.24.104.105.
Port 80 successfully added to cluster 9.24.104.105.
```

Figure 59. Adding a Server

Then type the server name and click **OK** in the panel that appears:



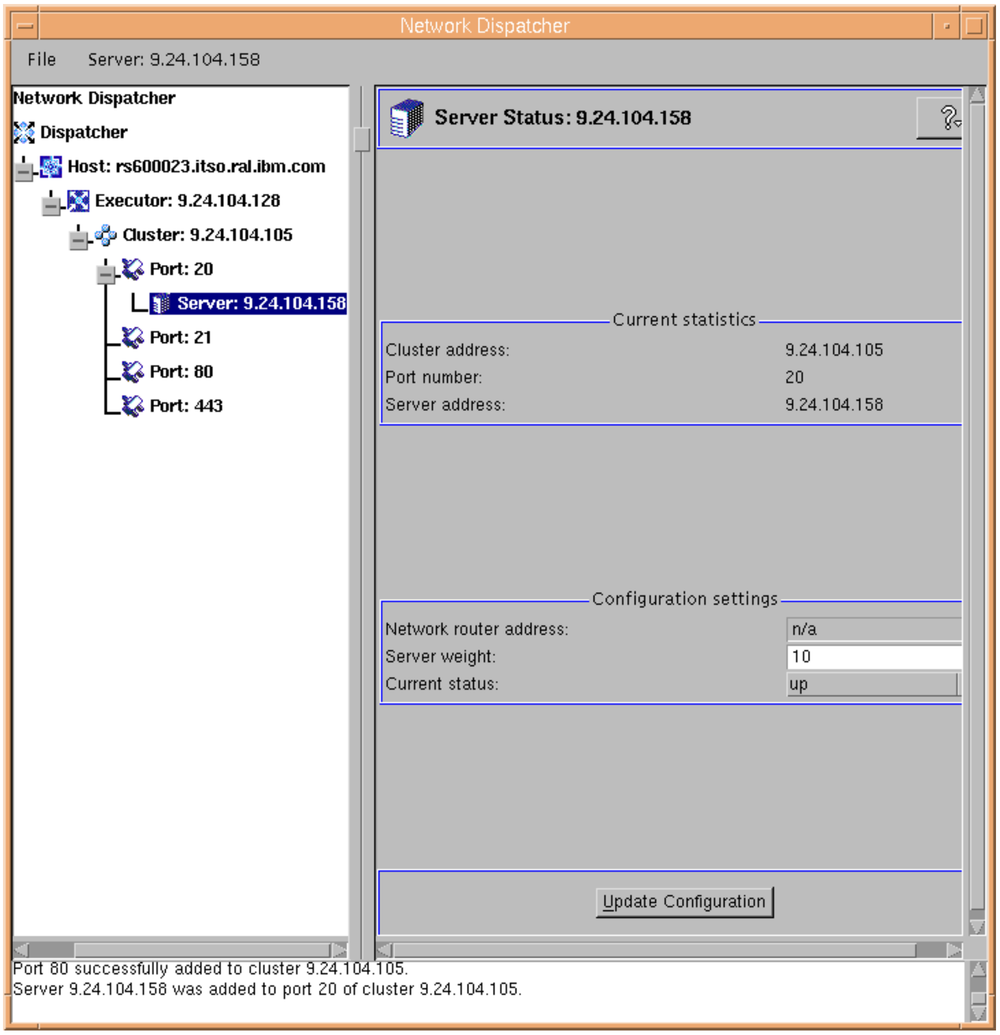
Server address: 9.24.104.158

Network router address: n/a

Figure 60. Enter Server Address

You can optionally click the **Network router address** radio button and fill in the network router address if you are making use of the Wide Area Network Dispatcher (WAND) support function. See Chapter 8, “Wide Area Network Dispatcher Support” on page 211, for further details on how to do this.

We clicked **OK**, and by expanding the Port 20 item, we ensured that the server with IP address 9.24.104.158 had been added to port 20 of the cluster with IP address 9.24.104.105, as shown in the following window:



Cluster address: 9.24.104.105
Port number: 20
Server address: 9.24.104.158

Network router address: n/a
Server weight: 10
Current status: up

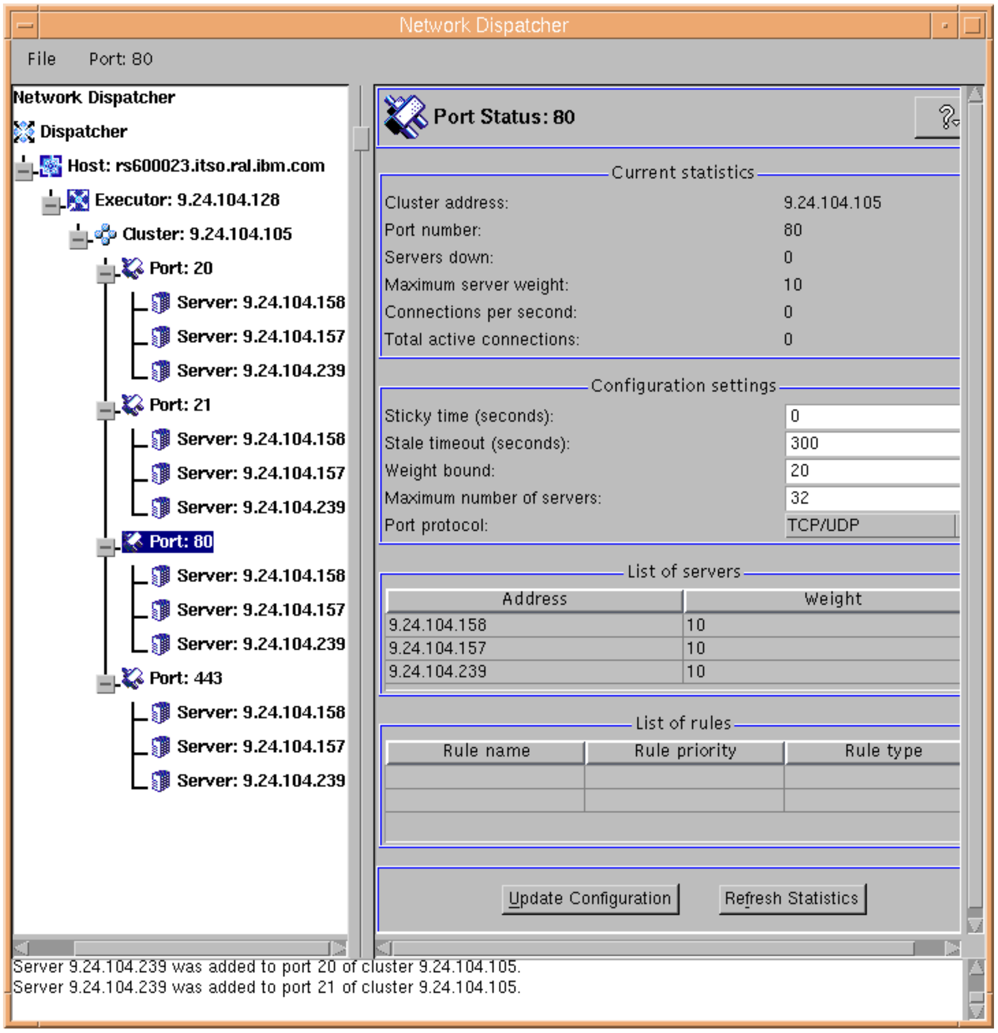
```
Port 80 successfully added to cluster 9.24.104.105.
Server 9.24.104.158 was added to port 20 of cluster 9.24.104.105.
```

Figure 61. Verifying that the Server Has Been Added

Using the GUI, we added all the TCP servers of our environment to port 20 and 21. Then from a command line we added the same Web servers to port 80 and 443, by entering the following two commands:

```
ndcontrol server add 9.24.104.105:80:9.24.104.158+9.24.104.157+9.24.104.239
ndcontrol server add 9.24.104.105:443:9.24.104.158+9.24.104.157+9.24.104.239
```

Following is the updated ND GUI showing our configuration:



Current statistics

Cluster address:	9.24.104.105
Port number:	80
Servers down:	0
Maximum server weight:	10
Connections per second:	0
Total active connections:	0

Configuration settings

Sticky time (seconds):	0
Stale timeout (seconds):	300
Weight bound:	20
Maximum number of servers:	32
Port protocol:	TCP/UDP

List of servers

Address	Weight
9.24.104.158	10
9.24.104.157	10
9.24.104.239	10

List of rules

Rule name	Rule priority	Rule type

```
Server 9.24.104.239 was added to port 20 of cluster 9.24.104.105.
Server 9.24.104.239 was added to port 21 of cluster 9.24.104.105.
```

Figure 62. All Servers Added

At this point of the configuration, the Dispatcher machine is already capable of performing load balancing using *weighted round-robin scheduling*, which is based on the current servers' weights. Weighted round-robin scheduling is not yet the best way to implement load balancing, because you can change the weights of the TCP servers statically, but they are not modified dynamically during the load-balancing activities. So, for example, if one of the TCP servers becomes very busy and cannot activate any more sessions, the Dispatcher, in the configuration shown above, is not able to reduce the TCP server's weight and increase the other servers' weights, and will continue to forward requests to that TCP server irrespective of the workload. More importantly, the Dispatcher alone does not have the ability to detect whether one of your server daemons has died.

The Manager component of the Dispathcer solves this problem. The Manager can dynamically set weights of the TCP servers in the cluster, and determines each single weight, basing its decision on internal counters in the Executor, feedback from the Advisors, and feedback from a system-monitoring program, such as ISS. Notice that weights are applied to all servers on a given port.

In this particular configuration, we wanted to set weights manually, and so we did not run the Manager component. We accepted the default values for weights. As you can see in Figure 62, each server has a weight of 10 on the HTTP port 80. In this case, since all the servers would be balanced with the same weight, the weighted round-robin load balancing performed by the Executor is reduced to a standard round-robin balancing of client requests, done by the Dispatcher. We used this weighted (with equal weights) round-robin approach to load balancing until we started using the Manager component as shown in 4.1.9, "Activating the Managers and the Advisors" on page 124.

4.1.5 TCP Servers Configuration

Before the Dispatcher starts to forward TCP/IP connection requests to the TCP servers, it is necessary to set up the TCP server machines. For the reasons explained in 2.5, "How the Dispatcher Function Works" on page 34, on each server in the cluster, you must add an alias to the cluster address on the loopback interface, often called lo0. This is the only configuration necessary in the TCP server machines in order for them to be load balanced by the Dispatcher. The loopback IP address is usually 127.0.0.1 and is never forwarded as a destination on the network media.

OS/2 and HP-UX Operating Systems

If you have a server running on an operating system that does not support aliases, such as HP-UX and OS/2, you must *set* the loopback device to the cluster address.

Other operating systems, such as Windows NT, AIX and Solaris, support aliases, so you can follow the procedure we describe now.

The Dispatcher does not change the destination IP address in the TCP/IP packet before forwarding the packet to a TCP server machine. By setting or aliasing the loopback device to the cluster address, the TCP server machine will accept a packet that was addressed to the cluster address.

Before going on with the TCP server's configuration, you should understand the flow of the incoming and outgoing IP packets. See 2.5, "How the Dispatcher Function Works" on page 34 and 4.1.6, "How the Dispatcher Works – The Flow of the IP Packets" on page 114 for detailed information on the IP packet flow. However, if you prefer to go on with the configuration steps, you can safely continue with the next section.

4.1.5.1 Aliasing the Loopback Device on AIX

To alias the loopback device on an AIX machine, use the following command syntax:

```
ifconfig lo0 alias cluster_address netmask netmask
```

On our AIX Web server machines (see Table 4 on page 81), we entered the command:

```
ifconfig lo0 alias 9.24.104.105 netmask 255.0.0.0
```

Locating An Extra Route

We found that on AIX, when we added the alias on lo0 with an incorrect subnet mask value, a new route was automatically added by the system. The following two figures show output from the command:

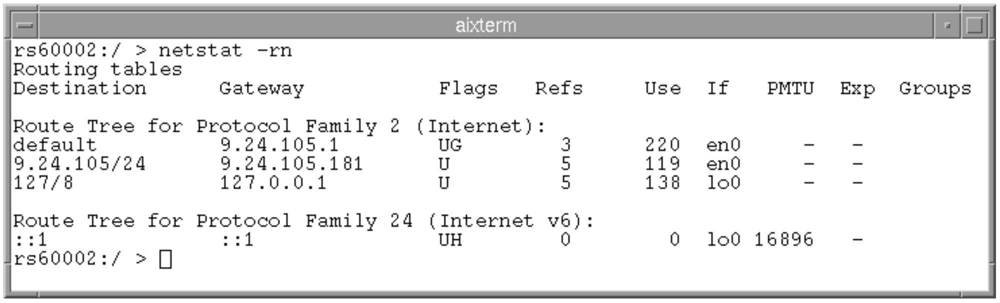
```
netstat -rn
```

executed before and after an alias was defined on an AIX 4.3.2 test system, which was not part of this scenario. We created the route with:

```
ifconfig lo0 alias 9.24.104.105 netmask 255.0.0.0
```

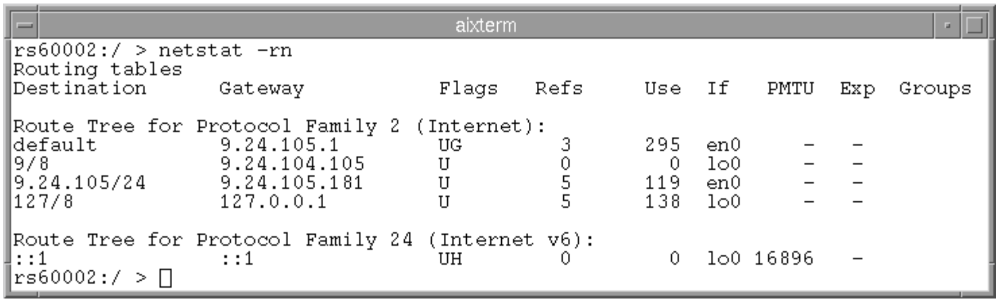
In this case we determined that there was an extra route. It was the one in the second row of the Route Tree shown in Figure 64 on page 108 and we deleted it with the following command:

```
route delete 9/8 9.24.104.105
```



```
rs60002:/ > netstat -rn
Routing tables
Destination      Gateway           Flags   Refs     Use  If   PMTU  Exp  Groups

Route Tree for Protocol Family 2 (Internet):
default          9.24.105.1        UG        3      220  en0     -   -
9.24.105/24      9.24.105.181      U         5      119  en0     -   -
127/8            127.0.0.1         U         5      138  lo0     -   -

Route Tree for Protocol Family 24 (Internet v6):
::1              ::1               UH        0        0  lo0 16896   -
rs60002:/ >
```

Figure 63. netstat -rn Output before the Alias Was Added



```
rs60002:/ > netstat -rn
Routing tables
Destination      Gateway           Flags   Refs     Use  If   PMTU  Exp  Groups

Route Tree for Protocol Family 2 (Internet):
default          9.24.105.1        UG        3      295  en0     -   -
9/8              9.24.104.105      U         0        0  lo0     -   -
9.24.105/24      9.24.105.181      U         5      119  en0     -   -
127/8            127.0.0.1         U         5      138  lo0     -   -

Route Tree for Protocol Family 24 (Internet v6):
::1              ::1               UH        0        0  lo0 16896   -
rs60002:/ >
```

Figure 64. netstat -rn Output after the Alias Was Added

You must alias the loopback device each time your AIX machine reboots, so it is recommended that you put the previous command in a script and instruct the machine to run it at reboot time. What we did in our case was to put the previous command into the file /etc/rc.tcpip. This file contains commands that start TCP/IP daemons (sendmail, inetd, etc.) on an AIX machine. It is launched at each reboot. We added at the end of that file the following lines (note, however, that the first and third lines contain only comments):

```
#Add alias on the loopback device:
ifconfig lo0 alias 9.24.104.105 netmask 255.0.0.0
#Remove the extra route:
route delete 9/8 9.24.104.105
```

4.1.5.2 Aliasing the Loopback Device on Solaris

To alias the loopback device on a Solaris system, use the following command syntax:

`ifconfig lo0:1 `*`cluster_address`*` 127.0.0.1 up`

For example:

```
ifconfig lo0:1 9.24.104.105 127.0.0.1 up
```

Also check for an extra route with the command:

```
netstat -rn
```

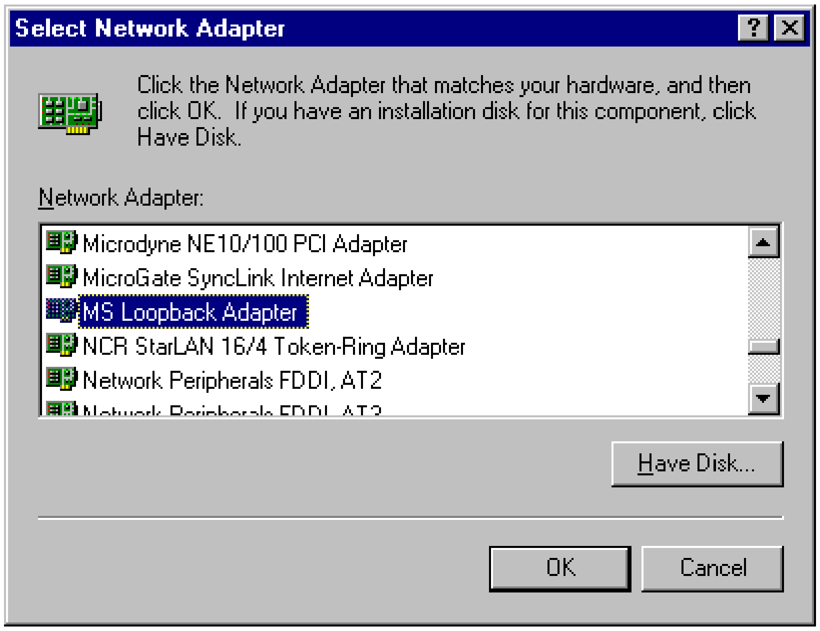
Then, if an extra route exists use the corresponding command:

```
route delete
```

4.1.5.3 Aliasing the Loopback Device on Windows NT

The procedure to add the alias on a Windows NT system is more complex, because you first have to add and then configure the loopback device, as reported in the following.

Click **Start** then **Settings**, and select **Control Panel**. Double-click the **Network** icon, and in the Network window click the **Adapters** tab. Then select the **Add...** button and from the list of adapters select **MS Loopback Adapter**, as shown in the following window:

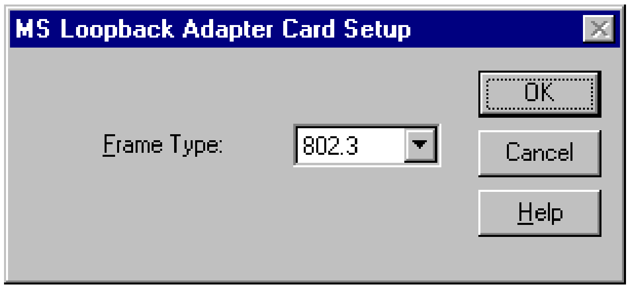


Click the Network Adapter that matches your hardware, and then click OK. If you have an installation disk for this component, click Have Disk.

Microdyne NE10/100 PCI Adapter
MicroGate SyncLink Internet Adapter
MS Loopback Adapter
NCR StarLAN 16/4 Token-Ring Adapter
Network Peripherals FDDI, AT2

Figure 65. Select Network Adapter: MS Loopback Adapter

Click **OK** and you will get the following screen:

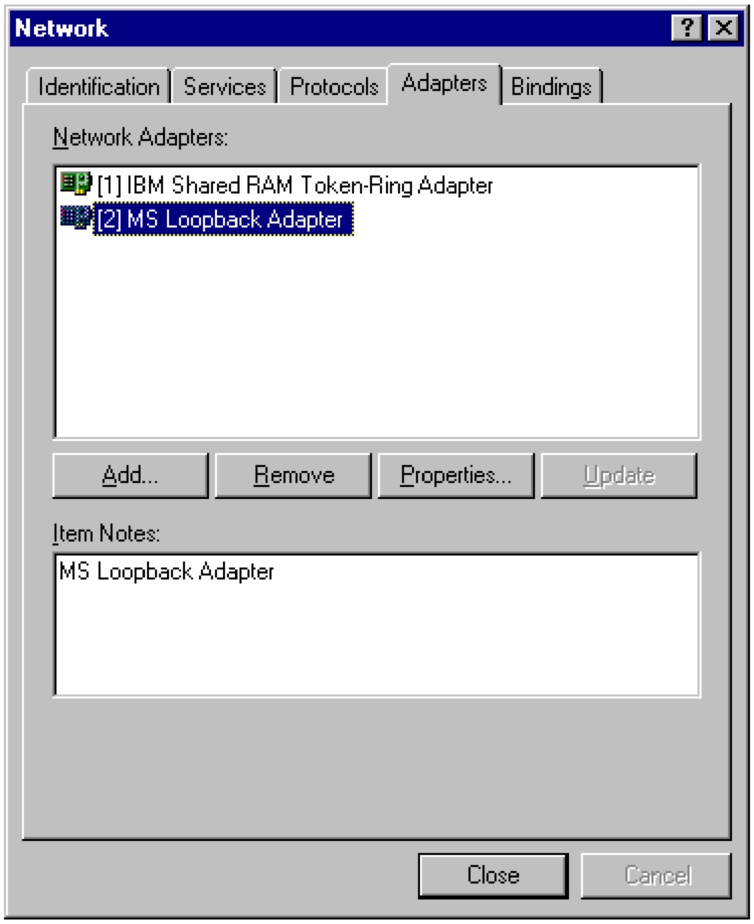


Frame Type: 802.3

Figure 66. MS Loopback Adapter Card Setup

We accepted the default configuration, clicked **OK** and, when prompted, inserted the installation Windows NT CD-ROM.

After these simple steps, you will see another adapter added in your Windows NT machine adapter list:



Network Adapters:

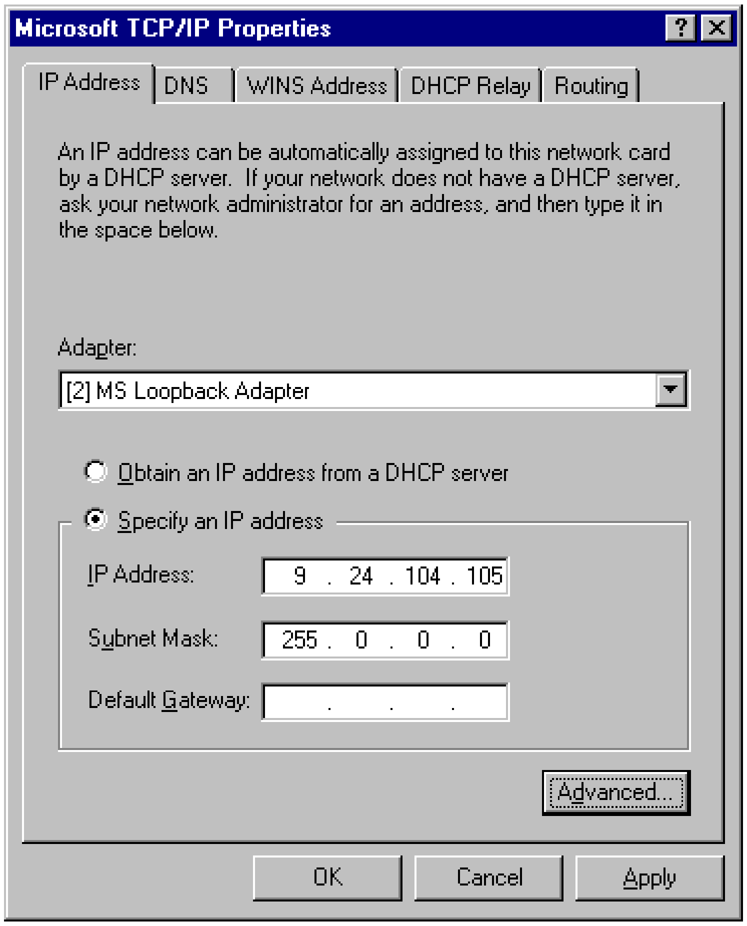
[1] IBM Shared RAM Token-Ring Adapter
[2] MS Loopback Adapter

Item Notes:

MS Loopback Adapter

Figure 67. Adapters List Refreshed

Now select the **Protocols** tab and double-click the **TCP/IP Protocol** item.You should then see the following window.


Microsoft TCP/IP Properties

An IP address can be automatically assigned to this network card by a DHCP server. If your network does not have a DHCP server, ask your network administrator for an address, and then type it in the space below.

Adapter: [2] MS Loopback Adapter

○ Obtain an IP address from a DHCP server

● Specify an IP address

IP Address: 9 . 24 . 104 . 105

Subnet Mask: 255 . 0 . 0 . 0

Default Gateway: . . .

Figure 68. Configuring MS Loopback Adapter

If you open the Adapter list box, the MS Loopback Adapter should have been added to the list.

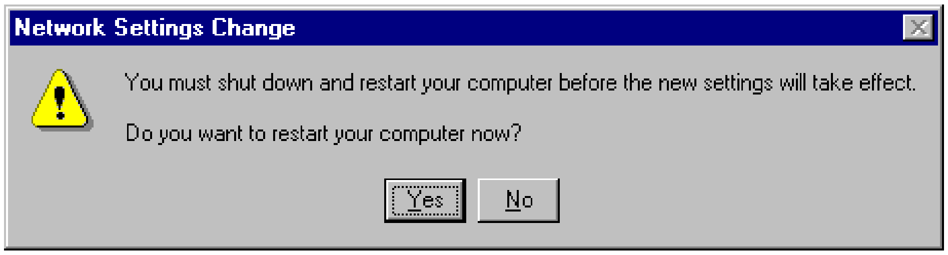
Adapter Missing?

If the MS Loopback Adapter does not appear in the Adapter list box under the TCP/IP protocol configuration window, this is because the Network panel needs to be refreshed. You need to close and re-open the Microsoft TCP/IP Properties window to see the new adapter.

As shown in Figure 68, select **MS Loopback Adapter** from the Adapter list box and set the IP Address to the cluster address.

We suggest that you type `255.0.0.0` in the Subnet Mask field and do not enter a gateway address. The setting of the subnet mask is not relevant on the loopback interface and the reason for using 255.0.0.0 will become clear shortly.

We used the default subnet mask 255.0.0.0 and did not enter a gateway address. Upon selecting the **OK** button, you will get the following window:



You must shut down and restart your computer before the new settings will take effect.

Do you want to restart your computer now?

Figure 69. Network Setting Change - Reboot Your System

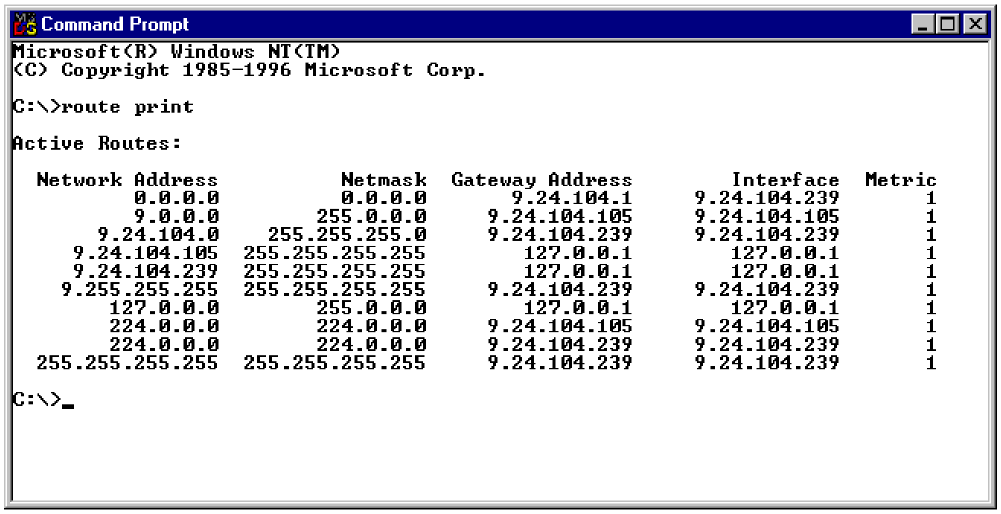
Choose **Yes** to reboot your system.

On Windows NT, after you install the MS Loopback Adapter, each time you reboot the system you do not need to redefine the alias on your loopback device. Instead, you have to accept the drawback that an extra route is created at each reboot, and you need to delete it.

To check for an extra route, from a command prompt enter the command:

```
route print
```

A table similar to the following will be displayed:



```
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>route print

Active Routes:

  Network Address          Netmask  Gateway Address        Interface  Metric
          0.0.0.0          0.0.0.0      9.24.104.1     9.24.104.239       1
          9.0.0.0        255.0.0.0    9.24.104.105     9.24.104.105       1
       9.24.104.0    255.255.255.0    9.24.104.239     9.24.104.239       1
     9.24.104.105  255.255.255.255       127.0.0.1        127.0.0.1       1
     9.24.104.239  255.255.255.255       127.0.0.1        127.0.0.1       1
    9.255.255.255  255.255.255.255    9.24.104.239     9.24.104.239       1
        127.0.0.0        255.0.0.0       127.0.0.1        127.0.0.1       1
        224.0.0.0        224.0.0.0    9.24.104.105     9.24.104.105       1
        224.0.0.0        224.0.0.0    9.24.104.239     9.24.104.239       1
  255.255.255.255  255.255.255.255    9.24.104.239     9.24.104.239       1

C:\>_
```

Figure 70. Locating The Extra Route

In order to find the extra route, you should look for your cluster address under the Gateway Address column. If the cluster address appears twice, it means you have an extra route. In our case, the cluster address 9.24.104.105 appeared in row 2 and row 8.

You need only one of those two entries; one of them is *extra*, and must be removed. To determine which route is the extra one that needs to be deleted, you can follow this simple rule, if you followed the suggestion to add the subnet mask for the loopback adapter with the value 255.0.0.0: *the extra route is the one that has in the Network Address column the first octet of the address you entered (the first number of the cluster address) followed by zeros in the other octets, and the Gateway Address is the address of the cluster.* Therefore, using a subnet mask of 255.0.0.0 makes it easier to spot the route that must be deleted.

If you used a different subnet mask, then the extra route is the one that has an address in the Network Address column that has the same octets as the cluster address in the network portion of the address with respect to the subnet mask used to create the alias in Figure 68. The host portion of the address in the Network Address column will be 0's.

In our case, the extra route is the one in row 2, since it has Network Address 9.0.0.0:

Table 6. Extra Route

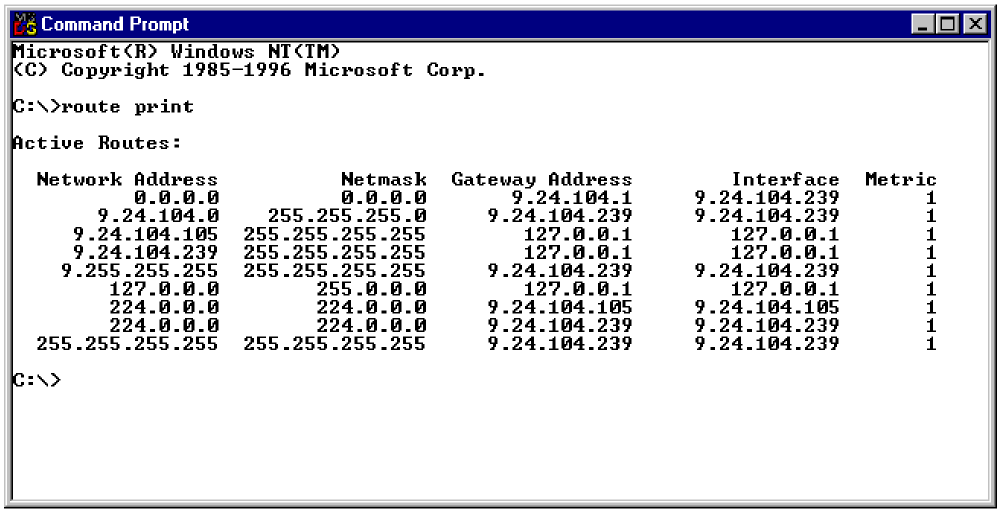
Network Address	Netmask	Gateway Address	Interface	Metric
9.0.0.0	255.0.0.0	9.24.104.105	9.24.104.105	1

We entered the following command to delete the extra route:

```
route delete 9.0.0.0 9.24.104.105
```

Then we checked that the extra route had been deleted by entering the following command:

```
route print
```



```
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>route print

Active Routes:

  Network Address          Netmask  Gateway Address        Interface  Metric
          0.0.0.0          0.0.0.0      9.24.104.1      9.24.104.239       1
       9.24.104.0    255.255.255.0    9.24.104.239      9.24.104.239       1
     9.24.104.105  255.255.255.255       127.0.0.1         127.0.0.1       1
     9.24.104.239  255.255.255.255       127.0.0.1         127.0.0.1       1
    9.255.255.255  255.255.255.255    9.24.104.239      9.24.104.239       1
        127.0.0.0        255.0.0.0       127.0.0.1         127.0.0.1       1
        224.0.0.0        224.0.0.0    9.24.104.105      9.24.104.105       1
        224.0.0.0        224.0.0.0    9.24.104.239      9.24.104.239       1
  255.255.255.255  255.255.255.255    9.24.104.239      9.24.104.239       1

C:\>
```

Figure 71. Extra Route Deleted

In order to automatically remove the extra route at each system reboot, you can copy the full `route delete` command in a batch text file, named for example Routedel.bat, and place it into your Windows NT Startup folder, as shown:

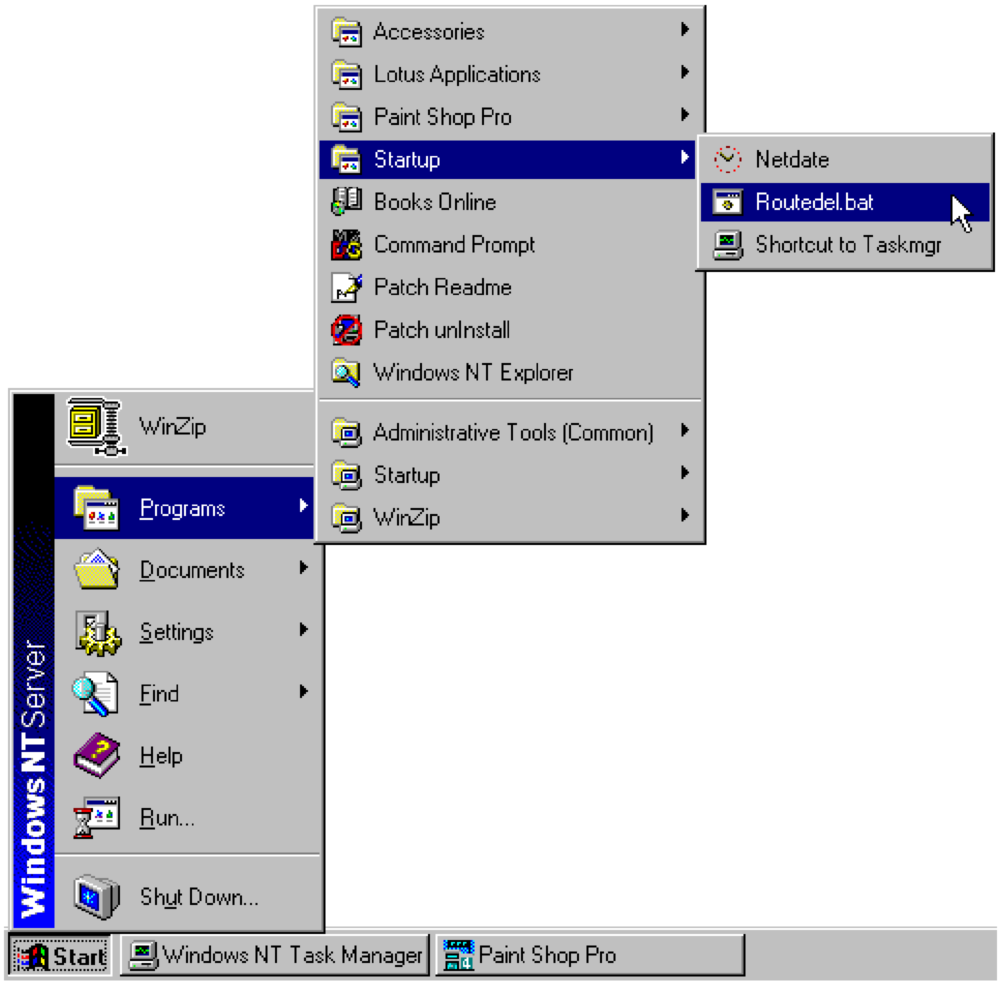


Figure 72. Startup Folder of Windows NT Server)

4.1.6 How the Dispatcher Works – The Flow of the IP Packets

In 2.5, “How the Dispatcher Function Works” on page 34, we explained how the IP packets start flowing from a TCP client to a TCP server passing through the Dispatcher machine, and then how the server’s response flows directly from the server to the client without any need to pass through the Dispatcher again. In this section, we want to explain in more detail how this important mechanism works.

You need to perform IP configuration on the Dispatcher machine and on the TCP servers. You also need to define aliases to the cluster address on the network interface on the Dispatcher machine and on the loopback device on all the cluster’s TCP servers. The reason for this is based on the way the Dispatcher works and treats incoming IP packets.

The Dispatcher is designed to make several TCP servers appear as one in the TCP/IP environment, typically for HTTP, FTP, and other protocols on the Internet. Let’s assume that a request for the service managed by the cluster arrives. The incoming IP packets have, among other data:

- The source address, which identifies who has sent the packet
- The destination address, which is the IP address of the cluster

For example, in the environment we built, an incoming IP packet has 9.24.104.218 as its source IP address, since this was the address of our Web client machine, and 9.24.104.105 as the destination IP address, since this is the cluster address (see Table 4 on page 81).

When the Dispatcher is installed, all the incoming IP packets sent by end users to the cluster address first arrive at the Dispatcher machine, not at one of the TCP servers. This is because the Dispatcher's network interface, besides having its own unique IP address, has been given an alias to the cluster address. The TCP servers in the cluster also have an alias to the cluster address, but this is defined on the loopback interface. The Dispatcher runs at a low level in the machine operating system so it can directly intercept all the IP packets.

Each time a new connection is initiated by a client, the Dispatcher selects which TCP server in the cluster should receive the connection packet. Now the Dispatcher should be able to send that packet to the selected TCP server. How can the Dispatcher do that? The answer is that the Dispatcher routes the packet based on the Media Access Control (MAC) address of the network adapter on the chosen TCP server.

MAC Address

Ethernet and token-ring devices require an adapter to physically attach to the LAN. This adapter must provide both physical and logical capabilities for the device. The adapter contains a unique 48-bit address, assigned to it during the manufacturer process, called MAC address. All the MAC addresses are assigned by the IEEE 802 committee. The IEEE provides the vendor building adapters with a range of MAC addresses to use for assigning adapters their unique 48-bit address so that no two adapters should ever have a duplicate address.

Ethernet and token-ring require the MAC address for both the origin and the destination adapters when communicating over a LAN. Besides the IP address, the MAC address also must be known when sending data to a LAN-attached device.

So let's assume that the Dispatcher has decided to send the connection IP packet to the TCP server that has the following particulars (see Table 4 on page 81):

- Unique IP address 9.24.104.239 and host name wtr05193
- Cluster address 9.24.104.105 as the alias on the loopback interface
- Adapter MAC address equal to 0006296AA301

The Dispatcher must now transmit the IP packet over the token-ring network (but in this context, Ethernet would be similar), so it must know the MAC address of the selected TCP server.

ARP Requests

If the TCP server's MAC address is unknown to the Dispatcher, the same Dispatcher is able to discover it by issuing an Address Protocol Request (ARP) containing a broadcast LAN MAC address that will be read by all the network interfaces attached to the same LAN. This request also specifies the IP address 9.24.104.239 of the TCP server. Each of the network adapters attached to the LAN will determine if its relative interface has been configured with the IP address 9.24.104.239. Only the network interface adapter of wtr05193 will respond with its own MAC address, 0006296AA301.

Once the Dispatcher knows the wanted TCP server's MAC address, it will be stored into the ARP cache to skip successive requests.

The original destination MAC address on the IP packet was the one of the network interface on the Dispatcher machine itself. When the Dispatcher knows the MAC address of the selected TCP server wtr05193, it changes the original destination MAC address on the packet to the MAC address 0006296AA301. This packet will now be encapsulated in the token-ring frame and transmitted to the chosen TCP server wtr05193.

The Dispatcher then sets up a connection table entry to make sure that subsequent incoming IP packets for this client continue to be forwarded to the same server, until the connection is interrupted.

When the TCP server wtr05193 receives the packet, the information related to the MAC addresses is eliminated and the source and destination IP addresses are extracted.

The destination IP address is still the cluster address 9.24.104.105, but the TCP server wtr05193 has its own IP address 9.24.104.239. However, wtr05193 can accept that packet, since the cluster address is configured as an alias on wtr05193's loopback interface.

At this point, wtr05193 sends a response to the Web client that originated the request. Now, let's see what happens to the outgoing packets.

Once the TCP server wtr05193 receives all the IP packets of the originating client's request, it performs the standard TCP processing that is commonly performed by all TCP servers while responding to a client. It switches the IP source and destination addresses for the outgoing packets that form the response to the client. The source address becomes in this case the cluster address 9.24.104.105, while the destination address is now the Web client IP address 9.24.104.218. This operation has an important consequence; the balancing function is transparent both to the client and the clustered servers:

1. The destination IP address is the client's IP address, not the Dispatcher's. For this reason, the TCP server wtr05193 can route the IP packets through its default router directly to the client, and all the outgoing packets do not pass back through the Dispatcher. There is no need to even return using the original physical path and a separate high-bandwidth connection can be used.

   This is very important, since in many cases, the volume of the outbound server-to-client traffic is substantially greater than the inbound traffic. Typically, HTML pages and imbedded images are at least 10 times the size of the client URLs that requested them.

2. The source IP address in the outgoing packet sent by the TCP server shows as the cluster address, and not as the TCP server's IP address. The cluster address was also the destination IP address in the IP packets sent by the client in its request, so the client is not able to understand the TCP architecture of the target server. This security feature offered by the Dispatcher ensures privacy for a site that is composed of multiple TCP servers load-balanced by a Dispatcher machine.

   A user at the client machine on which a network monitoring tool is running could understand that multiple TCP servers are part of the same cluster only if no router, firewall or gateway is between the client itself and the servers – in other words, the client and the servers must be part of the same intranet. In this case, in fact, the network monitoring tool would show the MAC address of the TCP server that actually served the request. In a real-life situation, the MAC address shown in the packet is that of the router closest to the client.

The following diagram offers a graphical representation of the flow we have just described:

1

	Source	Destination
IP Address	Client	Cluster
MAC Address	Client	Router

2

	Source	Destination
IP Address	Client	Cluster
MAC Address	Router	ND Machine

3

	Source	Destination
IP Address	Client	Cluster
MAC Address	ND Machine	TCP Server

4

	Source	Destination
IP Address	Cluster	Client
MAC Address	TCP Server	Router

5

	Source	Destination
IP Address	Cluster	Client
MAC Address	Router	Client

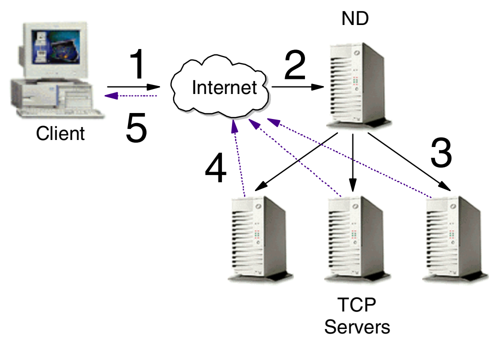


Figure 73. The Flow of the IP Packets

Because the Dispatcher does not participate in bidirectional communications with the client but simply forwards the incoming packets unchanged, its presence is transparent to both client and server. The real TCP/IP connection is between the client and the clustered server, and the Dispatcher soon disappears from the scene after forwarding the incoming packets.

The only requirement for the TCP server is that its loopback device be set or aliased to the cluster address. In this way, the server is capable of responding to a request that was addressed to the cluster address.

4.1.7 RoundRobin Load Balancing Scenario

Now that the TCP servers are configured, we want to show you how the Dispatcher is able to do standard round-robin load balancing. Before going in-depth, here is an explanation about the weights of the servers connecting to a port in the cluster. As we also mentioned in 4.1.4, “Dispatcher Configuration” on page 83, weights are applied to all servers on a given port, and for any particular port, the requests will be distributed between servers based on their weights relative to each other. For example, if the weight on one server is set to 10 and on the other server to 5, the first server gets twice as many requests as the second server.

In our configuration, since the weight for each server on port 80 had been set to 10 (see Figure 62 on page 106), we expected the Dispatcher to perform the standard (not weighted) round-robin load balancing.

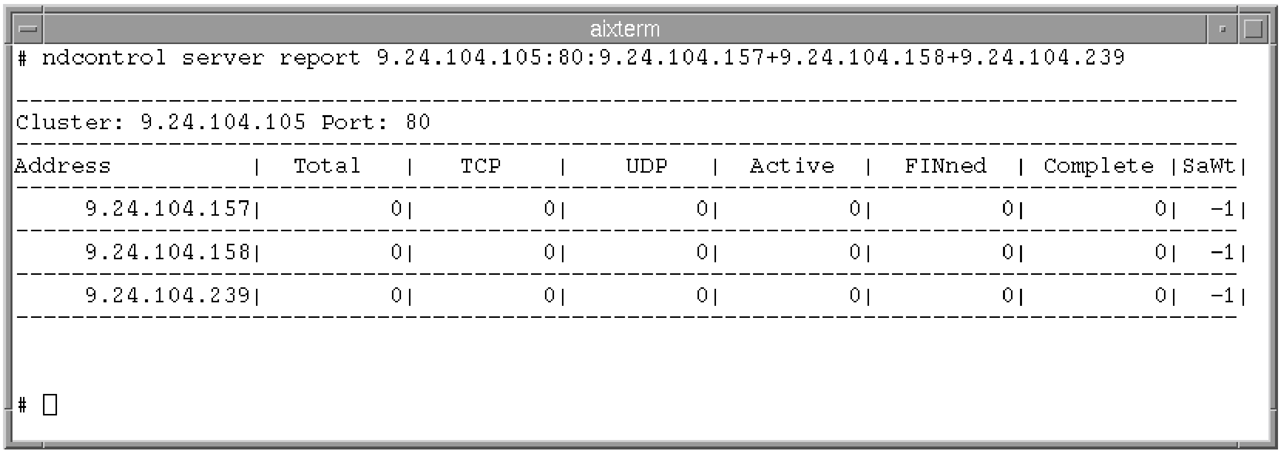
To do that, we performed the following demonstration. From a command line on the Dispatcher machine, we entered the following command:

```
ndcontrol server report 9.24.104.105:80:9.24.104.157+9.24.104.158+9.24.104.239
```

A shortcut to the command above would have been:

```
ndcontrol server report ::
```

We saw that the total number of TCP connections on port 80 to the servers was 0:



```
# ndcontrol server report 9.24.104.105:80:9.24.104.157+9.24.104.158+9.24.104.239

-----------------------------------------------------------------------------------------
Cluster: 9.24.104.105 Port: 80
-----------------------------------------------------------------------------------------
Address          |  Total   |   TCP    |    UDP   |  Active  |  FINned  | Complete |SaWt|
-----------------------------------------------------------------------------------------
     9.24.104.157|         0|         0|         0|         0|         0|         0|  -1|
-----------------------------------------------------------------------------------------
     9.24.104.158|         0|         0|         0|         0|         0|         0|  -1|
-----------------------------------------------------------------------------------------
     9.24.104.239|         0|         0|         0|         0|         0|         0|  -1|
-----------------------------------------------------------------------------------------


#
```

Figure 74. TCP Connections on Port 80

We could verify with this that no connection had yet been activated.

Then we wrote three HTML files, which were very simple and very similar to one another. The three of them were named cjl.html, and we distributed them all into the document root directories of the three clustered Web servers of our platform:

```
<html>
<title>Test Page from aixncf157</title>
<body>

This page comes from aixncf157.itso.ral.ibm.com
with ip address 9.24.104.157
</body>
</html>
```

Figure 75. /usr/lpp/HTTPServer/share/htdocs/cjl.html on Host 9.24.104.157

```
<html>
<title>Test Page from AIXAFS</title>
<body>

This page comes from AIXAFS.itso.ral.ibm.com
with ip address 9.24.104.158
</body>
</html>
```

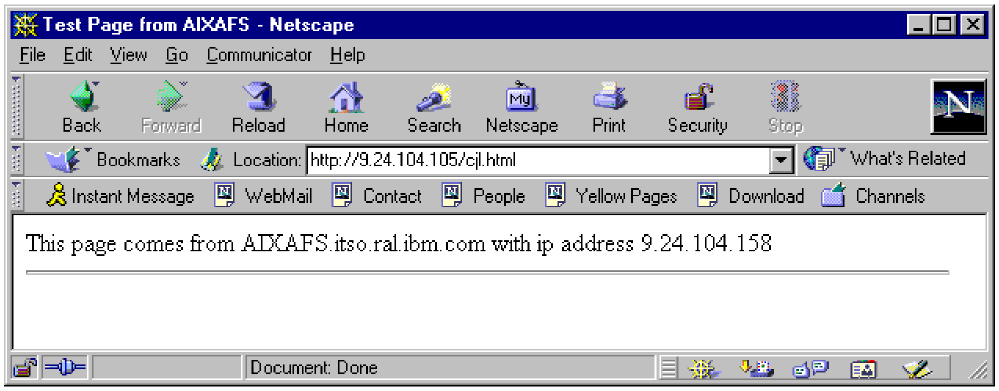
Figure 76. /usr/lpp/HTTPServer/share/htdocs/cjl.html on Host 9.24.104.158

```
<html>
<title>Test Page from aixncf157</title>
<body>

This page comes from aixncf157.itso.ral.ibm.com
with ip address 9.24.104.157
</body>
</html>
```

Figure 77. C:\Program Files\IBM HTTPServer\htdocs\cjl.htm on 9.24.104.239

Notice that the three files have the same structure. In each file it is specified what server in the cluster is actually the owner of the page. This will be displayed not only in the body of the Web page, but also in the title bar of the browser window, since we have inserted the host name of the TCP server between the HTML tags `<title>` and `</title>`.

We started our Web browser and de-activated its memory and disk caches; then we requested the URL `http://9.24.104.105/cjl.html` and got the following screen:


Test Page from AIXAFS - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://9.24.104.105/cjl.html What's Related
Instant Message WebMail Contact People Yellow Pages Download Channels
Document: Done

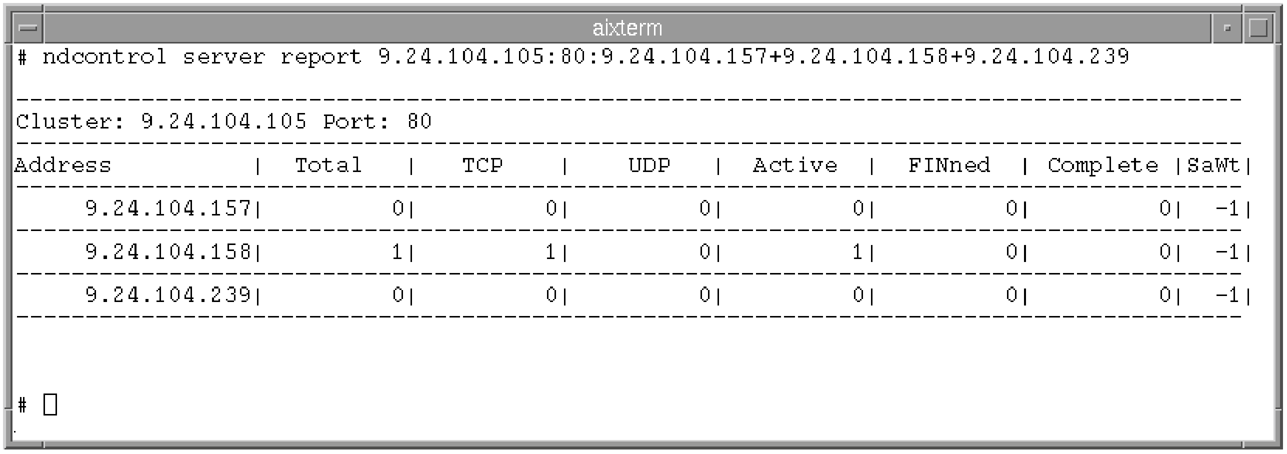
This page comes from AIXAFS.itso.ral.ibm.com with ip address 9.24.104.158

Figure 78. HTML Page Served by the First Web Server

Note that the requested page came from the Web server with IP address 9.24.104.158. By entering the following command again we saw the Dispatcher report:

```
ndcontrol server report 9.24.104.105:80:9.24.104.157+9.24.104.158+9.24.104.239
```

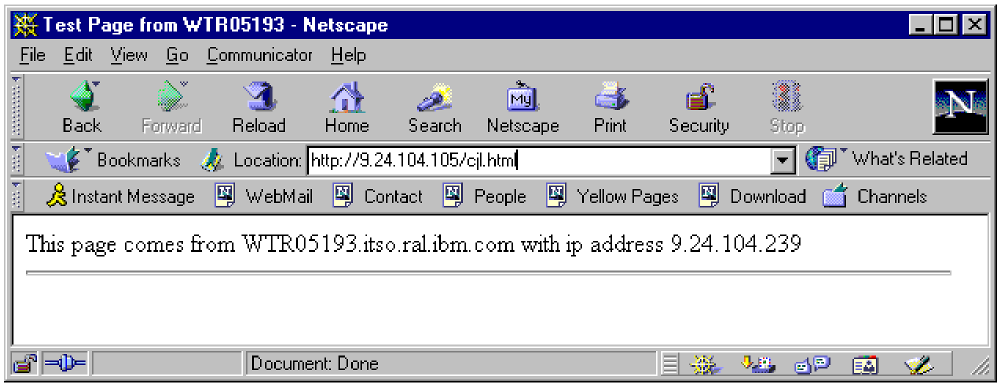
It confirmed that the server with IP address 9.24.104.158 had effectively served the request, as shown in the following figure:



```
# ndcontrol server report 9.24.104.105:80:9.24.104.157+9.24.104.158+9.24.104.239

-----------------------------------------------------------------------------------
Cluster: 9.24.104.105 Port: 80
-----------------------------------------------------------------------------------
Address            |  Total   |   TCP    |   UDP    |  Active  |  FINned  | Complete |SaWt|
-----------------------------------------------------------------------------------
     9.24.104.157|         0|         0|         0|         0|         0|         0|  -1|
-----------------------------------------------------------------------------------
     9.24.104.158|         1|         1|         0|         1|         0|         0|  -1|
-----------------------------------------------------------------------------------
     9.24.104.239|         0|         0|         0|         0|         0|         0|  -1|
-----------------------------------------------------------------------------------


# 
```

Figure 79. The First Web Server Has Served One Page

Then we reloaded the page from the browser by clicking the **Reload** button. This time the page came from another server, the one with IP address 9.24.104.239, as shown in the following figure:


Test Page from WTR05193 - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://9.24.104.105/cjl.html What's Related
Instant Message WebMail Contact People Yellow Pages Download Channels
Document: Done

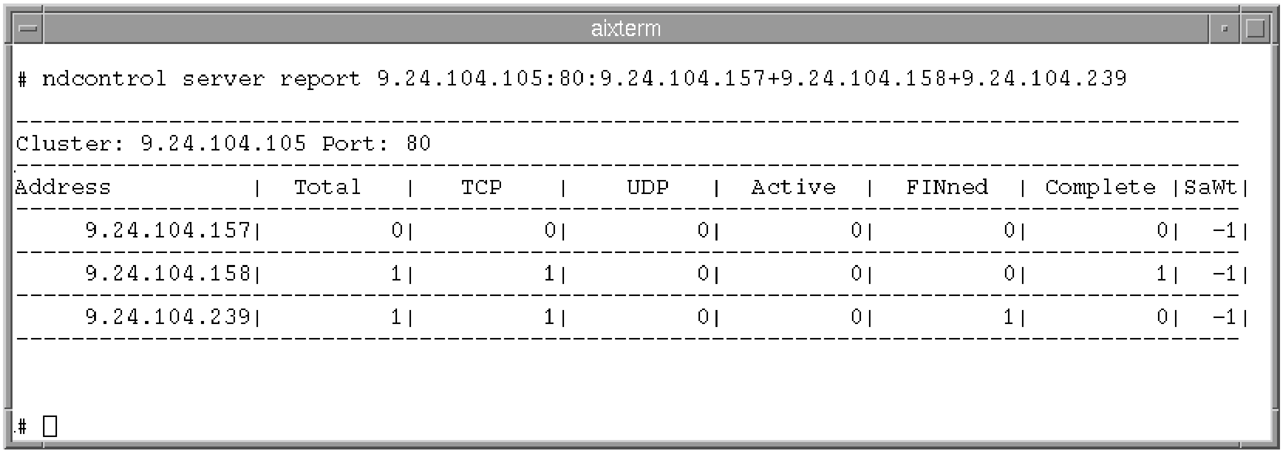
This page comes from WTR05193.itso.ral.ibm.com with ip address 9.24.104.239

Figure 80. HTML Page Served by the Second Web Server

Upon re-entering the following command we had another confirmation that the server with IP address 9.24.104.239 had effectively served the request:

```
ndcontrol server report 9.24.104.105:80:9.24.104.157+9.24.104.158+9.24.104.239
```

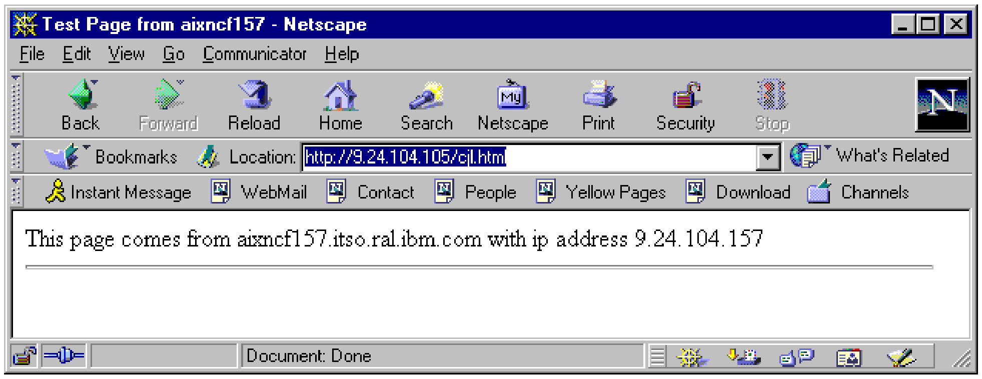
This confirmation is shown in the following figure:



```
# ndcontrol server report 9.24.104.105:80:9.24.104.157+9.24.104.158+9.24.104.239

-------------------------------------------------------------------------------------------
Cluster: 9.24.104.105 Port: 80
-------------------------------------------------------------------------------------------
Address              |   Total    |    TCP     |    UDP     |  Active    |  FINned    | Complete |SaWt|
-------------------------------------------------------------------------------------------
     9.24.104.157|          0|          0|          0|          0|          0|          0|  -1|
-------------------------------------------------------------------------------------------
     9.24.104.158|          1|          1|          0|          0|          0|          1|  -1|
-------------------------------------------------------------------------------------------
     9.24.104.239|          1|          1|          0|          0|          1|          0|  -1|
-------------------------------------------------------------------------------------------


#
```

Figure 81. The Second Web Server Has Served One Page

At the end, we reloaded the page on the browser for the third time and saw that the file cjl.html was served by the server with IP address 9.24.104.157, as shown in the following figure:


Test Page from aixncf157 - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://9.24.104.105/cjl.html What's Related
Instant Message WebMail Contact People Yellow Pages Download Channels
Document: Done

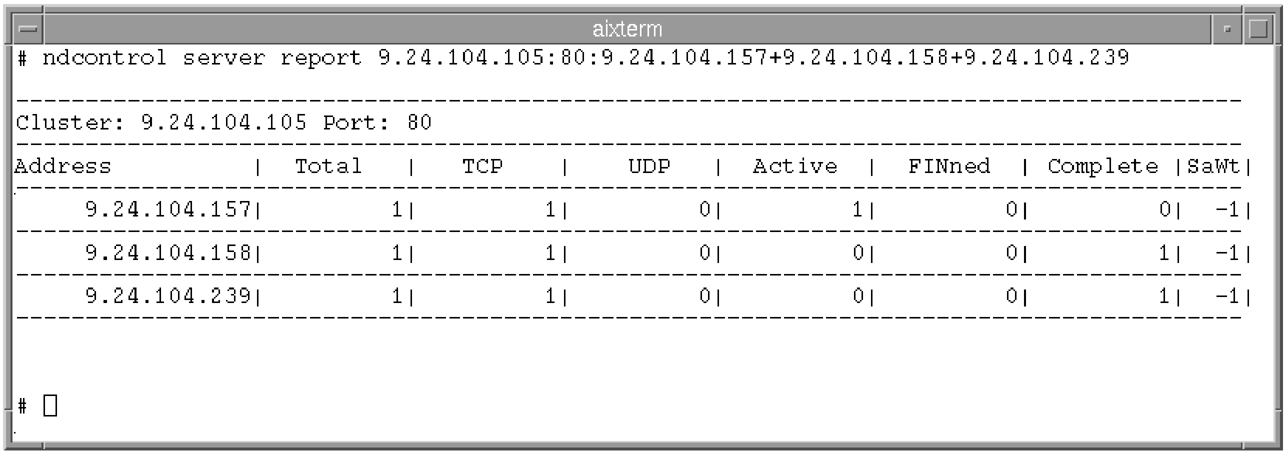
This page comes from aixncf157.itso.ral.ibm.com with ip address 9.24.104.157

Figure 82. HTML Page Served by the Third Server

We again entered the command:

```
ndcontrol server report 9.24.104.105:80:9.24.104.157+9.24.104.158+9.24.104.239
```

This time we saw the final report showing that all three machines had been accessed one time each:



```
# ndcontrol server report 9.24.104.105:80:9.24.104.157+9.24.104.158+9.24.104.239

-------------------------------------------------------------------------------------
Cluster: 9.24.104.105 Port: 80
-------------------------------------------------------------------------------------
Address           |  Total   |   TCP    |    UDP   |  Active  |  FINned  | Complete |SaWt|
-------------------------------------------------------------------------------------
     9.24.104.157|         1|         1|         0|         1|         0|         0|  -1|
-------------------------------------------------------------------------------------
     9.24.104.158|         1|         1|         0|         0|         0|         1|  -1|
-------------------------------------------------------------------------------------
     9.24.104.239|         1|         1|         0|         0|         0|         1|  -1|
-------------------------------------------------------------------------------------


# 
```

Figure 83. The Third Web Server Has Served One Page

So each time you reload the same Web page from the browser, the servers honoring the response alternate in a nonweighted round-robin way.

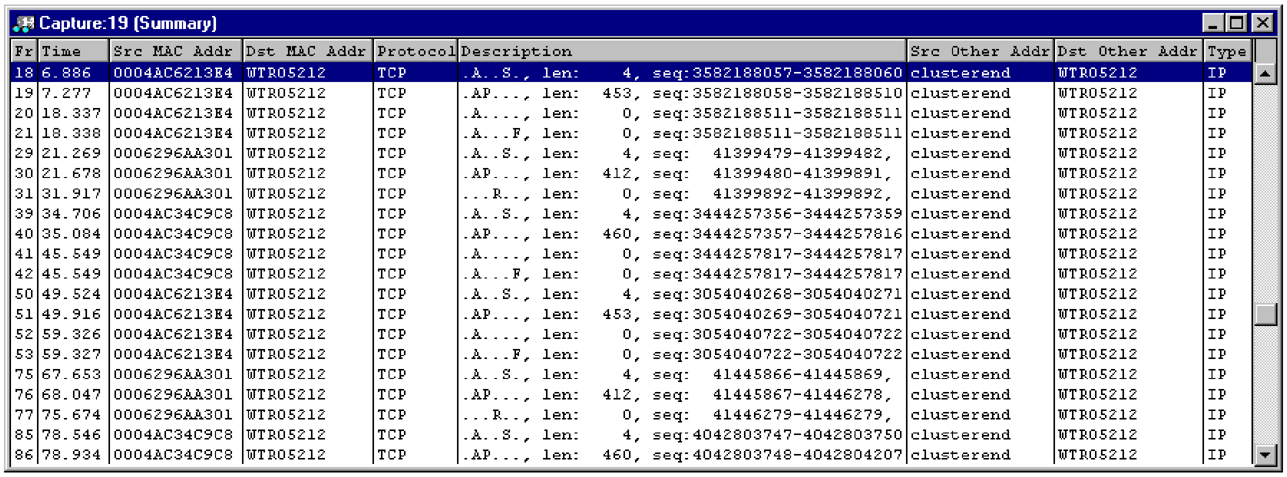
It is worth emphasizing that we used a different test page on each of the three servers. Each Web page was named cjl.html, but the contents were slightly different. We did this just to show the round-robin balancing of the Dispatcher. In a real situation, the pages would be identical, and this could be obtained by either duplicating the same Web page in all the Web servers or using AFS, the File Sharing component of IBM WebSphere Performance Pack, to dynamically distribute the same file in multiple locations.

When using the Dispatcher component of ND V2.1, if the page is really the same, the user cannot see any differences, since even the URL in the Location field of the browser and the information displayed on the Page Info window of the browser do not show that the page is served by different Web servers. The cluster address of your site is the only IP address that is shown. In this sense, the load balancing operation is transparent to the end user, since the Web server can respond directly to the client, as explained in 4.1.6, “How the Dispatcher Works – The Flow of the IP Packets” on page 114.

4.1.8 Analyzing the Flow with a Network Monitoring Tool

In this section we describe a scenario that helps us to understand how the Dispatcher function really works. In this scenario, we repeated the scenario described in 4.1.7, “RoundRobin Load Balancing Scenario” on page 118, but on the client machine we had Microsoft Network Monitor Version 4.00 for Windows NT Server running. This software can capture and display incoming packets being transmitted from other machines within the same LAN segment where the server running Network Monitor is attached.

The host name of the client machine was wtr05212, as specified in Table 4 on page 81. From the Web browser installed on this machine, we sent some requests to the clustered Web site we implemented, having cluster address 9.24.104.105, and we monitored the incoming network packets. We got the following table from the Network Monitor:



Fr	Time	Src MAC Addr	Dst MAC Addr	Protocol	Description	Src Other Addr	Dst Other Addr	Type
18	6.886	0004AC6213E4	WTR05212	TCP	.A..S., len: 4, seq:3582188057-3582188060	clusterend	WTR05212	IP
19	7.277	0004AC6213E4	WTR05212	TCP	.AP..., len: 453, seq:3582188058-3582188510	clusterend	WTR05212	IP
20	18.337	0004AC6213E4	WTR05212	TCP	.A...., len: 0, seq:3582188511-3582188511	clusterend	WTR05212	IP
21	18.338	0004AC6213E4	WTR05212	TCP	.A...F, len: 0, seq:3582188511-3582188511	clusterend	WTR05212	IP
29	21.269	0006296AA301	WTR05212	TCP	.A..S., len: 4, seq: 41399479-41399482,	clusterend	WTR05212	IP
30	21.678	0006296AA301	WTR05212	TCP	.AP..., len: 412, seq: 41399480-41399891,	clusterend	WTR05212	IP
31	31.917	0006296AA301	WTR05212	TCP	...R.., len: 0, seq: 41399892-41399892,	clusterend	WTR05212	IP
39	34.706	0004AC34C9C8	WTR05212	TCP	.A..S., len: 4, seq:3444257356-3444257359	clusterend	WTR05212	IP
40	35.084	0004AC34C9C8	WTR05212	TCP	.AP..., len: 460, seq:3444257357-3444257816	clusterend	WTR05212	IP
41	45.549	0004AC34C9C8	WTR05212	TCP	.A...., len: 0, seq:3444257817-3444257817	clusterend	WTR05212	IP
42	45.549	0004AC34C9C8	WTR05212	TCP	.A...F, len: 0, seq:3444257817-3444257817	clusterend	WTR05212	IP
50	49.524	0004AC6213E4	WTR05212	TCP	.A..S., len: 4, seq:3054040268-3054040271	clusterend	WTR05212	IP
51	49.916	0004AC6213E4	WTR05212	TCP	.AP..., len: 453, seq:3054040269-3054040721	clusterend	WTR05212	IP
52	59.326	0004AC6213E4	WTR05212	TCP	.A...., len: 0, seq:3054040722-3054040722	clusterend	WTR05212	IP
53	59.327	0004AC6213E4	WTR05212	TCP	.A...F, len: 0, seq:3054040722-3054040722	clusterend	WTR05212	IP
75	67.653	0006296AA301	WTR05212	TCP	.A..S., len: 4, seq: 41445866-41445869,	clusterend	WTR05212	IP
76	68.047	0006296AA301	WTR05212	TCP	.AP..., len: 412, seq: 41445867-41446278,	clusterend	WTR05212	IP
77	75.674	0006296AA301	WTR05212	TCP	...R.., len: 0, seq: 41446279-41446279,	clusterend	WTR05212	IP
85	78.546	0004AC34C9C8	WTR05212	TCP	.A..S., len: 4, seq:4042803747-4042803750	clusterend	WTR05212	IP
86	78.934	0004AC34C9C8	WTR05212	TCP	.AP..., len: 460, seq:4042803748-4042804207	clusterend	WTR05212	IP

Figure 84. Network Monitor Output Showing Mac Addresses of the Frames Source

In this table, take a look at the columns Src Other Addr (Source Other Address) and Dst Other Addr (Destination Other Address). You can see that the client machine cannot determine which server has honored the HTTP requests, since the source address from which the response arrived is always the cluster address (see 4.1.6, “How the Dispatcher Works – The Flow of the IP Packets” on page 114).

If you look closely at the table displayed by the Network Monitor, you can see that the Frame numbers are not consecutive. For clarity, we filtered out the frames that were being sent to our Windows NT machine from hosts that were not related to this demonstration.

The following table shows the MAC addresses of the Web server that were part of our cluster:

Table 7. Our Web Server MAC Addresses

Hostname	IP Address	Mac Address
aixafs	9.24.104.158	0004AC6213E4

Hostname	IP Address	Mac Address
wtr05193	9.24.104.239	0006296AA301
aixncf157	9.24.104.157	0004AC34C9C8

The Src MAC Addr (Source MAC Address) column shows us that Frames 18 through 21 were served by the Web Server with MAC address 0004AC6213E4. From Table 7 on page 123, we can see that the MAC address 0004AC6213E4 corresponds to host aixafs. Similarly, Frames 29 through 31 were served by wtr05193, Frames 39 through 42 were served by aixncf157 and so on. Each group of these frames represents the flow of data from the WebSphere machine to the Web Client machine for one page load operation. This confirms the analysis shown in 4.1.6, “How the Dispatcher Works – The Flow of the IP Packets” on page 114.

This demonstration would have not worked if the TCP servers had been located on a different LAN. In other words, we would not have been able to see the MAC addresses of the various TCP servers if such Web servers had been located on a different LAN. On the contrary, we would have seen the MAC address of the router closest to the client machine. For this reason, in a real-life situation, the TCP architecture of a clustered Web site remains hidden to remote clients, which results in a further security feature offered by the Dispatcher component of ND V2.1. This scenario will be demonstrated in 7.1.4, “Packet Flow” on page 208.

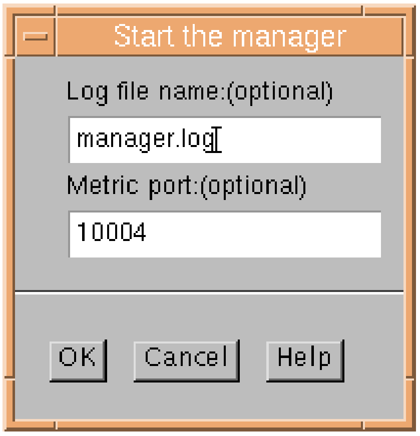
4.1.9 Activating the Managers and the Advisors

Now let’s go back to the GUI on the Dispatcher machine and activate the functions responsible for gathering and processing the information about the performance of the TCP servers for the purpose of improving load balancing: the Manager and the Advisors.

4.1.9.1 Activating the Manager

The main load balancing function is the Manager. The Manager is the function that collects the information from the Advisors about the servers’ conditions. Based on that information, it then dynamically adjusts the weights of the single server to reconfigure load distribution during run time.

If you want to start the Manager function through the GUI, right-click the **Host** item (see Figure 62 on page 106) and from the pop-up menu select **Start Manager...**. You are prompted to optionally type a log file name, in which the Manager will log all its activities, and a metric port used by the ISS function, if running, to report system loads. We used the default values for both the Log file name (`manager.log)` and Metric port (10004) fields, as shown in the following figure:

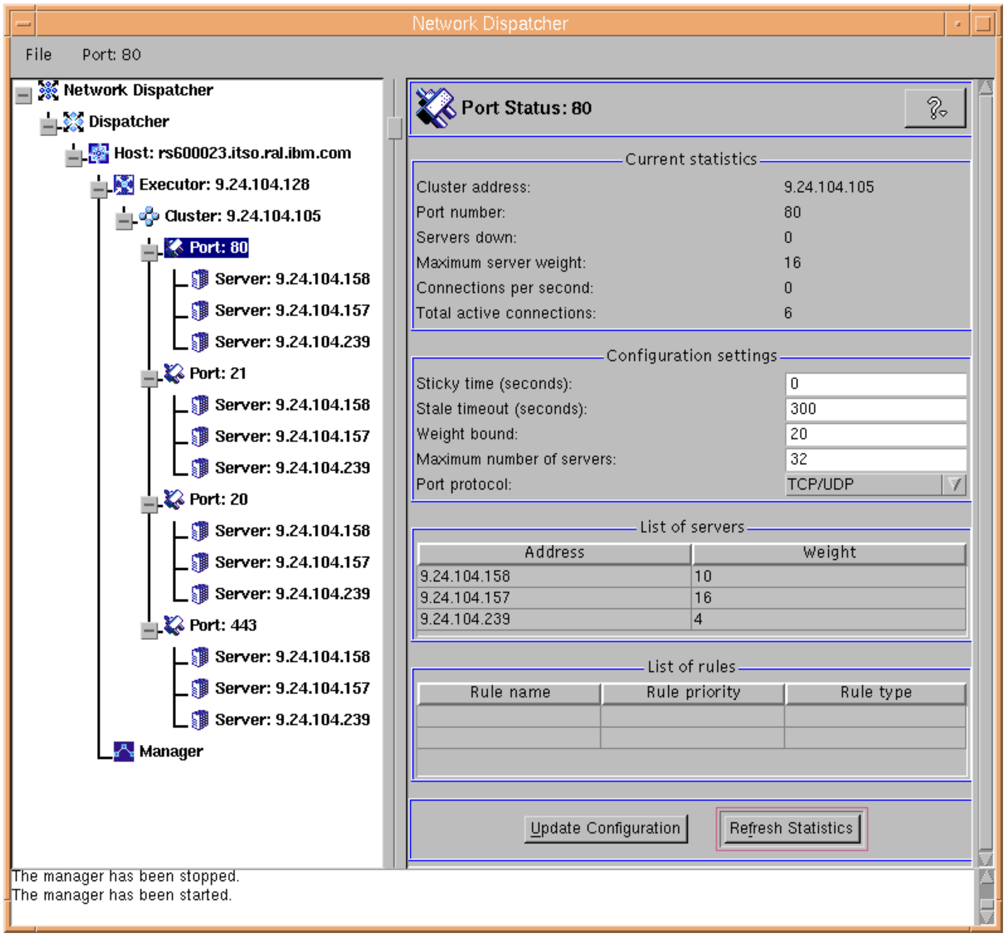


Log file name:(optional)
manager.log
Metric port:(optional)
10004

Figure 85. Starting the Manager

In 2.5.4, “TCP Ports Used by the Dispatcher” on page 38, we explained how to start the Manager function through the command line.

Once the Manager was started, the weights of the servers were changed by the Manager. To see this, select **Port: 80** and notice the Weight column in the List of Servers section. If your HTTP servers are busy, you can click the **Refresh Statistics** button and watch as the weights are dynamically changed by the Manager.



Current statistics

Cluster address:	9.24.104.105
Port number:	80
Servers down:	0
Maximum server weight:	16
Connections per second:	0
Total active connections:	6

Configuration settings

Sticky time (seconds):	0
Stale timeout (seconds):	300
Weight bound:	20
Maximum number of servers:	32
Port protocol:	TCP/UDP

List of servers

Address	Weight
9.24.104.158	10
9.24.104.157	16
9.24.104.239	4

List of rules

Rule name	Rule priority	Rule type

```
The manager has been stopped.
The manager has been started.
```

Figure 86. The Manager is Changing the Weights of the Port 80 Servers

4.1.9.2 Activating the Advisors

In order to feed the Manager with more information about the ability of the TCP servers to respond to requests, you need to start the Advisors. As we explained in 2.5.1, “Dispatcher Components” on page 35, the Advisors monitor each server defined on the assigned port, and forward the information about the server’s response time and availability to the Manager.

In the following table you find the list of the available Advisors along with their respective protocols and ports:

Table 8. Advisors and Related Protocol and Ports

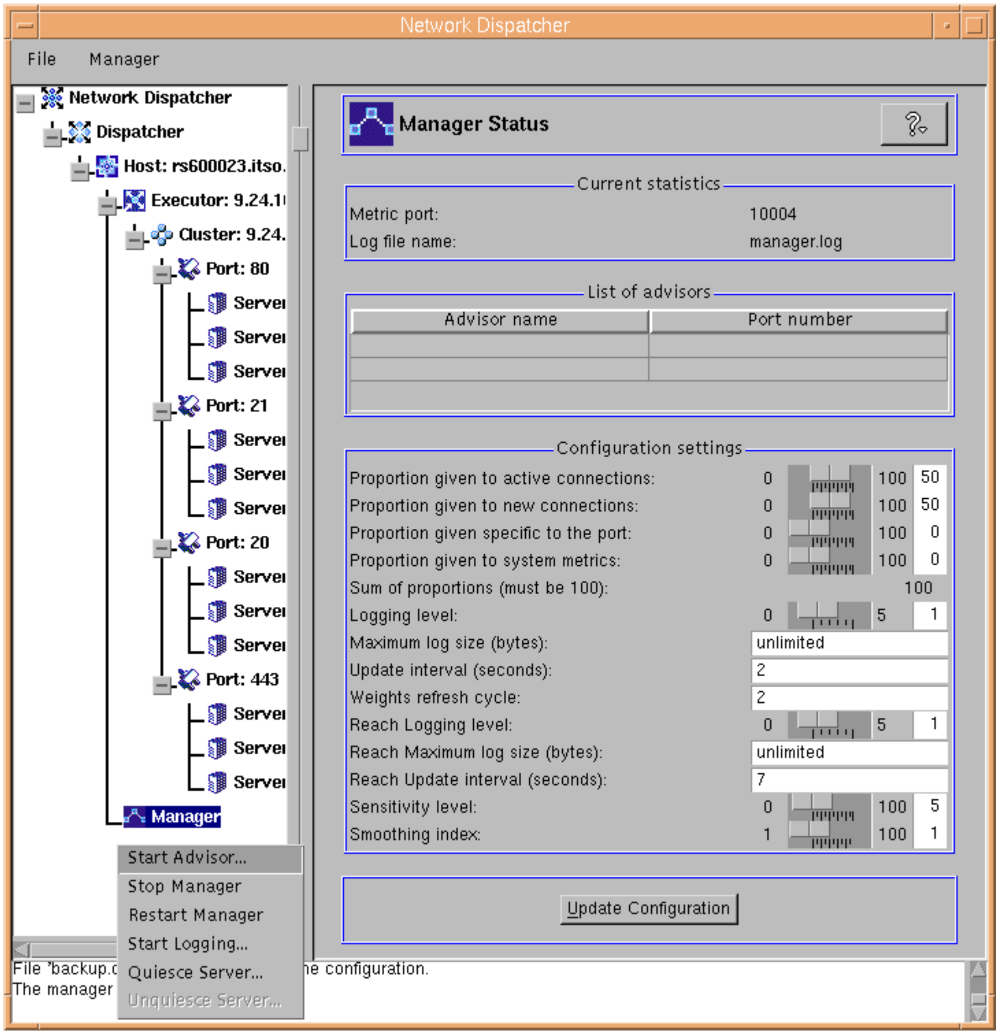
Advisor Name	Protocol	Port
ftp	FTP	21
telnet	Telnet	23
smtp	SMTP	25
http	HTTP	80
pop3	POP3	110
nntp	NNTP	119

Advisor Name	Protocol	Port
ssl	SSL	443
Workload Manager	private	10,007
WTE	HTTP	80
PING	ping	0

Note that if you use the FTP Advisor, it should be started only on the FTP control port 21, and not on the FTP data port 20 (see also Table 5 on page 101).

ND administrators have the ability to write a custom advisor for any protocol not mentioned in the table above.

To start the Advisor from the GUI, right-click **Manager** and select **Start Advisor...** from the pop-up menu that is automatically brought up, as shown in the following figure:



Current statistics

Metric port:	10004
Log file name:	manager.log

List of advisors

Advisor name	Port number

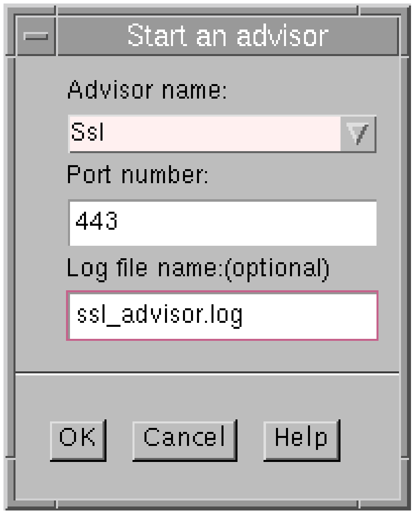
Configuration settings

Proportion given to active connections:	0	100	50
Proportion given to new connections:	0	100	50
Proportion given specific to the port:	0	100	0
Proportion given to system metrics:	0	100	0
Sum of proportions (must be 100):			100
Logging level:	0	5	1
Maximum log size (bytes):	unlimited		
Update interval (seconds):	2		
Weights refresh cycle:	2		
Reach Logging level:	0	5	1
Reach Maximum log size (bytes):	unlimited		
Reach Update interval (seconds):	7		
Sensitivity level:	0	100	5
Smoothing index:	1	100	1

```
File 'backup.c                                   he configuration.
The manager
```

Figure 87. Starting the Advisor

Select the Advisor you want to add from the list box in the following panel. We wanted to select the ssl Advisor, so we chose **Ssl** and the port was automatically set to `443`, as shown in the following figure:



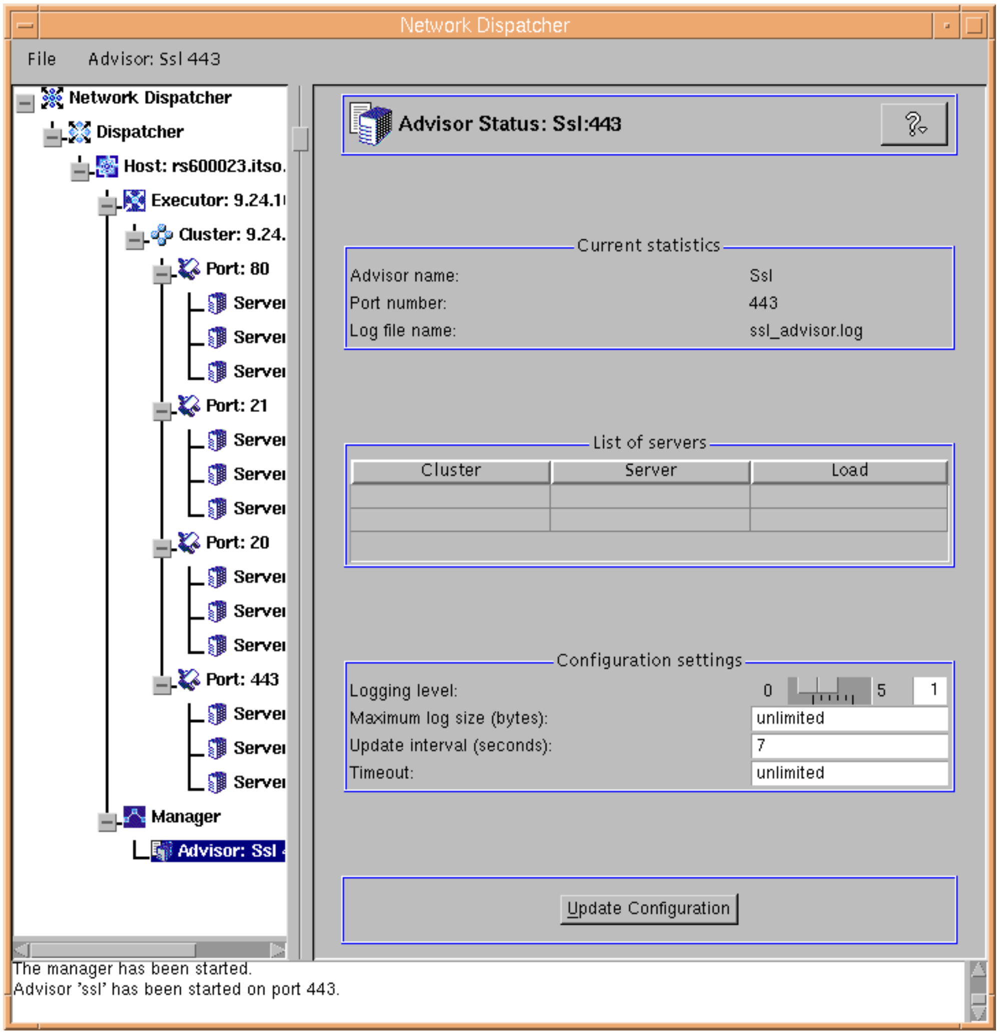
Advisor name: Ssl

Port number: 443

Log file name:(optional) ssl_advisor.log

Figure 88. Select Advisor to Start

If you click **OK**, you will see that the Advisor has been added to the configuration, as shown in the following panel:



Current statistics

Advisor name:	Ssl
Port number:	443
Log file name:	ssl_advisor.log

List of servers

Cluster	Server	Load

Configuration settings

Logging level:	0 5 1
Maximum log size (bytes):	unlimited
Update interval (seconds):	7
Timeout:	unlimited

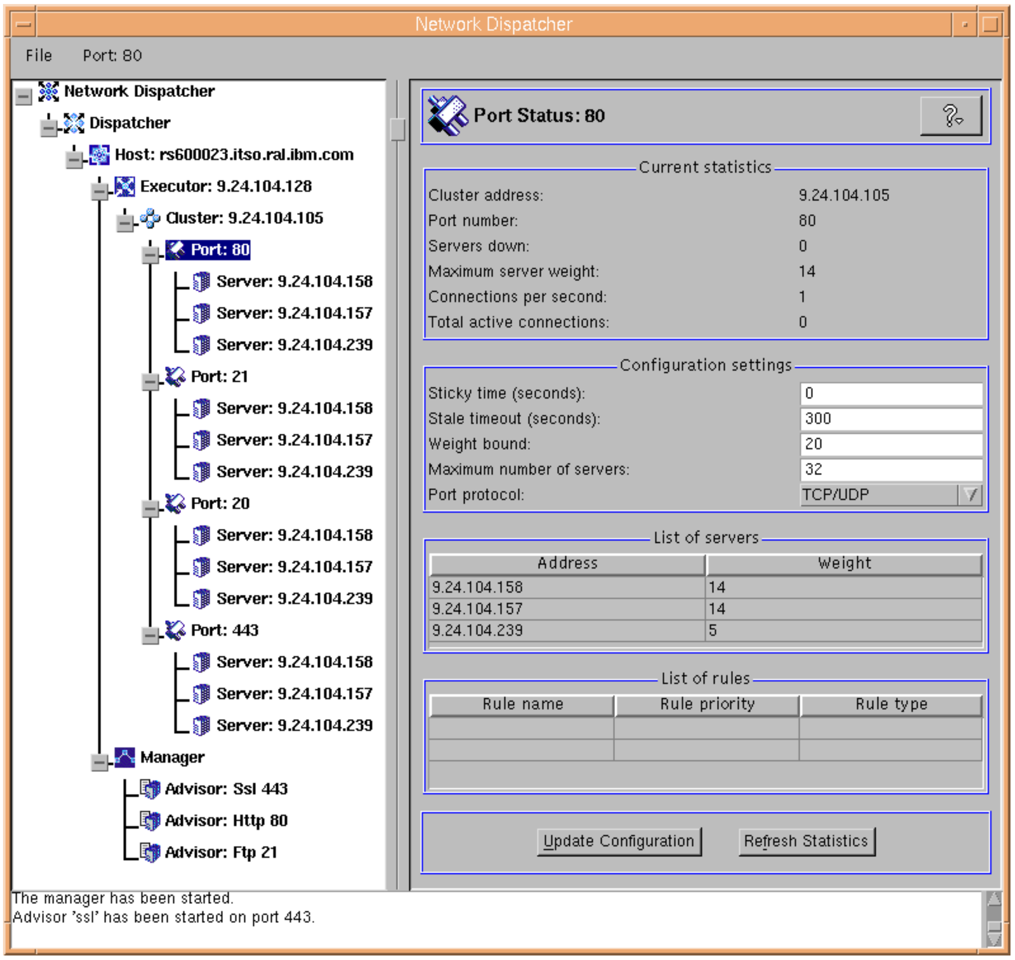
```
The manager has been started.
Advisor 'ssl' has been started on port 443.
```

Figure 89. Advisor Added

We also added the HTTP and FTP Advisors, but for these we used the command line, as follows:

```
ndcontrol advisor start http 80
ndcontrol advisor start ftp 21
```

From the GUI, we selected **Port: 80**, which was one of the ports that we just added an Advisor on, and we could see the updated Port Status, as shown in the following figure:



```
Network Dispatcher
  Dispatcher
    Host: rs600023.itso.ral.ibm.com
      Executor: 9.24.104.128
        Cluster: 9.24.104.105
          Port: 80
            Server: 9.24.104.158
            Server: 9.24.104.157
            Server: 9.24.104.239
          Port: 21
            Server: 9.24.104.158
            Server: 9.24.104.157
            Server: 9.24.104.239
          Port: 20
            Server: 9.24.104.158
            Server: 9.24.104.157
            Server: 9.24.104.239
          Port: 443
            Server: 9.24.104.158
            Server: 9.24.104.157
            Server: 9.24.104.239
      Manager
        Advisor: Ssl 443
        Advisor: Http 80
        Advisor: Ftp 21
```

Port Status: 80

Current statistics

Cluster address:	9.24.104.105
Port number:	80
Servers down:	0
Maximum server weight:	14
Connections per second:	1
Total active connections:	0

Configuration settings

Sticky time (seconds):	0
Stale timeout (seconds):	300
Weight bound:	20
Maximum number of servers:	32
Port protocol:	TCP/UDP

List of servers

Address	Weight
9.24.104.158	14
9.24.104.157	14
9.24.104.239	5

List of rules

Rule name	Rule priority	Rule type

```
The manager has been started.
Advisor 'ssl' has been started on port 443.
```

Figure 90. Other Advisors Added

4.1.10 Customization of the Manager for the Advisors

The load balancing action performed by the Dispatcher depends on several settings, which we are going to describe in the following sections.

4.1.10.1 Proportions of Importance Settings

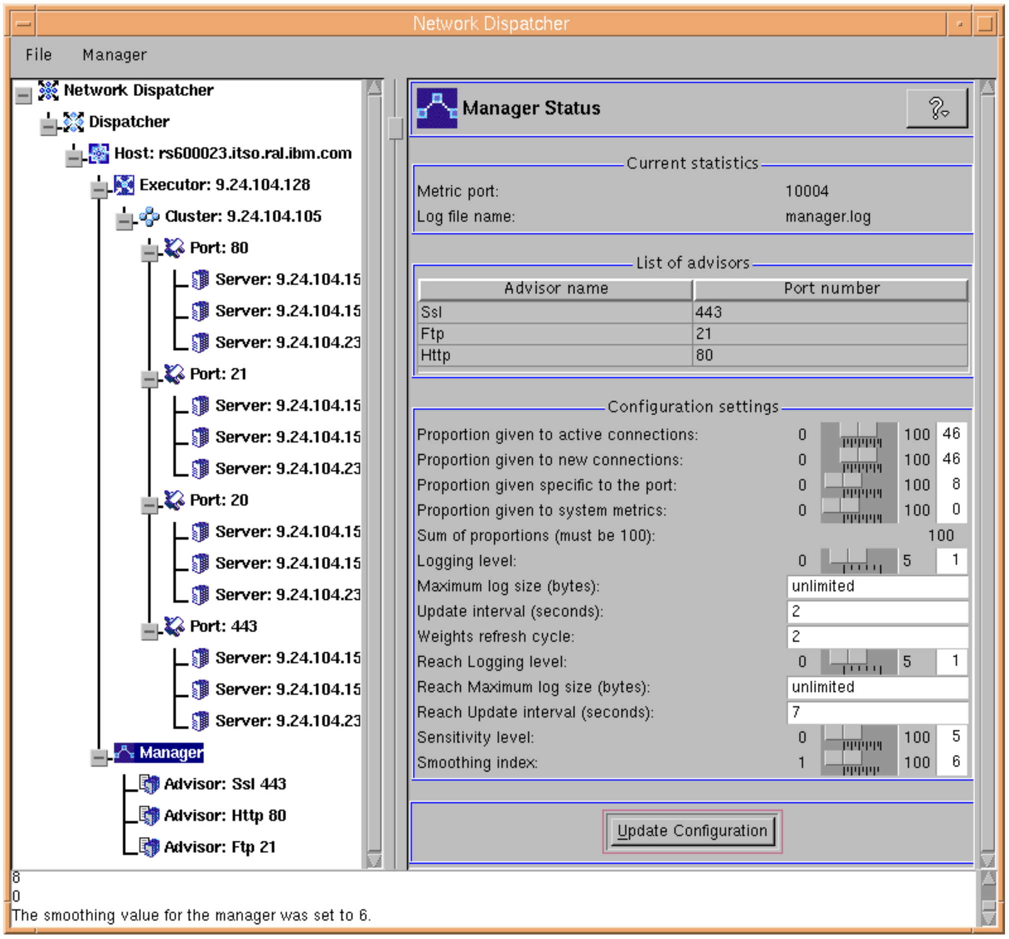
Now we show you how to set the proportions of importance for each of the policies (see 2.5.2.1, “Guidelines on Proportions of Importance Settings” on page 36). In the configuration window shown in Figure 90 on page 130, click the **Manager** item in the tree panel and in the right pane the Manager Status information will appear. As we said in 2.5.2.1, “Guidelines on Proportions of Importance Settings” on page 36, the default values are `50 50 0 0`: we tried the mix `46 46 8 0` by entering the new values directly into each of the respective fields, as shown in Figure 91 on page 131, so that:

- Both the number of active connections and new connections will contribute 46% in the weighting process.
- The Advisors will contribute 8%.
- Input from system monitoring tools, such as ISS, will not be considered in the weight decision. This is reasonable, because in this test we had not yet activated any monitoring tool.

4.1.10.2 Smoothing Index

Another important parameter to be considered when using the Advisors (and/or system monitoring tools such as ISS) is the *smoothing index*, which defines how smooth the change of the servers' weights will be. The Manager calculates and then updates the servers' weights dynamically. The weight values could change very rapidly, and in some circumstances this might result in an *oscillant* effect in the way the requests are load balanced. The distribution of the requests could be made in a nonoptimized way, and needs to be smoothed. Based on the smoothing index value, the Manager will change the servers' weights more or less quickly. The default value for smoothing index is 1.5, which could cause the Manager to change the weights rather dynamically. Values around 5 are preferred for having the Manager modify the weights slowly.

We set the Smoothing index value to `6`. After you modify all the values you want in the Manager Status window, click **Update** to have the configuration updated, as shown in the following window:



Current statistics

Metric port:	10004
Log file name:	manager.log

List of advisors

Advisor name	Port number
Ssl	443
Ftp	21
Http	80

Configuration settings

Proportion given to active connections:	0	100	46
Proportion given to new connections:	0	100	46
Proportion given specific to the port:	0	100	8
Proportion given to system metrics:	0	100	0
Sum of proportions (must be 100):			100
Logging level:	0	5	1
Maximum log size (bytes):			unlimited
Update interval (seconds):			2
Weights refresh cycle:			2
Reach Logging level:	0	5	1
Reach Maximum log size (bytes):			unlimited
Reach Update interval (seconds):			7
Sensitivity level:	0	100	5
Smoothing index:	1	100	6

```
8
0
The smoothing value for the manager was set to 6.
```

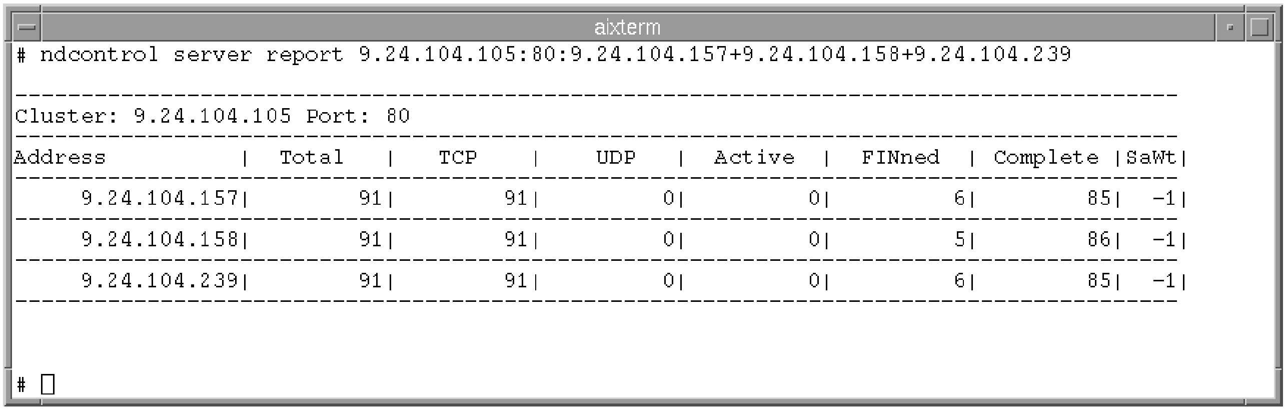
Figure 91. Setting the Manager Proportions and Smoothing Level

Note that as a result of starting the Advisors, we noticed an increase in the number of TCP connections reported in the `ndcontrol` server report.

After starting the HTTP Advisor on port 80 we again entered the command:

```
ndcontrol server report 9.24.104.105:80:9.24.104.157+9.24.104.158+9.24.104.239
```

Figure 83 on page 122 shows the command output before the Advisor was started. No new client Web requests were made in the interim, just TCP connections made by the Advisor:

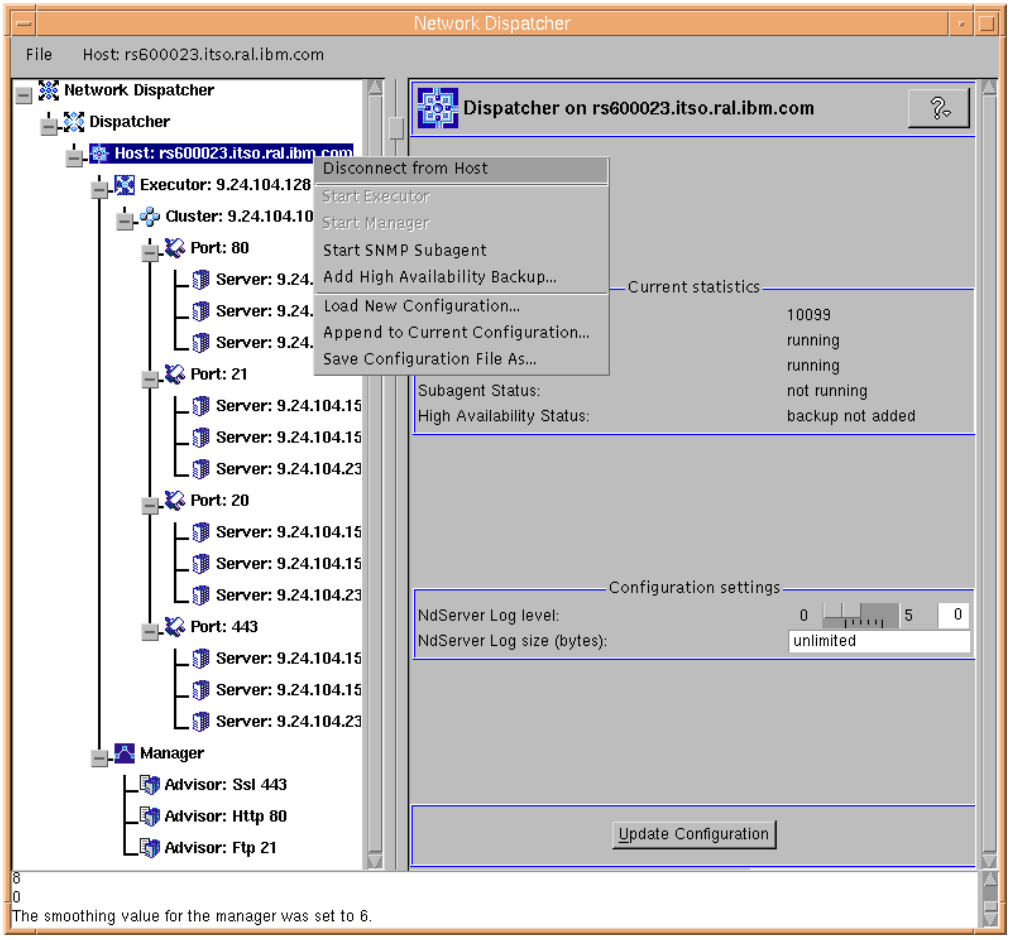


```
# ndcontrol server report 9.24.104.105:80:9.24.104.157+9.24.104.158+9.24.104.239

--------------------------------------------------------------------------------
Cluster: 9.24.104.105 Port: 80
--------------------------------------------------------------------------------
Address          |  Total   |   TCP    |    UDP   |  Active  |  FINned  | Complete |SaWt|
--------------------------------------------------------------------------------
     9.24.104.157|        91|        91|         0|         0|         6|        85|  -1|
--------------------------------------------------------------------------------
     9.24.104.158|        91|        91|         0|         0|         5|        86|  -1|
--------------------------------------------------------------------------------
     9.24.104.239|        91|        91|         0|         0|         6|        85|  -1|
--------------------------------------------------------------------------------


#
```

Figure 92. The Many TCP Connections Made by the Advisor Are Shown

4.1.11 Saving the Configuration

Once you finish configuring the Dispatcher, it is recommended you save the configuration of the Dispatcher to a file. To do this, right click **Host** and select **Save Configuration File As...**. This is shown in the following figure:



```
8
0
The smoothing value for the manager was set to 6.
```

Figure 93. Save Configuration File As... Option On the Host Pop-Up

In the resulting pop-up window, you will be prompted to enter the name of the configuration file where you would like to save this information.

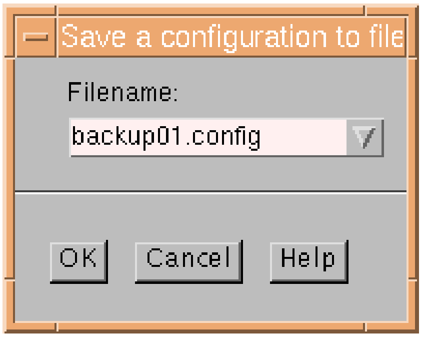


Figure 94. Enter Configuration File Name

By default this file is placed in the directory *installbase*/configurations/, where *installbase* varies by component (Dispatcher or CBR) and operating system. See Table 1 on page 69 for a list of the *installbase* locations.

In our testing, we noticed that the Dispatcher automatically makes a backup of the execution environment in the file called backup.config once an hour when the executor is running.

The configuration file is saved in ASCII format and contains the list of commands that would be necessary to reconfigure your setup:

```
ndcontrol executor start
ndcontrol executor set nfa 9.24.104.128

ndcontrol cluster add 9.24.104.105

ndcontrol port add 9.24.104.105:80

ndcontrol server add 9.24.104.105:80:9.24.104.158

ndcontrol server add 9.24.104.105:80:9.24.104.157

ndcontrol server add 9.24.104.105:80:9.24.104.239
ndcontrol server set 9.24.104.105:80:9.24.104.239 weight 9

ndcontrol port add 9.24.104.105:21
ndcontrol port set 9.24.104.105:21 staletimeout 900

ndcontrol server add 9.24.104.105:21:9.24.104.158

ndcontrol server add 9.24.104.105:21:9.24.104.157

ndcontrol server add 9.24.104.105:21:9.24.104.239
ndcontrol server set 9.24.104.105:21:9.24.104.239 weight 0

ndcontrol port add 9.24.104.105:20

ndcontrol server add 9.24.104.105:20:9.24.104.158
ndcontrol server set 9.24.104.105:20:9.24.104.158 weight 9

ndcontrol server add 9.24.104.105:20:9.24.104.157
ndcontrol server set 9.24.104.105:20:9.24.104.157 weight 9

ndcontrol server add 9.24.104.105:20:9.24.104.239
ndcontrol server set 9.24.104.105:20:9.24.104.239 weight 9

ndcontrol port add 9.24.104.105:443

ndcontrol server add 9.24.104.105:443:9.24.104.158

ndcontrol server add 9.24.104.105:443:9.24.104.157

ndcontrol server add 9.24.104.105:443:9.24.104.239
ndcontrol cluster configure 9.24.104.105 tr0 255.255.255.0

ndcontrol manager start manager.log 10004
ndcontrol manager proportions 46 46 8 0
ndcontrol manager smoothing 6.0

ndcontrol advisor start Http 80 Http_80.log

ndcontrol advisor start Ssl 443 Ssl_443.log

ndcontrol advisor start Ftp 21 Ftp_21.log
```

Figure 95. Configuration File backup01.config

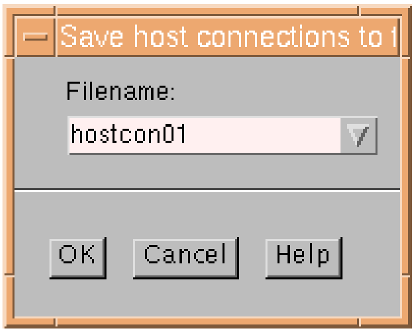
In our testing, we noticed that the line:

```
ndcontrol cluster configure 9.24.104.105 tr0 255.255.255.0
```

was included in the above configuration file in place of the UNIX command `ifconfig` or the Windows NT command `ndconfig` that we had used. The command above creates the cluster alias on the network interface and is indeed equivalent to the `ifconfig` and `ndconfig` commands.

4.1.12 Saving the Host Connections

Host Connections can be saved by selecting **File** on the menu bar of the configuration window (see Figure 91 on page 131) and then selecting **Save Host Connections As...**. In the panel that appears, enter the configuration file name that you would like the Host Connection information saved into.



Filename: hostcon01

Figure 96. Enter the Name of the Host Connection Save File

This file is saved in the directory *installbase*/configurations, where *installbase* is the install directory for the admin component that varies by operating system. See Table 1 on page 69 for a list of the *installbase* locations.

As a result of saving this information, on subsequent invocations of the GUI, this host connection will be listed as one of the choices in the Connect to Host selection screen (see Figure 46 on page 87).

4.1.13 Stopping the Executor and the GUI

To stop the Executor from the GUI right-click **Executor** and select **Stop Executor** (see Figure 48 on page 89). The command line version of this is:

```
ndcontrol executor stop
```

On Windows NT, the Executor cannot be stopped. As an administrator, you can select **IBM Network Dispatcher** from the Services folder of the Control Panel, and then click **Stop**. However, this operation only stops the server, not the Executor.

To exit the GUI, select **File->Exit.**

On Closing the GUI

Remember that if you close the GUI without stopping the Executor, this continues to run. In fact, if you restart the GUI with the command `ndadmin`, you will still be shown the current running configuration.

When you restart the GUI after connecting to a host, you can choose whether to load a saved configuration, if any, or create a new one. To load a configuration file, from the **Host** pop-up menu select **Load New Configuration** (see Figure 48 on page 89) and select the file name that contains your saved configuration from the list box.

4.2 Load Balancing Scenario Using the Dispatcher and ISS

By using the monitoring capabilities provided by ISS on the TCP server machines, you can provide the Dispatcher with load server information. In this case the ISS cooperates with the Dispatcher, but ISS does not make any load balancing decision. The ISS monitor collects specific server information such as CPU usage, memory usage and disk activity from the ISS agents running on the individual servers, and forwards it to the Dispatcher. The Dispatcher uses this load information, along with other sources of information, to determine which is the least loaded server of the cluster and then performs load balancing.

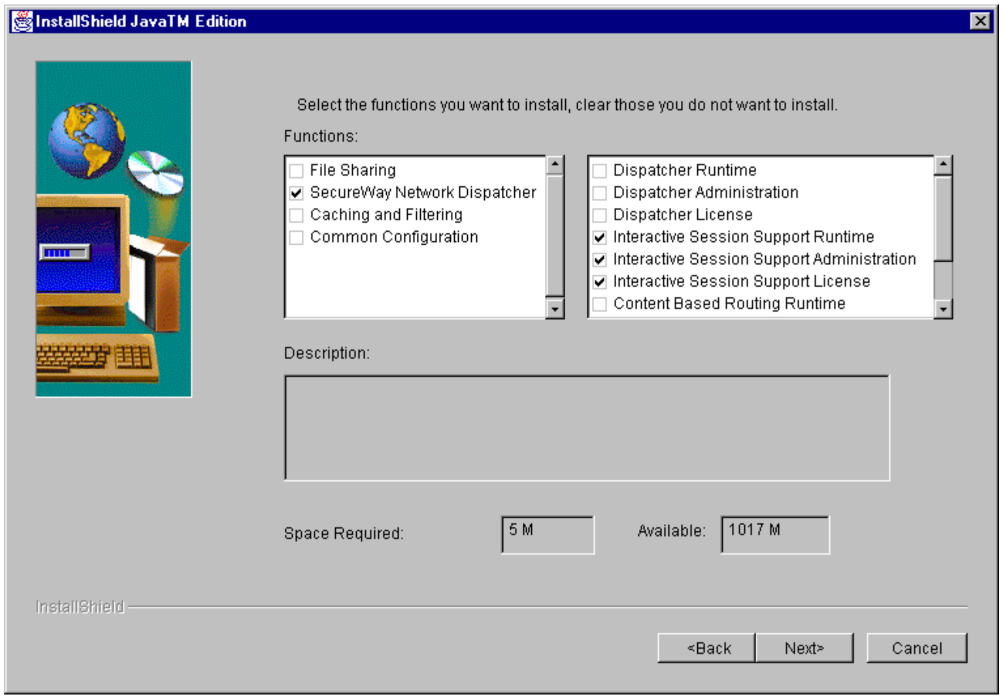
4.2.1 Installation of ISS

As already mentioned, the ND V2.1 component of IBM WebSphere Performance Pack Version 2 has three subcomponents: Dispatcher, ISS and CBR. In this section we discuss the installation of the ISS component.

ISS is supported on three operating systems: IBM AIX 4.2.1 or later, Microsoft Windows NT 4.0 and Sun Solaris 2.6 or later. Refer to the appropriate platform section of 3.1, “Installation of ND” on page 55 for details on how to use the Java InstallShield on your respective platform. When you reach the point where you choose which ND component to install (see Figure 17 on page 57 and Figure 24 on page 64), select these three components to install ISS on your machine:

- Interactive Session Support Runtime
- Interactive Session Support Administration
- Interactive Session Support License

This is shown in the following figure:


InstallShield JavaTM Edition

Select the functions you want to install, clear those you do not want to install.

Functions:

- [ ] File Sharing
- [x] SecureWay Network Dispatcher
- [ ] Caching and Filtering
- [ ] Common Configuration

- [ ] Dispatcher Runtime
- [ ] Dispatcher Administration
- [ ] Dispatcher License
- [x] Interactive Session Support Runtime
- [x] Interactive Session Support Administration
- [x] Interactive Session Support License
- [ ] Content Based Routing Runtime

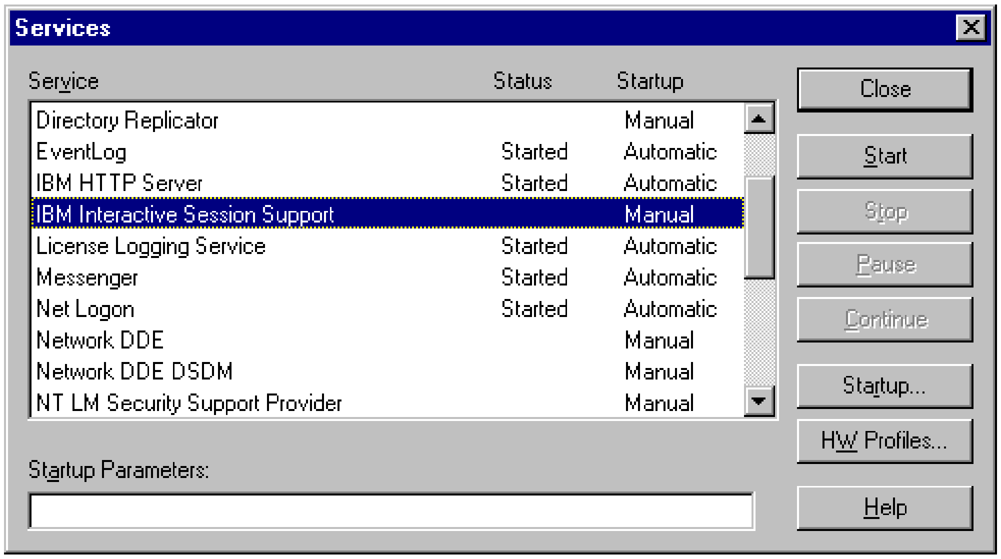
Description:

Space Required: 5 M   Available: 1017 M

Figure 97. Select the Three ISS Components to be Installed

ISS has the special function of system monitoring. If you want ISS to be part of your environment, you should select one machine to run as an ISS monitor, while on the TCP servers in the cluster an ISS agent process should run, gathering information about the status of the systems and feeding it back to the ISS monitor. The three ISS components listed above should be installed on the ISS monitor machine as well as on the TCP server machines.

On Windows NT, at the end of the ISS component installation, you will be instructed to reboot. After rebooting, notice that the IBM Interactive Session Support service is not automatically started, as we can see by viewing the Services folder of the Control Panel:



Service	Status	Startup
Directory Replicator		Manual
EventLog	Started	Automatic
IBM HTTP Server	Started	Automatic
IBM Interactive Session Support		Manual
License Logging Service	Started	Automatic
Messenger	Started	Automatic
Net Logon	Started	Automatic
Network DDE		Manual
Network DDE DSDM		Manual
NT LM Security Support Provider		Manual

Figure 98. The ISS Server Is Not Automatically Started after Rebooting

You should click **Start** to activate the ISS daemon. Refer to 4.2.4.2, “Starting the ISS Daemon” on page 142 for further information on this.

4.2.2 Scenario Configuration

In this section we show you how to create a scenario where both the Dispatcher and ISS components of ND V2.1 are used. We worked on the same environment shown in Table 4 on page 81. However, we enhanced that scenario by adding the ISS function. In the environment shown in Figure 41 on page 82, we decided to have the Dispatcher machine run as the ISS monitor, too. ISS is used here only in its capacity to gather server load information from the TCP servers, where the ISS agent process runs. This information is then passed back to the ISS monitor process, running in this case on the same Dispatcher machine. The monitor interacts with the Dispatcher and provides it with information about the loads on the servers. The Dispatcher uses this information, along with other sources to perform the load balancing.

The steps required to configure ISS in this information-gathering capacity start out essentially the same as if ISS were to perform the load balancing. They key point that makes the difference between the information gathering function and the load balancing function of ISS is in the choice of Observer type, as explained below.

4.2.3 Network Environment

The following figure offers a graphical representation of the environment we used:

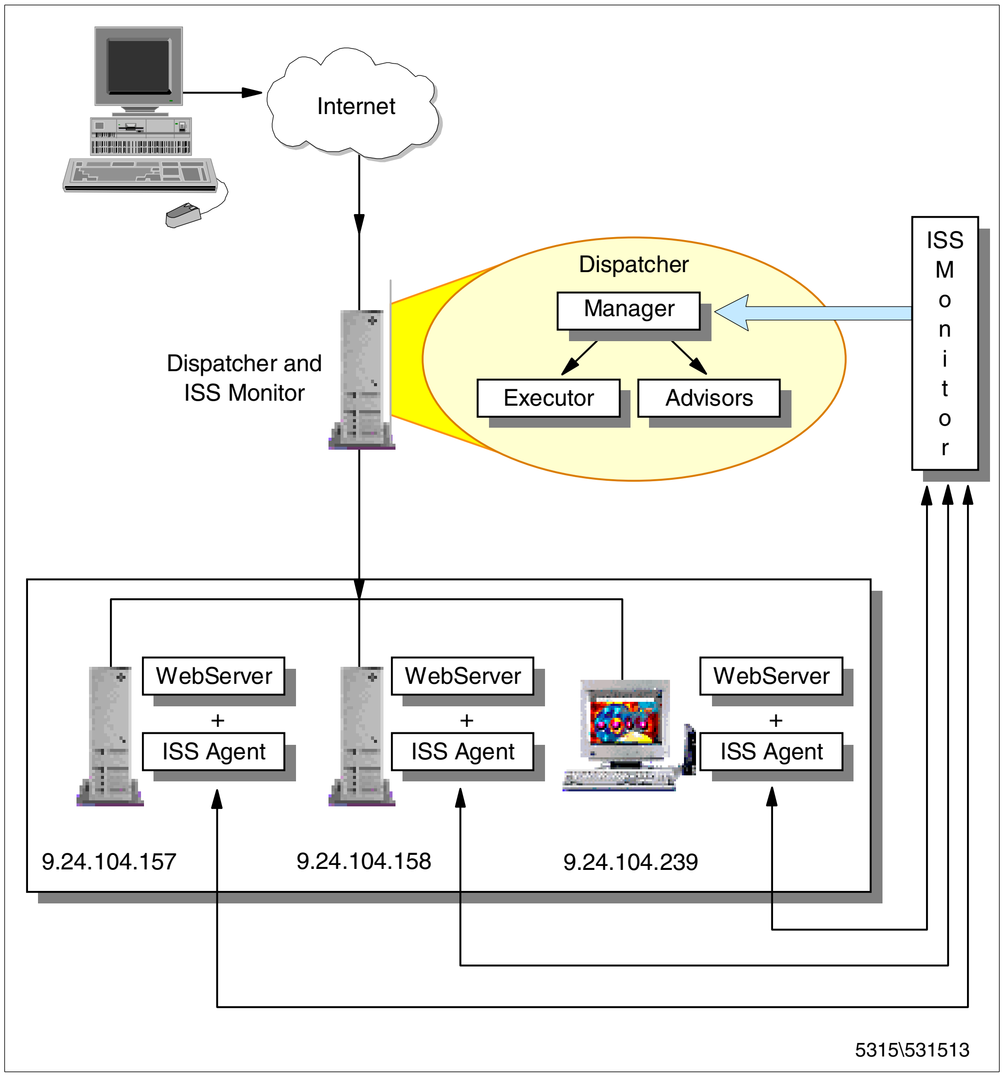


Figure 99. Graphical Representation of the Dispatcher and ISS Scenario Environment

Notice that, as soon as you make ISS part of the load balancing process, the Manager configuration must reflect the presence of ISS. To do this, change the proportions of importance (see 4.1.10.1, “Proportions of Importance Settings” on page 130) to take into account the input coming from ISS. In this case, we set the proportions of importance for the Manager to `48 48 2 2` (see also 2.5.2.1, “Guidelines on Proportions of Importance Settings” on page 36).

The example we are going to show you is also very interesting because it demonstrates how easy it is to implement ISS high availability.

4.2.4 ISS Configuration

In 4.1, “Load Balancing Basic Scenario Using the Dispatcher” on page 81, we described the network environment we had set up using the Dispatcher, but without using ISS. In this section we show how to add ISS to the already existing environment.

4.2.4.1 ISS Configuration Methods

In the previous version of WebSphere Performance Pack, we would have had to modify the ISS configuration file by manually editing it. New to this version is the ability to modify the ISS configuration file by using the ND configuration GUI.

In this section, we show how to make some changes to the Basic.Sample configuration file shipped with the product and then how to complete the customization with the GUI. Each customization that we performed with the GUI could also have been performed on the command line with `isscontrol` commands. We will supply the syntax of the corresponding `isscontrol` command that could be used for several of the tasks that we use the GUI for. For further details on the `isscontrol` command, please refer to *SecureWay Network Dispatcher User's Guide V2.1 for AIX, Solaris and Windows NT*, GC31-8496.

The Basic.Sample configuration file is located in the directory *installbase*/samples, where *installbase* is for the ISS component and varies by operating system. See Table 1 on page 69, for a list of the *installbase* locations.

The original Basic.Sample configuration file for the AIX platform, as it is shipped with the product, is shown here:

```
# This is the basic configuration sample. You must create
# this minimal configuration file before you use the ISS
# GUI, if you do not have any pre-existing configuration
# files. Replace yourCellName and yourHostName with names
# appropriate for your configuration.

        cell yourCellName local
        node yourHostName 1
```

Figure 100. ISS Shipped Configuration File Basic.Sample

See "Script Modifications on Windows NT" on page 99 for details on how script files need to be modified on the Windows NT platform.

The following figure shows our customized version of Basic.Sample. We renamed the modified version of Basic.Sample iss.cfg, and placed it in /etc on AIX (this is also where it should be located on Solaris) and in \Program Files\nd on Windows NT.

Note: It is important that when you start using ISS, all the machines in your cell have the same ISS configuration file.

As the comments in the Basic.Sample file indicated, you must create this minimal configuration file before you use the ISS GUI, if you do not have any preexisting configuration files. We replaced yourCellName and yourHostName with Cary and rs600023 respectively:

```
# This is our initial customized copy of the basic configuration
# sample. We created this minimal configuration file before
# using the ISS GUI. We replaced yourCellName and yourHostName
# with names appropriate for our configuration.

        cell Cary local
        node rs600023 1
```

Figure 101. Customized Basic.Sample ISS Configuration File

In order to perform the configuration steps listed below, you must be the root user on AIX and Solaris or, if the Dispatcher is installed on Windows NT, a system administrator.

4.2.4.2 Starting the ISS Daemon

ISS operates as either monitor or agent depending on the contents of the configuration file or parameters entered.

To start the ISS daemon, if on AIX and Solaris, from a command line enter the command:

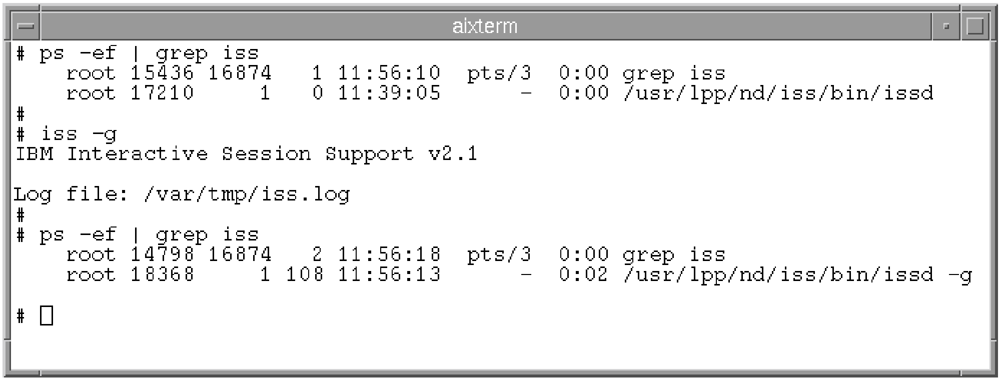
```
iss -g
```

The `-g` flag instructs the ISS daemon to monitor for communication from the ND GUI on port 12099. If your iss.cfg file is not located in /etc or called iss.cfg, then you will need to use the `-c` flag followed by the full path to the configuration file, to let ISS know where the file is. The `-l` flag can be used to specify a log file.

Note that on Windows NT, ISS is a Windows NT service and is not automatically started. To start ISS as administrator, open the Services window of the Windows NT Control Panel (see Figure 98 on page 139), select **IBM Interactive Session Support** and then click **Start**. In the Startup Parameters text field, you can specify the arguments for the `iss` command. Notice, however, that if you want to code a back lash (\), you have to type a double backslash (\\). For example, if you want to specify the configuration file D:\issconf\isspistoia.cfg and the log file D:\issconf\isspistoia.log, you have to type:

```
-c D:\\issconf\\isspistoia.cfg -l D:\\issconf\\isspistoia.log
```

On AIX, if you enter the `ps` command after launching the ISS daemon, you can see that the ISS daemon process is running:



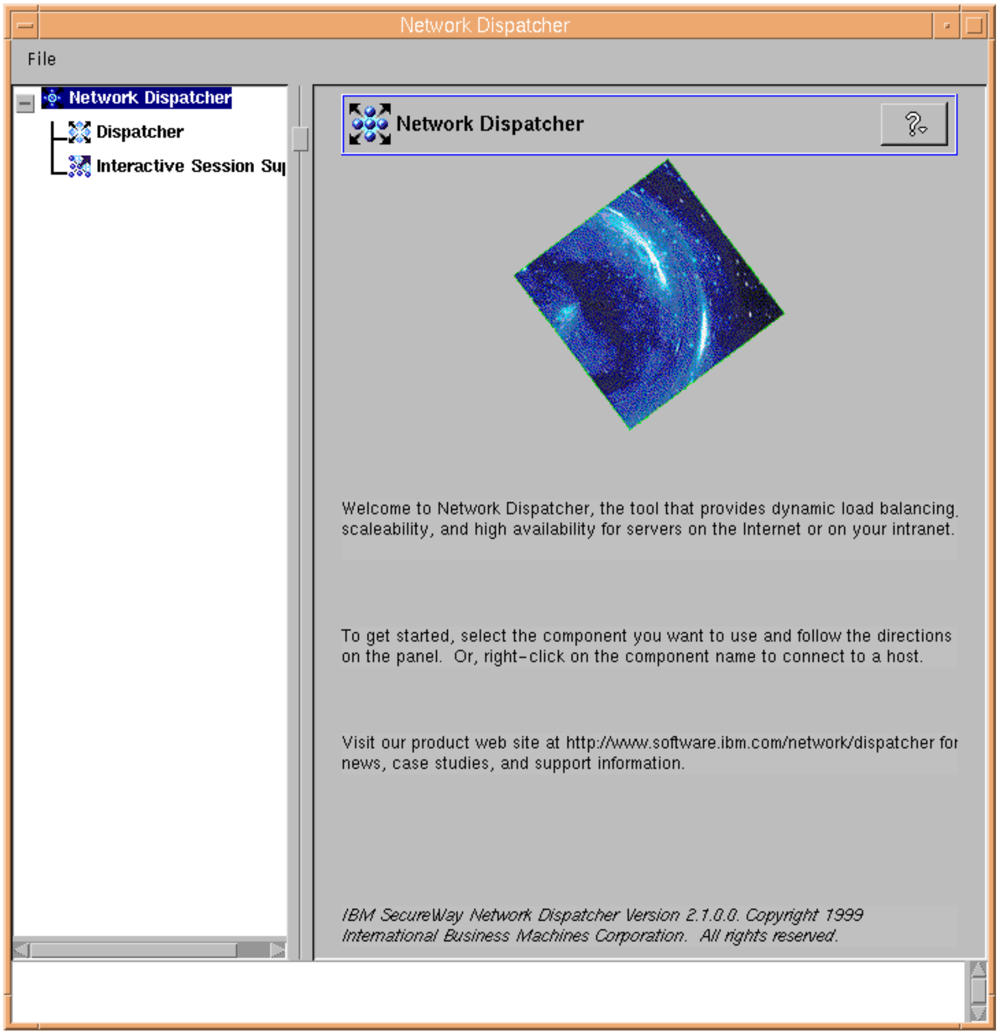
```
# ps -ef | grep iss
    root 15436 16874   1 11:56:10  pts/3  0:00 grep iss
    root 17210     1   0 11:39:05      -  0:00 /usr/lpp/nd/iss/bin/issd
#
# iss -g
IBM Interactive Session Support v2.1

Log file: /var/tmp/iss.log
#
# ps -ef | grep iss
    root 14798 16874   2 11:56:18  pts/3  0:00 grep iss
    root 18368     1 108 11:56:13      -  0:02 /usr/lpp/nd/iss/bin/issd -g

#
```

Figure 102. Confirming that the ISS Daemon Process is Running

4.2.4.3 Starting the Configuration GUI

Now you can start the GUI. To do so, enter the command `ndadmin`.

If your platform is Windows NT, you can also click **Start->Programs->Network Dispatcher**. The Network Dispatcher item in the Programs menu was created during the installation and is another method of launching the GUI.



Welcome to Network Dispatcher, the tool that provides dynamic load balancing, scaleability, and high availability for servers on the Internet or on your intranet.

To get started, select the component you want to use and follow the directions on the panel. Or, right-click on the component name to connect to a host.

Visit our product web site at http://www.software.ibm.com/network/dispatcher for news, case studies, and support information.

IBM SecureWay Network Dispatcher Version 2.1.0.0. Copyright 1999 International Business Machines Corporation. All rights reserved.

Figure 103. ND Configuration GUI Window

The left pane displays a tree structure, with Network Dispatcher at the top level, and Dispatcher, Interactive Session Support and Content Based Routing as components if they are installed. You can select elements in the tree structure by clicking mouse button one (typically the left button), and you can display pop-up menus by clicking mouse button two (typically the right button). The pop-up menus for the tree elements are also accessible from the menu bar located at the top of the window. Each item in the tree is marked with a plus sign (**+**) or a minus sign (**-**). Click the plus sign (**+**) to expand the items within it and the minus sign (**-**) to contract the items.

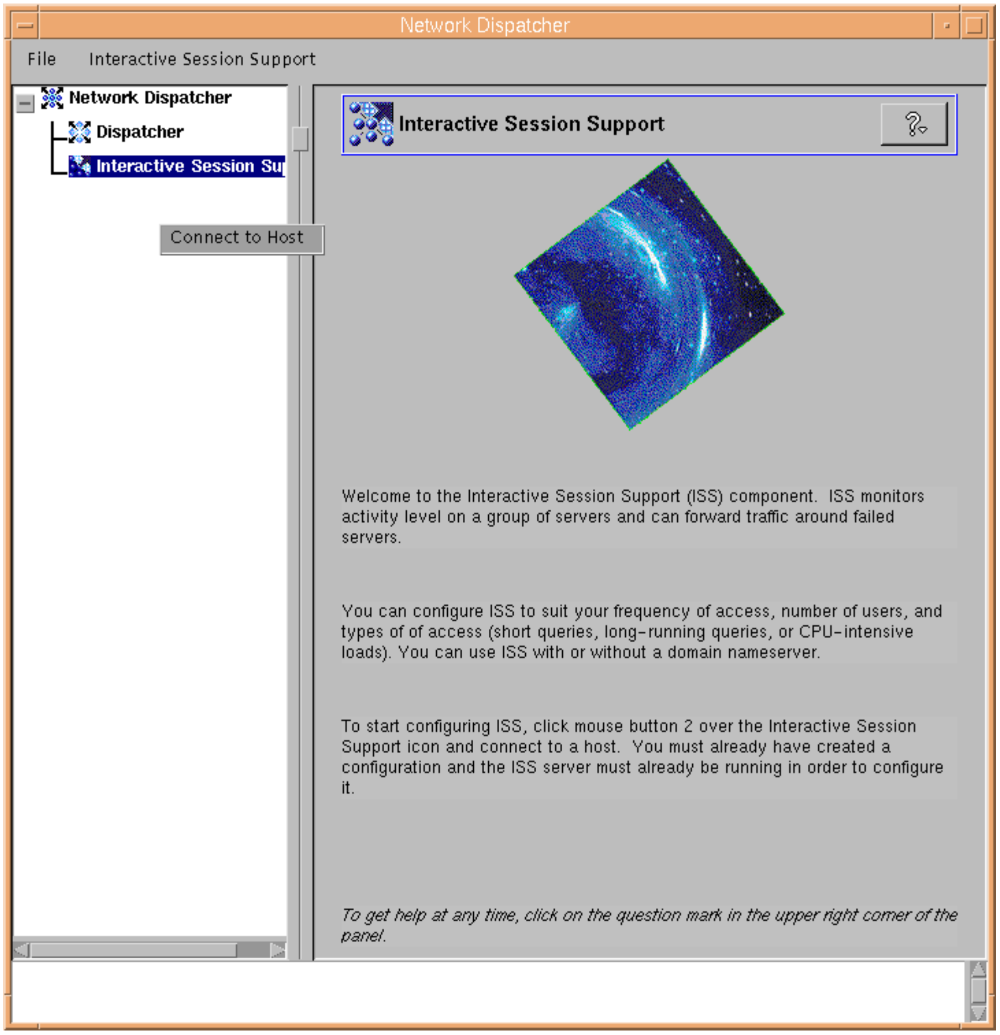
On AIX, if you enter the `ps` command after launching the command `ndadmin`, you can see that ndadmin is implemented as a Java class, running through the JVM `java` executable file (see Figure 44 on page 86)

4.2.4.4 Connecting to a Host

The next step in configuring ISS is to make a connection to a host where the ISS daemon is running. This is an effect of the new Remote Authenticated Administration feature, which enables ND component configuration to be done on a remote client machine. See 3.3.1, “Remote Authenticated Administration” on page 75 for further detail on remote configuration.

In this case, we are performing this configuration on the ISS manager machine itself. However, the connection still must take place.

To make the connection, right-click **Interactive Session Support** in the tree structure. In the pop-up window that appears, select **Connect to Host...**, as shown in the following figure:



Welcome to the Interactive Session Support (ISS) component. ISS monitors activity level on a group of servers and can forward traffic around failed servers.

You can configure ISS to suit your frequency of access, number of users, and types of of access (short queries, long–running queries, or CPU–intensive loads). You can use ISS with or without a domain nameserver.

To start configuring ISS, click mouse button 2 over the Interactive Session Support icon and connect to a host. You must already have created a configuration and the ISS server must already be running in order to configure it.

To get help at any time, click on the question mark in the upper right corner of the panel.

Figure 104. ISS Host Connection Menu

We selected the host name of the machine where we are doing the initial ISS configuration, which was the same machine where we would be running the ISS monitor and the Dispatcher (see Figure 99 on page 140) in the following window:

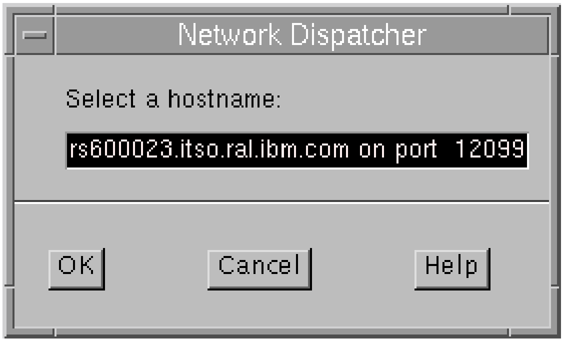
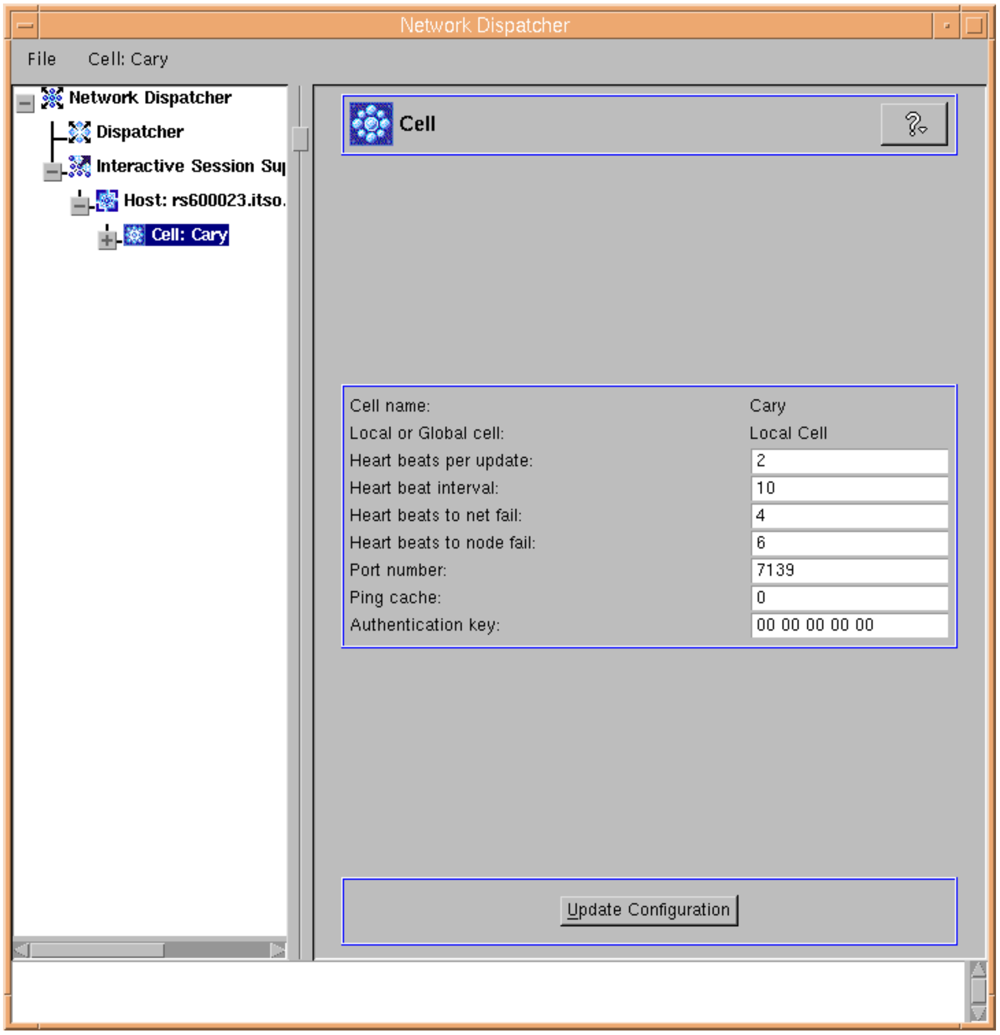


Figure 105. Selecting a Host to Perform the ISS Configuration On

After clicking **OK**, the GUI was refreshed to show the ISS host connection and the single Cary cell that we had defined in our iss.cfg file. Selecting **Cell: Cary** shows us the default values:

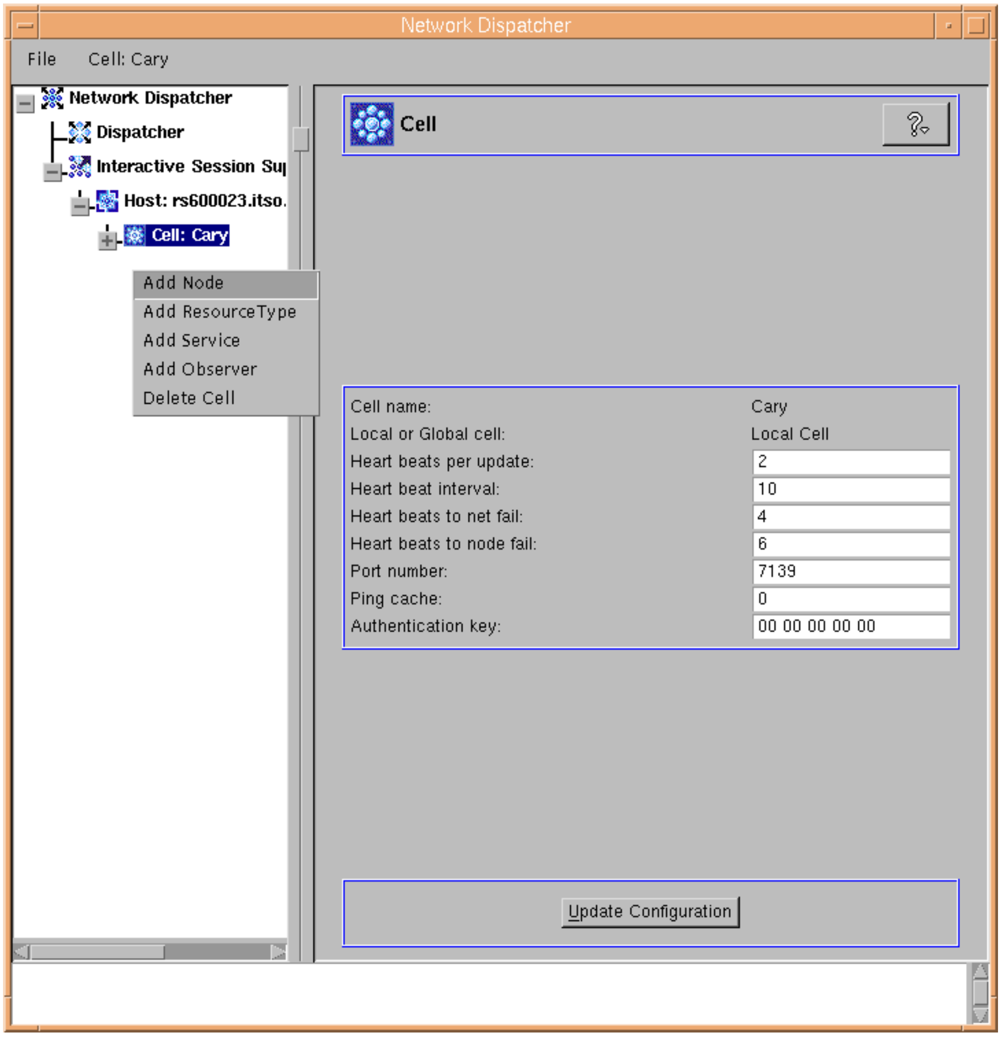


Cell name:	Cary
Local or Global cell:	Local Cell
Heart beats per update:	2
Heart beat interval:	10
Heart beats to net fail:	4
Heart beats to node fail:	6
Port number:	7139
Ping cache:	0
Authentication key:	00 00 00 00 00

Figure 106. Default Cell Statistics

4.2.4.5 Adding Nodes

The next step is to add nodes to your configuration. To do this, right-click **Cell: Cary** and then select **Add Node** in the Cell menu, as shown in the following figure:



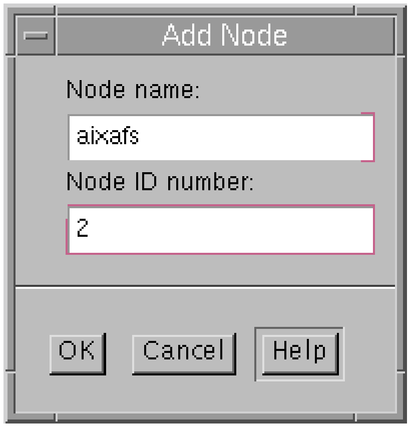
Cell name:	Cary
Local or Global cell:	Local Cell
Heart beats per update:	2
Heart beat interval:	10
Heart beats to net fail:	4
Heart beats to node fail:	6
Port number:	7139
Ping cache:	0
Authentication key:	00 00 00 00 00

Figure 107. ISS Cell Menu

You will be prompted for the node name and the order number for the node as seen in Figure 108.

We configured the Dispatcher machine rs600023 to be the primary ISS monitor, and the three Web servers aixafs, aixncf157 and wtr05193 to be ISS agents. While wtr05193 is explicitly set as incapable of becoming a monitor by using the `NotMonitor` keyword, aixafs and aixncf157 were backups to the monitor node. The `NotMonitor` keyword is not specified for aixafs and aixncf157 and the priority numbers are set as `1` for rs600023, `2` for aixafs and `3` for aixncf157.

This means that aixafs will be the first ISS machine to take over and become the ISS monitor, should the primary ISS monitor rs600023 fail. Should aixafs fail, then it would be aixncf157 that takes over. These simple steps are enough to implement ISS high availability.



Node name:

aixafs

Node ID number:

2

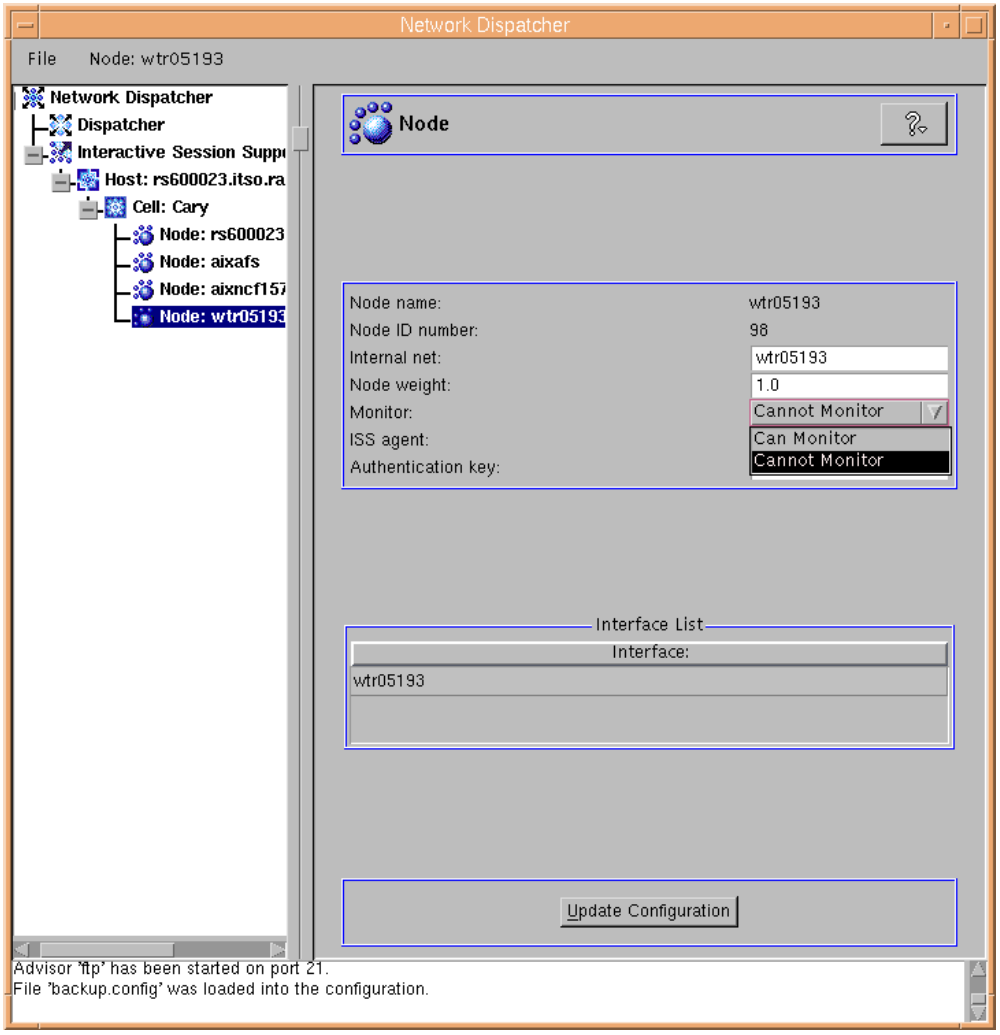
Figure 108. Supplying Node Information to the GUI

At this point, the GUI is updated to reflect the new node that you added.

From the command line, we then added the remaining nodes with these commands:

```
isscontrol add node aixncf157 3
isscontrol add node wtr05193 98
```

At this point, the configuration tree on the left side of the GUI was updated to show the two nodes that were added from the command line. For node wtr05193, we changed the ISS monitor attribute to `NotMonitor` by selecting **Cannot Monitor** from the monitor selection menu, accessible by clicking the drop-down menu indicator to the right of the monitor status value:



Node name: wtr05193
Node ID number: 98
Internal net: wtr05193
Node weight: 1.0
Monitor: Cannot Monitor
ISS agent:
Authentication key:

wtr05193

```
Advisor 'ftp' has been started on port 21.
File 'backup.config' was loaded into the configuration.
```

Figure 109. Changing the Monitor Attribute for Node wtr05193

The command-line version of this would be:

```
isscontrol set node wtr05193 CanMonitor false
```

When values are entered directly into the GUI window, as we did with the above Monitor pull-down menu, it is important to click the **Update Configuration** button at the bottom of the display. This writes the changes to the ISS configuration file, iss.cfg.

If the nodes you are adding have multiple network interfaces on them, then you can define these individually by right-clicking the **Node** entry; this would open the Node menu, which can be used to select **Add Interface to Node.** In our scenario, our nodes had only one network interface and so we did not need to do this step.

4.2.4.6 Defining Resource Types

The next step is to add resources to your configuration. To do this, right-click **Cell: Cary** and then select **Add ResourceType** in the Cell menu (see Figure 107 on page 147).

In this case, we would like to add a CPU ResourceType and so we type `CPU` in the Add ResourceType window, as shown in the following figure:

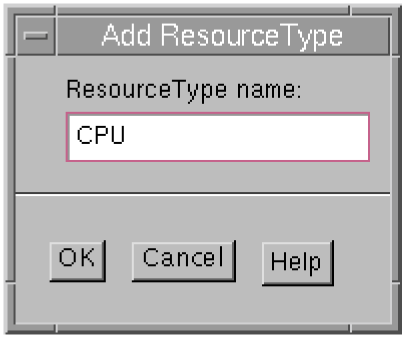
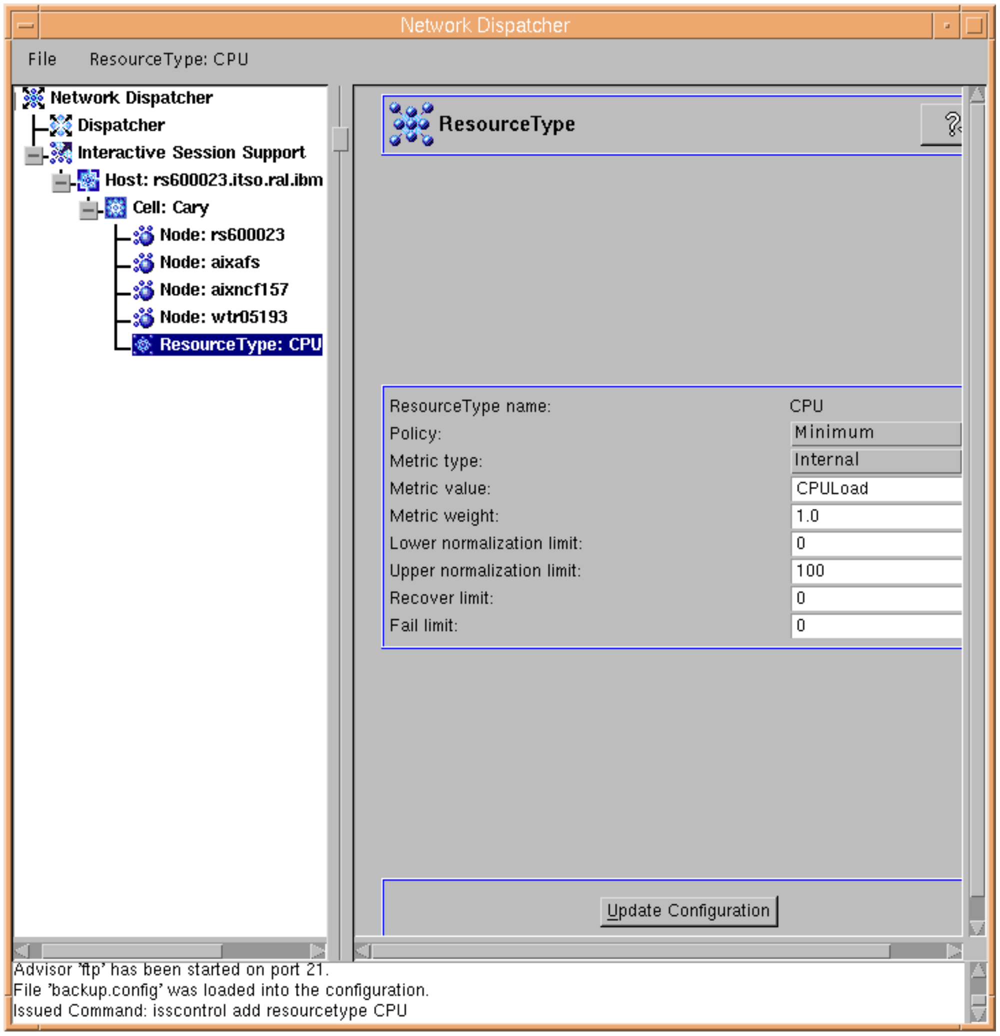


Figure 110. Adding a CPU Resource Type

ResourceType entries are given the following default values, which in this case are appropriate for the CPU ResourceType:



ResourceType name:	CPU
Policy:	Minimum
Metric type:	Internal
Metric value:	CPULoad
Metric weight:	1.0
Lower normalization limit:	0
Upper normalization limit:	100
Recover limit:	0
Fail limit:	0

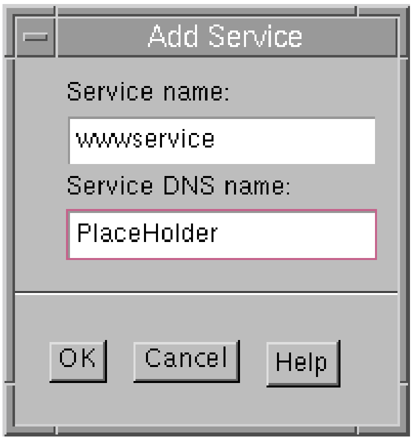
```
Advisor 'ftp' has been started on port 21.
File 'backup.config' was loaded into the configuration.
Issued Command: isscontrol add resourcetype CPU
```

Figure 111. The Default Values for the CPU Resource

We then modified the CPU ResourceType Recover limit and Fail limit by entering the new values `80` and `95`, respectively, directly into the GUI fields. Following this, we clicked on **Update Configuration** to update the ISS configuration file, iss.cfg.

4.2.4.7 Defining Services

The next step was to add services to our configuration. To do this, we right-clicked **Cell: Cary** and then selected **Add Service** in the Cell menu (see Figure 107 on page 147). In the Add Service window, we gave the service a name, `wwwservice`, and also supplied a service DNS name. The Service DNS name is not needed when you are using ISS to update the Dispatcher as in this case. However, you must fill in the blank with a placeholder name; for this reason, we typed in `PlaceHolder`, as shown in the following figure:

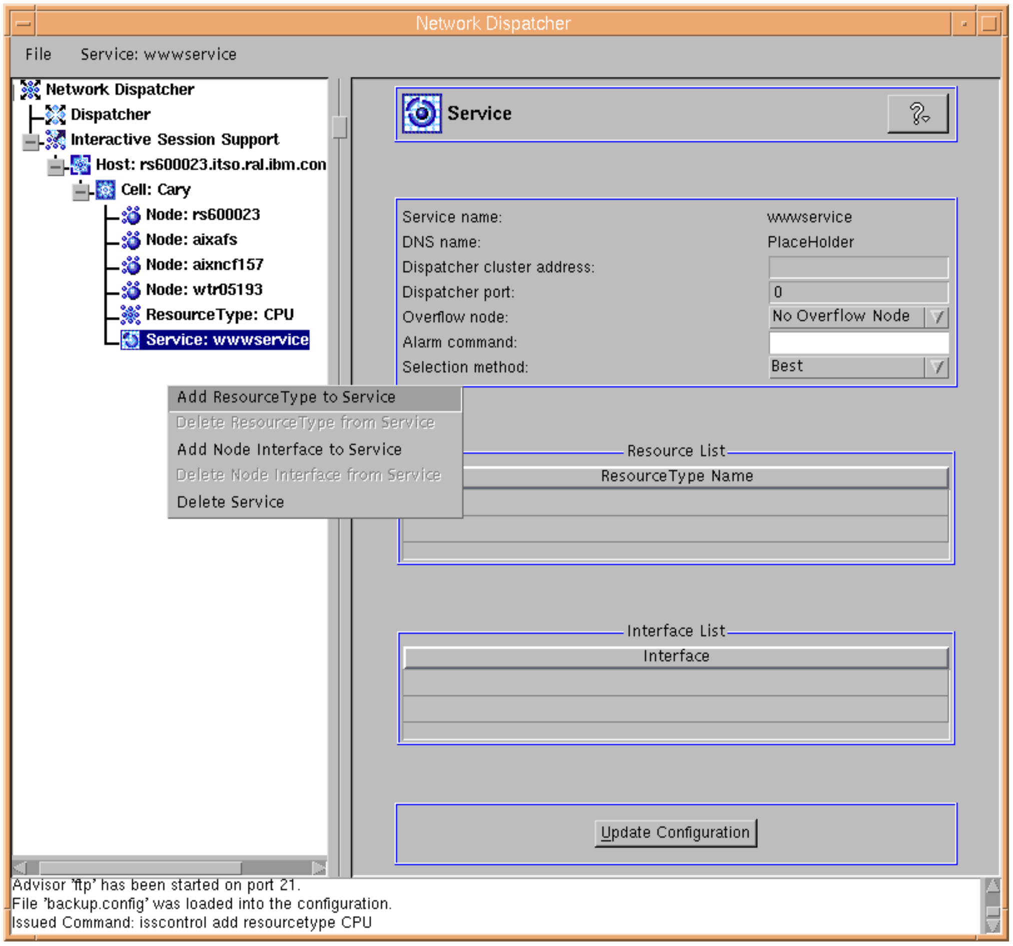


Service name: wwwservice

Service DNS name: PlaceHolder

Figure 112. Adding a Service

Once we added the service name, from the Service menu we selected **Add ResourceType to Service**, as shown in the following figure:

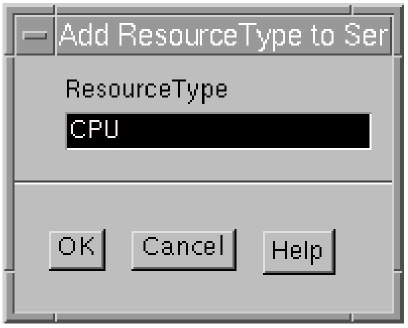


Service name:	wwwservice
DNS name:	PlaceHolder
Dispatcher cluster address:	
Dispatcher port:	0
Overflow node:	No Overflow Node
Alarm command:	
Selection method:	Best

```
Advisor 'ftp' has been started on port 21.
File 'backup.config' was loaded into the configuration.
Issued Command: isscontrol add resourcetype CPU
```

Figure 113. The Service Menu

In the Add ResourceType to Service window, we selected the CPU ResourceType that we had defined in 4.2.4.6, “Defining Resource Types” on page 149:



ResourceType: CPU

Figure 114. Adding a ResourceType to the Service

Following this, we selected **Add Node Interface to Service** from the Service menu (see Figure 113) and then we were presented with the Add Node Interface to Service window, showing the list of nodes we had added in 4.2.4.5, “Adding Nodes” on page 147:

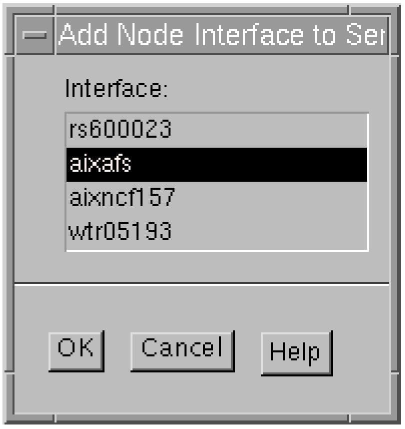
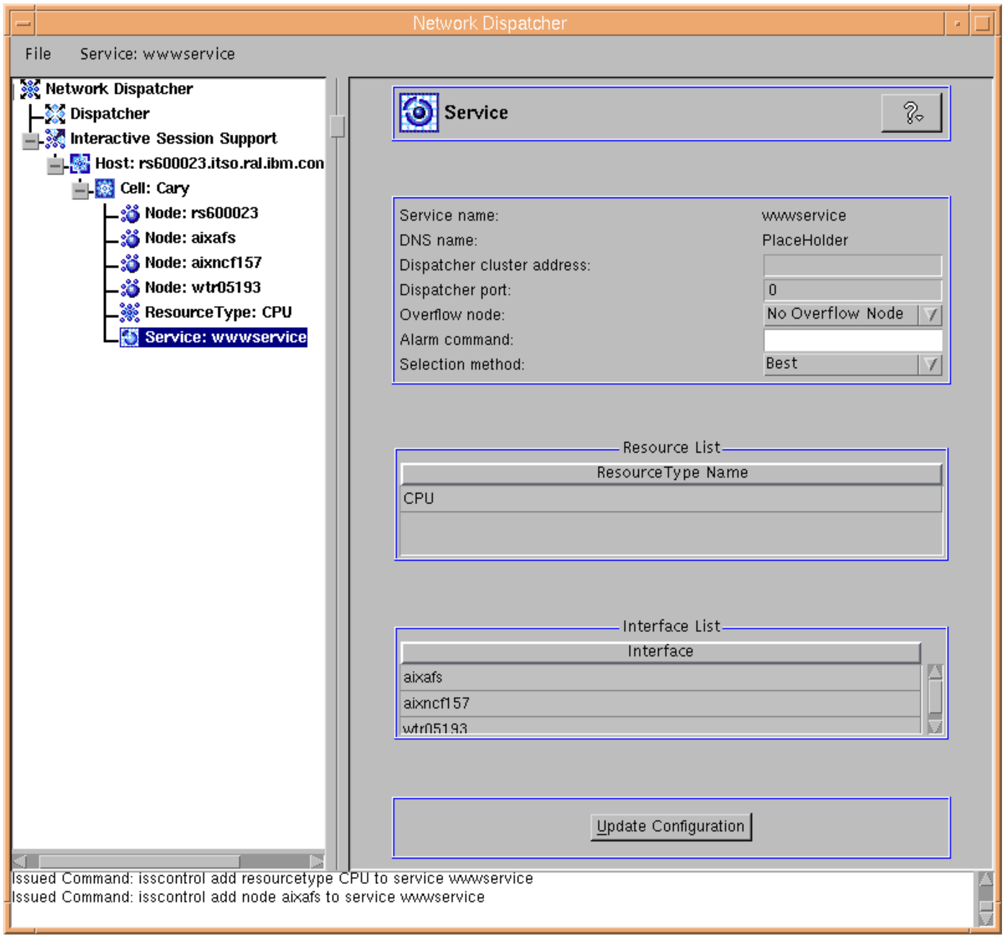


Figure 115. Add Node Interface to Service Selection List

We were able to only select one node at a time from this window. We added aixafs through the GUI and added the other two nodes to our wwwservice from the command line as follows:

```
isscontrol add node aixncf157 to service wwwservice
isscontrol add node wtr05193 to service wwwservice
```

The Interface List section of the Service information for the wwwservice service was updated to reflect the three nodes we had added:



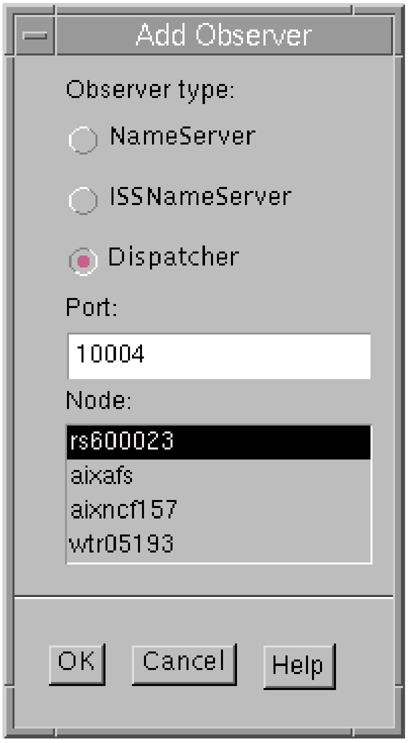
Service name:	wwwservice
DNS name:	PlaceHolder
Dispatcher cluster address:	
Dispatcher port:	0
Overflow node:	No Overflow Node
Alarm command:	
Selection method:	Best

```
Issued Command: isscontrol add resourcetype CPU to service wwwservice
Issued Command: isscontrol add node aixafs to service wwwservice
```

Figure 116. Service Statistics Showing ResourceType and Nodes (Interfaces) Added

4.2.4.8 Defining Observers

The next step was to add observers to our configuration. To do this, we right-clicked **Cell: Cary** and then selected **Add Observer** in the cell menu (see Figure 107 on page 147). The Add Observer window features three radio buttons at the top where you select which type of Observer you are defining. In this case, we selected Dispatcher and the default port number changed to 10004. This is the port that is used to send the metric load information to the Dispatcher. In the selection list at the bottom of the Add Observer window, we selected the host where the Dispatcher server was running, **rs600023**. This host is the machine where the Dispatcher was running and that ISS would be providing load statistics to.



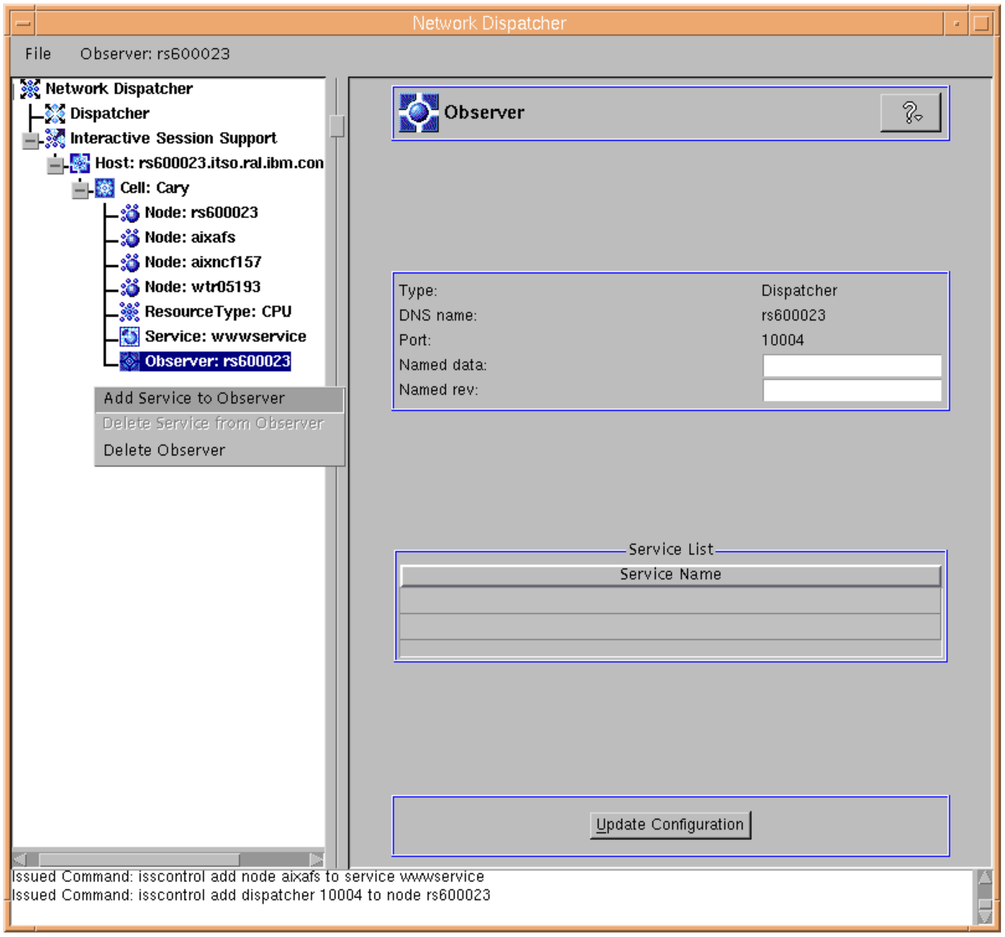
Observer type: [✓ Dispatcher]

Port: 10004

Node:
rs600023
aixafs
aixncf157
wtr05193

Figure 117. Add Observer

After we clicked **OK**, we went back to the GUI and selected the new Observer, **Observer: rs600023**. The settings for this Observer were immediately displayed. Next, we added a service to this Observer by right-clicking **Observer: rs600023** in the tree and selecting **Add Service to Observer**, as shown in the following figure:

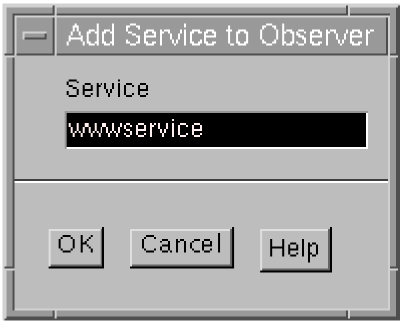


Type:	Dispatcher
DNS name:	rs600023
Port:	10004
Named data:	
Named rev:	

```
Issued Command: isscontrol add node aixafs to service wwwservice
Issued Command: isscontrol add dispatcher 10004 to node rs600023
```

Figure 118. Observer Menu

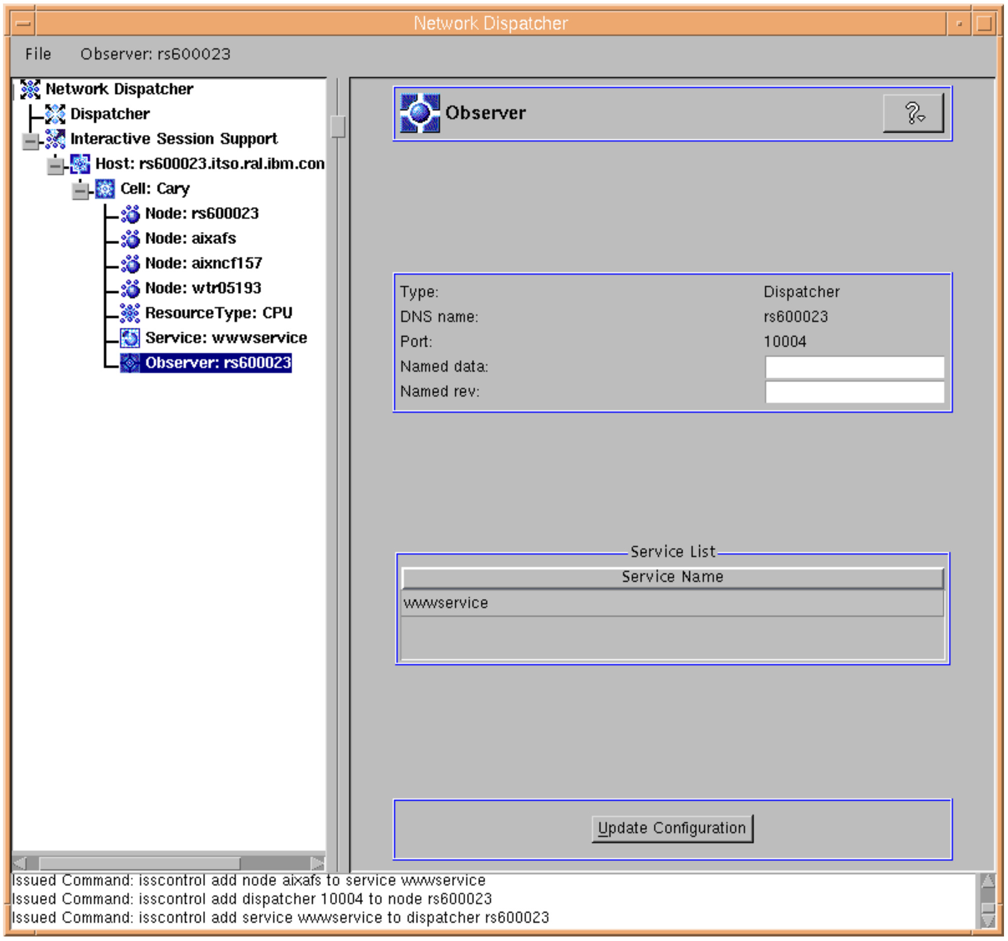
We were then shown a list of the defined Services in the Add Service to Observer window, and in this case there was only one: wwwservice. We selected it and clicked **OK**:



Service: wwwservice

Figure 119. Add Service to Observer Window

As a result of this, the Service List section for our rs600023 Observer was updated to reflect the service name we had just added to our Observer:



Type:	Dispatcher
DNS name:	rs600023
Port:	10004
Named data:	
Named rev:	

Service List: wwwservice

```
Issued Command: isscontrol add node aixafs to service wwwservice
Issued Command: isscontrol add dispatcher 10004 to node rs600023
Issued Command: isscontrol add service wwwservice to dispatcher rs600023
```

Figure 120. Updated Observer Information Showing the Service Name

At this point, we examined our ISS configuration file, /etc/iss.cfg, and saw that it had been modified to include the ISS keywords required to create the ISS environment that we had configured through the GUI, as shown below:

```
# ISS Configuration file
# Automatically generated at  1999/03/01 16:46:21

Cell                    Cary            LOCAL
PortNumber              7139
LogLevel                INFO
HeartbeatInterval       10
HeartbeatsPerUpdate     2
HeartbeatsToNetFail     4
HeartbeatsToNodeFail    6

Node                    rs600023                1
Node                    aixafs          2
Node                    aixncf157               3
Node                    wtr05193                98
NotMonitor

ResourceType            CPU
Metric                  INTERNAL        CPULoad
Policy                  MIN
MetricWeight            1.0
MetricNormalization     0.0             100.0
MetricLimits            80.0            95.0

Service                 wwwservice      PlaceHolder           0
SelectionMethod         BEST
ResourceList            CPU
NodeList                aixafs
                        aixncf157
                        wtr05193

Dispatcher              rs600023                10004
ServiceList             wwwservice
```

Figure 121. System Modified iss.cfg File

Once the configuration of our environment was complete, we placed a copy of this configuration file on the three other nodes and then started the ISS daemon on them as well. On the three Web server nodes, the ISS daemon started in its capacity as an agent as a result of the format of its own Node entry in the configuration file.

4.2.5 Managing ISS

In this section we explain how to stop and start both ISS and its configuration GUI. See 4.2.4.2, “Starting the ISS Daemon” on page 142 for details on how to start ISS, and see 4.2.4.3, “Starting the Configuration GUI” on page 143 for instructions on how to start the configuration GUI.

In the following section, we see how ISS can be controlled and reconfigured during run time.

4.2.5.1 Dynamically Reconfiguring ISS

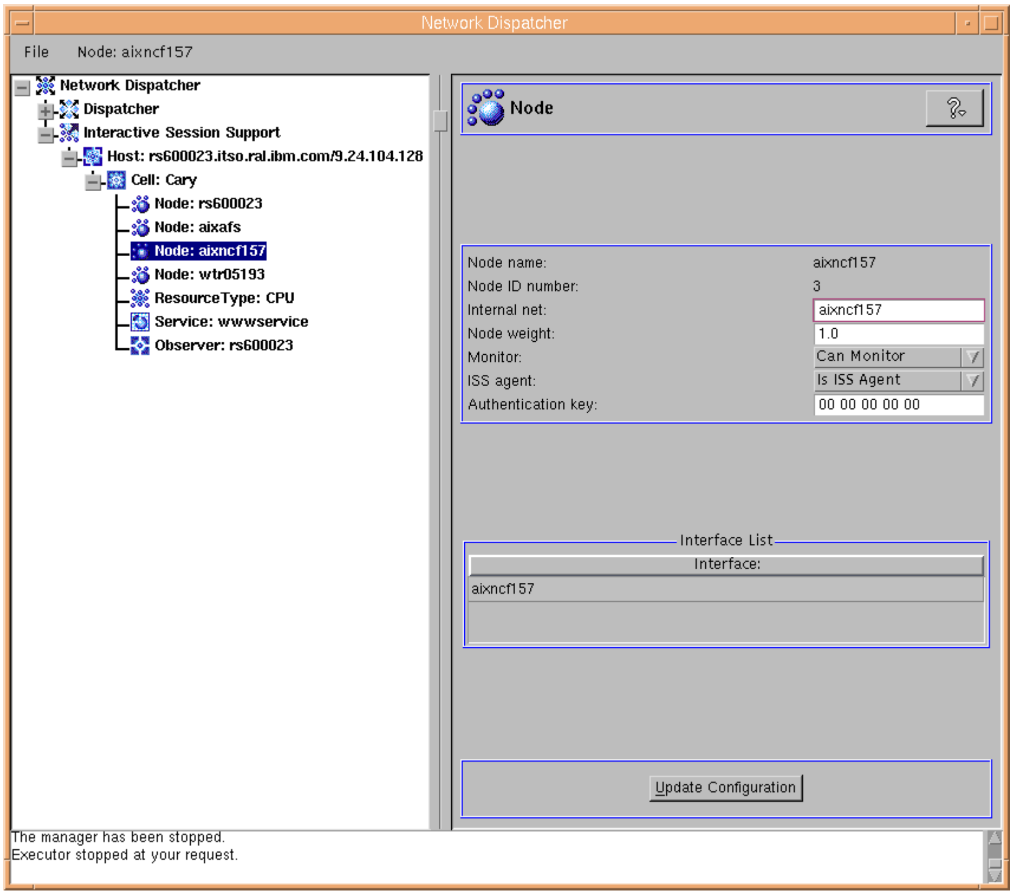
You can control and reconfigure ISS while it is running by using the `isscontrol` command or the GUI.

To demonstrate this, we used the environment described in 4.2, “Load Balancing Scenario Using the Dispatcher and ISS” on page 137. In that scenario, Web server nodes aixafs and aixncf157 were configured as being capable of taking over the function of ISS monitor if required. On the contrary, Web server node wtr05193 was configured to not take over the function of ISS monitor, as shown in Figure 109 on page 149. The demonstration of changing the ISS configuration was done in two steps:

1. We changed the designation of Web server node aixncf157 so that it could no longer be selected as an ISS monitor.
2. We monitored for this change to be reflected in the configuration file of one of our other nodes, for example aixafs.

Prior to the change being made, we used FTP to propagate the iss.cfg file to each of the nodes in our cell. Following this, we started ISS on each of these nodes.

We then verified both in the ISS configuration file and via the GUI, that aixncf157 had been designated as a potential monitor. We did this verification on both rs600023 and on aixafs. Following is the aixncf157 node status as it appeared on rs600023:

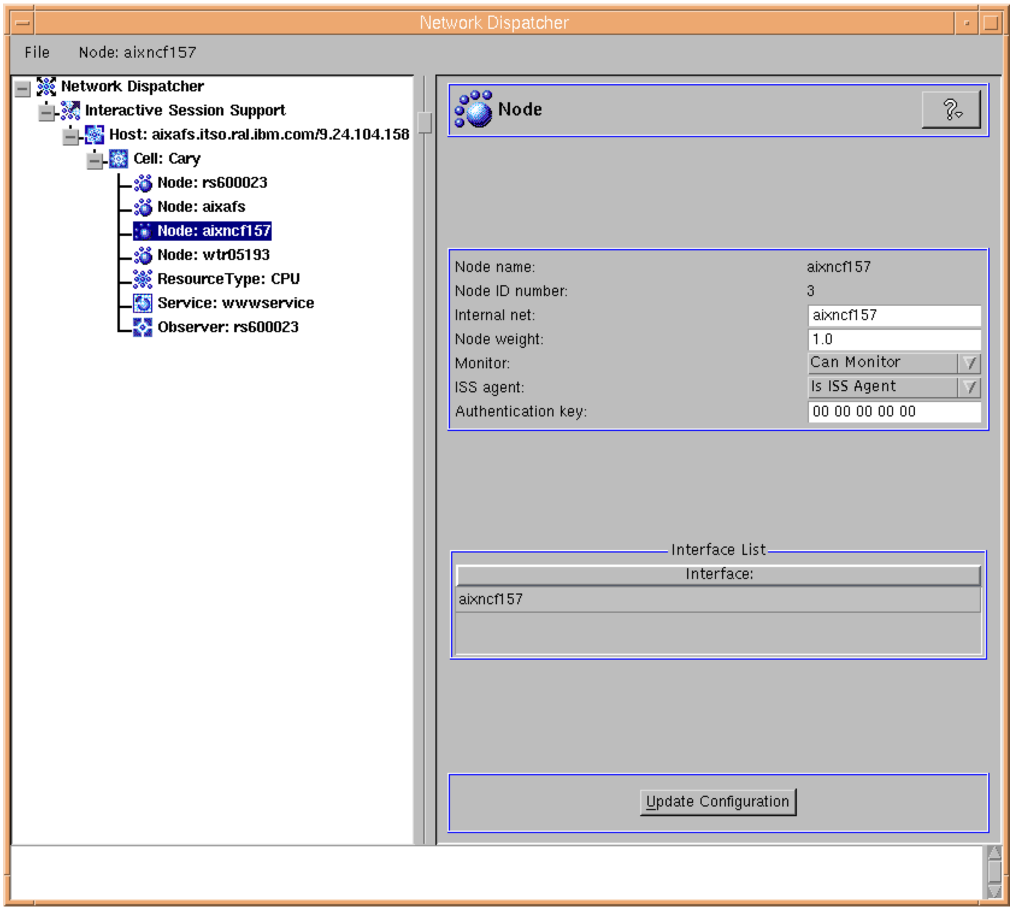


Node name:	aixncf157
Node ID number:	3
Internal net:	aixncf157
Node weight:	1.0
Monitor:	Can Monitor
ISS agent:	Is ISS Agent
Authentication key:	00 00 00 00 00

```
The manager has been stopped.
Executor stopped at your request.
```

Figure 122. Web Server Node aixncf157 Can Monitor Status as Shown on rs600023

The same status is shown on the other Web server node aixafs:



Node name:	aixncf157
Node ID number:	3
Internal net:	aixncf157
Node weight:	1.0
Monitor:	Can Monitor
ISS agent:	Is ISS Agent
Authentication key:	00 00 00 00 00

Interface: aixncf157

Figure 123. Web Server Node aixncf157 Can Monitor Status as Shown on aixafs

We then used the GUI on rs600023 to change the status of aixncf157. We did this by selecting **Cannot Monitor** on the Monitor pull-down menu.

Using the Command Line

The same change could have been accomplished by using the `isscontrol` command from the command line:

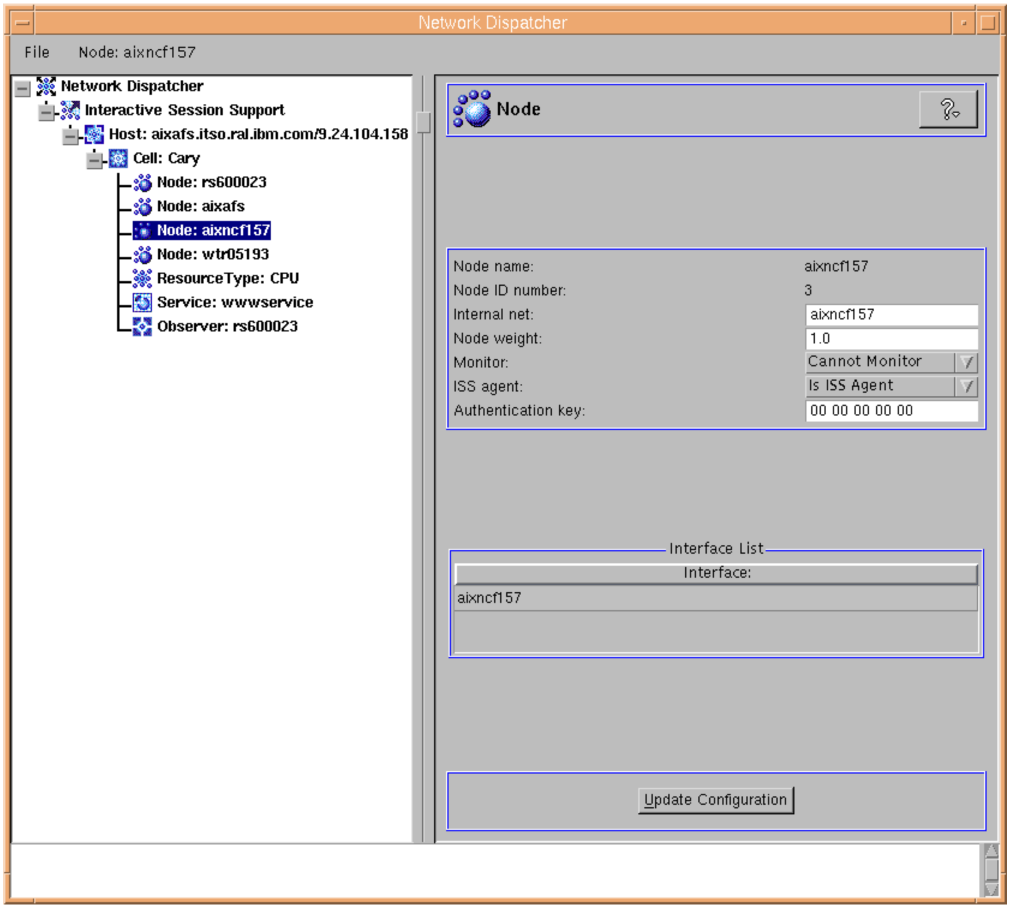
```
isscontrol set node nodename CanMonitor false
```

In our case, this would have appeared as:

```
isscontrol set node aixncf157 CanMonitor false
```

Then we clicked the **Update Configuration** button at the bottom of the GUI. We also confirmed that the ISS configuration files were updated on all of the machines by looking at the /etc/iss.cfg file on both rs600023 (where the change was made) and aixafs (one of the nodes participating in the cell).

Following this, we refreshed the GUI on our other Web server machine aixafs and saw that the Monitor status of aixncf157 had changed correspondingly:



Node name:	aixncf157
Node ID number:	3
Internal net:	aixncf157
Node weight:	1.0
Monitor:	Cannot Monitor
ISS agent:	Is ISS Agent
Authentication key:	00 00 00 00 00

Interface: aixncf157

Figure 124. Web Server Node aixncf157 Cannot Monitor Status as Shown on aixafs

We confirmed by the timestamp on the ISS configuration file on aixafs that the change had been automatically propagated from rs600023.

4.2.5.2 Saving the Host Connections

The GUI host connections can be saved by selecting **File** on the menu bar of the configuration window (see Figure 120 on page 157) and then clicking **Save Host Connections As...**. In the panel that appears, enter the configuration file name that you would like the Host Connection information saved into (see Figure 96 on page 136). This file is saved in the directory *installbase*/configurations, where *installbase* is the install directory for the admin component that varies by operating system. See Table 1 on page 69 for a list of the installbase locations.

As a result of saving this information, on subsequent invocations of the GUI this host connection will be listed as one of the choices in the Connect to Host selection screen (see Figure 46 on page 87).

4.2.5.3 Disconnecting from the Host

To disconnect from the host, right-click the **Host** item, either in the tree or on the menu bar, and then select **Disconnect from Host**.

4.2.5.4 Terminating the GUI

To exit the GUI, select **File->Exit**. Your ISS configuration file should automatically be updated at this time.

4.2.5.5 Stopping ISS

On AIX and Solaris, as root, enter:

```
isscontrol shutdown nodename
```

On Windows NT, as a system administrator, from the Services folder of the Control Panel, select **IBM Interactive Session Support**, then click **Stop**.

Chapter 5. Rules-Based Load Balancing

You can use *rules-based load balancing* to fine tune when and why packets are sent to which servers. Rules-based load balancing is a feature available only for the Dispatcher and Content Based Routing (CBR) components of IBM SecureWay Network Dispatcher (ND) V2.1. These components review any rules you add from first priority to last priority, stopping on the first rule that they find to be true, and then balance the load between any servers associated with the rule.

The Dispatcher already has the capability of balancing the load based on destination and port. Using rules expands your ability to distribute connections and offers another possible way to manage the load distribution in a cluster. Rules are particularly useful when you want to distribute the load to a subset of your servers for any reason. You must always use rules with the CBR component.

5.1 Types of Rules

You can use the following types of rules:

- Client IP address

  You might want to use rules based on the client IP address if you want to screen the customers and allocate resources based on where they are coming from.

  For example, you have noticed that your network is getting a lot of unpaid and therefore unwanted traffic from clients coming from a specific set of IP addresses. You can add a rule that instructs the Dispatcher not to serve those requests, or to distribute them among a limited subset of your servers.

- Time of day

  You may want to use rules based on the time of day for capacity planning reasons. For example, if your Web site gets accessed most during the same group of hours every day, you might want to dedicate five servers to HTTP full-time, then add another five during the peak time period.

  Another reason you might use a rule based on the time of day could be, for example, if you want to bring some of the servers down for maintenance every night at midnight. In this case, you would set up a rule that excludes those servers during the necessary maintenance period.

- Connections per second on a port

  You might want to use rules based on connections per second on a port if you need to share some of your servers with other applications.

  For example, you set two rules:

  1. If the number of connections per second on port 80 is greater than 100, then distribute the load on two specific Web servers.
  2. If the number of connections per second on port 80 is greater than 2000, then distribute the load on 10 specific Web servers.

  Or you might be using Telnet and want to reserve two of your five servers for Telnet use, except when the connections per second increase above a certain level. This way, the Dispatcher would balance the load across all five servers at peak times.

Note that the Manager must be running for the above to work.

- Active connections total for a port

  You might want to use rules based on the total number of active connections on a specific port. This is very useful for when your servers get overloaded and start throwing packets away.

  In fact certain Web servers will continue accepting connections even though they do not have enough threads to respond to the requests. As a result, the client requests time out and the customer coming to your Web site is not served. You can set rules based on the total number of active connections to balance capacity within a pool of servers.

  For example, you know from your experience that your servers will stop serving requests after they have accepted 250 connections. Then, you can create a rule that instructs the Dispatcher to use your current servers, but some additional servers are automatically added when the total number of active connections becomes greater than 250. Those additional servers will otherwise be used for other processing.

  Note that the Manager must be running for the above to work. This type of rule is available to both the Dispatcher and CBR components.

- Client port

  You may want to use rules based on the client port if your clients are running software that uses a specific client port when making requests.

  For example, you could create a rule that says that any requests with a client port of 10002 will have to use a set of special fast servers because you know that any client requests with that port are coming from an elite group of customers.

  The client port rule type is available only for the Dispatcher component.

- Content of a request

  Request content type rules are used to send requests to sets of servers specifically set up to handle some subset of your sites traffic. For example, you may want to use one set of servers to handle all CGI-BIN requests, another set to handle all streaming audio requests, and a third set to handle all other requests. To do this, you will add one rule with a pattern that matches the path to your CGI-BIN directory, another that matches the file type or your streaming audio files, and a third always true rule to handle the rest of the traffic. You will then add the appropriate servers to each of the rules.

  This rule type is available only to the CBR component. For further information about configuring content type rules, see Chapter 14, “Content Based Routing” on page 329.

- Always true

  A rule can be created that is *always true*. Such a rule will always be selected, unless all the servers associated with it are down. For this reason, it should ordinarily be at a lower priority than other rules. You can even have multiple always-true rules, each with a set of servers associated with it. The first true rule with an available server is chosen.

  For example, assume you have six servers. You want two of them to handle your traffic under all circumstances, unless they are both down. If the first two servers are down, you want a second set of servers to handle the traffic. If all four of these servers are down, then you will use the final two servers to handle the traffic. To do this, you can set up three always-true rules. Then, the first set of servers will always be chosen as long as at least one is up. If they are both down, one from the second set will be chosen, and so forth.

You can define more than one always-true rule, and thereafter adjust which one gets executed by changing their priority levels.

We recommend that you make a plan of the logic that you want Dispatcher to follow before you start adding rules to your configuration.

5.2 How Rules Are Evaluated

All rules have a name, type, priority, begin range, and end range, and may have a set of servers. Rules are evaluated in priority order, with lower priority rules evaluated first. In other words, a rule with a priority of 1 will be evaluated before a rule with a priority of 2. The first rule that is satisfied will be used. Once a rule has been satisfied, no further rules are evaluated.

For a rule to be satisfied, it must meet two conditions:

1. The predicate of the rule must be `true`. That is, the value it is evaluating must be between the begin and end ranges. For always-true rules, the predicate is always satisfied, regardless of the begin and end ranges.
2. If there are servers associated with the rule, at least one of them must be available to forward packets to.

If a rule has no servers associated with it, the rule needs to meet only the first condition to be satisfied. If no rules are satisfied, a server will be selected from the full set of servers available on the port.

5.3 Rules-Based Load Balancing Scenario

In this section we show you how it is possible to integrate the load balancing rules in a working scenario. Our scenario involved a Web client machine sending requests to a Web site, composed of three clustered Web servers balanced by a Dispatcher machine. We started by confirming that all three clustered servers were receiving Web requests in a round-robin fashion. We then added a rule eliminating one of the servers during certain hours of the day. We made some requests during those hours and then examined again the distribution of requests to verify the implication of the rule.

5.3.1 Network Environment

The network environment we used in this scenario is the same one that was used in 4.2, “Load Balancing Scenario Using the Dispatcher and ISS” on page 137, and is described in the following table:

Table 9. Basic Scenario - Hardware, Software, and Network Configuration

Workstation	Host Name	IP Address	Operating System	Service
IBM PC 365	wtr05212	9.24.104.218	Windows NT Server 4.0	Web Client

Workstation	Host Name	IP Address	Operating System	Service
IBM RS/6000 43P	rs600023	9.24.104.128	AIX 4.3.1	Dispatcher
	clusterend	9.24.104.105		
IBM RS/6000 43P	aixncf157	9.24.104.157	AIX 4.3.1	Web Server
IBM RS/6000 43P	aixafs	9.24.104.158	AIX 4.3.1	Web Server
IBM PC 365	wtr05193	9.24.104.239	Windows NT Server 4.0	Web Server

All the above machines were provided with a token-ring interface and connected to the same local area network (LAN).

Notice that:

1. The load balancing function was provided by the Dispatcher component of ND V2.1.
2. Netscape Navigator 4.5 was the Web browser running on the Web client machine.
3. The Web server function on the three clustered Web servers was provided by the IBM HTTP Server Version 1.3.3.

The following figure offers a logical representation of the network environment where we performed this scenario:

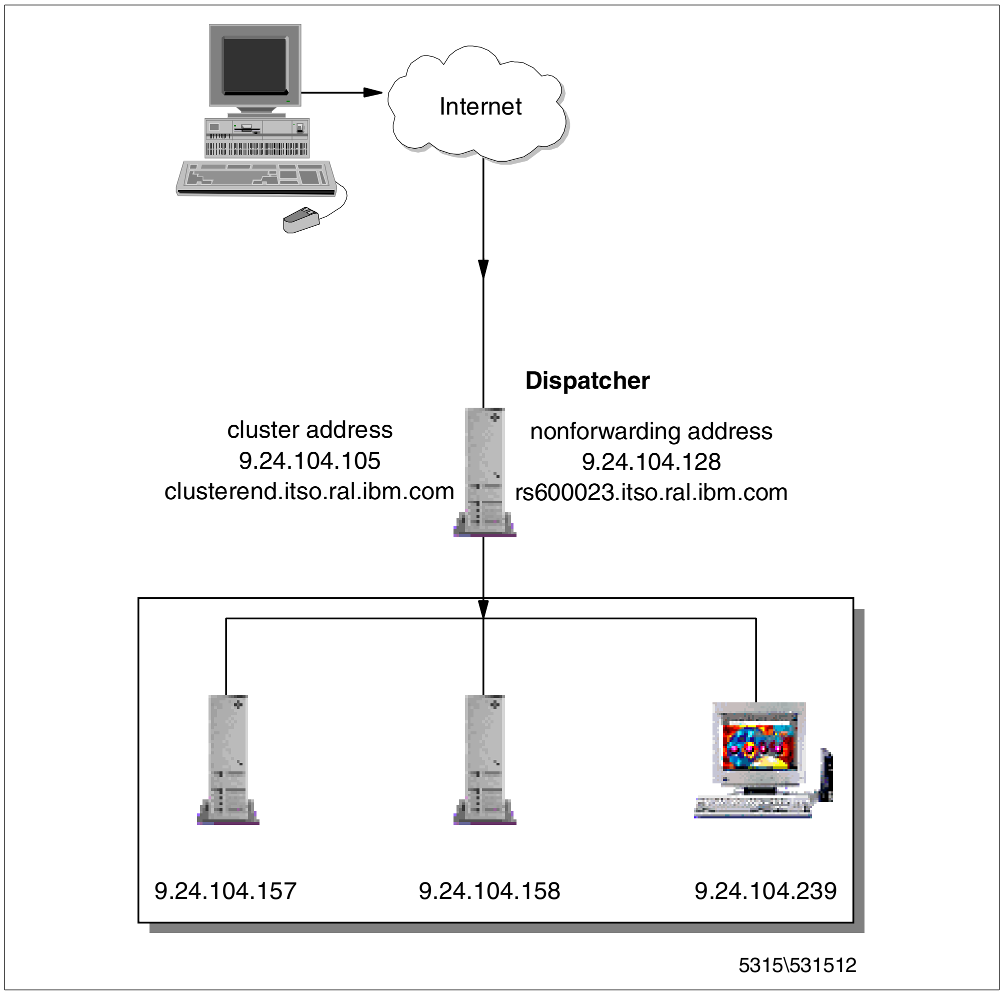
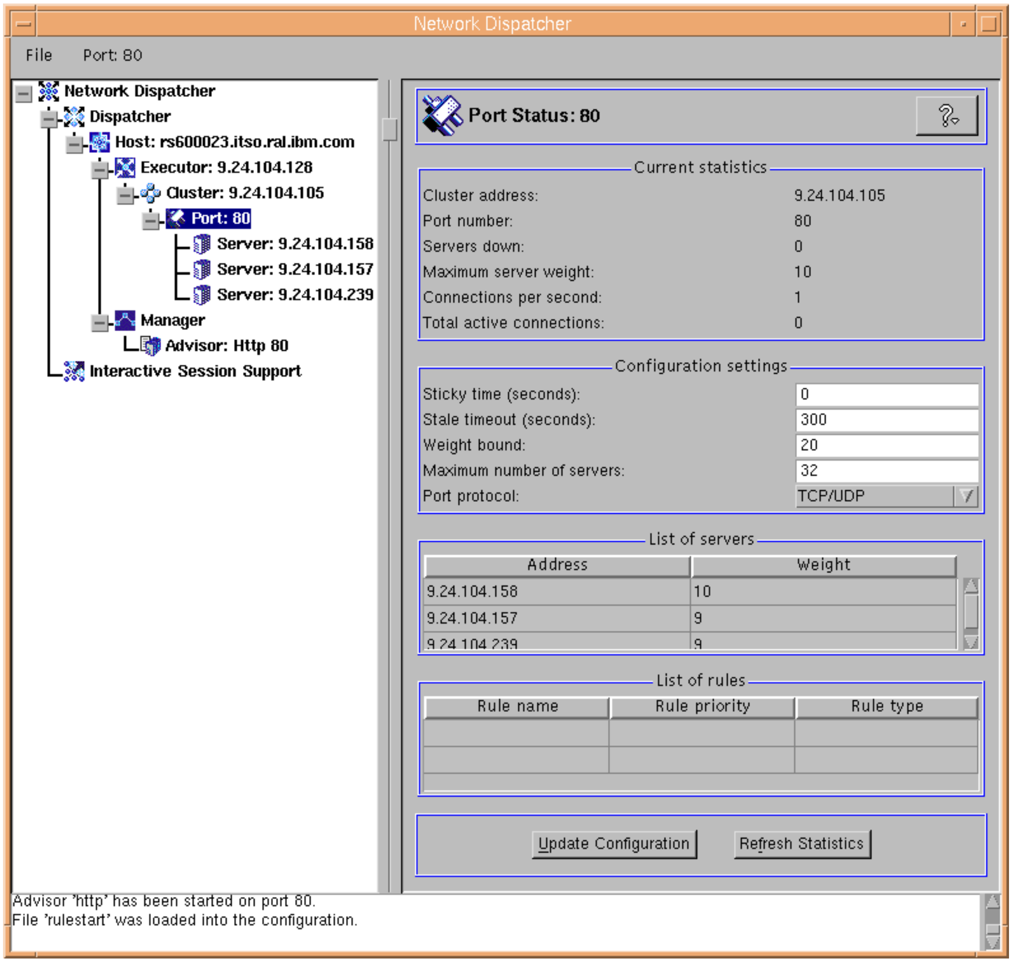


Figure 125. Graphical Representation of the Rules-Based Scenario

5.3.2 Rules Configuration

We added rules to our Dispatcher using the graphical user interface (GUI).

The following figure is the ND GUI that displays the basic running configuration of the Dispatcher that we set up:



Current statistics

Cluster address:	9.24.104.105
Port number:	80
Servers down:	0
Maximum server weight:	10
Connections per second:	1
Total active connections:	0

Configuration settings

Sticky time (seconds):	0
Stale timeout (seconds):	300
Weight bound:	20
Maximum number of servers:	32
Port protocol:	TCP/UDP

List of servers

Address	Weight
9.24.104.158	10
9.24.104.157	9
9.24.104.239	9

List of rules

Rule name	Rule priority	Rule type

```
Advisor 'http' has been started on port 80.
File 'rulestart' was loaded into the configuration.
```

Figure 126. Basic Dispatcher Configuration

To generate the HTTP requests, we deactivated the browser cache on the Web client machine, and set up a direct connection to the Internet without passing through a proxy server.

Before adding a rule, we submitted a set of requests from the Web client and verified that the Dispatcher was able to select each of the three servers members of the cluster. We did this by using the ND GUI monitor to view the number of new connections on port 80. To bring up the ND GUI monitor, we right-clicked on the **Port: 80** tree element, and in the pop-up menu we selected **Monitor**. The manager component must be active in order for **Monitor** to be selectable. The monitor output is shown in the following figure:

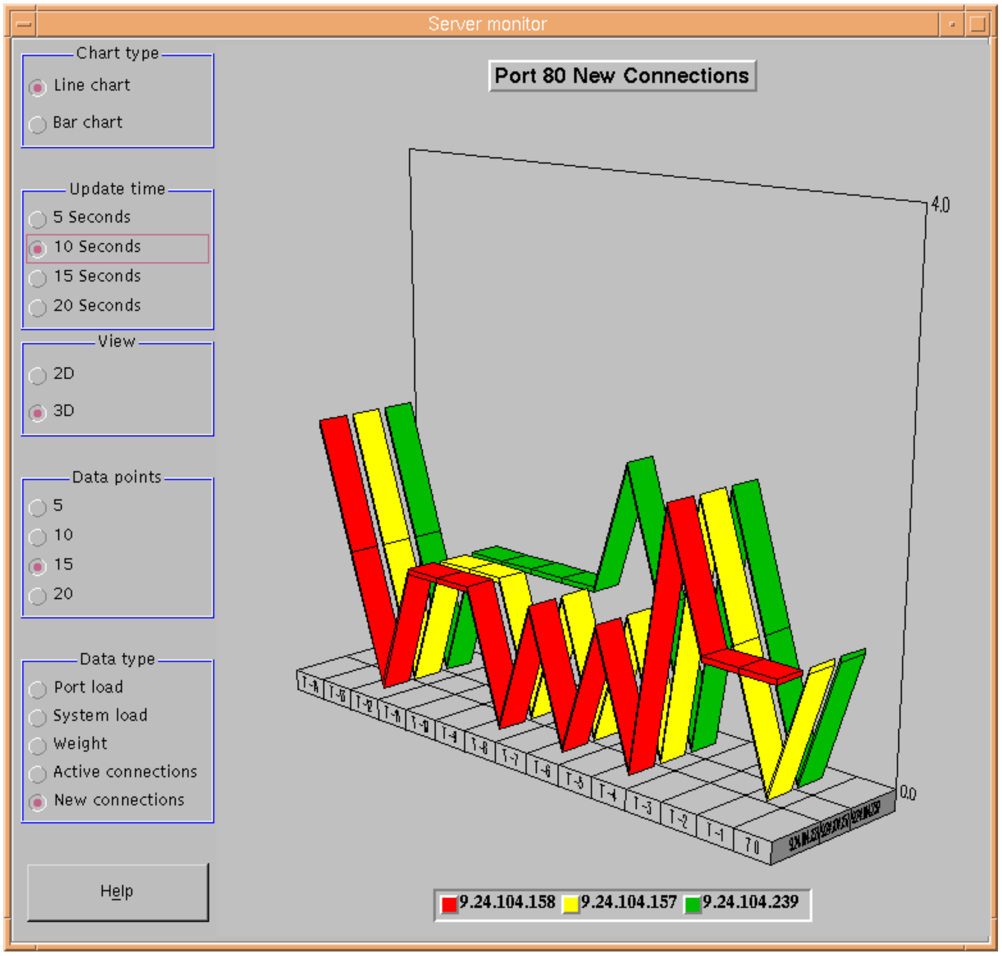

Server monitor

Chart type: [✓ Line chart] Bar chart

Update time: 5 Seconds [✓ 10 Seconds] 15 Seconds 20 Seconds

View: 2D [✓ 3D]

Data points: 5 10 [✓ 15] 20

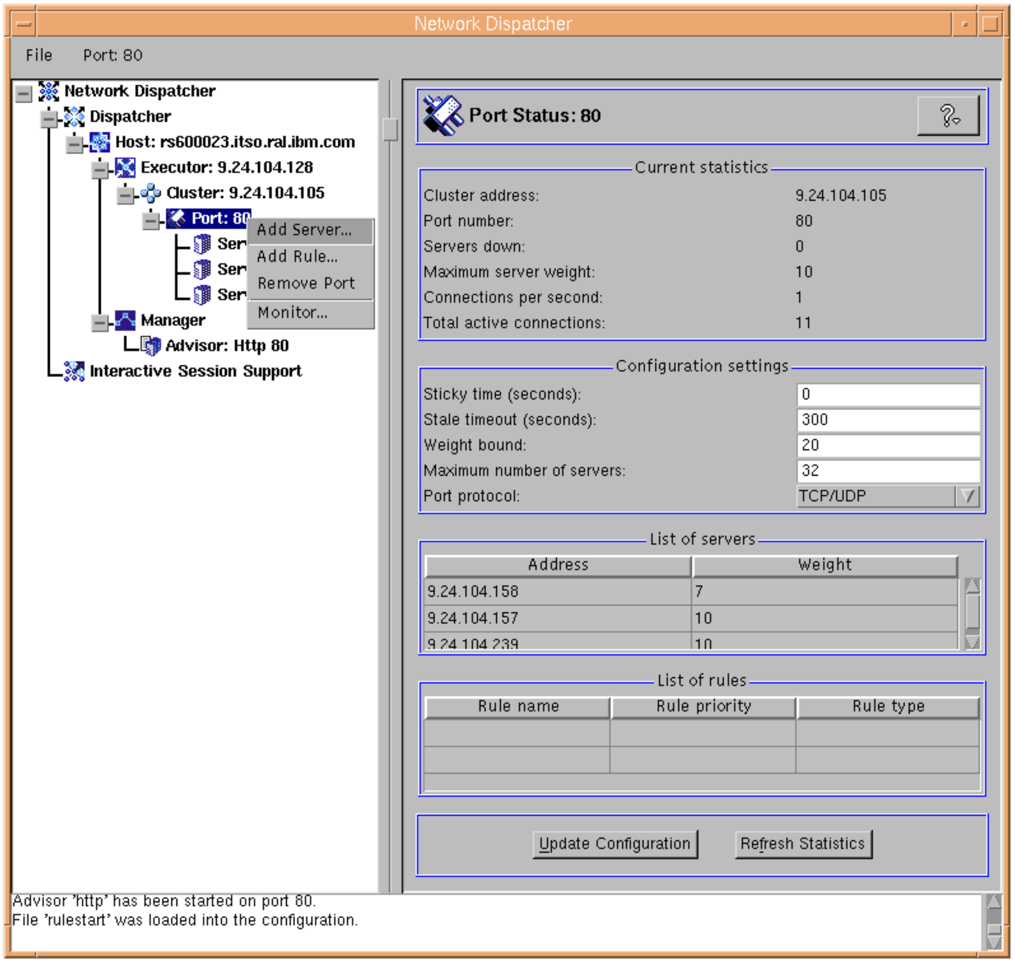
Data type: Port load System load Weight Active connections [✓ New connections]

Help

Figure 127. Dispatcher GUI Monitor

At this point, we wanted to add a rule based on the time of day. What we wanted the Dispatcher to do was to distribute the incoming HTTP requests from 19:00 through 19:59 only to the two Web server machines with IP addresses 9.24.104.157 and 9.24.104.158. Outside of the specified time, all the three Web servers should have been active.

To add the rule, we right-clicked the item **Port: 80** in the left panel of the ND GUI, and in the pop-up menu, selected **Add Rule...**:



Current statistics

Cluster address:	9.24.104.105
Port number:	80
Servers down:	0
Maximum server weight:	10
Connections per second:	1
Total active connections:	11

Configuration settings

Sticky time (seconds):	0
Stale timeout (seconds):	300
Weight bound:	20
Maximum number of servers:	32
Port protocol:	TCP/UDP

List of servers

Address	Weight
9.24.104.158	7
9.24.104.157	10
9.24.104.239	10

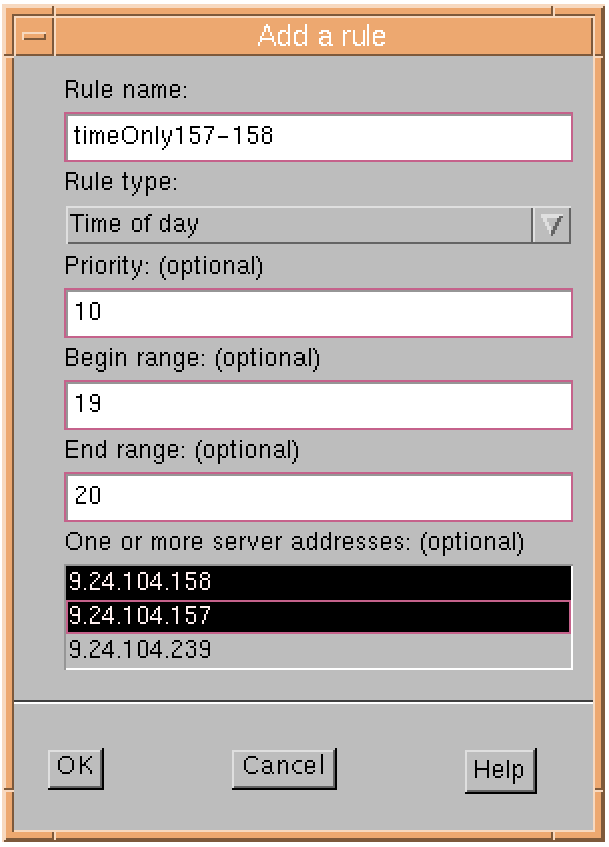
List of rules

Rule name	Rule priority	Rule type

```
Advisor 'http' has been started on port 80.
File 'rulestart' was loaded into the configuration.
```

Figure 128. Adding a Rule

Then the Add a Rule dialog window was displayed:



Rule name: timeOnly157-158

Rule type: Time of day

Priority: (optional) 10

Begin range: (optional) 19

End range: (optional) 20

One or more server addresses: (optional)

9.24.104.158
9.24.104.157
9.24.104.239

Figure 129. Add a Rule Dialog Window

We filled in the dialog window as follows:

- We assigned a mnemonic name to the rule. In our case, we entered `timeOnly157-158`.
- We selected the type of the rule. In our case, we chose **Time of Day**.

  Other choices for the rule type would be:

  - **IP address**

    The rule will be based on the client IP address.

  - **Total connections (per second)**

    The rule will be based on the number of connections per second on the port. This rule will work only if the Manager is running.

  - **Active connections (total)**

    The rule will be based on the number of active connections total on the port. This rule will work only if the Manager is running.

  - **Client port**

    The rule will be based on the client port number.

  - **Always true**

    The rule will always be true.

- Then we specified the priority of the rule. Priorities establish the order in which rules will be reviewed. This parameter accepts integer values, and we set it to `10`.

  If you do not specify the priority of the first rule you add, Dispatcher will set it by default to `1`. When a subsequent rule is added, by default its priority is calculated to be 10 plus the current lowest priority of any existing rule. For example, assume you have an existing rule whose priority is 30. You add a new rule and set its priority at `25` (which, remember, is a higher priority than 30). Then you add a third rule without setting a priority. The priority of the third rule is calculated to be 30 + 10 = 40.

- Then we specified the begin range and the end range parameters.

  The begin range parameter represents the lower value in the range used to determine whether or not the rule is true. The type of values you can assign to it depends on the type of the rule. The type of values and its default are listed here, depending on the rule type:

  - IP address

    This is the address of the client as either a symbolic name or in dotted-decimal format. The default is 0.0.0.0.

  - Time of day

    An integer is expected here. The default is 0, representing midnight.

  - Total connections (per second)

    An integer is expected here. The default is 0.

  - Active connections (total)

    This is an integer. The default is 0.

  - Client port

    This is an integer. The default is 0.

  The end range parameter represents the higher value in the range used to determine whether or not the rule is true. The type of values you can assign to it depends on the type of the rule. The type of value and its default are listed here, depending on the rule type:

  - IP address

    This is the address of the client as either a symbolic name or in dotted-decimal format. The default is 255.255.255.254.

  - Time of day

    An integer is expected here. The default is 24, representing midnight.

    Note that when defining the begin range and end range parameters for time intervals, each value must be an integer representing only the hour portion of the time; portions of an hour are not specified. For this reason, to specify a single hour, say, the hour between 3:00 and 4:00 a.m., you would specify a begin range of 3 and an end range also of 3. This will signify all the minutes between 3:00 and 3:59. Specifying a begin range of 3 and an end range of 4 would cover the two-hour period from 3:00 through 4:59.

    In our case, we wanted the rule to be true during the time from 19:00 through 20:59, and so we set the begin range to `19` and the end range to `20`.

- Total connections (per second)

  This is an integer. The default is 2^{32} - 1.

- Active connections (total)

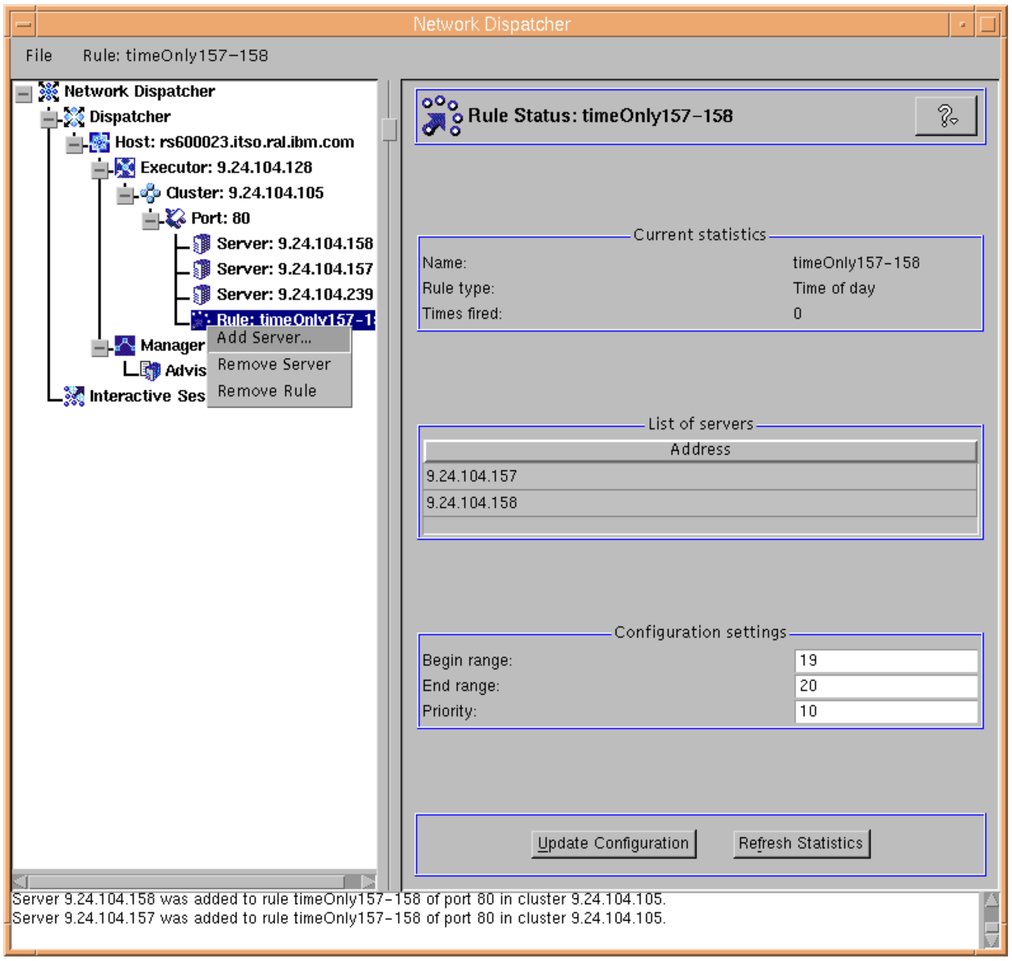
  This is an integer. The default is 2^{32} - 1.

- Client port

  An integer is expected here. The default is 65535.

- The last item on the Add a rule dialog window is a new feature of this version. You are presented with a scrolled list of server addresses to optionally choose from. In this case we chose **9.24.104.158** and **9.24.104.157**, the two servers that we wanted to serve the client requests between 19:00 and 19:59. We used the Shift key to select the second server after the first one was selected.

At this point we clicked the **OK** button in Figure 129 on page 171. The GUI reported the configuration was updated with the added rule, as shown in the following figure:



Name:	timeOnly157–158
Rule type:	Time of day
Times fired:	0

Address
9.24.104.157
9.24.104.158

Begin range:	19
End range:	20
Priority:	10

```
Server 9.24.104.158 was added to rule timeOnly157–158 of port 80 in cluster 9.24.104.105.
Server 9.24.104.157 was added to rule timeOnly157–158 of port 80 in cluster 9.24.104.105.
```

Figure 130. Updated Rule Status Window

Once the rule is added, if you want to change which servers to use if the rule is true, right-click the **Rule** and select **Add Server...** or **Remove Server** as shown in Figure 130.

This step completed the rule configuration.

The same configuration could have been accomplished from the command line by using these commands:

```
ndcontrol rule add 9.24.104.105:80:timeOnly157-158 type time priority 10 beginrange 19 endrange 20
ndcontrol rule useserver 9.24.104.105:80:timeOnly157-158 9.24.104.158
ndcontrol rule useserver 9.24.104.105:80:timeOnly157-158 9.24.104.157
```

To verify that the configuration shown was working correctly, we submitted again a set of requests from the Web browser to the clustered Web site, but this time we set it between 19:00 and 19:59. The following screen shows the Dispatcher monitor GUI displaying the new connections on port 80:

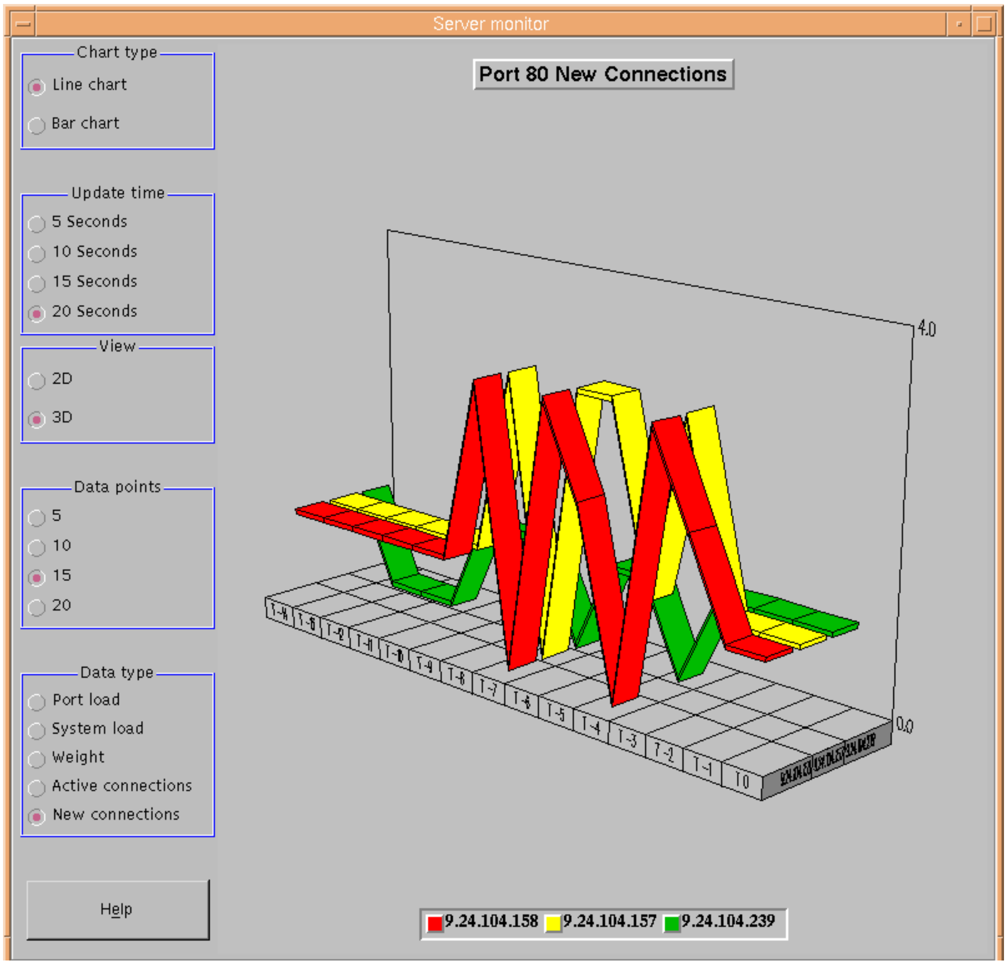


Chart type: Line chart [✓ Line chart] Bar chart

Update time: 5 Seconds, 10 Seconds, 15 Seconds, [✓ 20 Seconds]

View: 2D, [✓ 3D]

Data points: 5, 10, [✓ 15], 20

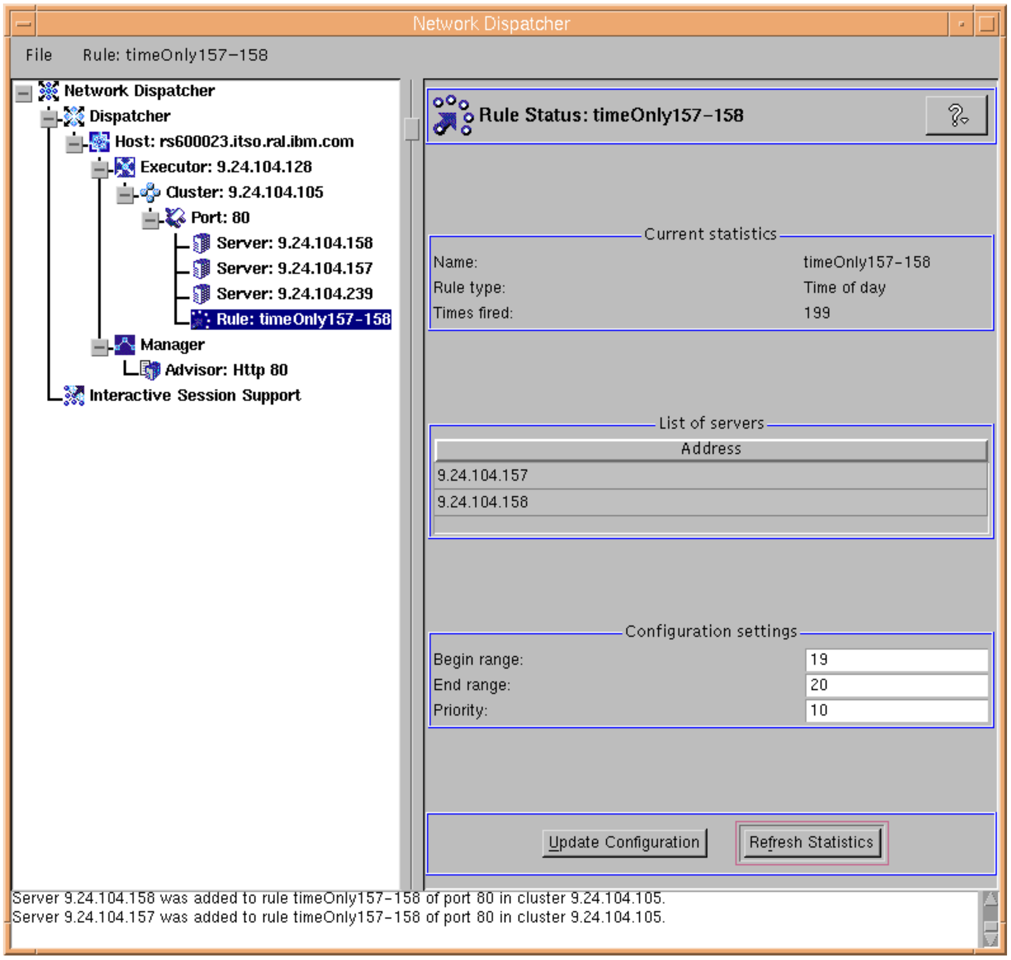
Data type: Port load, System load, Weight, Active connections, [✓ New connections]

🟥 9.24.104.158 🟨 9.24.104.157 🟩 9.24.104.239

Figure 131. Verifying that the Rule Configuration Has Been Successful

This shows that the Dispatcher was not routing new requests to 9.24.104.239. The base level for all three servers is not 0.0 because we have an active HTTP advisor that is sending HEAD requests to each of the servers to ensure they are still responding and to assess the load on each of them.

The Rule Status window, in the Current Statistics section, shows the number of times the rule is activated:



Current statistics

Name:	timeOnly157–158
Rule type:	Time of day
Times fired:	199

List of servers

Address
9.24.104.157
9.24.104.158

Configuration settings

Begin range:	19
End range:	20
Priority:	10

```
Server 9.24.104.158 was added to rule timeOnly157–158 of port 80 in cluster 9.24.104.105.
Server 9.24.104.157 was added to rule timeOnly157–158 of port 80 in cluster 9.24.104.105.
```

Figure 132. Rule Status Window Showing the Number of Times the Rule Was Activated

The timeOnly157-158 rule triggered 199 times during our test. After 20:00, we generated another HTTP request load from our Web browser. This time, as expected, the Dispatcher made use of all three Web servers in the cluster.

Figure 131 and Figure 132 confirmed that the rule configuration was successful.

Chapter 6. ND High Availability Support

In 4.2, “Load Balancing Scenario Using the Dispatcher and ISS” on page 137, we showed how to implement ISS high availability.

The Dispatcher component of IBM WebSphere Performance Pack Version 2 offers built-in high availability also for the Dispatcher function. A standby Dispatcher machine remains ready at all times to take over load balancing should the primary Dispatcher machine fail.

The high-availability configuration detects and recovers from network and server failures. The Dispatcher component of IBM WebSphere Performance Pack Version 2 is smart enough to determine that a server or a network is down. In case of failure, clients lose only the current connections, but they can immediately establish a new connection to the remaining servers with no problems.

The high-availability environment involves two Dispatcher machines with connectivity to the same clients, and to the same cluster of servers, as well as connectivity between the Dispatchers. Both the Dispatchers must be using the same operating systems.

The two Dispatcher machines are usually referred to as *primary* machine and *backup* machine:

- The primary machine works normally as a Dispatcher, and is in the *active* state while it is balancing the load among the servers of its clusters.
- The backup machine, configured in a very similar way to the primary machine, stays in *standby* mode unless the primary fails.

The two machines are synchronized, and only the primary machine routes packets, while the backup machine is continually updated.

The two machines establish communication to monitor the status of each other, referred to as a *heartbeat*, using a port that you can choose. If the primary machine fails, the backup machine detects this failure, switches to active state, and begins to take over the routing of packets. When the primary machine is operational again, but in standby state, you can either decide that it again *automatically* becomes the active machine, or leave it in standby mode. In this case, you will have to act *manually* if you want it to become the active machine again.

We see in 6.1, “Dispatcher High-Availability Scenario” on page 177 how to implement Dispatcher high availability. In 11.1, “Firewall High Availability Using the Dispatcher” on page 273, we show how it is possible to use the high-availability feature incorporated in the Dispatcher to provide *firewall high availability*.

6.1 Dispatcher High-Availability Scenario

In this section we describe the procedure we followed to configure the Dispatcher high-availability feature on our test environment.

6.1.1 Network Architecture

A summary of the hardware, software and network configuration of the environment where we performed our test is reported in the following table:

Table 10. Basic Scenario - Hardware, Software, and Network Configuration

Workstation	Host Name	IP Address	Operating System	Service
IBM RS/6000 43P	rs600023	9.24.104.128	AIX 4.3.1	Primary Dispatcher
	clusterend	9.24.104.105		
IBM RS/6000 43P	rs600030	9.24.104.97	AIX 4.3.1	Backup Dispatcher
	clusterend	9.24.104.105		
IBM RS/6000 43P	aixncf157	9.24.104.157	AIX 4.3.1	Web Server
IBM RS/6000 43P	aixafs	9.24.104.158	AIX 4.3.1	Web Server
IBM PC 365	wtr05193	9.24.104.239	Windows NT Server 4.0	Web Server

As you can see from the table above, the network interface on each Dispatcher machine is configured to respond to two IP addresses and host names: the nonforwarding address and the cluster address respectively.

All the above machines had a token-ring interface and were connected to the same local area network (LAN). The domain name for all the machines was itso.ral.ibm.com.

Dispatcher Operating Systems when Using High Availability

You can see that we installed both the primary and the backup Dispatcher machines on AIX 4.3.1. The most important thing you should remember when you make your plans for Dispatcher high availability is that the two machines implementing the Dispatcher function are currently only supported if they run the same operating system. You should not configure the high availability feature if, for example, one Dispatcher machine runs Windows NT and the other AIX or Solaris.

The following figure offers a graphical representation of the Dispatcher high-availability scenario that we implemented:

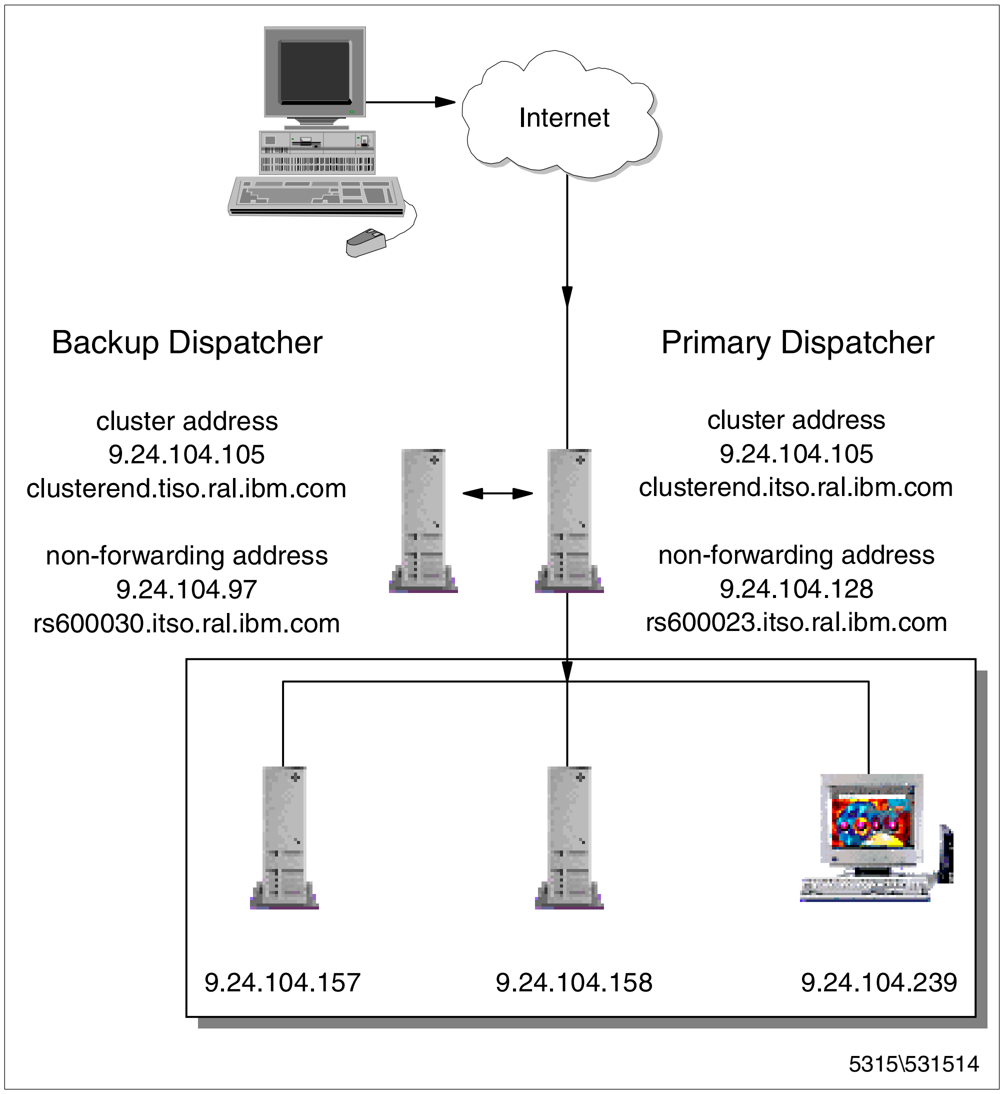


Figure 133. High-Availability Scenario Configuration

6.1.2 Configuration Steps Common on Both the Machines

Before starting the configuration, you need to define the cluster address, which must be the same for both the Dispatcher machines. Both the primary and the backup machines must have their own nonforwarding address for remote administration and configuration.

Note that the primary and backup machines must be configured in the same way. The same processes (Executor, Manager and Advisors) must run in both machines.

In the following we show the configuration steps for the Dispatcher that we elected to use as the backup Dispatcher machine (the machine with hostname rs600030 and IP address 9.24.104.97). *The same steps* must be done on the other machine that will act as the primary Dispatcher machine (the machine with host name rs600023 and IP address 9.24.104.128).

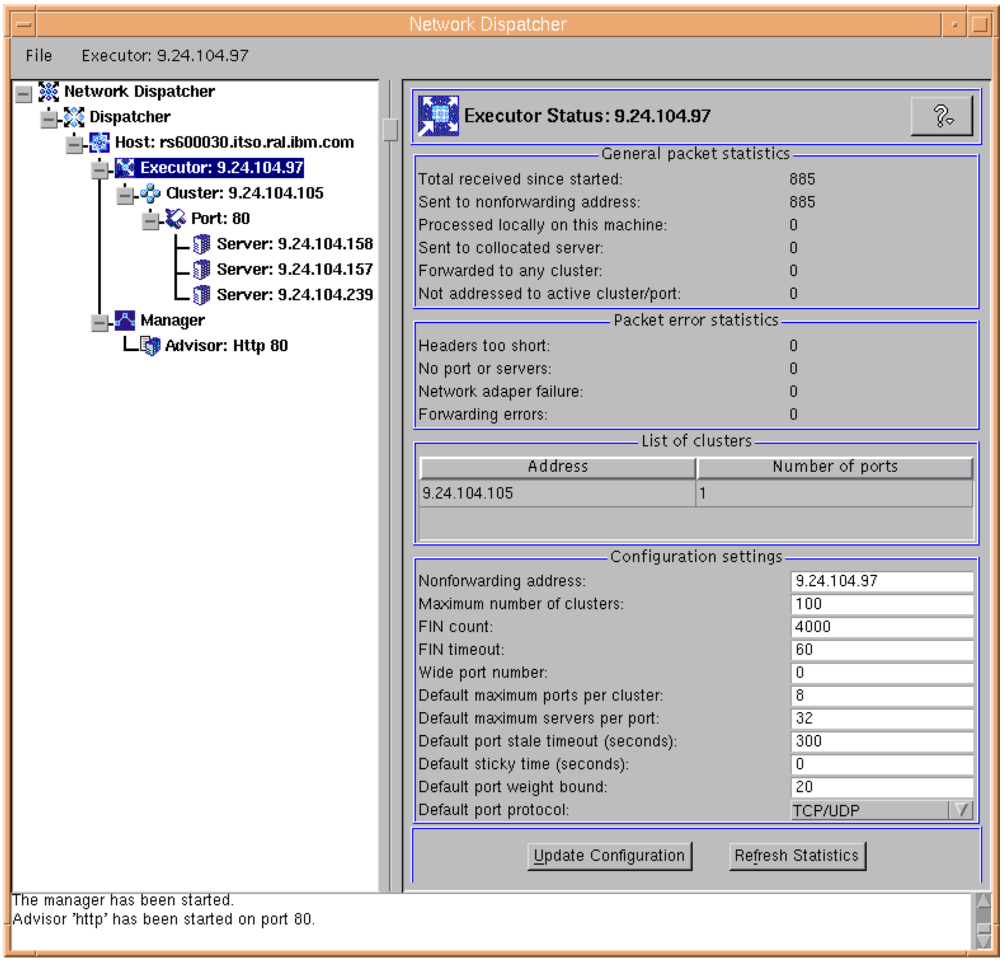
First we started the ndserver component. To do this, from a command line we entered the `ndserver` command.

We decided to perform all the configuration activities for the Dispatcher by using the GUI. The same steps could be performed by using the command line, as we see in 11.1, “Firewall High Availability Using the Dispatcher” on page 273. We launched the GUI by entering the `ndadmin` command.

Using the GUI, we started the Executor, with 9.24.104.97 as the nonforwarding address; then we defined a cluster for the HTTP service on TCP port 80. On port 80 we added the three Web servers. Of course it was necessary to configure the three Web servers to be nodes of the cluster. Then, on the Dispatcher machine, we started the Manager and launched the HTTP Advisor on port 80. We also set the proportions for the Manager and the smoothing index. For details on how to perform all these configuration steps, see 4.1, “Load Balancing Basic Scenario Using the Dispatcher” on page 81. You can add ISS to this configuration, using the ISS high availability support. In this case we recommend you see 4.2, “Load Balancing Scenario Using the Dispatcher and ISS” on page 137.

In the following figures we show the panels that summarize the configuration.

The first window shows the Executor Status panel for the backup Dispatcher machine. We accepted all the default values:



Executor Status: 9.24.104.97

General packet statistics

Total received since started:	885
Sent to nonforwarding address:	885
Processed locally on this machine:	0
Sent to collocated server:	0
Forwarded to any cluster:	0
Not addressed to active cluster/port:	0

Packet error statistics

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

List of clusters

Address	Number of ports
9.24.104.105	1

Configuration settings

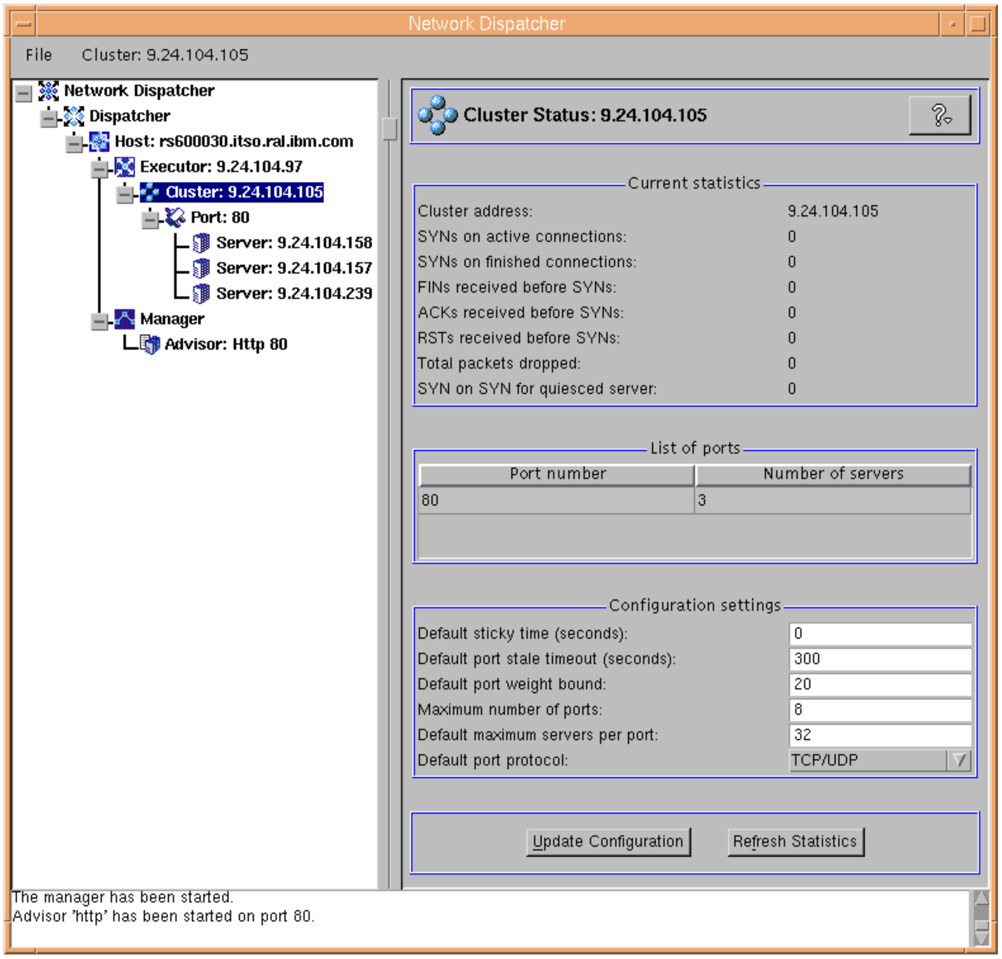
Nonforwarding address:	9.24.104.97
Maximum number of clusters:	100
FIN count:	4000
FIN timeout:	60
Wide port number:	0
Default maximum ports per cluster:	8
Default maximum servers per port:	32
Default port stale timeout (seconds):	300
Default sticky time (seconds):	0
Default port weight bound:	20
Default port protocol:	TCP/UDP

```
The manager has been started.
Advisor 'http' has been started on port 80.
```

Figure 134. Backup Dispatcher – Executor Status

The Executor Status window for the primary Dispatcher machine was very similar. The only difference was that the nonforwarding address for the primary Dispatcher showed as `9.24.104.128`.

The Cluster Status panel is shown in the next figure:



Current statistics

Cluster address:	9.24.104.105
SYNs on active connections:	0
SYNs on finished connections:	0
FINs received before SYNs:	0
ACKs received before SYNs:	0
RSTs received before SYNs:	0
Total packets dropped:	0
SYN on SYN for quiesced server:	0

List of ports

Port number	Number of servers
80	3

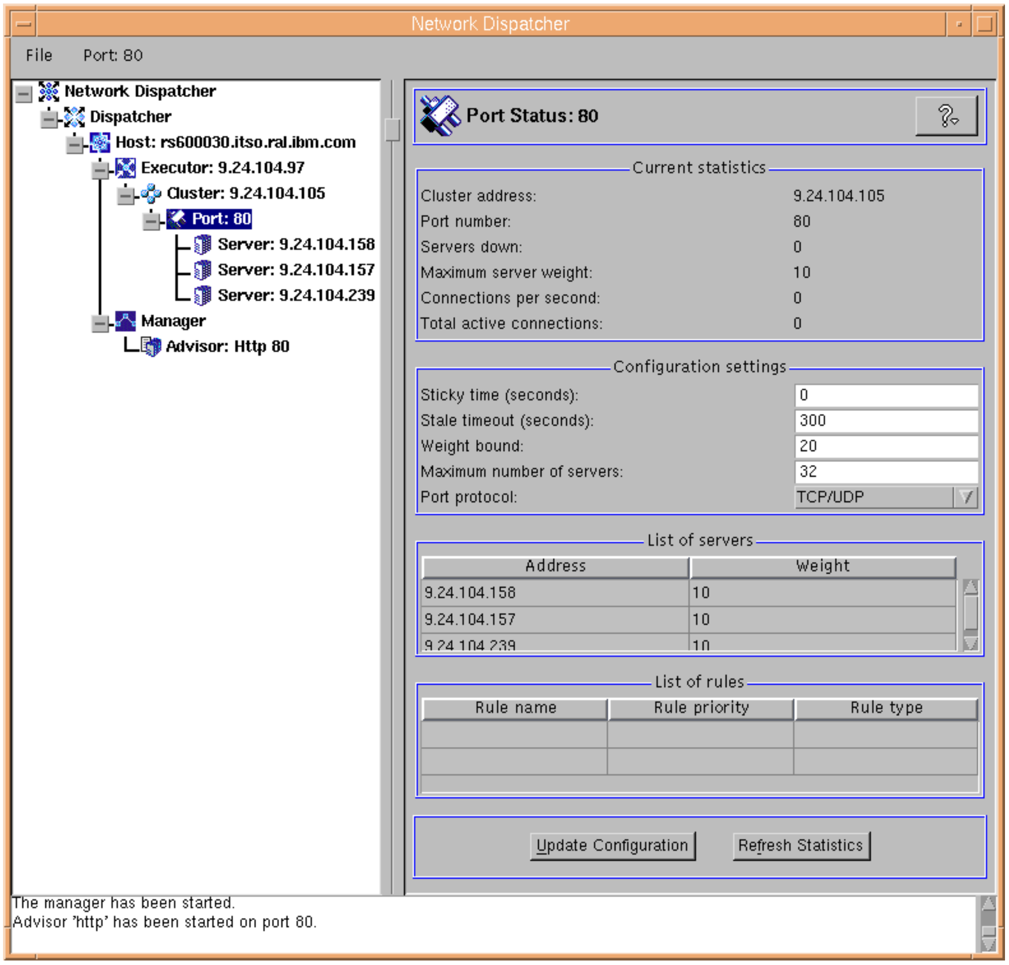
Configuration settings

Default sticky time (seconds):	0
Default port stale timeout (seconds):	300
Default port weight bound:	20
Maximum number of ports:	8
Default maximum servers per port:	32
Default port protocol:	TCP/UDP

```
The manager has been started.
Advisor 'http' has been started on port 80.
```

Figure 135. Backup Dispatcher – Cluster Status

In Figure 136 you will find the Port Status panel. The addresses of the three Web servers are visible in the figure:



Current statistics

Cluster address:	9.24.104.105
Port number:	80
Servers down:	0
Maximum server weight:	10
Connections per second:	0
Total active connections:	0

Configuration settings

Sticky time (seconds):	0
Stale timeout (seconds):	300
Weight bound:	20
Maximum number of servers:	32
Port protocol:	TCP/UDP

List of servers

Address	Weight
9.24.104.158	10
9.24.104.157	10
9.24.104.239	10

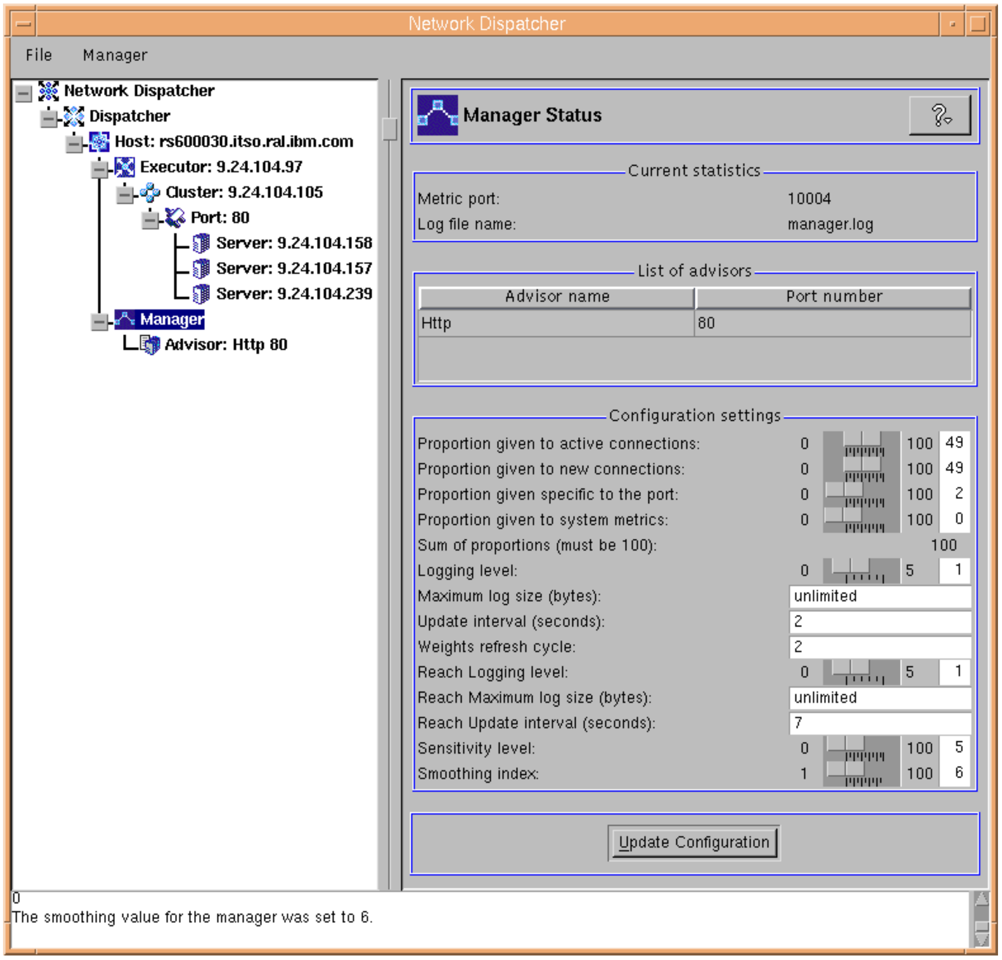
List of rules

Rule name	Rule priority	Rule type

```
The manager has been started.
Advisor 'http' has been started on port 80.
```

Figure 136. Backup Dispatcher – Port Status

Finally, we show the Manager Status panel. In this case, we did not accept the default values, but we changed the proportions to `49 49 2 0` and the smoothing index values to `6`, as shown in the following figure:



Current statistics

Metric port:	10004
Log file name:	manager.log

List of advisors

Advisor name	Port number
Http	80

Configuration settings

Proportion given to active connections:	0	100	49
Proportion given to new connections:	0	100	49
Proportion given specific to the port:	0	100	2
Proportion given to system metrics:	0	100	0
Sum of proportions (must be 100):			100
Logging level:	0	5	1
Maximum log size (bytes):			unlimited
Update interval (seconds):			2
Weights refresh cycle:			2
Reach Logging level:	0	5	1
Reach Maximum log size (bytes):			unlimited
Reach Update interval (seconds):			7
Sensitivity level:	0	100	5
Smoothing index:	1	100	6

```
0
The smoothing value for the manager was set to 6.
```

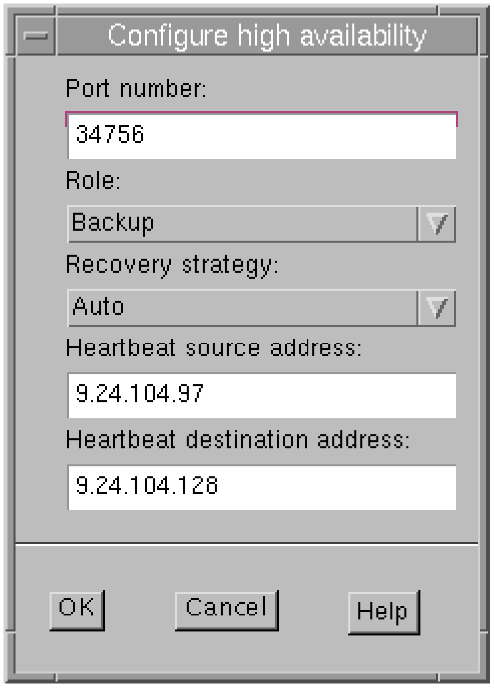
Figure 137. Backup Dispatcher – Manager Status

The reason we set the Proportions given to system metrics to `0` is that in this particular scenario we are not making use of ISS. In fact, our purpose here is to show how to configure Dispatcher high availability, and the use of ISS would not make any difference here. However, you can add ISS to the scenario, and configure ISS high availability as explained in 4.2, “Load Balancing Scenario Using the Dispatcher and ISS” on page 137.

6.1.3 Configuration Steps for High Availability

Now we start the configuration of the Dispatcher high-availability feature. This operation can easily be done through the GUI configuration panels.

On the backup Dispatcher machine, click **Host** and from the pop-up menu select **Add High Availability Backup**. You will get a dialog similar to the following, which requires you to specify several parameters:



Port number:
34756
Role:
Backup
Recovery strategy:
Auto
Heartbeat source address:
9.24.104.97
Heartbeat destination address:
9.24.104.128

Figure 138. Backup Dispatcher – Add Backup Panel

In the Port number field enter the port over which the two Dispatcher machines will communicate to exchange data for the synchronization of the Dispatcher information. Entering a correct port number is up to you. No default value will appear. We entered a random port number, `34756`, but we had to ensure that the selected port was unused on both the Dispatcher machines. To check for that, we issued on both machines the following command:

```
netstat -an | grep 34756
```

We received no output on both machines, which meant that port 34756 was not already in use.

We then elected this machine to have the backup role. This can be done by selecting **Backup** from the Role pop-up menu.

We also decided to set the recovery strategy to **Auto**. The recovery strategy determines which course of action is taken when a failed primary becomes active again. There are two kinds of recovery strategies:

1. Automatic

   The primary machine resumes routing packets as soon as it becomes operational again.

2. Manual

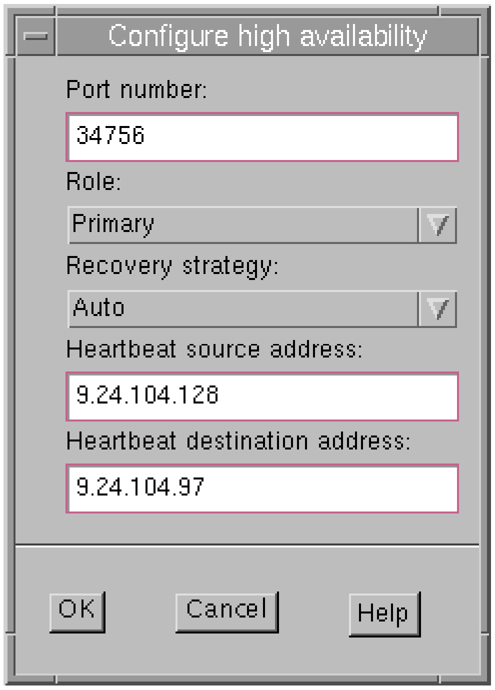
   The backup machine continues routing packets even after the primary machine becomes operational. Manual intervention is required to return the primary machine to active state and reset the backup machine to standby.

Notice that the Recovery strategy parameter must be set the same for both machines.

The manual recovery strategy allows you to force the routing of packets to a particular machine, using the `takeover` command, which is also available through the GUI. Manual recovery is useful when maintenance is being performed on the other machine. The automatic recovery strategy is designed for normal unattended operation. We explain the recovery strategy in more detail at the end of this section.

In Figure 138 we also added the heartbeat information, the mechanism through which the two Dispatchers continuously detect each other's state.

The following figure shows the values we entered for the other Dispatcher machine, the one we elected as primary Dispatcher:


Configure high availability

Port number: 34756

Role: Primary

Recovery strategy: Auto

Heartbeat source address: 9.24.104.128

Heartbeat destination address: 9.24.104.97

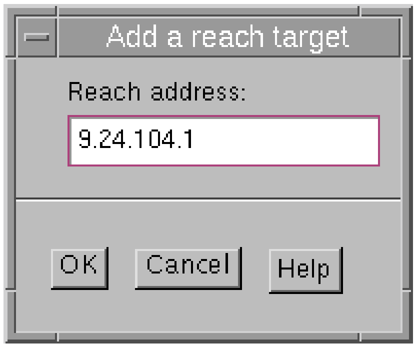
Figure 139. Primary Dispatcher - Add Backup Panel

Notice that in this case the heartbeat addresses are inverted as compared with the backup Dispatcher machine (see Figure 138). The Role field was set to **Primary** and the port number was the same we used on the backup Dispatcher, since we had already verified that it was not used by any other process.

The heartbeat is a mechanism of message exchange between the two Dispatchers to detect Dispatcher failure. Besides this mechanism of failure detection, there is another failure detection mechanism named *reachability criteria*. When you configure the Dispatcher, you provide a list of hosts that each of the Dispatchers should be able to reach by pinging in order to work correctly. You should use at least one host for each subnet your Dispatcher machine uses. The hosts could be routers, gateways, IP servers or other types of hosts. Host reachability is obtained by the reach Advisor, which pings the host. Switchover takes place if the heartbeat messages cannot go through or if the reachability criteria are met better by the standby Dispatcher than by the primary Dispatcher. To make the decision based on all the available information, the active Dispatcher sends the standby Dispatcher its reachability capabilities. The standby

Dispatcher then compares those capabilities with its own and decides whether to switch.

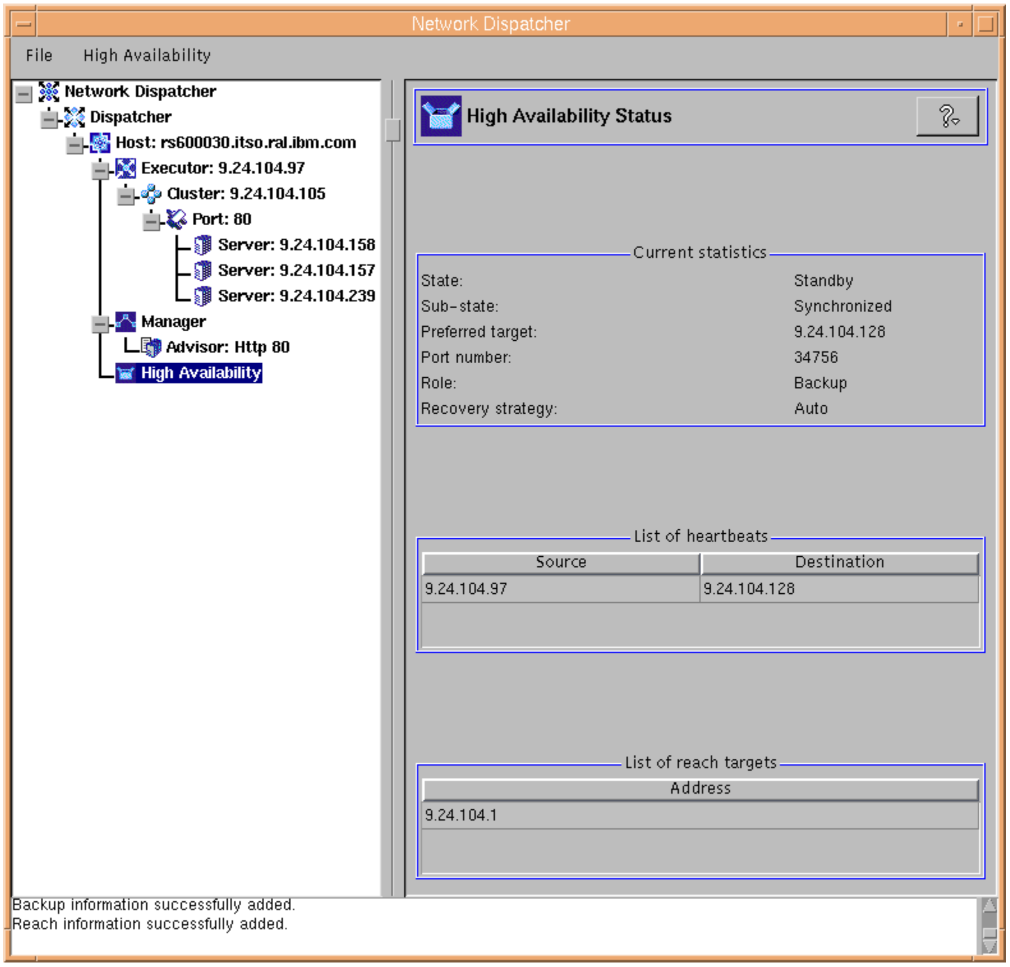
We added the information about the reachability criteria in both the Dispatcher machines in the same way. We right-clicked the **High Availability** item in the left panel, then selected **Add Reach Target** from the pop-up menu. We were presented with a dialog and in it we entered `9.24.104.1`, which was the IP address of the gateway machine in our intranet, as shown in the following figure:



Reach address: 9.24.104.1

Figure 140. Primary and Backup Dispatcher - Add a Reach Target

The following figure shows the High Availability status panel for the backup Dispatcher:



Current statistics

State:	Standby
Sub-state:	Synchronized
Preferred target:	9.24.104.128
Port number:	34756
Role:	Backup
Recovery strategy:	Auto

List of heartbeats

Source	Destination
9.24.104.97	9.24.104.128

List of reach targets

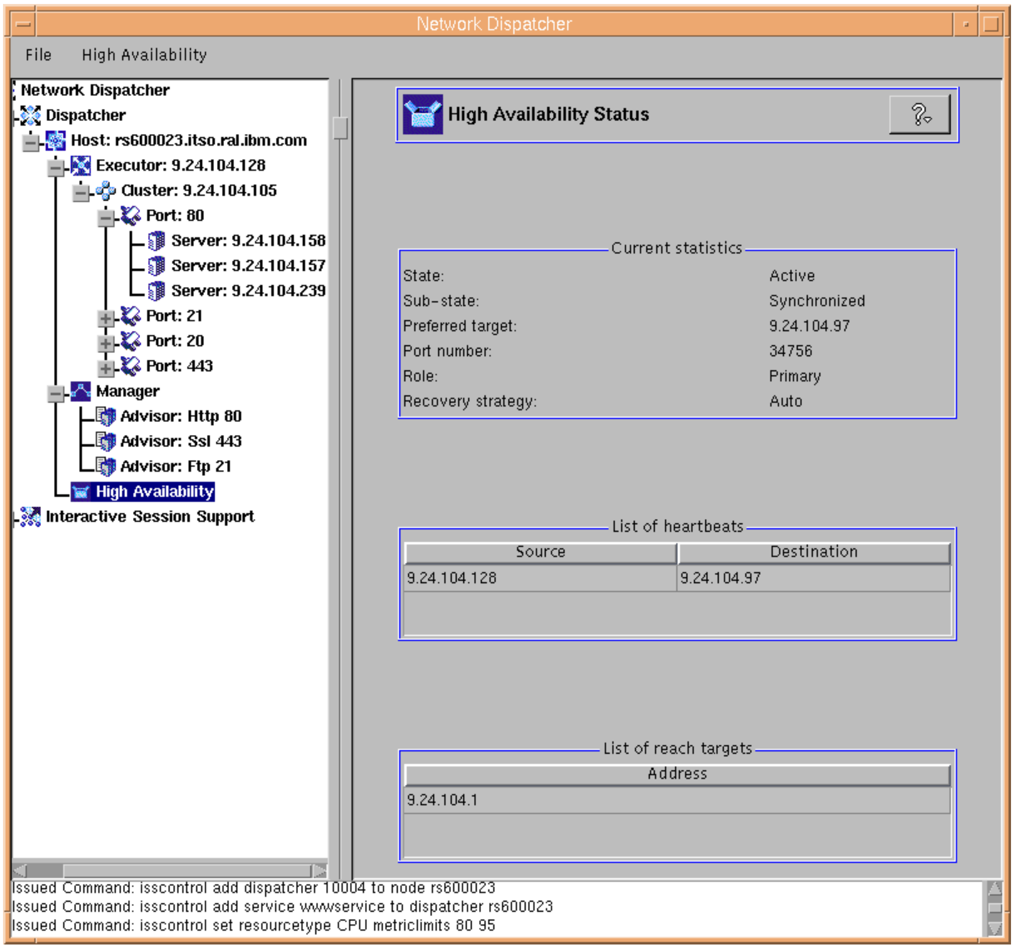
Address
9.24.104.1

```
Backup information successfully added.
Reach information successfully added.
```

Figure 141. Backup Dispatcher - High Availability Status

In the Current Statistics section of the High Availability Status panel, you can see that this machine (the backup) is in standby state and is synchronized with the other machine (the primary). This means that the heartbeat mechanism is working correctly.

The following figure illustrates the High Availability Status panel for the primary Dispatcher, which is in active state and is synchronized with the standby Dispatcher as well:



Current statistics

State:	Active
Sub-state:	Synchronized
Preferred target:	9.24.104.97
Port number:	34756
Role:	Primary
Recovery strategy:	Auto

List of heartbeats

Source	Destination
9.24.104.128	9.24.104.97

List of reach targets

Address
9.24.104.1

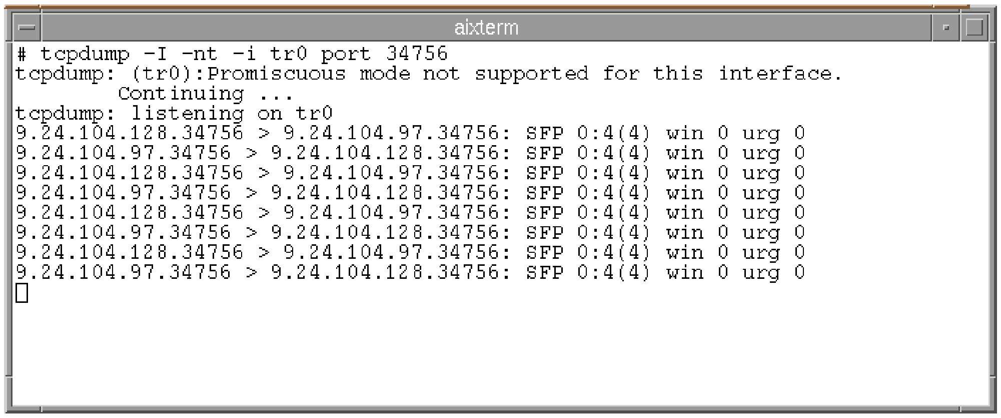
```
Issued Command: isscontrol add dispatcher 10004 to node rs600023
Issued Command: isscontrol add service wwwservice to dispatcher rs600023
Issued Command: isscontrol set resourcetype CPU metriclimits 80 95
```

Figure 142. Primary Dispatcher - High Availability Status

To further check whether the heartbeat mechanism was on, from a command line we entered the following command:

```
tcpdump -I -nt -i tr0 port 34756
```

We got the output shown in the following figure:



```
# tcpdump -I -nt -i tr0 port 34756
tcpdump: (tr0):Promiscuous mode not supported for this interface.
        Continuing ...
tcpdump: listening on tr0
9.24.104.128.34756 > 9.24.104.97.34756: SFP 0:4(4) win 0 urg 0
9.24.104.97.34756 > 9.24.104.128.34756: SFP 0:4(4) win 0 urg 0
9.24.104.128.34756 > 9.24.104.97.34756: SFP 0:4(4) win 0 urg 0
9.24.104.97.34756 > 9.24.104.128.34756: SFP 0:4(4) win 0 urg 0
9.24.104.128.34756 > 9.24.104.97.34756: SFP 0:4(4) win 0 urg 0
9.24.104.97.34756 > 9.24.104.128.34756: SFP 0:4(4) win 0 urg 0
9.24.104.128.34756 > 9.24.104.97.34756: SFP 0:4(4) win 0 urg 0
9.24.104.97.34756 > 9.24.104.128.34756: SFP 0:4(4) win 0 urg 0
```

Figure 143. Checking for Heartbeat

You can see that the two machines continuously exchange IP packets with each other.

At this point the two Dispatchers are not yet actually able to route packets. The reason is that in a high availability environment, preparations must be made to alias the cluster IP address to the correct interface (either an active network interface or the loopback interface) on both Dispatcher machines:

- On the machine currently in active state, you must alias the cluster address (or each cluster address, if you defined more than one cluster) to a network interface card.
- On the machine currently in standby state, you must alias the cluster address, (or each cluster address, if you defined more than one single cluster) to the loopback device lo0.
- On any Dispatcher machine (whatever its own current state), you should remove all defined aliases for the cluster address (both on network interface card and loopback device) when the Executor is stopped or before it is started for the first time. In this way no conflicts are possible.

The above-mentioned changes must be made automatically on each Dispatcher machine when its state changes. To ensure that this happens correctly, you need to use three script files, named goActive, goStandby and goInOp, that must be put into a specific directory, *installbase*/bin, where *installbase* varies by operating system. See Table 1 on page 69 for a list of the *installbase* locations. On the Windows NT platform, the script file must have the .cmd extension.

Sample versions of the script files are shipped with the product. The sample script files are located in the directory *installbase*/samples. Notice that the sample files shipped with the product carry a .sample extension, which must be removed if the sample script files are customized and used.

A description of the three script files follows, along with our modified version of the script files used in our AIX configuration:

- goActive script

  This script deletes cluster loopback aliases and adds cluster device aliases. It will be executed when a Dispatcher machine, either the primary or backup in a high-availability configuration, goes into active state and begins routing packets.

```
#
# goActive script
#
# Configure this script when using the high availability feature of
# Network Dispatcher.
#
# This script is executed when Network Dispatcher goes into the
# 'Active' state and begins routing packets.
#
# This script must be placed in Network Dispatcher's bin directory
# (by default this is /usr/lpp/nd/dispatcher) and it needs to
# have root execute permission.
#
# Modify NETWORK, CLUSTER, INTERFACE and NETMASK to match your environment.
#
# en0=first Ethernet adapter, tr0=first Token ring adapter, fi0=first FDDI adapter.
#
# NETMASK must be the netmask of your LAN.  It may be hexadecimal or octal notation.
#
NETWORK=9.24.104
INTERFACE=tr0
NETMASK=0xffffff00
#
 echo "Adding cluster alias(es)"
   for CLUSTER in 105; do
      ifconfig lo0 delete $NETWORK.$CLUSTER
      ifconfig $INTERFACE alias $NETWORK.$CLUSTER netmask $NETMASK
  done
```

Figure 144. goActive Script

- goStandby script

  This script deletes every cluster device alias and adds cluster loopback aliases. It will be executed when a Dispatcher machine in a high-availability configuration goes into standby state.

```
#!/bin/ksh
#
# goStandby script
#
# Configure this script when using the high availability feature of
# eNetwork Dispatcher.
#
# This script is executed when Network Dispatcher goes into the
# 'Standby' state.  Monitoring the health of the 'Active' machine
# but not routing packets.
#
# This script must be placed in Network Dispatcher's bin directory
# (by default this is /usr/lpp/nd/dispatcher) and it needs to
# have root execute permission.
#
# Modify NETWORK, CLUSTER, INTERFACE and NETMASK to match your environment.
#
# en0=first Ethernet adapter, tr0=first Token ring adapter, fi0=first FDDI adapter
#
# NETMASK must be the netmask of your LAN.  It may be hexadecimal or octal notation.
#
NETWORK=9.24.104
INTERFACE=tr0
NETMASK=0xffffff00
#
echo "Deleting the device aliases and adding the loopback aliases"
 for CLUSTER in 105 ; do
    ifconfig $INTERFACE delete $NETWORK.$CLUSTER
    ifconfig lo0 alias $NETWORK.$CLUSTER netmask $NETMASK
 done
```

Figure 145. goStandby Script

- goInOp script

  This script deletes all cluster device and loopback aliases. This script is executed when a Dispatcher Executor is stopped and before it is started for the first time.

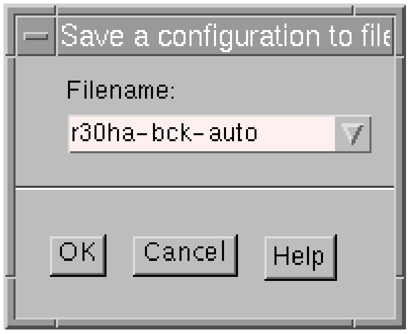
```
#!/bin/ksh
#
# goInOp script
#
# Configure this script when using the high availability feature of
# Network Dispatcher and optionally when using Network Dispatcher in a
# standalone environment.
#
# This script is executed when the Network Dispatcher executor is stopped
# (and before the executor is initially started).
#
# This script must be placed in Network Dispatcher's bin directory
# (by default this is /usr/lpp/nd/dispatcher) and it needs to
# have root execute permission.
#
# Modify NETWORK, CLUSTER and INTERFACE to match your environment.
#
# en0=first Ethernet adapter, tr0=first Token ring adapter, fi0=first FDDI adapter
#
NETWORK=9.24.104
INTERFACE=tr0
#
echo "Removing all loopback and device aliases"
for CLUSTER in 105; do
   ifconfig lo0 delete $NETWORK.$CLUSTER
   ifconfig $INTERFACE delete $NETWORK.$CLUSTER
 done
```

Figure 146. goInOp script

The script files used on Windows NT and Solaris are similar. In particular, you can consult “Script Modifications on Windows NT” on page 99 to understand how the script syntax differs on the Windows NT platform.

We made copies of the three files, customized them for our environment and placed them in the *installbase*/bin directory. However, just placing them in the right location was not enough to have the two Dispatcher machines be aware of the exact configuration and deliver a highly available environment. We needed to make the Dispatchers change their state in order to establish the correct configuration for the aliases on the two machines. In other words, it was necessary to stop and restart the Executor on both the machines.

First of all, we suggest that you save the current configuration. You can do this at any time by right-clicking the **Host** item in the left panel, then selecting **Save Configuration File As...** from the pop-up menu. When we did this on the backup Dispatcher machine, we got the following panel, where we typed `r30ha-bck-auto` as the name that we wanted to save the configuration file as:


Save a configuration to file

Filename:

r30ha-bck-auto

Figure 147. Save Configuration File – Backup Machine

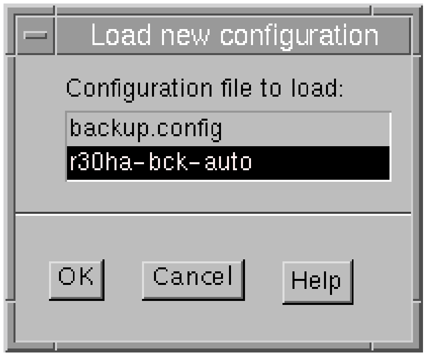
The name we entered for the configuration file in the primary Dispatcher machine was `r23ha-pri-auto`.

Then we stopped the Executor on both the machines by right-clicking the **Executor** item on the left pane of the GUI, then clicking **Stop Executor**. It also would have been possible to save the configuration when we stopped the Executor, since there is a dialog box asking if you would like to save the configuration before stopping the Executor.

When the Executor was stopped, the goInOp script was executed on both machines, but without changing anything, since we did not previously define any aliases.

Then we restarted the Executor on both machines by right-clicking the **Host** item on the left pane of the GUI, clicking **Load New Configuration**, choosing the configuration file name from the dialog that appears, and then clicking **OK**.

This is what we did on our backup Dispatcher machine to restore the `r30ha-bck-auto` configuration:


Load new configuration

Configuration file to load:

backup.config

r30ha-bck-auto

Figure 148. Load Configuration File - Backup Machine

Similarly, on the primary Dispatcher machine, we loaded the configuration file r23ha-pri-auto.

After restarting the Executors on both the machines, the primary Dispatcher machine switched its State field to read `Active` (and the goActive script was executed on that machine) while the State field in the backup Dispatcher machine changed to `Standby` (and the goStandby script was executed on that machine). Moreover, both the machines switched their Sub-state to `Synchronized`. Notice that after restarting the Executors, it might be necessary to click the **Refresh** buttons on the GUI to see the State and the Sub-state fields updated.

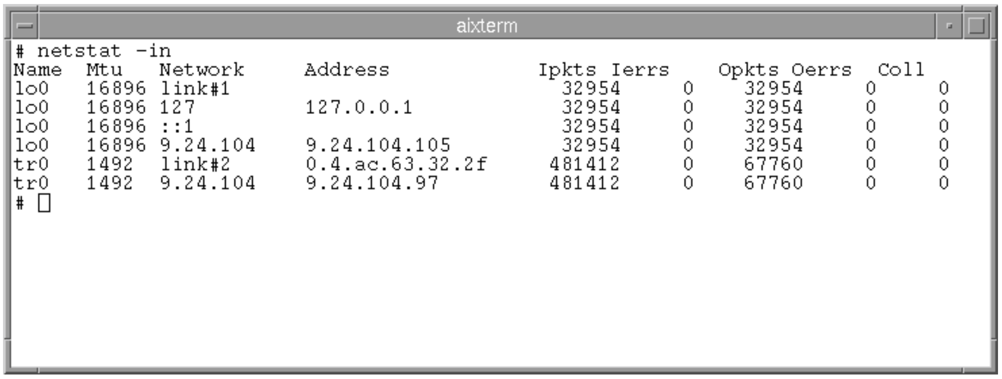
Executing the Scripts

Notice that it would not really be necessary to restart the Executor on the primary and backup machines. The scripts are executable, so you can just run the goActive script on the primary machine and the goStandby script on the backup machine, and the network interface cards and loopback devices would be configured accordingly.

We wanted to check whether the aliases were correctly added, so from a command line we issued the following command on both the machines:

```
netstat -in
```

We got the following output on the backup Dispatcher machine, which was in standby state:

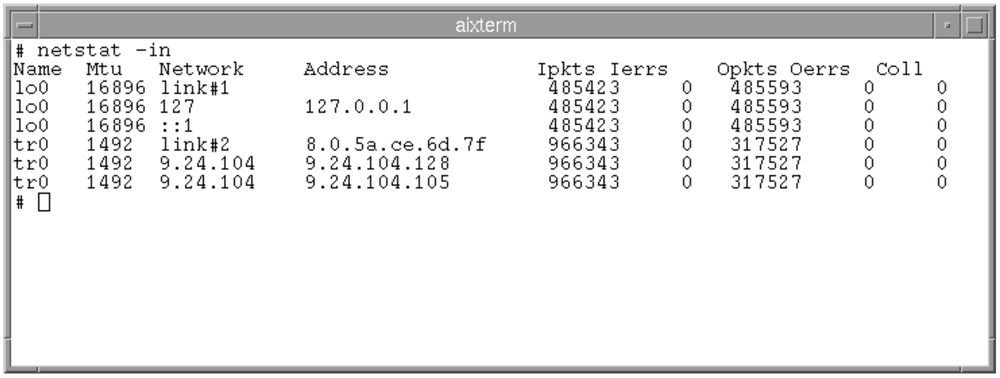


```
# netstat -in
Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
lo0   16896 link#1                           32954     0    32954     0     0
lo0   16896 127         127.0.0.1            32954     0    32954     0     0
lo0   16896 ::1                              32954     0    32954     0     0
lo0   16896 9.24.104    9.24.104.105         32954     0    32954     0     0
tr0   1492  link#2      0.4.ac.63.32.2f     481412     0    67760     0     0
tr0   1492  9.24.104    9.24.104.97         481412     0    67760     0     0
# 
```

Figure 149. Network Interface Status - Backup Machine

This output confirmed that the alias to the cluster address 9.24.104.105 was added on the loopback device lo0.

We got the following output on the primary Dispatcher machine, whose state was marked as active:



```
# netstat -in
Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
lo0   16896 link#1                          485423     0   485593     0     0
lo0   16896 127         127.0.0.1           485423     0   485593     0     0
lo0   16896 ::1                             485423     0   485593     0     0
tr0   1492  link#2      8.0.5a.ce.6d.7f     966343     0   317527     0     0
tr0   1492  9.24.104    9.24.104.128        966343     0   317527     0     0
tr0   1492  9.24.104    9.24.104.105        966343     0   317527     0     0
# 
```

Figure 150. Network Interface Status - Primary Machine

This output verified that the alias to the cluster address 9.24.104.105 had been added on the network adapter interface tr0.

Now all network and loopback interfaces are correctly configured. The primary Dispatcher is able to route IP packets to the servers belonging to the defined cluster, and the backup Dispatcher is able to take over in case the primary Dispatcher should have a failure.

6.1.4 Experimenting with High Availability

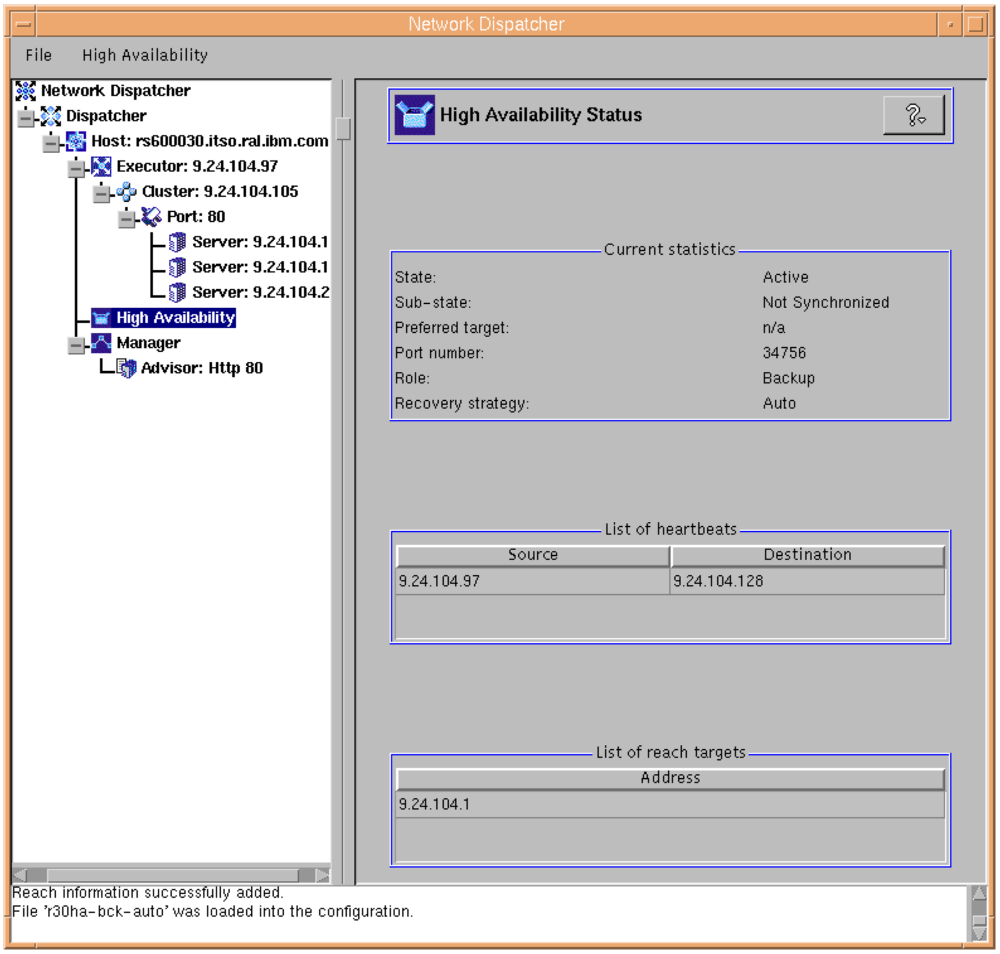
We wanted to perform a test to demonstrate that the automatic takeover worked correctly. Several tests could be done, such as disconnecting the network interface on the primary machine, shutting down the primary machine, or even turning it off.

We selected a solution that would not damage the stability of our test machines. While the primary machine was in active state, on that machine we issued the following command to shut down the network interface:

```
ifconfig tr0 down
```

This command effectively brought the token-ring interface down.

The backup machine detected that the primary machine was no longer available, and changed to active state, as shown in the following figure:



Current statistics

State:	Active
Sub-state:	Not Synchronized
Preferred target:	n/a
Port number:	34756
Role:	Backup
Recovery strategy:	Auto

List of heartbeats

Source	Destination
9.24.104.97	9.24.104.128

List of reach targets

Address
9.24.104.1

```
Reach information successfully added.
File 'r30ha-bck-auto' was loaded into the configuration.
```

Figure 151. Backup Dispatcher Automatically Takes Over

Of course, the Sub-state field appeared as `Not Synchronized`, since the primary machine was no longer able to exchange heartbeat messages.

In general, as soon as the primary machine fails, a gratuitous Address Protocol Request (ARP) is issued by the backup Dispatcher machine when it changes states (see “ARP Requests” on page 116). After that, the goActive script file is executed on the backup machine. Then we issued the following command on the primary machine, to make it operational again:

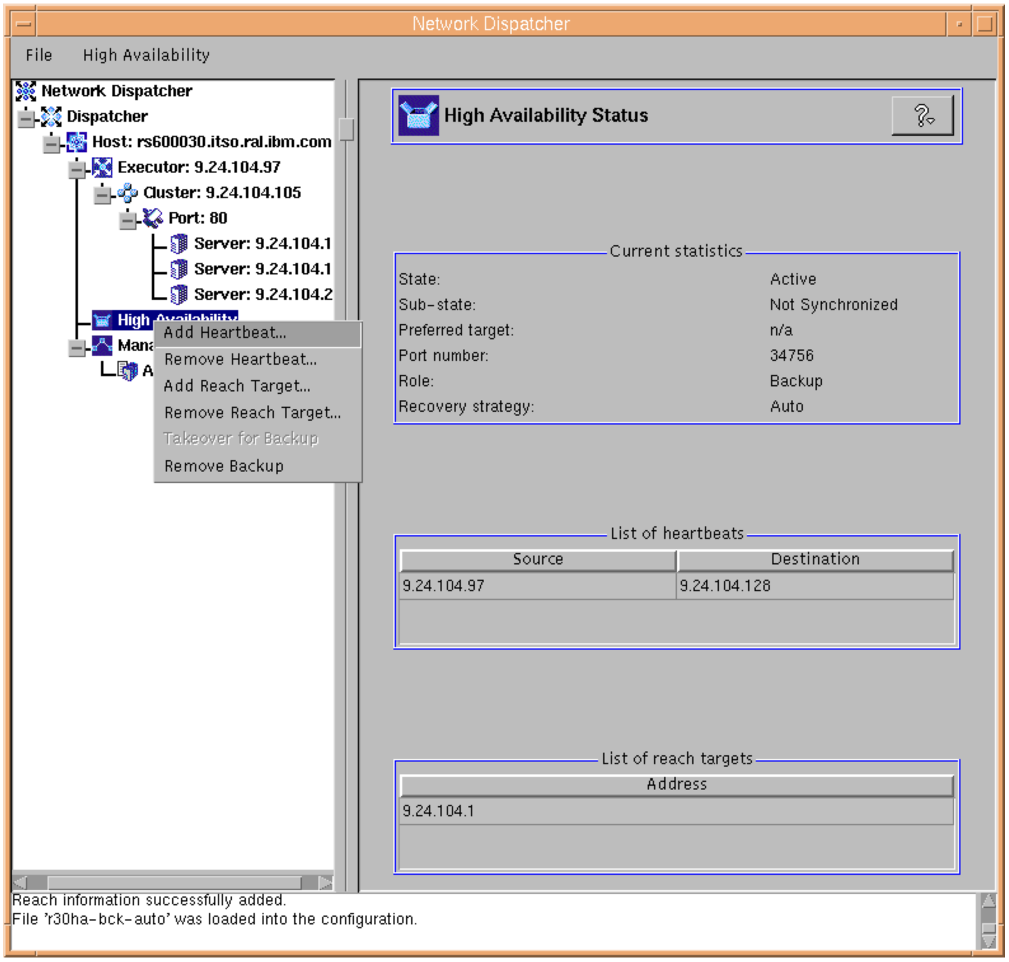
```
ifconfig tr0 up
```

Since we had set the recovery strategy to **Auto**, the primary machine went immediately to active state, while the backup machine changed its state back to standby, by executing the goStandby script.

6.1.5 Experimenting with the Recovery Strategy

Now we want to explain to you in more detail the recovery strategy. Notice that in the configuration we just described, where we set the Recovery strategy parameter to **Auto** in both Dispatcher machines (see Figure 138 on page 185), we found that the **Takeover for Backup** menu item had been disabled *on both machines*. This was how the menu appeared after right-clicking the **High**

Availability item. The following screen was captured on the backup Dispatcher machine:



Current statistics

State:	Active
Sub-state:	Not Synchronized
Preferred target:	n/a
Port number:	34756
Role:	Backup
Recovery strategy:	Auto

List of heartbeats

Source	Destination
9.24.104.97	9.24.104.128

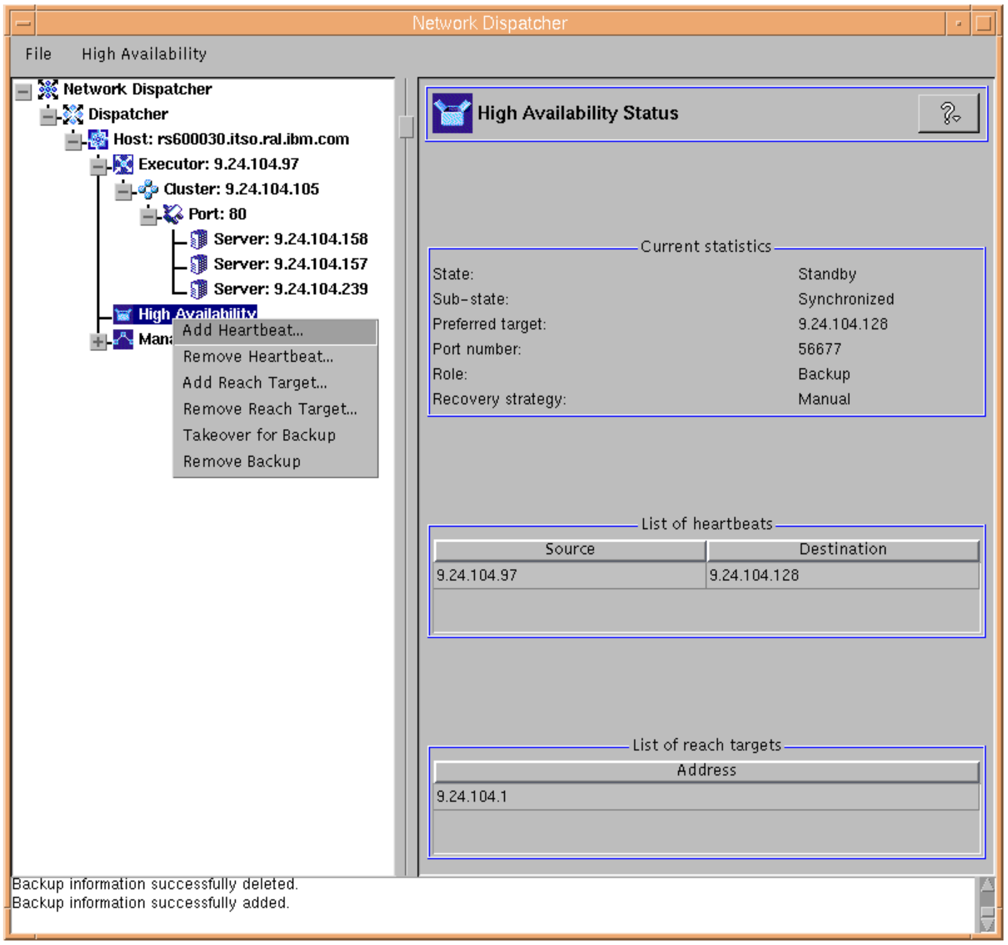
List of reach targets

Address
9.24.104.1

```
Reach information successfully added.
File 'r30ha-bck-auto' was loaded into the configuration.
```

Figure 152. Takeover Disabled when the Recovery Strategy Is Set to Auto

Then we tried another experiment. We defined two new configurations for the two Dispatcher machines, still in high availability mode, but setting the Recovery strategy field to **Manual**. After that we noticed that the Takeover for Backup item from the High Availability pop-up menu, corresponding to the `takeover` command, had been enabled on the backup Dispatcher machine, as shown in the following figure:



Current statistics

State:	Standby
Sub-state:	Synchronized
Preferred target:	9.24.104.128
Port number:	56677
Role:	Backup
Recovery strategy:	Manual

List of heartbeats

Source	Destination
9.24.104.97	9.24.104.128

List of reach targets

Address
9.24.104.1

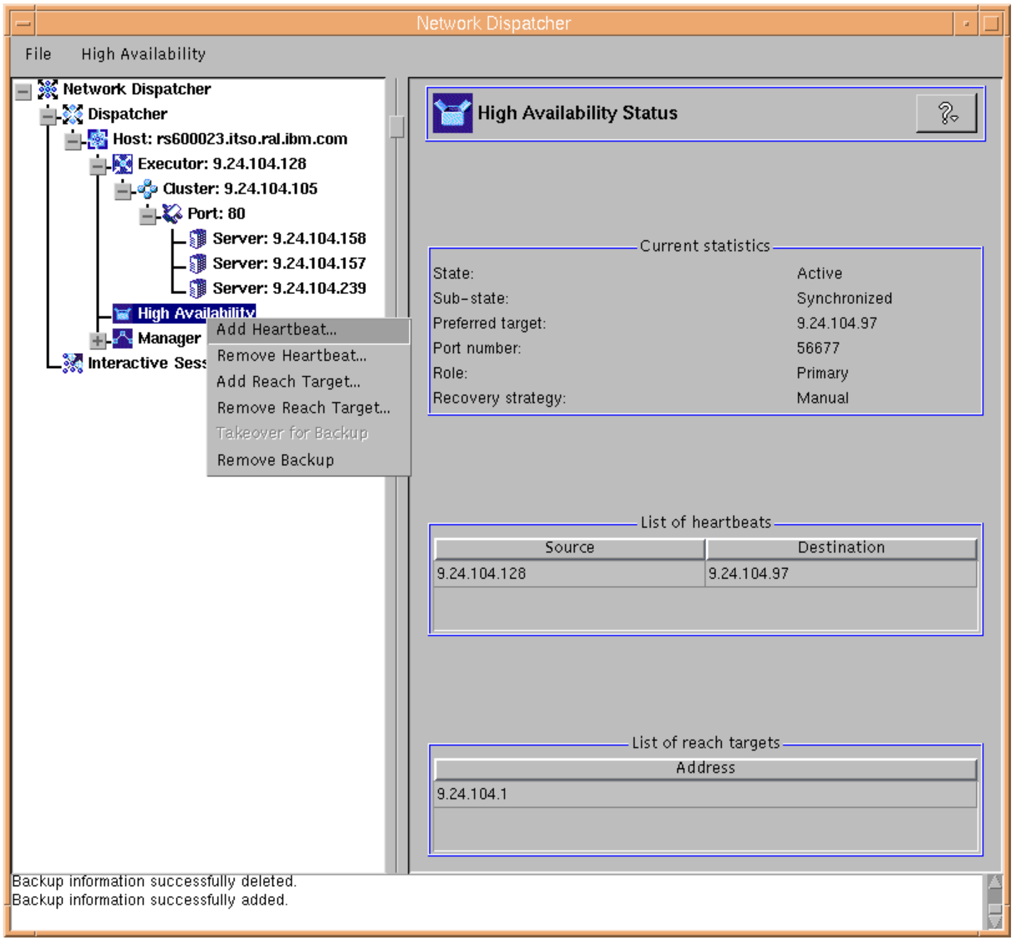
```
Backup information successfully deleted.
Backup information successfully added.
```

Figure 153. Takeover Enabled when the Recovery Strategy Is Set to Manual

In the above figure you can see other details:

- We defined another port number, 56677, for the heartbeat mechanism, after verifying that no other processes were using that specific port number.
- We saved the configuration in a new configuration file, which we named r30ha-bck-man.

The Takeover for Backup item in the High Availability pop-up menu was still disabled in the primary Dispatcher machine, since the state of this Dispatcher machine was already set to `Active`, as shown in the following figure:



Current statistics

State:	Active
Sub-state:	Synchronized
Preferred target:	9.24.104.97
Port number:	56677
Role:	Primary
Recovery strategy:	Manual

List of heartbeats

Source	Destination
9.24.104.128	9.24.104.97

List of reach targets

Address
9.24.104.1

```
Backup information successfully deleted.
Backup information successfully added.
```

Figure 154. Takeover Disabled on the Primary Dispatcher Machine

In the configuration of the primary Dispatcher machine, we made the same changes that were made on the backup machine configuration. We saved the new configuration on the primary Dispatcher machine in a configuration file that we named r23ha-pri-man.

Then, in the GUI of the backup Dispatcher machine we selected **Takeover for Backup**, as shown in Figure 153 on page 199, and we saw the following:

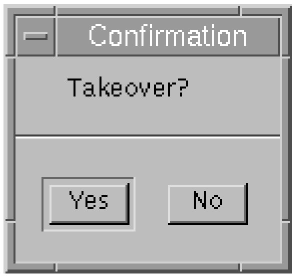
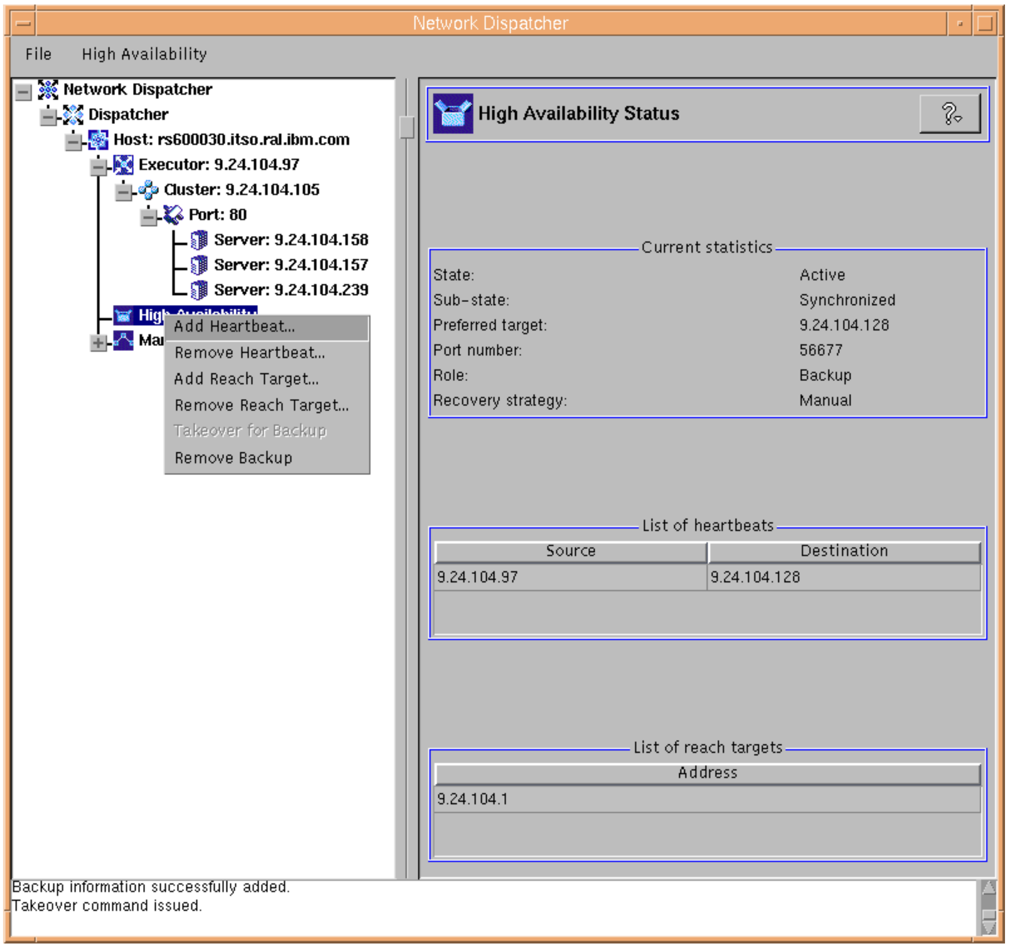


Figure 155. Takeover Confirmation Window

The reason for this further confirmation window is that the takeover operation is very important in a network environment where two Dispatchers are running in high availability mode. We selected **Yes** and the GUI for the backup Dispatcher machine appeared as shown in the following figure:



Current statistics

State:	Active
Sub-state:	Synchronized
Preferred target:	9.24.104.128
Port number:	56677
Role:	Backup
Recovery strategy:	Manual

List of heartbeats

Source	Destination
9.24.104.97	9.24.104.128

List of reach targets

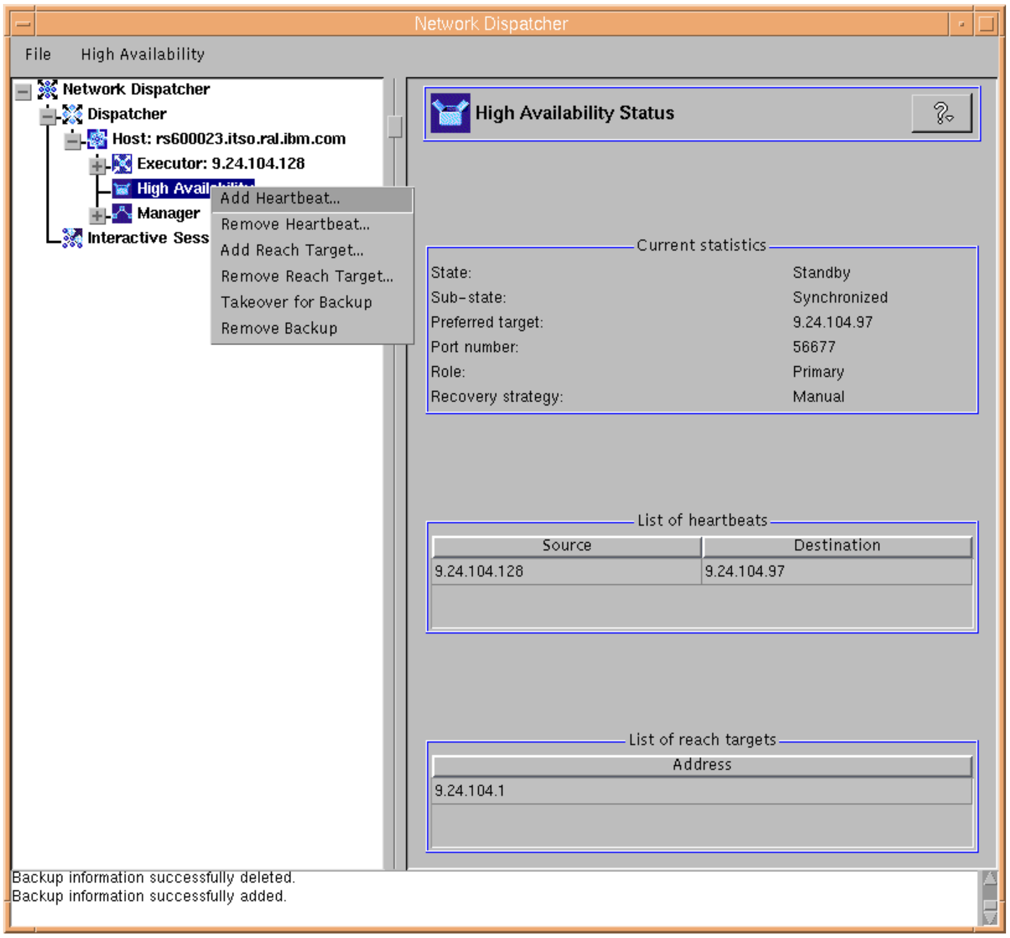
Address
9.24.104.1

```
Backup information successfully added.
Takeover command issued.
```

Figure 156. State Changes to Active on the Backup Machine after the Takeover

Notice that the Role of that Dispatcher machine remained set to `Backup`, but its state changed to read `Active` and, for this reason, the Takeover for Backup item was disabled again from the High Availability pop-up menu.

On the primary Dispatcher machine, the Dispatcher state appears set to `Standby`, and the Takeover for Backup item became enabled, as you can see in the following figure:



Current statistics

State:	Standby
Sub-state:	Synchronized
Preferred target:	9.24.104.97
Port number:	56677
Role:	Primary
Recovery strategy:	Manual

List of heartbeats

Source	Destination
9.24.104.128	9.24.104.97

List of reach targets

Address
9.24.104.1

```
Backup information successfully deleted.
Backup information successfully added.
```

Figure 157. Takeover Enabled on the Primary Dispatcher Machine

This confirms that even if the primary machine is operational when the recovery strategy is set to **Manual**, only a manual intervention can return the primary machine to active state and reset the backup machine to standby. Otherwise the backup machine continues routing packets.

Chapter 7. Dispatcher Colocation Option

The Dispatcher component of IBM SecureWay Network Dispatcher (ND) Version 2.1 can be installed on the same machine as one of the clustered TCP servers. This feature is known as *colocation option* and is currently available on the AIX and Solaris platforms.

One of the most important benefits derived from using colocation is that you can take advantage of having a highly available load-balanced TCP server environment with a minimum of investment. In addition to this, by having a Dispatcher in your configuration, you have a site that is highly scalable, simply by adding one or more TCP servers to your cluster. Moreover, again with a minimum of investment, you can easily implement colocation for both the primary and backup Dispatcher in a Dispatcher high-availability configuration (see Chapter 6, “ND High Availability Support” on page 177).

7.1 Dispatcher Colocation Scenario

In this scenario, we demonstrate setting up a load-balanced clustered Web server site with only two machines.

7.1.1 Network Environment

A summary of the hardware, software, and network configuration of the environment where we performed our colocation scenario is reported in the following table:

Table 11. Colocation Scenario – Hardware, Software, and Network Configuration

Workstation	Host Name	IP Address	Operating System	Service
IBM PC 365	wtr05212	9.24.104.218	Windows NT Server 4.0	Web client
IBM RS/6000 C20	tricia	9.29.124.179	AIX 4.3.1	Dispatcher Colocated Web server
	elvis	9.29.124.163		
IBM RS/6000 F50	fortytwo	9.29.124.183	AIX 4.3.2	Web server

Please note the following points:

- The load-balancing function was provided by the Dispatcher component of ND V2.1, running on tricia with nonforwarding address 9.29.124.179.
- The cluster was defined at address 9.29.124.163, corresponding to the host name elvis.
- The colocated Dispatcher machine and the other Web server machine were both connected to the same Ethernet local area network (LAN).
- Lotus Domino Go Webserver Version 4.6 was the Web server software running on tricia, whereas the Web server function on the other clustered Web server, fortytwo, was provided by IBM HTTP Server Version 1.3.3.
- The two machines tricia and fortytwo were both part of the vanisc.can.ibm.com domain.

- In this case, the Web Client machine was on a subnet different from the other two machines. Netscape Navigator 4.5 was the Web browser running on the Web client machine.

The following figure shows a logical representation of our colocated network environment.

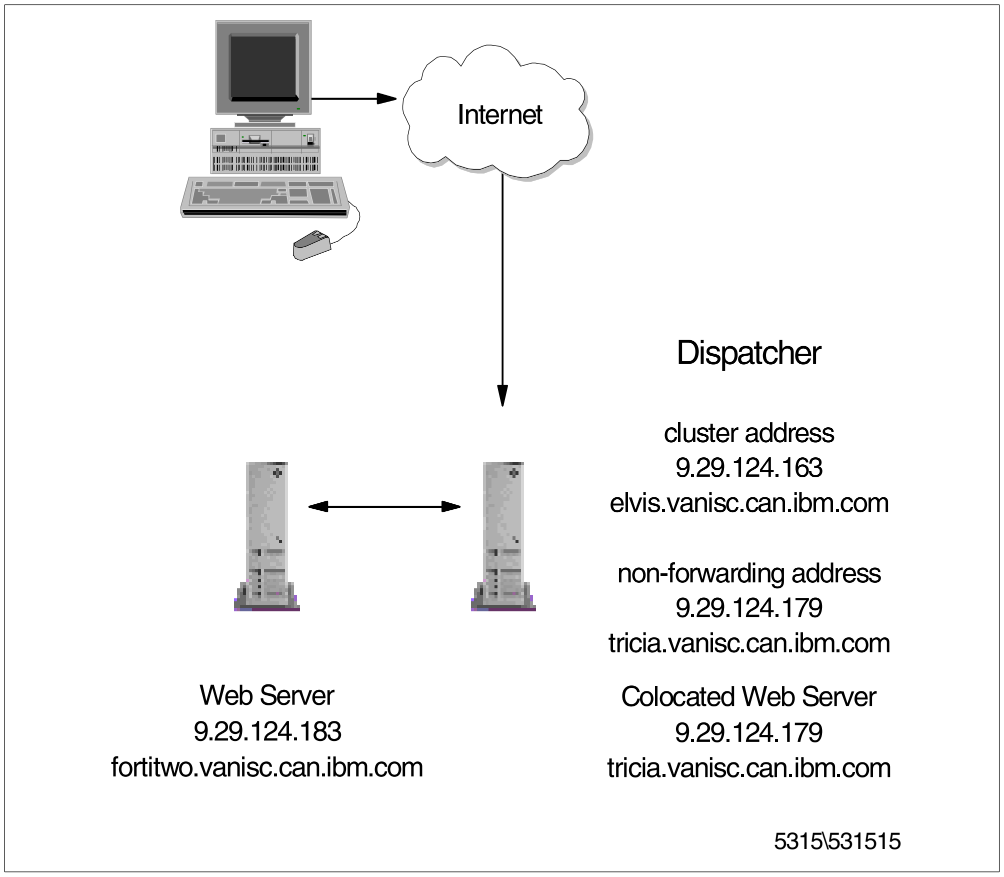


Figure 158. Dispatcher Colocation Scenario

The first step in configuring our colocation scenario was to start the ndserver process on tricia. To do this, from a command line we entered the `ndserver` command.

We decided to perform all the configuration activities for the Dispatcher by using the graphical user interface (GUI). The same steps could be performed by using the command line, as we see in 11.1, “Firewall High Availability Using the Dispatcher” on page 273. We launched the GUI by entering the `ndadmin` command.

Using the GUI, we started the Executor, with 9.29.124.179 as the nonforwarding address; then we defined a cluster for the HTTP service on the TCP port 80. On port 80 we added the two Web servers. For the colocated Web server we specified tricia’s nonforwarding address, since the Web server was running on the same machine as the Dispatcher. We then added the other Web server at IP address 9.29.124.183.

Of course it was necessary to start the Web servers on both of these machines. Then, on the Dispatcher machine, we started the Manager and launched the HTTP Advisor on port 80. We also set the proportions for the Manager and the smoothing index. For details on how to perform these configuration steps, see 4.1, “Load Balancing Basic Scenario Using the Dispatcher” on page 81.

7.1.2 Setting up the Aliases

In this section, we describe the alias configuration.

- On the Dispatcher machine, we added an alias for the cluster address on tricia’s en0 interface with the following command:

  ```
  ifconfig en0 alias 9.29.124.163
  ```

  See 4.1.3, “Cluster Address and Nonforwarding Address” on page 83 for alternative methods of creating this alias.

- On the TCP server machine fortytwo, we created an alias of the local cluster address on the loopback interface with the following command:

  ```
  ifconfig lo0 alias 9.29.124.163 netmask 255.0.0.0
  ```

  We then removed the duplicate route as follows:

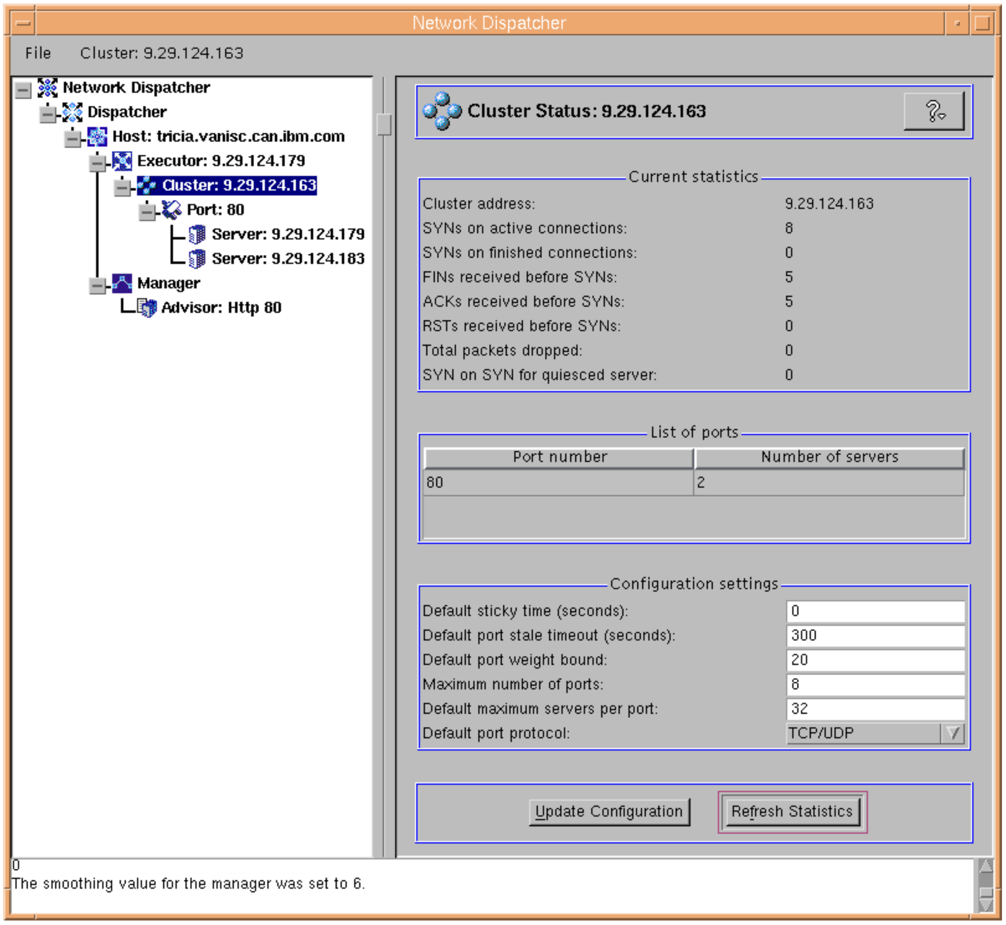
  ```
  route delete 9/8 9.29.124.163
  ```

  See 4.1.5, “TCP Servers Configuration” on page 107 for further details on aliasing the cluster to the loopback interface and removing duplicate routes.

Cluster Aliasing on a Colocated Machine

We did not create an alias for the cluster address on the loopback interface on the colocated server, even though there was a Web server running on that machine. The reason for this is that when the Dispatcher logic determines that the destination TCP server has the cluster’s nonforwarding address as its address, the packet is given back to the underlying operating system, not routed out the network interface.

The following window summarizes the configuration by showing the ND GUI Cluster Status information on the Dispatcher machine:



Current statistics

Cluster address:	9.29.124.163
SYNs on active connections:	8
SYNs on finished connections:	0
FINs received before SYNs:	5
ACKs received before SYNs:	5
RSTs received before SYNs:	0
Total packets dropped:	0
SYN on SYN for quiesced server:	0

List of ports

Port number	Number of servers
80	2

Configuration settings

Default sticky time (seconds):	0
Default port stale timeout (seconds):	300
Default port weight bound:	20
Maximum number of ports:	8
Default maximum servers per port:	32
Default port protocol:	TCP/UDP

```
0
The smoothing value for the manager was set to 6.
```

Figure 159. Colocated Cluster Status

7.1.3 Access From the Client's Perspective

To demonstrate that both Web servers responded to our client requests to access the clustered Web site, we wrote two simple similar HTML files. They were both named cjl.html, and placed in the document root directories of the two clustered Web servers.

Each of the files were of the following form, but each uniquely identified the server that it was located on. This is the file from the colocated Web server, having the host name tricia:

```
<html>
<title> Test Page from tricia</title>
<body>

This page comes from tricia.vanisc.can.ibm.com
with ip address 9.29.124.179
</body>
</html>
```

Figure 160. cjl.html on the Colocated Web Server tricia

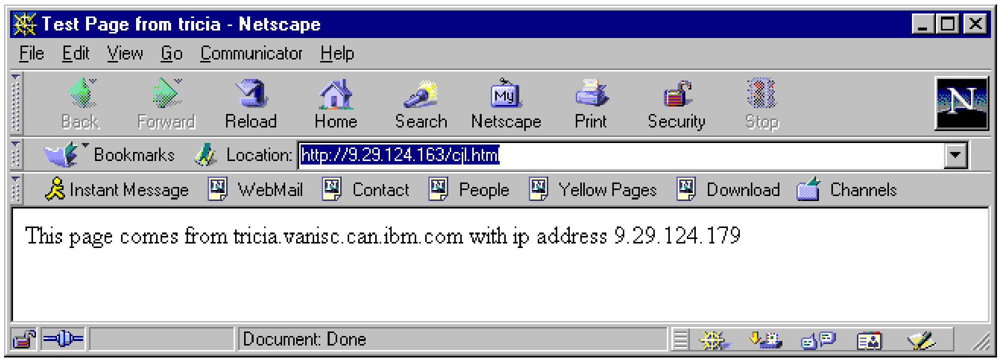
On the other Web server machine, fortytwo, the file cjl.html appeared as shown in the following figure:

```
<html>
<title> Test Page from fortytwo</title>
<body>

This page comes from fortytwo.vanisc.can.ibm.com
with ip address 9.29.124.183
</body>
</html>
```

Figure 161. cjl.html on the Web Server fortytwo

Similar to the scenario described in 4.1.7, “RoundRobin Load Balancing Scenario” on page 118, we then started our Web browser and de-activated its memory and disk caches. Then we requested the URL http://9.29.124.163/cjl.html several times and saw that the page was provided in turn by both of the Web server machines.

The following figure demonstrated that the page was effectively served by the colocated Web server, tricia, having IP address 9.29.124.179:

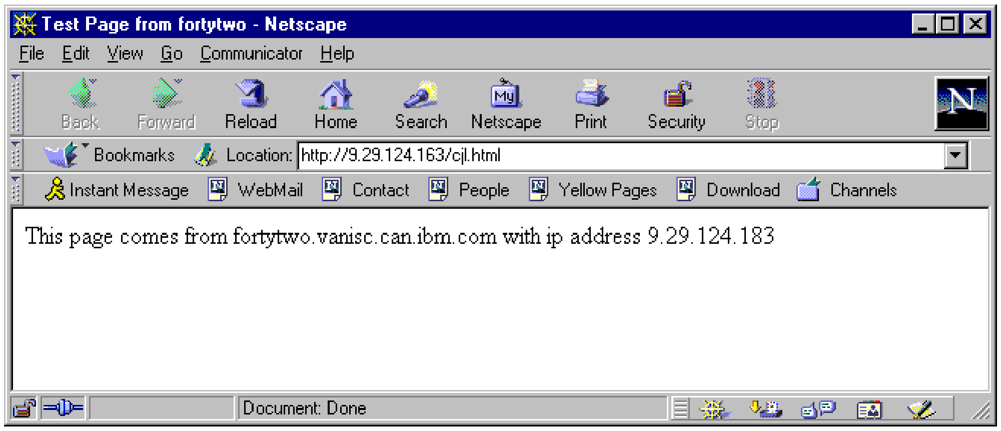

Test Page from tricia - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location:
Instant Message WebMail Contact People Yellow Pages Download Channels
Document: Done

http://9.29.124.163/cjl.html

This page comes from tricia.vanisc.can.ibm.com with ip address 9.29.124.179

Figure 162. HTML Page Served by the Colocated Web Server

The subsequent request for this page was responded to by the Web server running on fortytwo:


Test Page from fortytwo - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://9.29.124.163/cjl.html
Instant Message WebMail Contact People Yellow Pages Download Channels
Document: Done

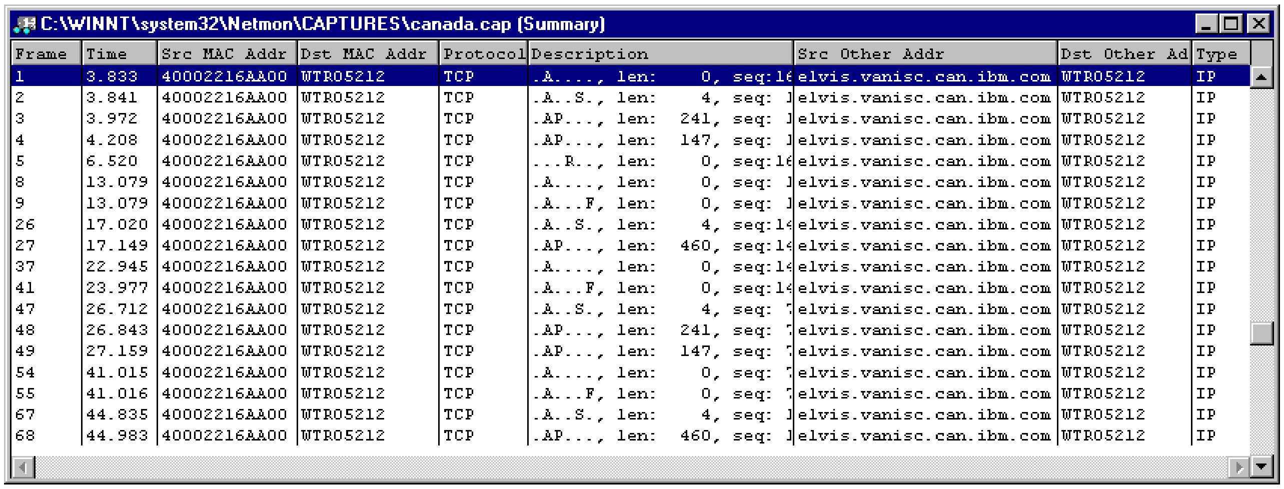
This page comes from fortytwo.vanisc.can.ibm.com with ip address 9.29.124.183

Figure 163. HTML Page Served by the Second Web Server in the Cluster

7.1.4 Packet Flow

In this section we demonstrate an important characteristic of the underlaying network frames transmitted to our local browser machine, in response to our request for this clustered Web server on a remote LAN. On the client machine we had Microsoft Network Monitor Version 4.00 for Windows NT Server running. This package can capture and display incoming packets being transmitted from other machines.

The host name of the client machine was wtr05212, as specified in Table 11 on page 203. While we requested the pages shown in Figure 162 and Figure 163, we monitored the incoming network packets. We got the following table from the Network Monitor:



Frame	Time	Src MAC Addr	Dst MAC Addr	Protocol	Description	Src Other Addr	Dst Other Ad	Type
1	3.833	40002216AA00	WTR05212	TCP	.A...., len: 0, seq:16	elvis.vanisc.can.ibm.com	WTR05212	IP
2	3.841	40002216AA00	WTR05212	TCP	.A..S., len: 4, seq: 1	elvis.vanisc.can.ibm.com	WTR05212	IP
3	3.972	40002216AA00	WTR05212	TCP	.AP..., len: 241, seq: 1	elvis.vanisc.can.ibm.com	WTR05212	IP
4	4.208	40002216AA00	WTR05212	TCP	.AP..., len: 147, seq: 1	elvis.vanisc.can.ibm.com	WTR05212	IP
5	6.520	40002216AA00	WTR05212	TCP	...R.., len: 0, seq:16	elvis.vanisc.can.ibm.com	WTR05212	IP
8	13.079	40002216AA00	WTR05212	TCP	.A...., len: 0, seq: 1	elvis.vanisc.can.ibm.com	WTR05212	IP
9	13.079	40002216AA00	WTR05212	TCP	.A...F, len: 0, seq: 1	elvis.vanisc.can.ibm.com	WTR05212	IP
26	17.020	40002216AA00	WTR05212	TCP	.A..S., len: 4, seq:14	elvis.vanisc.can.ibm.com	WTR05212	IP
27	17.149	40002216AA00	WTR05212	TCP	.AP..., len: 460, seq:14	elvis.vanisc.can.ibm.com	WTR05212	IP
37	22.945	40002216AA00	WTR05212	TCP	.A...., len: 0, seq:14	elvis.vanisc.can.ibm.com	WTR05212	IP
41	23.977	40002216AA00	WTR05212	TCP	.A...F, len: 0, seq:14	elvis.vanisc.can.ibm.com	WTR05212	IP
47	26.712	40002216AA00	WTR05212	TCP	.A..S., len: 4, seq: 7	elvis.vanisc.can.ibm.com	WTR05212	IP
48	26.843	40002216AA00	WTR05212	TCP	.AP..., len: 241, seq: 7	elvis.vanisc.can.ibm.com	WTR05212	IP
49	27.159	40002216AA00	WTR05212	TCP	.AP..., len: 147, seq: 7	elvis.vanisc.can.ibm.com	WTR05212	IP
54	41.015	40002216AA00	WTR05212	TCP	.A...., len: 0, seq: 7	elvis.vanisc.can.ibm.com	WTR05212	IP
55	41.016	40002216AA00	WTR05212	TCP	.A...F, len: 0, seq: 7	elvis.vanisc.can.ibm.com	WTR05212	IP
67	44.835	40002216AA00	WTR05212	TCP	.A..S., len: 4, seq: 1	elvis.vanisc.can.ibm.com	WTR05212	IP
68	44.983	40002216AA00	WTR05212	TCP	.AP..., len: 460, seq: 1	elvis.vanisc.can.ibm.com	WTR05212	IP

Figure 164. Network Monitor Output Showing Mac Addresses of the Frames Source

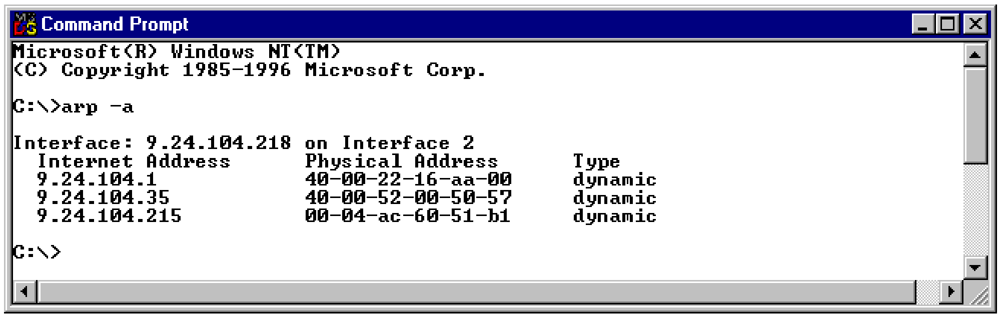
If you look closely at the table displayed by the Network Monitor, you can see that the numbers in the frame column are not consecutive. In fact, for clarity, we filtered out the frames that were being sent to our Windows NT machine from hosts that were not related to this demonstration.

In this table, by looking at the Src Other Addr (Source Other Address) columns, you can see that the client machine cannot determine which server has honored the HTTP requests, since the source from which the response arrived is always the cluster elvis.vanisc.can.ibm.com. We had already demonstrated this result in 4.1.8, “Analyzing the Flow with a Network Monitoring Tool” on page 123. The reason for this feature was explained in 4.1.6, “How the Dispatcher Works – The Flow of the IP Packets” on page 114.

However, this time there is a new particular that did not show up in 4.1.8, “Analyzing the Flow with a Network Monitoring Tool” on page 123. The Src MAC Addr (Source MAC Address) column shows us that all of the frames from elvis.vanisc.can.ibm.com came to us from the same machine having a Media Access Control (MAC) address 40002216AA00 (see “MAC Address” on page 115). We entered the following command:

```
arp -a
```

With the command above, we could determine that this MAC address belonged to the network adapter of the router on our local LAN, which has the IP address 9.24.104.1, as shown in the following figure:



```
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>arp -a

Interface: 9.24.104.218 on Interface 2
  Internet Address      Physical Address      Type
  9.24.104.1            40-00-22-16-aa-00     dynamic
  9.24.104.35           40-00-52-00-50-57     dynamic
  9.24.104.215          00-04-ac-60-51-b1     dynamic

C:\>
```

Figure 165. arp -a Associates the MAC Address to an IP Address

Because the MAC addresses of the TCP servers cannot be seen when the Web servers are located on a different LAN, the TCP architecture of a clustered Web site remains hidden to remote clients. This is a further security feature offered by the ND Dispatcher component.

In the scenario described in 4.1.8, “Analyzing the Flow with a Network Monitoring Tool” on page 123, the MAC addresses of the Web servers could be discovered with a network monitoring tool because the client machine and the Web server machines all belonged to the same LAN.

Chapter 8. Wide Area Network Dispatcher Support

IBM SecureWay Network Dispatcher (ND) Version 2.1, the load balancing component of IBM WebSphere Performance Pack Version 2, adds a wide area network (WAN) Dispatcher enhancement that offers support for *remote servers*. A remote server consists of a remote Dispatcher machine and its locally attached servers. A client's packet can now go from the Internet to a Dispatcher machine, from there to a geographically remote Dispatcher machine, then to one of its locally attached servers, and from the server directly back to the Internet and the client.

This allows one cluster address to support all worldwide client requests while distributing the load to servers around the world.

The Dispatcher machine initially receiving the packet can still have the local servers attached to it and can distribute the load between its local servers as well as the remote servers.

Notice that wide area support is currently available only for the Dispatcher component of ND.

8.1 Wide Area Network Dispatcher Scenario

In this section we describe the procedure we followed to configure our WAN Dispatcher support test environment.

8.1.1 Network Architecture

A summary of the hardware, software and network configuration of the environment where we performed our test is reported in the following two tables. Table 12 represents the local Dispatcher environment and Table 13 represents the remote Dispatcher environment. We linked the two Dispatchers together with the WAN Dispatcher support tools provided by the Dispatcher component of ND Version 2.1.

Table 12. Local Network Dispatcher Environment – Hardware, Software, and Network Configuration

Workstation	Host Name	IP Address	Operating System	Service
IBM PC 365	wtr05212	9.24.104.218	Windows NT Server 4.0	Web Client
IBM RS/6000 43P	rs600023	9.24.104.128	AIX 4.3.1	Entry Dispatcher
	clusterend	9.24.104.105		
IBM RS/6000 43P	aixncf157	9.24.104.157	AIX 4.3.1	Web Server
IBM PC 365	wtr05193	9.24.104.239	Windows NT Server 4.0	Web Server

As you can see, the local component of our WAN Dispatcher support scenario is very similar to the one that was described in 4.1, "Load Balancing Basic Scenario Using the Dispatcher" on page 81. In particular, notice that:

- The *entry Dispatcher* machine is the Dispatcher machine in the local area network (LAN) to which all client machines directly point. This machine has the nonforwarding address 9.24.104.128, corresponding with the host name rs600023, and cluster address 9.24.104.105, corresponding to the host name clusterend.

- All the above machines had a token-ring interface and were connected to the same LAN.
- Both Web servers were IBM HTTP Server Version 1.3.3.
- The Web client was running Netscape Navigator 4.5.
- The domain name for all the machines was itso.ral.ibm.com.

Table 13. Remote Network Dispatcher Environment – Hardware, Software, and Network Configuration

Workstation	Host Name	IP Address	Operating System	Service
IBM RS/6000 C20	tricia	9.29.124.179	AIX 4.3.1	Remote Dispatcher Colocated Web server
IBM RS/6000 F50	fortytwo	9.29.124.183	AIX 4.3.2	Web server

Notice that:

- Both of the above machines were located together in the same LAN and each had one Ethernet network interface card.
- The Web server on tricia was Lotus Domino Go Webserver Version 4.6.
- The Web server on fortytwo was IBM HTTP Server Version 1.3.3.
- The Dispatcher server was tricia with nonforwarding address 9.29.124.179.
- Both machines were in the vanisc.can.ibm.com domain.

Dispatcher Operating Systems when Using Wide Area Support

You can see that we installed both the local and remote Dispatcher machines on AIX 4.3.1. The most important thing you should remember when you make your plans for WAN Dispatcher support is that the machines implementing the Dispatcher function are currently required to run the same operating system. You should not configure the WAN Dispatcher support feature if, for example, one Dispatcher machine runs Windows NT and the other AIX or Solaris.

The following figure offers a graphical representation of the WAN Dispatcher support scenario that we implemented:

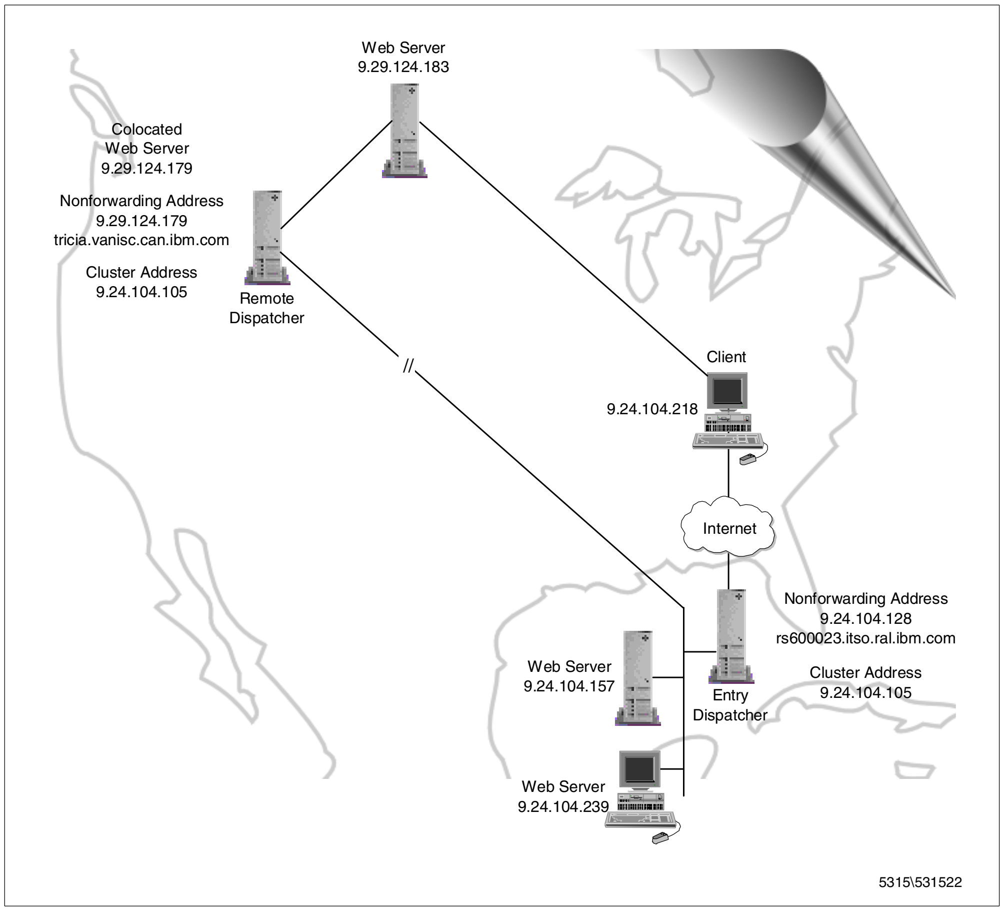


Figure 166. Wide Area Scenario Configuration

The figure above shows that in our environment, the entry Dispatcher and its local cluster were located in the United States, whereas the remote Dispatcher and its remote cluster were located in Canada. This reflects the fact that the WAN Dispatcher support allows one cluster address to support all worldwide client requests while distributing the load to servers around the world.

8.1.2 Local Dispatcher Configuration Steps

In this section, we show the configuration steps for the Dispatcher that we elected to use as the local Dispatcher machine (the machine with host name rs600023 and IP address 9.24.104.128).

For this scenario, we used the existing cluster address and cluster name that was used in the scenario described in 4.1, “Load Balancing Basic Scenario Using the Dispatcher” on page 81. We performed the following steps to set up the basic cluster environment on our local Dispatcher server, rs600023.

1. First we started the ndserver component. To do this, from a command line we entered the `ndserver` command.
2. We decided to perform all the configuration activities for the Dispatcher by using the ND graphical user interface (GUI). The same steps could be performed by using the command line, as we see in 11.1, “Firewall High

Availability Using the Dispatcher" on page 273. We launched the GUI by entering the `ndadmin` command.

3. Then, we defined the cluster. Using the GUI, we started the Executor, with 9.24.104.128 as the nonforwarding address; then we defined a cluster for the HTTP service on the TCP port 80. On the port 80, we added the two local Web servers, having IP addresses 9.24.104.157 and 9.24.104.239 respectively. Of course it was necessary to start the Web servers on both of the server machines. Then we started the Manager.
4. Following this, we aliased the cluster address 9.24.104.105 to the network interface card on the Dispatcher machine and to the loopback interface on both of the Web Server machines. For details on how to perform these aliasing steps, see 4.1.4.8, "Methods of Aliasing the Cluster to the Network Interface" on page 94 and 4.1.5, "TCP Servers Configuration" on page 107.

The next steps are unique to the WAN Dispatcher support environment:

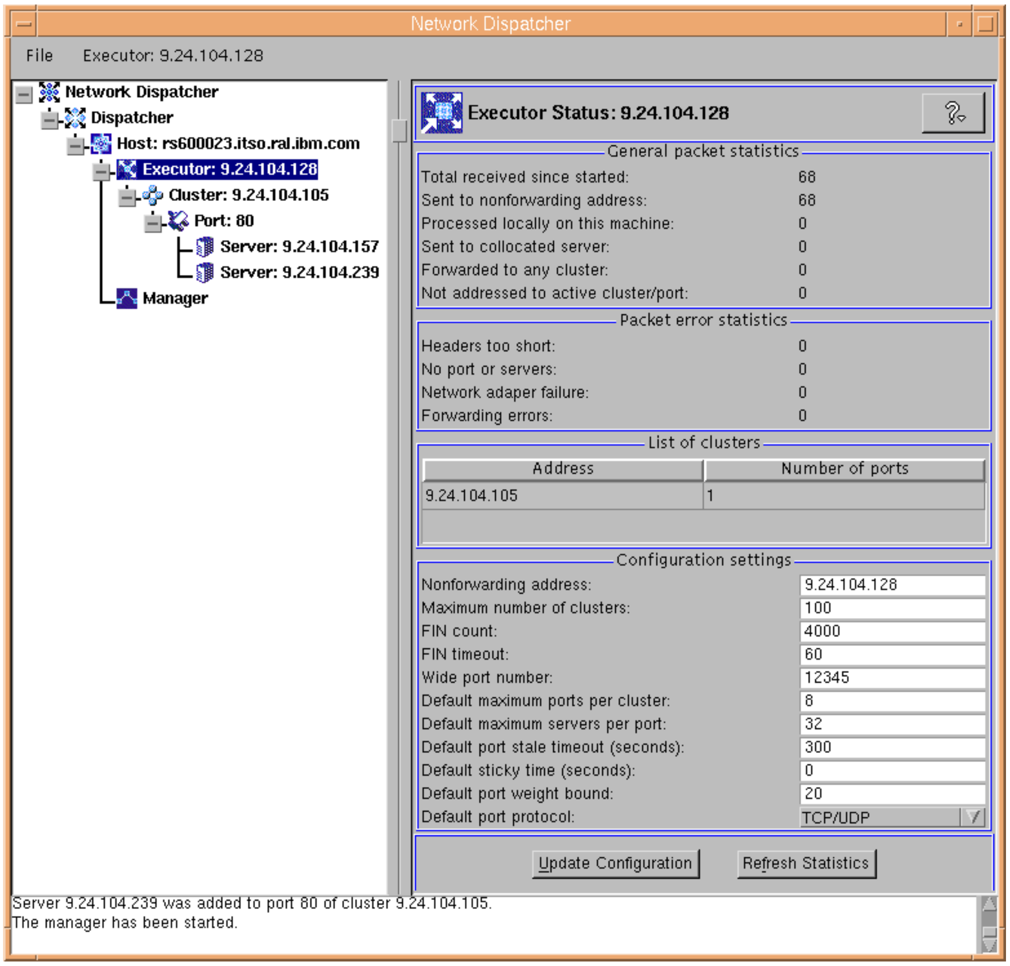
1. We selected a port that would be used for WAN communication between the two Dispatcher machines. The same port number is used on both Dispatchers. We arbitrarily selected port 12345 after confirming with this command that the port was not already in use on either of the machines.

   ```
   netstat -a | grep 12345
   ```

   No output from this command signified that the port was not already in use. We set the WAN port with this command:

   ```
   ndcontrol executor set wideportnumber 12345
   ```

   We could also have entered this port number in the Wide port number field on the Executor Status pane. This is the fifth field in the Configuration Settings section of the window as seen in the following figure:



General packet statistics

Total received since started:	68
Sent to nonforwarding address:	68
Processed locally on this machine:	0
Sent to collocated server:	0
Forwarded to any cluster:	0
Not addressed to active cluster/port:	0

Packet error statistics

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

List of clusters

Address	Number of ports
9.24.104.105	1

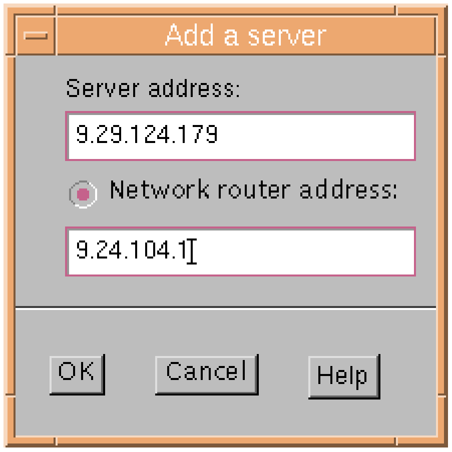
Configuration settings

Nonforwarding address:	9.24.104.128
Maximum number of clusters:	100
FIN count:	4000
FIN timeout:	60
Wide port number:	12345
Default maximum ports per cluster:	8
Default maximum servers per port:	32
Default port stale timeout (seconds):	300
Default sticky time (seconds):	0
Default port weight bound:	20
Default port protocol:	TCP/UDP

```
Server 9.24.104.239 was added to port 80 of cluster 9.24.104.105.
The manager has been started.
```

Figure 167. Local Dispatcher Executor Status Showing Wide Area Port Number

2. The next step is to add the remote Dispatcher as a third server to port 80. To do this we right clicked the **Port: 80** entry in the tree on the left side of the GUI, and in the pop-up menu selected **Add Server**. In the server address field we entered the remote Dispatcher’s nonforwarding address, and to indicate that the server is remote, we specified the router through which the Dispatcher must send the packets in order to reach the remote server. We put the IP address of our router in the router field as follows:


Add a server

Server address: 9.29.124.179

Network router address: 9.24.104.1

Figure 168. Add the Remote Dispatcher as a Port 80 Server on Local Dispatcher

We could have used the command-line version to add the server. This would have been:

```
ndcontrol server add 9.24.104.105:80:9.29.124.179 router 9.24.104.1
```

3. As we had previously aliased the cluster address to the network interface on the Dispatcher machine and to the loopback interface on each of the Web Server machines, no further alias configuration was necessary either on the local Dispatcher machine or the local Web Server machines.

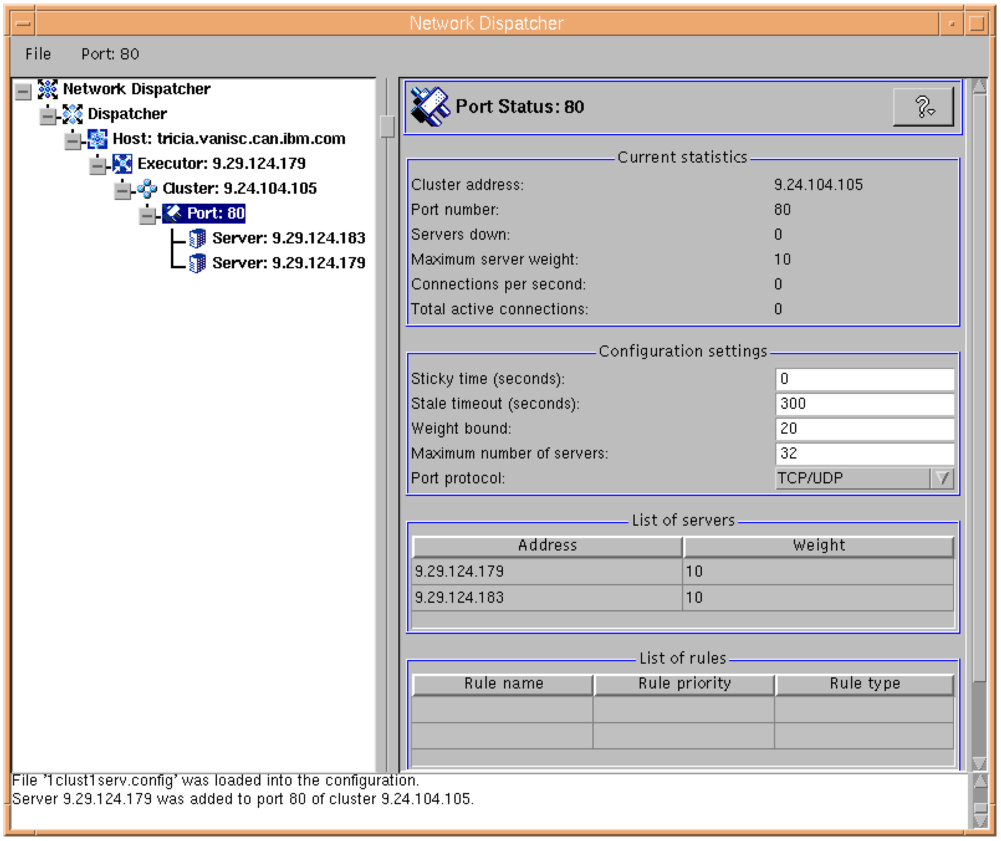
8.1.3 Remote Dispatcher Configuration Steps

In this section we show the configuration steps that were performed on the Dispatcher that we elected to use as the remote Dispatcher machine (the machine with host name tricia and IP address 9.29.124.179):

1. First we started the ndserver process. To do this, from a command line we entered the `ndserver` command.
2. Again, we performed all the configuration activities for the Dispatcher by using the GUI. The same steps could be performed by using the command line, as we see in 11.1, “Firewall High Availability Using the Dispatcher” on page 273. We launched the GUI by entering the `ndadmin` command.
3. Using the GUI, we started the Executor, with 9.29.124.179 as the nonforwarding address.
4. Then we defined the cluster with the same IP address that we used on our local Dispatcher, 9.24.104.105. Next we added the TCP port 80 to our cluster for the HTTP service. To port 80 we added the two Web servers. One of the Web servers was running on the same machine as the Dispatcher, so we used the 9.29.124.179 nonforwarding address for it. The other Web server was defined with its address of 9.29.124.183. Both Web servers had been previously configured and were running. Then we started the Manager.
5. Next, we configured the same port for wide area communication between the two Dispatcher machines that was configured on the local Dispatcher. This was done by entering the following command:

```
ndcontrol executor set wideportnumber 12345
```

Following is the Cluster Status window on the remote Dispatcher that summarizes the configuration:



Current statistics

Cluster address:	9.24.104.105
Port number:	80
Servers down:	0
Maximum server weight:	10
Connections per second:	0
Total active connections:	0

Configuration settings

Sticky time (seconds):	0
Stale timeout (seconds):	300
Weight bound:	20
Maximum number of servers:	32
Port protocol:	TCP/UDP

List of servers

Address	Weight
9.29.124.179	10
9.29.124.183	10

List of rules

Rule name	Rule priority	Rule type

```
File '1clust1serv.config' was loaded into the configuration.
Server 9.29.124.179 was added to port 80 of cluster 9.24.104.105.
```

Figure 169. Remote Dispatcher Port Status

6. The last step was to alias the cluster address to the loopback interface on each of the Web Server machines. Further information about how to create the alias can be found in 4.1.4.8, “Methods of Aliasing the Cluster to the Network Interface” on page 94 and 4.1.5, “TCP Servers Configuration” on page 107. In this case we used the following command on the fortytwo machine:

```
ifconfig lo0 alias 9.24.104.105 netmask 0xffffff00
```

   Notice, however, that the cluster address is not aliased to the loopback interface on the colocated Web server machine (which was also the Dispatcher). If the Dispatcher logic determines that the destination server has the same IP address as its own nonforwarding address, then the packet is given back to the underlying operating system and not routed out the network interface. This implies that *in order for colocation to work, the colocated Web server must be added with the nonforwarding address*.

Notice also that the cluster address is not aliased to the network interface on the remote Dispatcher machine.

8.1.4 Wide Area Load Balancing Scenario Results

We wrote four very simple similar HTML files. They were all named cjl.html, and we distributed them all into the document root directories of the three local clustered Web servers and the two remote clustered Web servers.

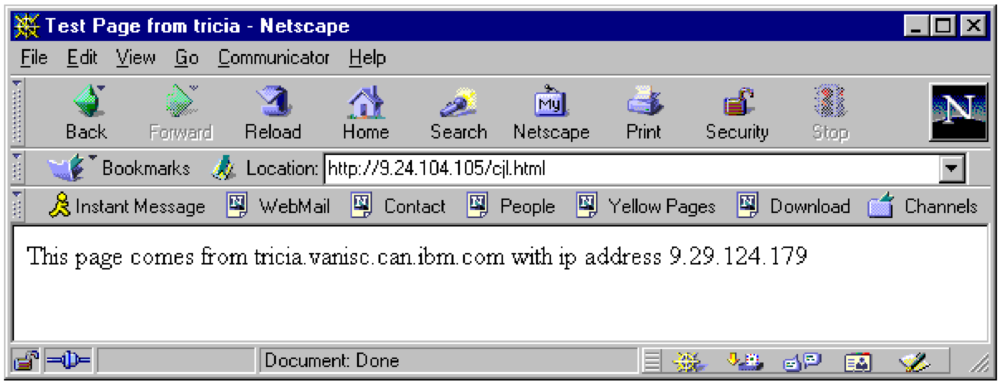
Each of the files were of the following form, but each uniquely identified the server that it was located on. This is the file from one of the local Web servers, aixncf157:

```
<html>
<title> Test Page from aixncf157</title>
<body>

This page comes from aixncf157.itso.ral.ibm.com
with ip address 9.24.104.157
</body>
</html>
```

Figure 170. cjl.html on Host 9.24.104.157

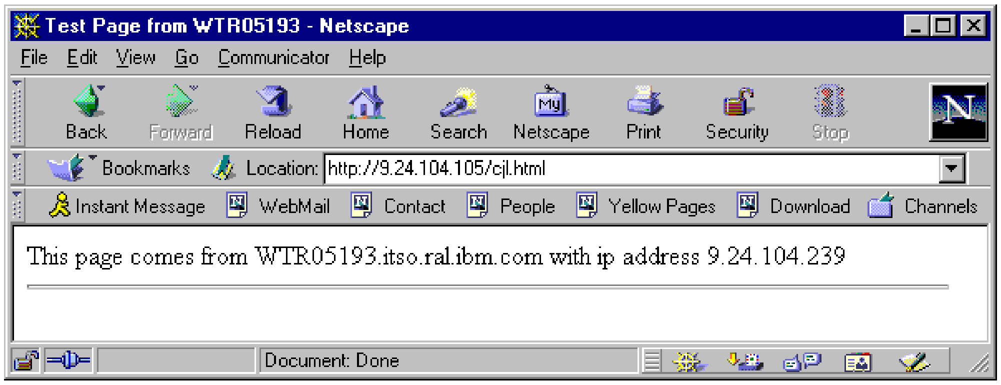
The other HTML files, located on the other Web server machines, were all very similar.

Similar to the scenario described in 4.1, “Load Balancing Basic Scenario Using the Dispatcher” on page 81, we then started our Web browser on the Web client machine and deactivated its memory and disk caches. From the browser on our client machine we requested the URL `http://9.24.104.105/cjl.html` several times and saw that the page was provided in turn by each of the Web server machines, as shown in the following four figures:


Test Page from tricia - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://9.24.104.105/cjl.html
Instant Message WebMail Contact People Yellow Pages Download Channels
Document: Done

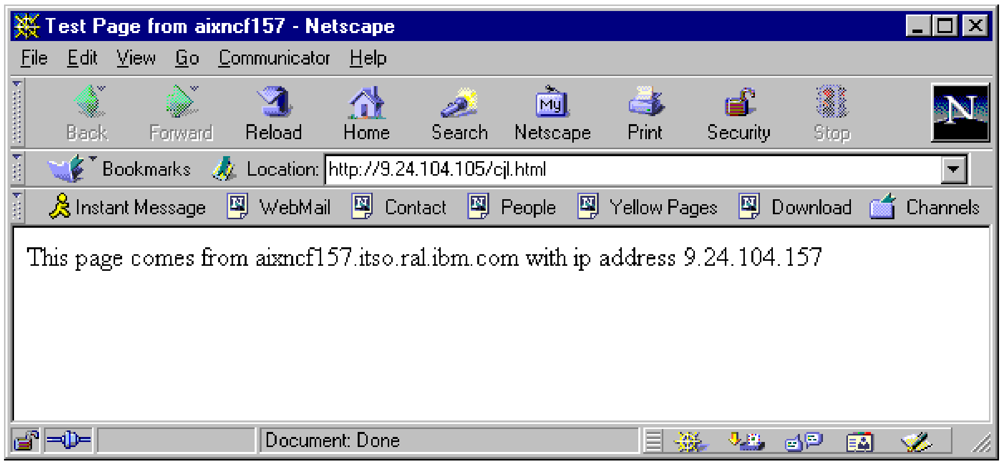
This page comes from tricia.vanisc.can.ibm.com with ip address 9.29.124.179

Figure 171. HTML Page Served by the First Web Server


Test Page from WTR05193 - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://9.24.104.105/cjl.html
Instant Message WebMail Contact People Yellow Pages Download Channels
Document: Done

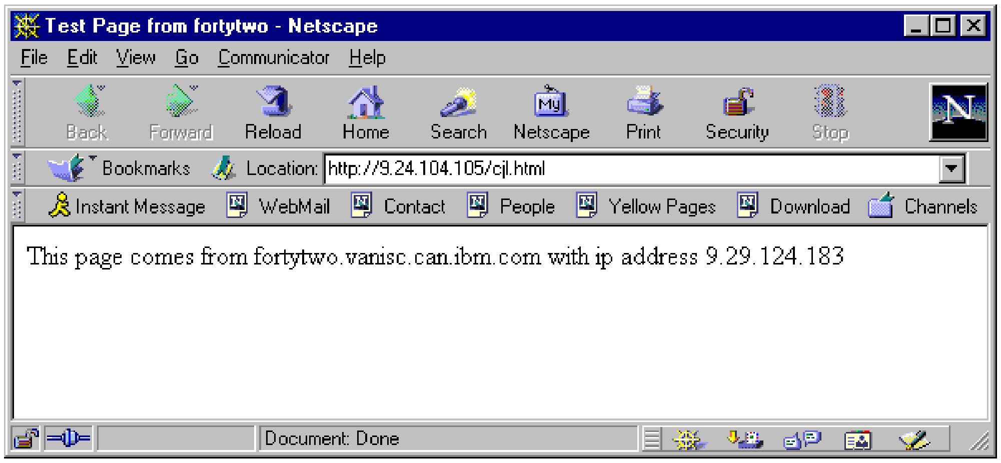
This page comes from WTR05193.itso.ral.ibm.com with ip address 9.24.104.239

Figure 172. HTML Page Served by the Second Web Server


Test Page from aixncf157 - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://9.24.104.105/cjl.html
Instant Message WebMail Contact People Yellow Pages Download Channels
Document: Done

This page comes from aixncf157.itso.ral.ibm.com with ip address 9.24.104.157

Figure 173. HTML Page Served by the Third Web Server


Test Page from fortytwo - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://9.24.104.105/cjl.html
Instant Message WebMail Contact People Yellow Pages Download Channels
Document: Done

This page comes from fortytwo.vanisc.can.ibm.com with ip address 9.29.124.183

Figure 174. HTML Page Served by the Fourth Web Server

In the above figures, the requested page came from the four Web servers starting with one of the remote Web servers, having IP address 9.29.124.179, then in turn from the Web servers with IP addresses 9.24.104.239 and 9.24.104.158 respectively, and finally from the remaining remote Web server, with IP address 9.29.124.183.

The ND GUI Port 80 monitor shows a graphical representation of the distribution of new connections made to the local Dispatcher:

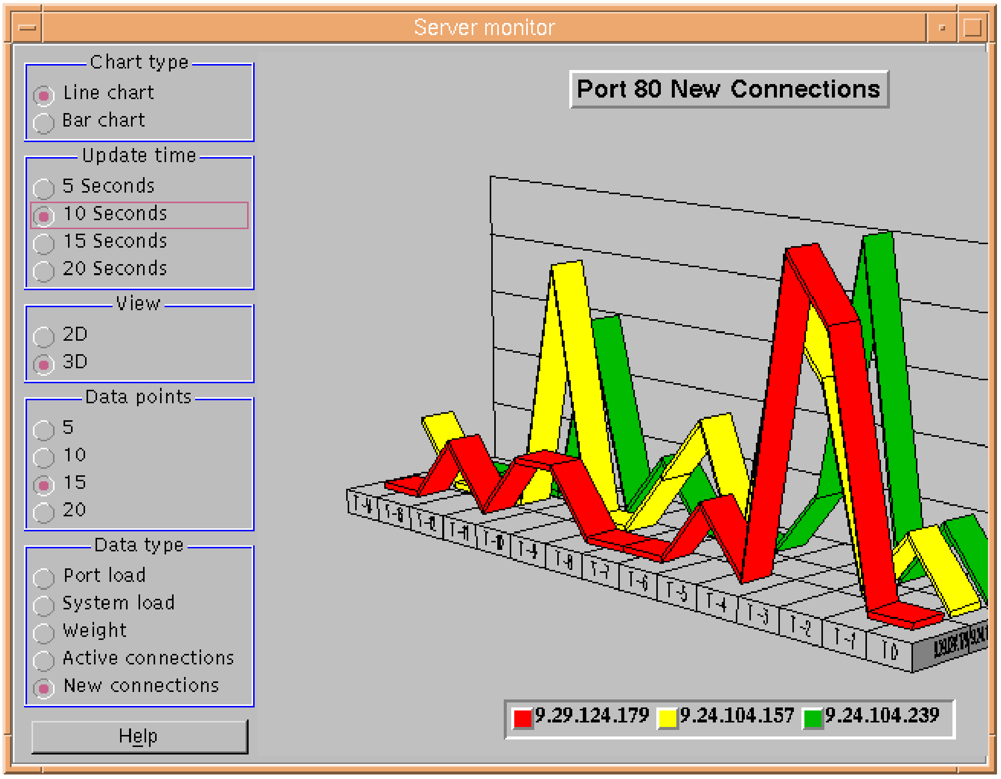
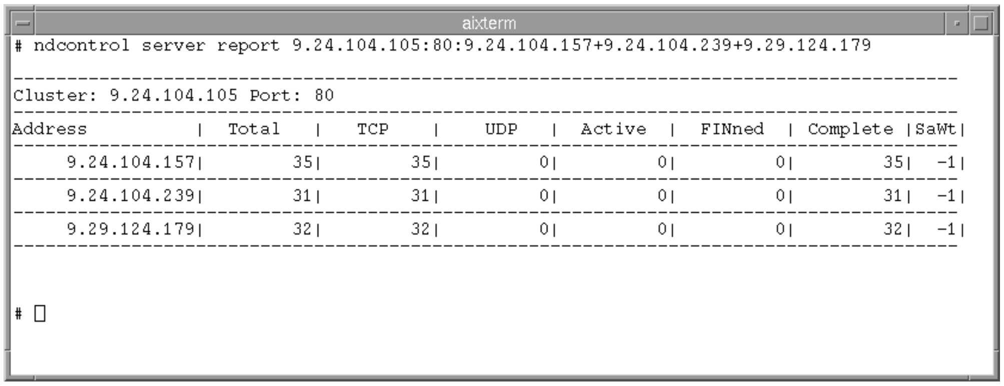


Figure 175. Port 80 Graphical Monitor

The same information can be seen in report format by entering the following command on the local Dispatcher machine rs600023:

```
ndcontrol server report 9.24.104.105:80:9.24.104.157+9.24.104.158+9.29.124.179
```

The following is the local Dispatcher report, which includes the nonforwarding address of the remote Dispatcher as one of its servers. We used the above command to produce the local report:



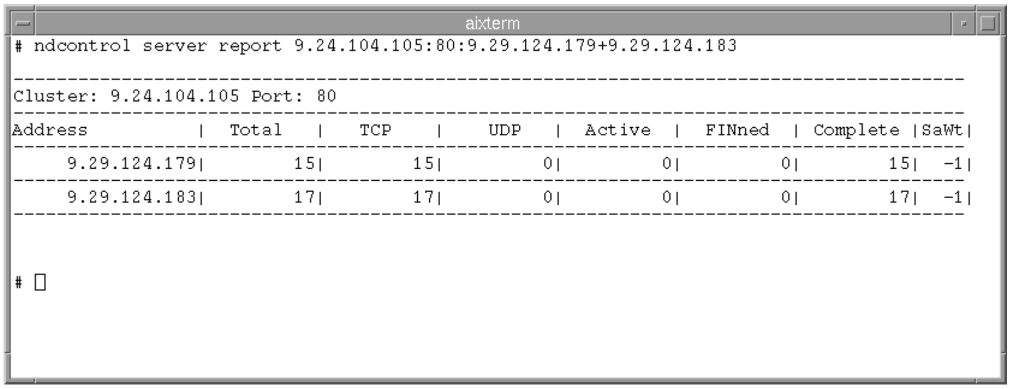
```
# ndcontrol server report 9.24.104.105:80:9.24.104.157+9.24.104.239+9.29.124.179

--------------------------------------------------------------------------------
Cluster: 9.24.104.105 Port: 80
--------------------------------------------------------------------------------
Address          |  Total   |   TCP    |   UDP    |  Active  |  FINned  | Complete |SaWt|
--------------------------------------------------------------------------------
    9.24.104.157|        35|        35|         0|         0|         0|        35|  -1|
--------------------------------------------------------------------------------
    9.24.104.239|        31|        31|         0|         0|         0|        31|  -1|
--------------------------------------------------------------------------------
    9.29.124.179|        32|        32|         0|         0|         0|        32|  -1|
--------------------------------------------------------------------------------


# 
```

Figure 176. Local ndcontrol Server Report Showing the Round-Robin Distribution of Requests

We used a similar command to produce the report on the remote Dispatcher:

```
ndcontrol server report 9.24.104.105:80:9.29.124.179+9.29.179.183
```

The output of the command above is shown in the following figure:



```
# ndcontrol server report 9.24.104.105:80:9.29.124.179+9.29.124.183

------------------------------------------------------------------------------------
Cluster: 9.24.104.105 Port: 80
------------------------------------------------------------------------------------
Address          |  Total   |   TCP    |    UDP   |  Active  |  FINned  | Complete |SaWt|
------------------------------------------------------------------------------------
     9.29.124.179|       15|        15|         0|         0|         0|        15|  -1|
------------------------------------------------------------------------------------
     9.29.124.183|       17|        17|         0|         0|         0|        17|  -1|
------------------------------------------------------------------------------------


# 
```

Figure 177. Remote ndcontrol Server Report Showing the Round-Robin Distribution of Requests

This verified that the local, or entry, Dispatcher was correctly routing requests in a round-robin fashion to the remote Dispatcher which in turn distributed the client requests to its local Web servers, also in a round-robin fashion. As we will see in the next section, the remote Web server machine then sends the response back to the client directly, with no requirement to pass the packets back through either of the Dispatcher machines.

8.1.5 How WAN Dispatcher Support Works – The Packet Flow

In this section, we build upon the information given in 4.1.6, “How the Dispatcher Works – The Flow of the IP Packets” on page 114 to include the details related to using a remote Dispatcher.

In 2.5, “How the Dispatcher Function Works” on page 34, we explained how the IP packets start flowing from a TCP client to a TCP server passing through the Dispatcher machine, and then how the server’s response flows directly from the server to the client without any need to pass through the Dispatcher again. When we add a remote Dispatcher to the environment, the path that the IP packet flows starts out the same as in the previous scenario. However, if the entry Dispatcher chooses to forward the packet to a remote Dispatcher rather than one of its local TCP servers, this is where the difference begins.

Recall that if the entry Dispatcher chooses to route the client TCP request to one of its local servers, it changes the original Media Access Control (MAC) address on the packet to the MAC address of the selected TCP server (for details on MAC addresses, see “MAC Address” on page 115). This works because all of the TCP servers are on the same subnet as the Dispatcher, since the MAC addresses cannot be used to route the packets outside a router. The fundamental point is that the destination IP address on the packet is not changed.

This presents a challenge to the WAN Dispatcher support because the MAC address cannot be used to route the packet beyond the router. As well, we know that the remote Dispatcher will not accept a packet destined to the cluster IP address because the cluster address is not aliased to the network interface on the remote Dispatcher.

From this we can infer that the destination IP address of the IP packet must be changed from the cluster IP address to the nonforwarding address of the remote

Dispatcher machine. However, doing this would mean we lose the advantage of anonymity that was gained by not changing the IP destination of the packet.

All of these issues are addressed by encapsulating the entire IP packet from the client within another IP packet. The encapsulating packet has in its header the cluster address as the source IP address and the nonforwarding address of the remote Dispatcher as the destination IP address. This encapsulated packet is sent from the WAN port on the entry point Dispatcher to the WAN port on the remote Dispatcher.

Before it is transmitted, it is of course encapsulated again in the frame type of the hardware transport medium (token-ring, Ethernet, etc.) it travels on. The first leg of its journey is to the router specified when the remote Dispatcher was added as a server.

A graphical representation of this explanation is offered in the following figure:

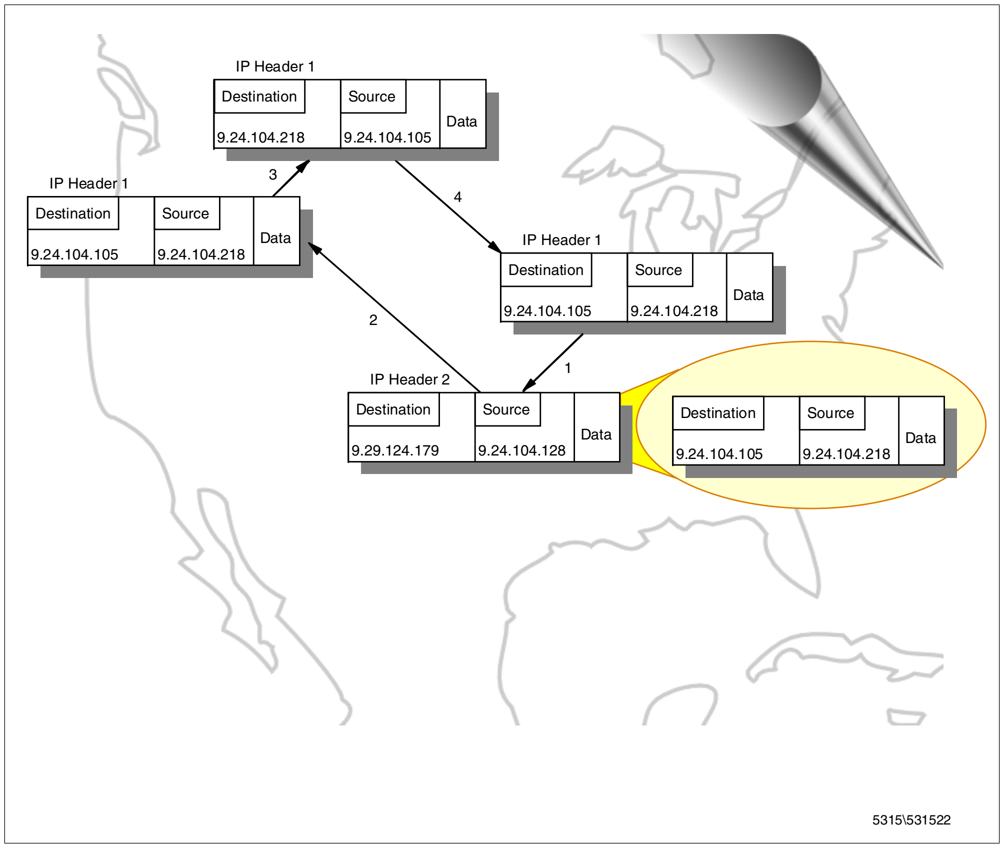


Figure 178. IP Header Source and Destination Values with WAN Dispatcher Support

When the frame is received by the remote Dispatcher, the outer IP wrapper is then stripped off and the inner client IP packet is then forwarded on one of the local load-balanced TCP servers. Code is built into the Dispatcher to prevent the packet from being forwarded again to another remote Dispatcher.

As before, the TCP servers have the cluster address aliased to the loopback interface enabling them to accept the packet. Also, as before, the TCP server selected by the remote Dispatcher responds by swapping the source and destination IP addresses to go directly back to the client machine, bypassing both of the network Dispatchers that have been involved in getting the client request here in the first place.

Both Dispatchers set up an entry in their own connection table to make sure that subsequent incoming IP packets for this client continue to be forwarded to the same server. The entry Dispatcher's connection table entry will point to the remote Dispatcher, and the remote Dispatcher's connection table entry will point to the correct TCP server.

Because the Dispatchers do not participate in bidirectional communications with the client but simply forward the incoming packets unchanged, their presence is transparent to both client and server. The real TCP/IP connection is between the client and the clustered server, and the Dispatchers soon disappear from the scene after forwarding the incoming packets.

8.2 Using Remote Advisors with WAN Dispatcher Support

The purpose of a remote advisor in this case is to send requests to the remote back-end servers to measure actual client response time for a particular protocol. These results are then fed to the Manager component of the remote Dispatcher to adjust the load-balancing weights.

On entry-point Dispatchers, advisors will work correctly without any special configuration. On remote Dispatchers, for each remote cluster, you need to alias the remote cluster address to the loopback interface. Recall that aliasing the cluster address to the loopback interface was also necessary on the TCP server machines and so we refer you back to the instructions in 4.1.4.8, “Methods of Aliasing the Cluster to the Network Interface” on page 94 and 4.1.5, “TCP Servers Configuration” on page 107. We provided those instructions to create the aliases for each platform.

8.3 WAN Dispatcher Support and ISS

Prior to IBM eNetwork Dispatcher V2.0, where WAN Dispatcher support was introduced, the only way remote sites could be configured was by using ISS in a two-tier or ping triangulation configuration. These options are still available, and also can be combined with the WAN Dispatcher support enhancement.

The combining of ISS and WAN Dispatcher support can be implemented in several ways:

- ISS ping triangulation accessing a remote Dispatcher site, as shown for Dispatcher 3 in Tier 1 in Figure 179.
- Dispatcher 3 could implement WAN Dispatcher support and include a remote Dispatcher, as shown for Dispatcher 3remote in Tier 2 in Figure 179.
- Dispatcher 3remote and its local TCP servers could implement a local ISS cell. A Dispatcher type observer could be defined in this cell to provide system load information about the TCP servers to Dispatcher 3remote for it to use in adjusting the load-balancing weights of the TCP servers (see Tier 3 in Figure 179).

It is important to note that a single local cell ISS implementation cannot contain a remote node entry.

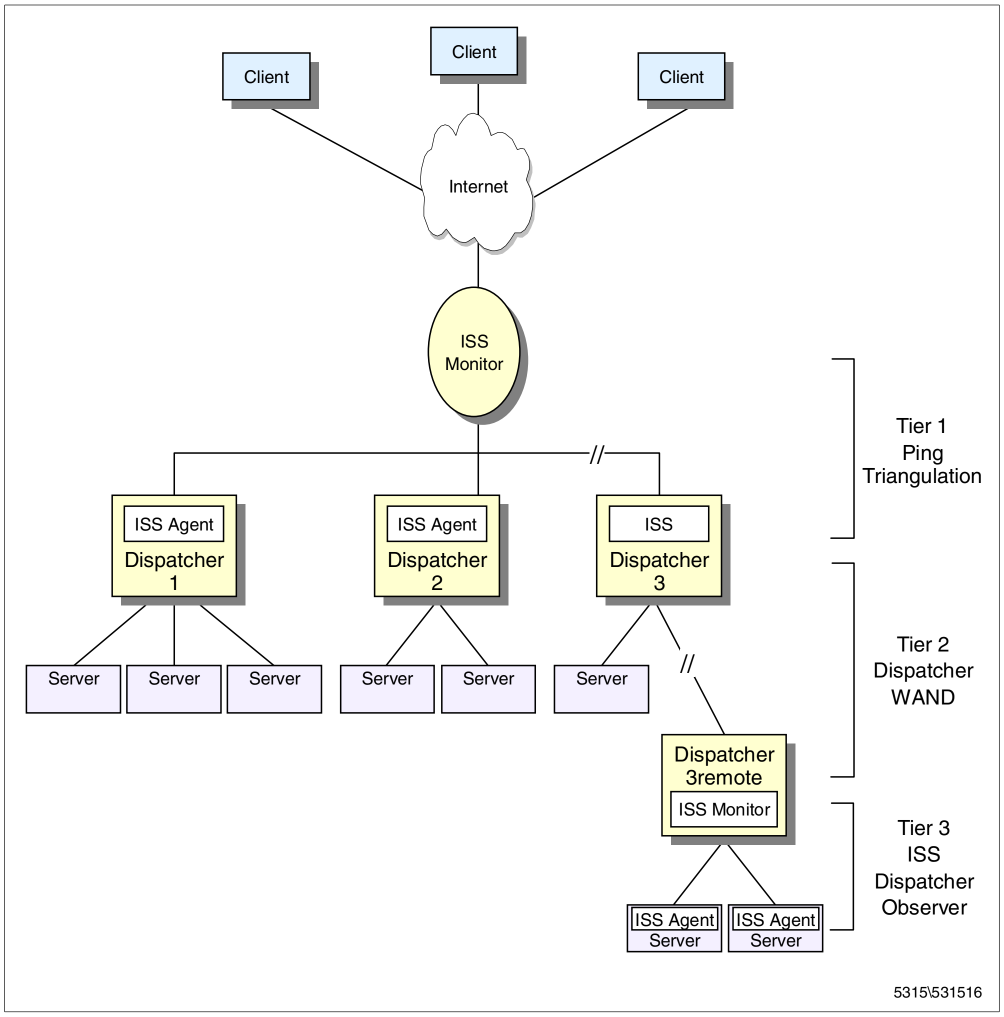


Figure 179. Managing Local and Remote Servers with ISS and WAN Dispatcher Support

8.4 WAN Dispatcher Support with Remote Dispatcher High Availability

Dispatcher high availability can be implemented at either end or both ends of a WAN Dispatcher support configuration. Both servers in the highly available environment must be configured the same with respect to their role in the WAN configuration. This means, for example, on the local or entry point Dispatcher side of a wide area configuration, both the primary and backup Dispatcher machines must have the WAN port configured and the remote Dispatcher's nonforwarding address set up as a server for the local cluster. Other than this, the remaining steps required to configure Dispatcher high availability on the local side of a WAN Dispatcher support environment are the same as those shown in Chapter 6, "ND High Availability Support" on page 177.

Similarly, on the remote side of the WAN Dispatcher support configuration, both the primary and backup Dispatcher machines must have the WAN port configured, and have a definition for the remote cluster being serviced by their own local TCP servers. However, one aspect of a remote high availability configuration differs from a local high availability configuration.

Recall that in a typical WAN Dispatcher support environment, as seen in 8.1.5, “How WAN Dispatcher Support Works – The Packet Flow” on page 221, the packets are sent to the remote Dispatcher by being encapsulated in an IP packet in whose header the destination IP address is the nonforwarding address of the remote Dispatcher. However, when there is a highly available Dispatcher on the remote side of a WAN Dispatcher support configuration, if a failover occurs, the active Dispatcher’s nonforwarding address changes. This issue is addressed by using a third IP address at the remote site. Both the primary and backup Dispatcher machines at the remote site will have their own nonforwarding address and will dynamically claim ownership of a *third, common non-forwarding address* when they become active. To facilitate the change of ownership of the third address, when a failover occurs the active Dispatcher sends out a gratuitous ARP to inform the router of the new MAC address associated with this common nonforwarding address.

In the following scenario, we make use of a third IP address on the remote primary and backup Dispatcher machines.

Several times throughout the following scenario, we refer to three sections in this book that describe basic Dispatcher configuration as well as the basics of implementing WAN Dispatcher support and high availability. It may be helpful for you to become familiar with those concepts before examining this scenario, because in this scenario we build upon the details provided in those sections. The three referenced sections are:

1. 4.1, “Load Balancing Basic Scenario Using the Dispatcher” on page 81
2. 6.1, “Dispatcher High-Availability Scenario” on page 177
3. 8.1, “Wide Area Network Dispatcher Scenario” on page 211

8.4.1 Scenario

We implemented this scenario in three stages:

1. In the first stage, we configured a simple local cluster.
2. In the second stage, we added a remote Dispatcher and its two local servers to the configuration by making use of the WAN Dispatcher support capabilities.
3. In the third stage, we changed the single remote Dispatcher to a highly available remote Dispatcher by adding a backup Dispatcher.

After implementing each stage, we did a simple test to ensure the configuration was correct. To do these tests, we placed a simple HTML file in the document root directory of each Web server in our scenario. The page contained text that uniquely identified the server that was serving the page. The tests were carried out by starting our Web browser and deactivating its memory and disk caches; then we requested the cluster address in the URL `http://9.24.105.18/cjl.html`.

The HTML files placed on the TCP servers were similar to the following, but each contained the host name and IP address of the respective Web server:

```
<html>
<title> Test Page from AIXAFS</title>
<body>

This page comes from AIXAFS.itso.ral.ibm.com
with ip address 9.24.104.158
</body>
</html>
```

Figure 180. cjl.html on Host 9.24.104.157

8.4.1.1 Network Environment

The following figure offers a graphical representation of our scenario environment:

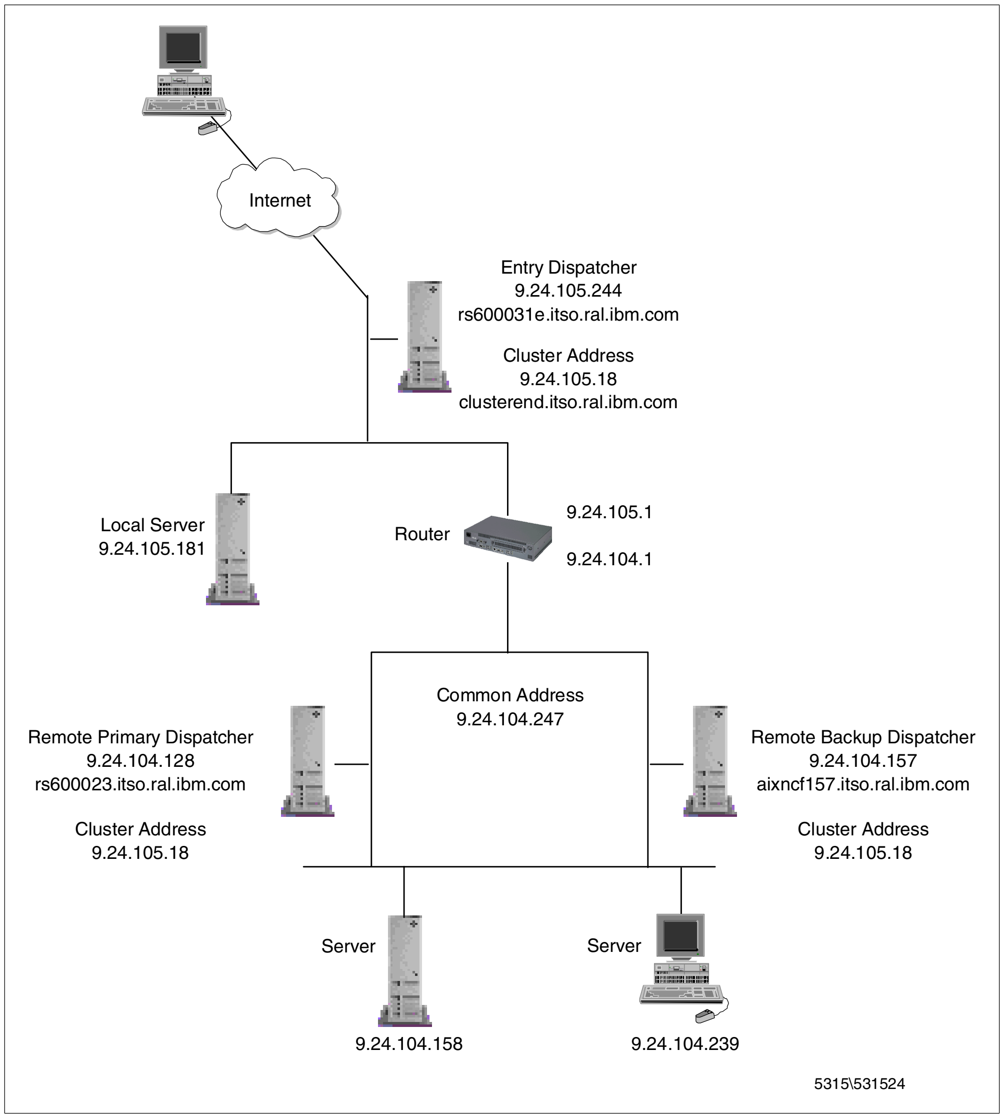


Figure 181. WAN Dispatcher Support with Remote Dispatcher High Availability Scenario

A summary of the hardware, software and network configuration of the machines involved in each of the stages of our implementation is shown in the following three tables:

Table 14. Stage 1 – Local Network Dispatcher Environment

Workstation	Host Name	IP Address	Operating System	Service
IBM PC 365	wtr05212	9.24.104.218	Windows NT Server 4.0	Web client
IBM RS/6000 43P	rs600031e	9.24.105.244	AIX 4.3.1	Entry Dispatcher
	clusterend	9.24.105.18		
IBM RS/6000 43P	rs60002	9.24.105.181	AIX 4.3.2	Web server

Table 15. Stage 2 – WAN Dispatcher Support with Local Network Dispatcher Environment

Workstation	Host Name	IP Address	Operating System	Service
IBM RS/6000 43P	rs600023	9.24.104.128	AIX 4.3.1	Remote (primary) Dispatcher
	clusterend	9.24.105.18		
IBM RS/6000 43P	aixafs	9.24.104.158	AIX 4.3.1	Web server
IBM PC 365	wtr05193	9.24.104.239	Windows NT Server 4.0	Web server

Table 16. Stage 3 – WAN Dispatcher Support with Remote High Availability Local Network Dispatcher Environment

Workstation	Host Name	IP Address	Operating System	Service
IBM RS/6000 43P	aixncf157	9.24.104.157	AIX 4.3.1	Remote backup Dispatcher
	clusterend	9.24.105.18		

The version of the ND component that was used on all Dispatcher machines in this scenario was 2.1.0.1.

8.4.1.2 Stage 1 – Local Cluster Configuration

The local component of our WAN Dispatcher support scenario is very similar to the one that was described in 4.1, “Load Balancing Basic Scenario Using the Dispatcher” on page 81.

We started the Dispatcher on rs600031e with the `ndserver` command and used the ND GUI to start the executor with the default nonforwarding address, 9.24.104.128. We then configured the cluster with address 9.24.105.18 and to this cluster we added port 80 as well as the server for it, 9.24.105.181.

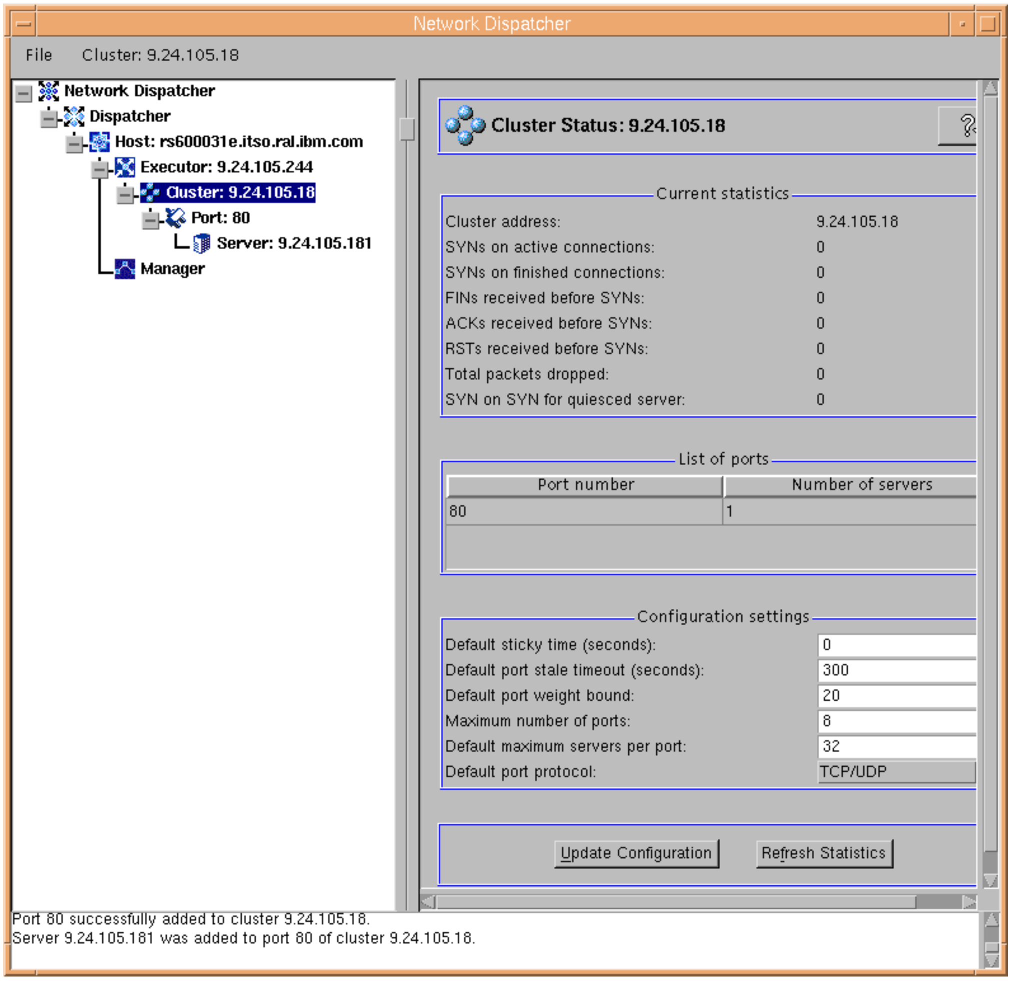
We aliased the cluster address to the network interface on rs600031e (the Dispatcher machine) with the command:

```
ifconfig en0 alias 9.24.105.18 netmask 255.255.255.0
```

We aliased the cluster address to loopback on rs60002 (the Web server machine) and deleted the associated route with the following commands:

```
ifconfig lo0 alias 9.24.105.18 netmask 255.0.0.0
route delete 9/8 9.24.105.18
```

The Cluster Status from the ND GUI showing a summary of our configuration appeared as follows:



Current statistics

Cluster address:	9.24.105.18
SYNs on active connections:	0
SYNs on finished connections:	0
FINs received before SYNs:	0
ACKs received before SYNs:	0
RSTs received before SYNs:	0
Total packets dropped:	0
SYN on SYN for quiesced server:	0

List of ports

Port number	Number of servers
80	1

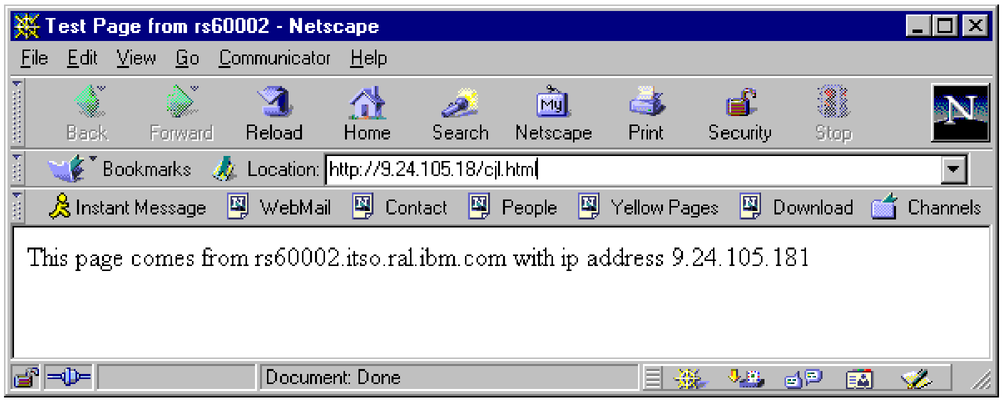
Configuration settings

Default sticky time (seconds):	0
Default port stale timeout (seconds):	300
Default port weight bound:	20
Maximum number of ports:	8
Default maximum servers per port:	32
Default port protocol:	TCP/UDP

```
Port 80 successfully added to cluster 9.24.105.18.
Server 9.24.105.181 was added to port 80 of cluster 9.24.105.18.
```

Figure 182. Cluster Status from the Local Dispatcher in Stage 1

To verify that the cluster was operational, we opened a browser on our Web client machine and requested the cjl.html page from the cluster address. The page was returned successfully from the single server that was defined in the cluster, as shown in the following figure:


Test Page from rs60002 - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://9.24.105.18/cjl.html
Instant Message WebMail Contact People Yellow Pages Download Channels
Document: Done

This page comes from rs60002.itso.ral.ibm.com with ip address 9.24.105.181

Figure 183. HTML Page Served by the Only Web Server in the Cluster

8.4.1.3 Stage 2 – Add WAN Dispatcher Support to the Local Cluster

In this stage we performed the following steps to configure rs600023 to be a remote Dispatcher for the cluster with IP address 9.24.105.18, which we configured in 8.4.1.2, “Stage 1 – Local Cluster Configuration” on page 227:

1. On the machine rs600023, we started the Dispatcher with the `ndserver` command, started the ND GUI with the `ndadmin` command and from the GUI, started the Executor.
2. In preparation for the configuration required in 8.4.1.4, “Stage 3 – Add High Availability to the Remote Dispatcher” on page 233, we used the third IP address that we selected for use at the remote end of our WAN Dispatcher support environment as the nonforwarding address of our Executor. We did this by typing `9.24.104.247` over the default nonforwarding address assigned to the Executor, within the Executor Configuration settings in the ND GUI.
3. On the machine rs600023, we added the cluster with address 9.24.105.18. To this cluster we added port 80 and the two local servers for it. These two servers had IP address 9.24.104.158 and 9.24.104.239, respectively.
4. We followed the instructions in 8.1, “Wide Area Network Dispatcher Scenario” on page 211 to pick a port to be used for wide area communications between rs600031e and rs600023. We used the ND GUI to set the Executor WAN port number on both Dispatchers, rs600031e and rs600023.
5. We aliased the new third IP address to the network interface on rs600023 with the command:

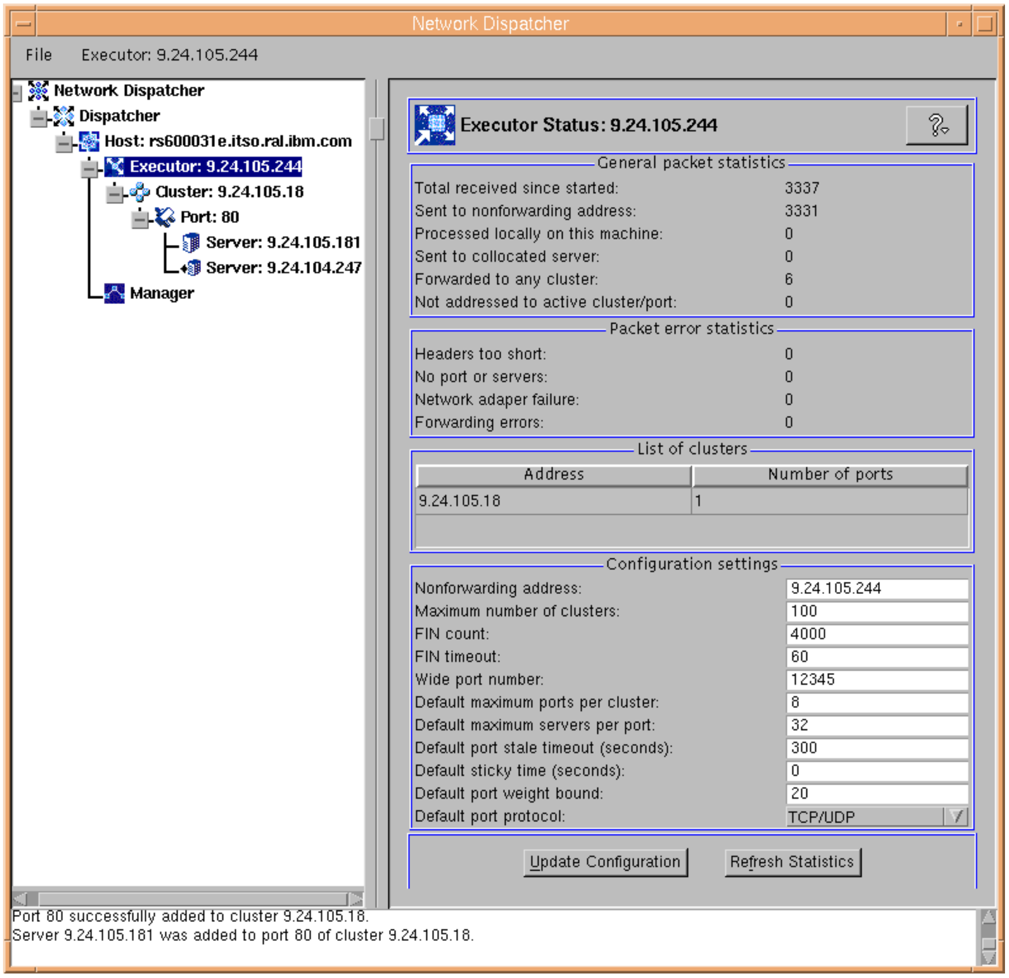
   ```
   ifconfig tr0 alias 9.24.104.247 netmask 255.255.255.0
   ```

   We aliased the cluster address to loopback on the two Web server machines, having IP addresses 9.24.104.158 and 9.24.104.239 respectively. On the AIX Web server machine we used the following command to alias the cluster address and delete the associated route:

   ```
   ifconfig lo0 alias 9.24.105.18 netmask 255.0.0.0
   route delete 9/8 9.24.105.18
   ```

   Refer to 4.1.5.2, “Aliasing the Loopback Device on Solaris” on page 109 and 4.1.5.3, “Aliasing the Loopback Device on Windows NT” on page 109 for instructions on how to alias an IP address to the loopback device on Solaris and Windows NT Web server machines.
6. Back on the entry Dispatcher machine (rs600031e, configured in 8.4.1.2, “Stage 1 – Local Cluster Configuration” on page 227), we added the remote Dispatcher’s 9.24.104.247 IP address as a port 80 server to the 9.24.105.18 cluster.

   The entry Dispatcher ND GUI appeared as follows:



General packet statistics

Total received since started:	3337
Sent to nonforwarding address:	3331
Processed locally on this machine:	0
Sent to collocated server:	0
Forwarded to any cluster:	6
Not addressed to active cluster/port:	0

Packet error statistics

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

List of clusters

Address	Number of ports
9.24.105.18	1

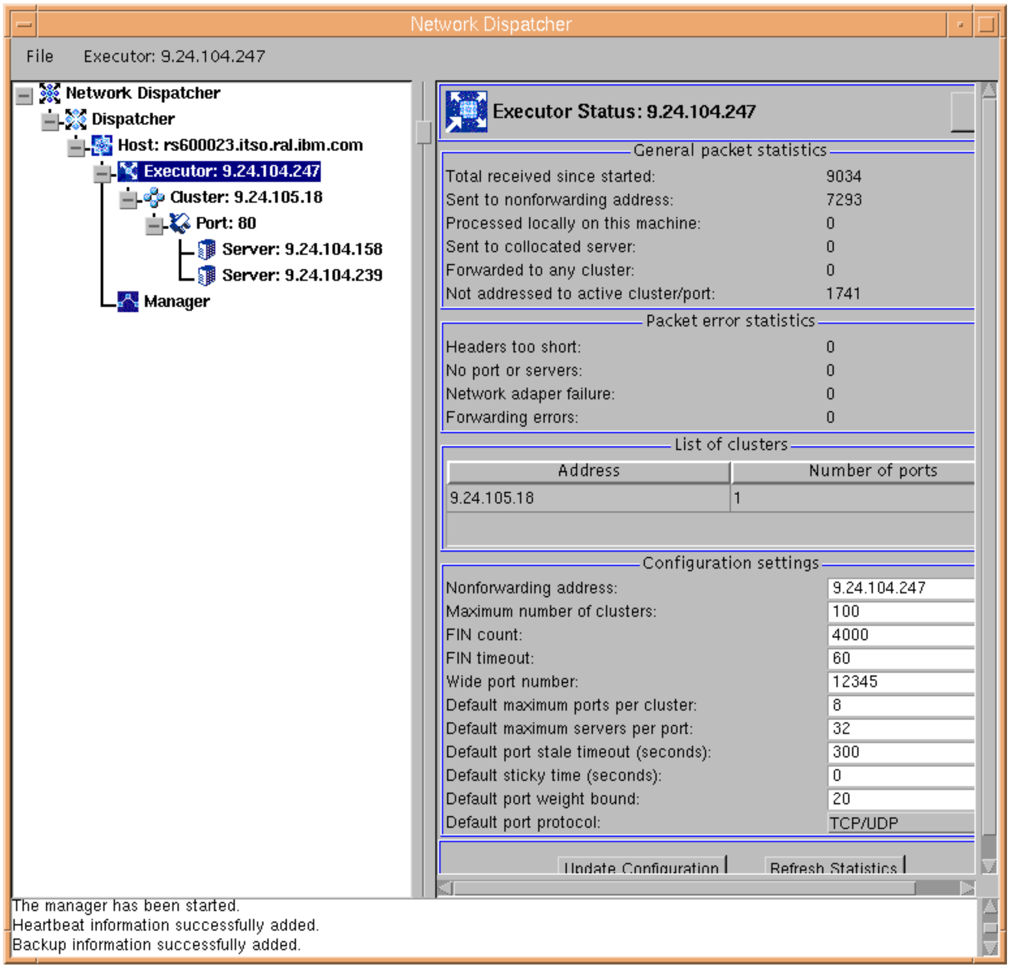
Configuration settings

Nonforwarding address:	9.24.105.244
Maximum number of clusters:	100
FIN count:	4000
FIN timeout:	60
Wide port number:	12345
Default maximum ports per cluster:	8
Default maximum servers per port:	32
Default port stale timeout (seconds):	300
Default sticky time (seconds):	0
Default port weight bound:	20
Default port protocol:	TCP/UDP

```
Port 80 successfully added to cluster 9.24.105.18.
Server 9.24.105.181 was added to port 80 of cluster 9.24.105.18.
```

Figure 184. Entry Dispatcher Executor Configuration

The remote Dispatcher ND GUI appeared as follows:



Executor Status: 9.24.104.247

General packet statistics

Total received since started:	9034
Sent to nonforwarding address:	7293
Processed locally on this machine:	0
Sent to collocated server:	0
Forwarded to any cluster:	0
Not addressed to active cluster/port:	1741

Packet error statistics

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

List of clusters

Address	Number of ports
9.24.105.18	1

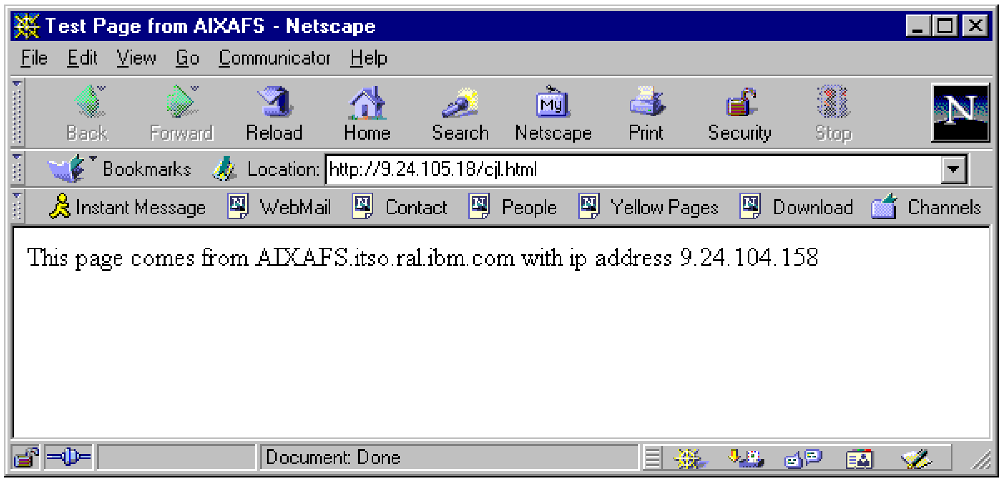
Configuration settings

Nonforwarding address:	9.24.104.247
Maximum number of clusters:	100
FIN count:	4000
FIN timeout:	60
Wide port number:	12345
Default maximum ports per cluster:	8
Default maximum servers per port:	32
Default port stale timeout (seconds):	300
Default sticky time (seconds):	0
Default port weight bound:	20
Default port protocol:	TCP/UDP

```
The manager has been started.
Heartbeat information successfully added.
Backup information successfully added.
```

Figure 185. Remote Dispatcher Executor Status

At this point we tested the configuration by making several requests for the cjl.html page at the cluster address. The results confirmed that the entry Dispatcher was correctly load balancing requests to both the local server and the remote Dispatcher, as we saw on the following page served by the remote server aixafs, as one of the pages we received in response to our request:


Test Page from AIXAFS - Netscape

File Edit View Go Communicator Help

Location: http://9.24.105.18/cjl.html

This page comes from AIXAFS.itso.ral.ibm.com with ip address 9.24.104.158

Document: Done

Figure 186. HTML Page Served by One of the Cluster's Remote Web Servers

The Port 80 monitor output from the entry Dispatcher also confirms that requests are being sent to the remote Dispatcher with address 9.24.104.247:

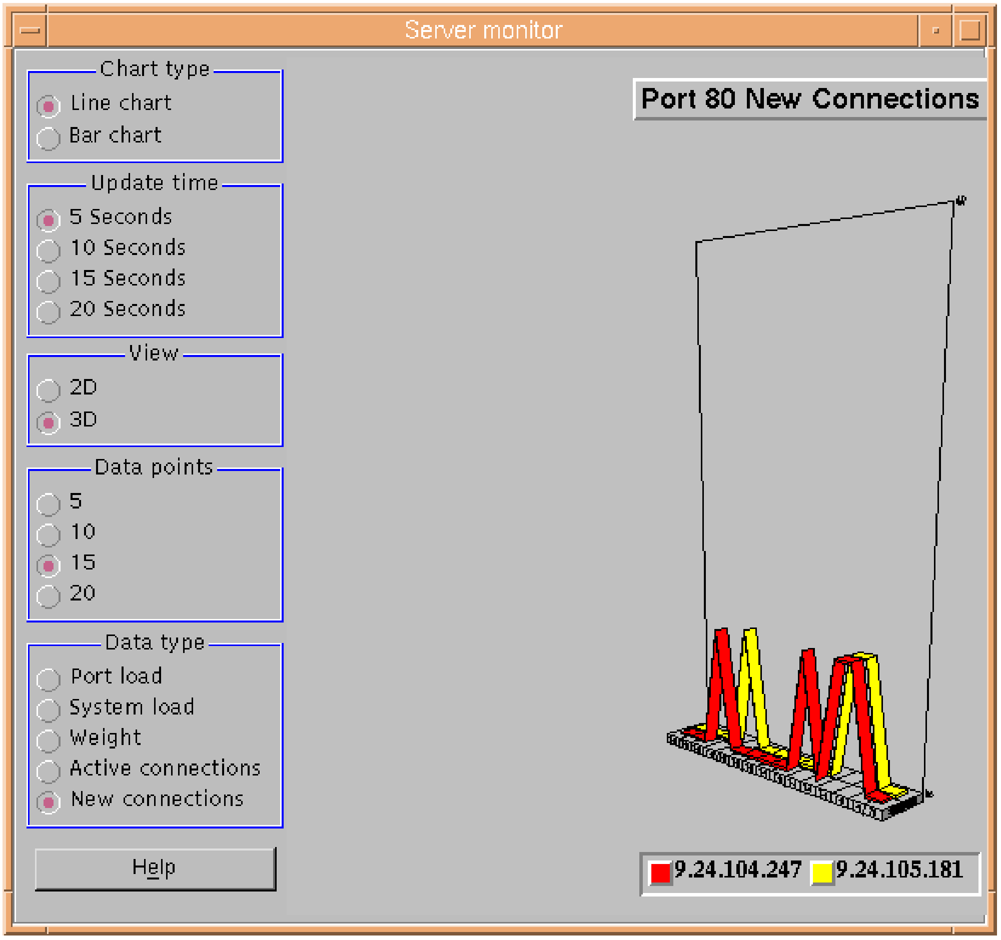

Server monitor

Chart type: [✓ Line chart] Bar chart

Update time: [✓ 5 Seconds] 10 Seconds 15 Seconds 20 Seconds

View: 2D [✓ 3D]

Data points: 5 10 [✓ 15] 20

Data type: Port load System load Weight Active connections [✓ New connections]

Port 80 New Connections

🟥 9.24.104.247 🟨 9.24.105.181

Figure 187. Port 80 GUI Monitor Output on the Entry Dispatcher Machine

8.4.1.4 Stage 3 – Add High Availability to the Remote Dispatcher

In this stage we configured the machine aixncf157, with IP address 9.24.104.157, as the high-availability *remote backup Dispatcher* for our Remote Dispatcher with IP address 9.24.104.128 (which we now refer to as the *remote primary Dispatcher*). The first four steps that we followed were essentially the same as the first four steps in 8.4.1.3, "Stage 2 – Add WAN Dispatcher Support to the Local Cluster" on page 229 (see Step 1 on page 229 through Step 4 on page 229), except that we performed them on the backup remote Dispatcher machine, aixncf157.

1. On aixncf157 we started the Dispatcher with the `ndserver` command, started the ND GUI with the `ndadmin` command, and from the GUI started the Executor.
2. We typed `9.24.104.247` over the default non-forwarding address assigned to the Executor, within the Executor Configuration settings in the ND GUI.
3. On aixncf157 we added the cluster with address 9.24.105.18. To this cluster we added port 80 and the two local servers for it, 9.24.104.158 and 9.24.104.239.
4. We used the ND GUI to set the Executor WAN port number that was selected on aixncf157 in 8.4.1.3, "Stage 2 – Add WAN Dispatcher Support to the Local Cluster" on page 229.
5. Recall that in a WAN Dispatcher support environment, the cluster address is not aliased to the network interface on the remote Dispatcher machine. However, what needs to be done in this case is to alias the third IP address to the network interface on the Remote Dispatcher. Recall, however, that as we described in 6.1, "Dispatcher High-Availability Scenario" on page 177, the network interface and loopback aliasing in a high-availability environment is done by using three scripts that are run automatically when the Executor changes state. We modified the goActive, goStandby and goInOp scripts shown in the following figures, so that it was not the cluster IP address that was being aliased, but the third common IP address, 9.24.104.247. We placed the three scripts in the /usr/lpp/nd/dispatcher/bin directory on both the remote primary Dispatcher machine and the remote backup Dispatcher machine. Our modified scripts follow:
   - goActive script

     This script deletes the loopback aliases and adds an alias to our third common IP address to the network interface. It is executed when either Dispatcher goes into active state and begins load balancing requests.

```
#
# goActive script
#
# Configure this script when using the high availability feature of
# Network Dispatcher.
#
# This script is executed when Network Dispatcher goes into the
# 'Active' state and begins routing packets.
#
# This script must be placed in Network Dispatcher's bin directory
# (by default this is /usr/lpp/nd/dispatcher) and it needs to
# have root execute permission.
#
# Modify NETWORK, CLUSTER, INTERFACE and NETMASK to match your environment.
#
# en0=first Ethernet adapter, tr0=first Token ring adapter, fi0=first FDDI adapter.
#
# NETMASK must be the netmask of your LAN.  It may be hexadecimal or octal notation.
#
NETWORK=9.24.104
INTERFACE=tr0
NETMASK=0xffffff00
#
 echo "Adding cluster alias(es)"
   for CLUSTER in 247; do
      ifconfig lo0 delete $NETWORK.$CLUSTER
      ifconfig $INTERFACE alias $NETWORK.$CLUSTER netmask $NETMASK
  done
```

Figure 188. goActive Script

- goStandby script

  This script deletes the device alias and aliases the third common IP address to loopback. It is executed when either Dispatcher goes into standby state.

```
#!/bin/ksh
#
# goStandby script
#
# Configure this script when using the high availability feature of
# eNetwork Dispatcher.
#
# This script is executed when Network Dispatcher goes into the
# 'Standby' state.  Monitoring the health of the 'Active' machine
# but not routing packets.
#
# This script must be placed in Network Dispatcher's bin directory
# (by default this is /usr/lpp/nd/dispatcher) and it needs to
# have root execute permission.
#
# Modify NETWORK, CLUSTER, INTERFACE and NETMASK to match your environment.
#
# en0=first Ethernet adapter, tr0=first Token ring adapter, fi0=first FDDI adapter
#
# NETMASK must be the netmask of your LAN.  It may be hexadecimal or octal notation.
#
NETWORK=9.24.104
INTERFACE=tr0
NETMASK=0xffffff00
#
echo "Deleting the device aliases and adding the loopback aliases"
 for CLUSTER in 247 ; do
    ifconfig $INTERFACE delete $NETWORK.$CLUSTER
    ifconfig lo0 alias $NETWORK.$CLUSTER netmask $NETMASK
 done
```

Figure 189. goStandby Script

- goInOp script

  This script deletes all cluster device and loopback aliases. This script is executed when a Dispatcher Executor is stopped and before it is started for the first time.

```
#!/bin/ksh
#
# goInOp script
#
# Configure this script when using the high availability feature of
# Network Dispatcher and optionally when using Network Dispatcher in a
# standalone environment.
#
# This script is executed when the Network Dispatcher executor is stopped
# (and before the executor is initially started).
#
# This script must be placed in Network Dispatcher's bin directory
# (by default this is /usr/lpp/nd/dispatcher) and it needs to
# have root execute permission.
#
# Modify NETWORK, CLUSTER and INTERFACE to match your environment.
#
# en0=first Ethernet adapter, tr0=first Token ring adapter, fi0=first FDDI adapter
#
NETWORK=9.24.104
INTERFACE=tr0
#
echo "Removing all loopback and device aliases"
for CLUSTER in 247; do
   ifconfig lo0 delete $NETWORK.$CLUSTER
   ifconfig $INTERFACE delete $NETWORK.$CLUSTER
 done
```

Figure 190. goInOp script

The scripts shown above have to respect a different syntax on the Windows NT platform. See “Script Modifications on Windows NT” on page 99 for further details.

Script Modifications

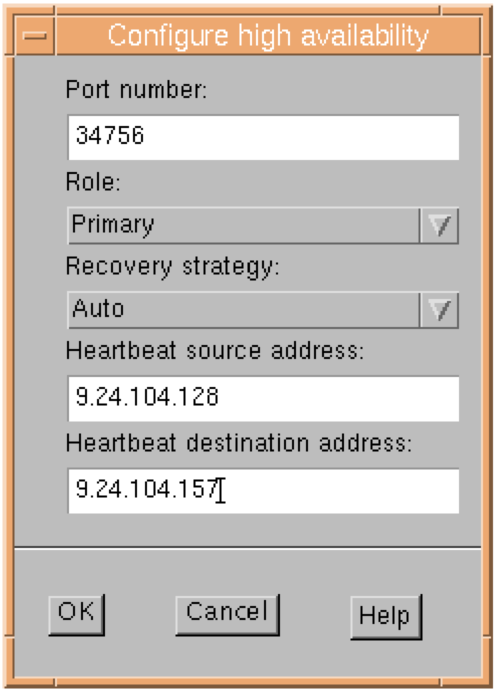
The scripts shown above should be modified under two circumstances:

- If you are using an Advisor on the remote Dispatcher machine, the scripts must be modified to alias the cluster address to loopback on the active Disptacher and add an entry to the ARP table for the remote cluster address. See 8.2, “Using Remote Advisors with WAN Dispatcher Support” on page 223 for further details on how to do this.
- If your remote highly-available Dispatcher is acting as an entry-point Dispatcher for some other cluster addresses, the scripts must be modified to include the aliasing for these as well. See 6.1.3, “Configuration Steps for High Availability” on page 184 for further details.

6. The last step is to add the high-availability remote backup Dispatcher. On both of the remote Dispatcher machines, from the ND GUI host menu, we selected **Add High Availability Backup**. In the Configure high availability dialog window on both Dispatchers, we added the port number over which the two Dispatcher machines will communicate to exchange data for the synchronization of the Dispatcher information. On the primary Dispatcher, we

set the Role to **Primary** and added `9.24.104.128` as the heartbeat source and `9.24.104.157` as the heartbeat destination. On the backup Dispatcher we set the Role to **Backup** and added `9.24.104.157` as the heartbeat source and `9.24.104.128` as the heartbeat destination. On both, we set the Recovery strategy to **Auto**, meaning that, after recovering from a failure, the primary Dispatcher will automatically take over.

Following is the Configure high availability dialog from rs600023:


Configure high availability

Port number:
34756

Role:
Primary

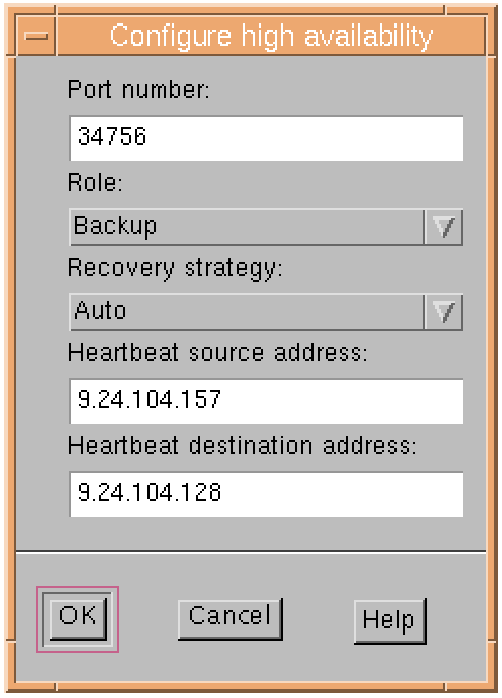
Recovery strategy:
Auto

Heartbeat source address:
9.24.104.128

Heartbeat destination address:
9.24.104.157

Figure 191. Configure High Availability Dialog from rs600023

Following is the Configure high availability dialog from aixncf157:


Configure high availability

Port number:
34756

Role:
Backup

Recovery strategy:
Auto

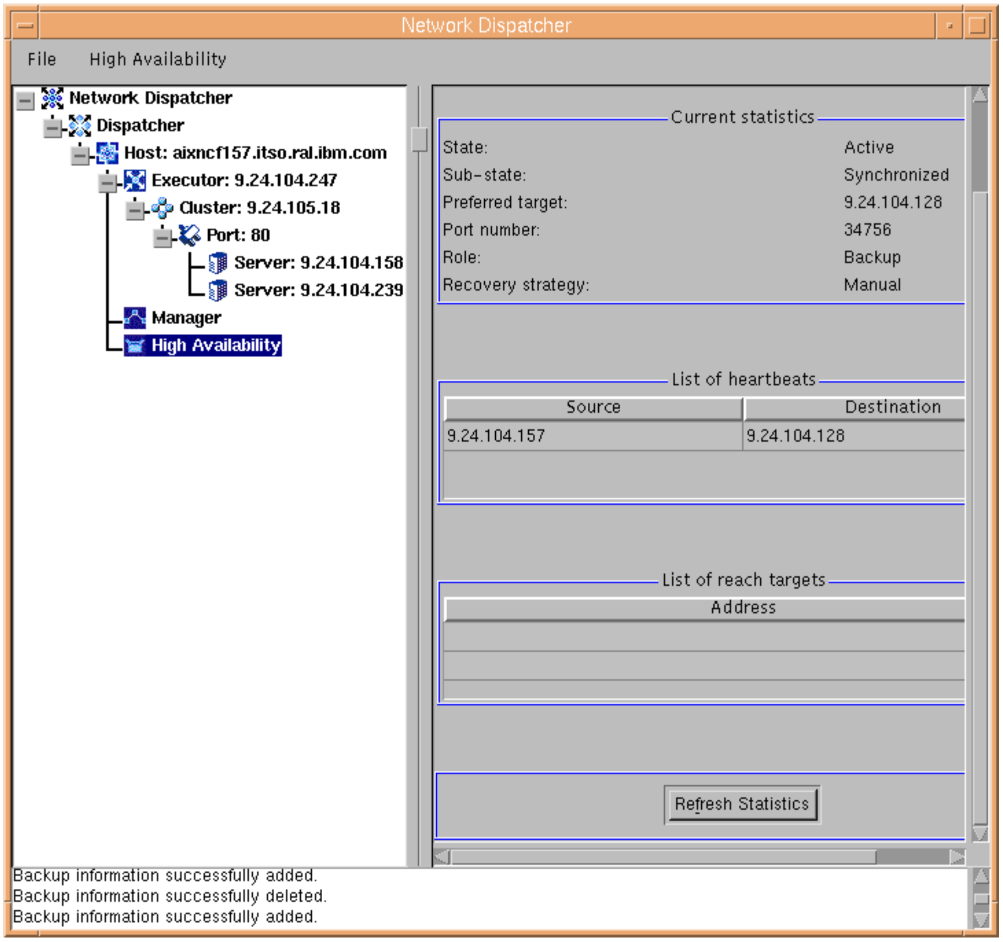
Heartbeat source address:
9.24.104.157

Heartbeat destination address:
9.24.104.128

Figure 192. Configure High Availability Dialog from aixncf157

For testing purposes, we changed the Recovery strategy on both the Dispatcher machines, setting it to **Manual**. Then, to test the high availability configuration, we forced a takeover by selecting the **Takeover for Backup** item from the High Availability menu in the ND GUI on aixncf157.

When the takeover was complete, we clicked the **Refresh Statistics** button on the High Availability Status window on both the primary and backup Dispatcher machines. The following window confirms that the remote backup Dispatcher has taken over for the remote primary Dispatcher as the active Dispatcher:



Current statistics

State:	Active
Sub-state:	Synchronized
Preferred target:	9.24.104.128
Port number:	34756
Role:	Backup
Recovery strategy:	Manual

List of heartbeats

Source	Destination
9.24.104.157	9.24.104.128

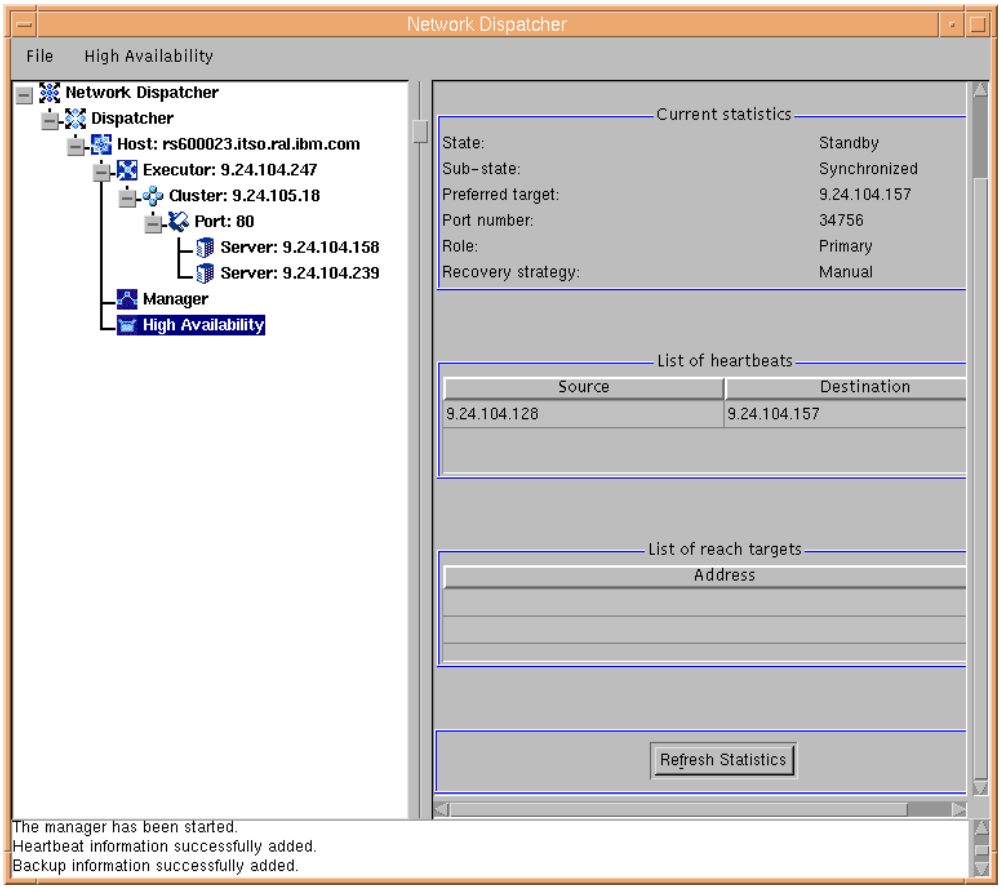
List of reach targets

Address

```
Backup information successfully added.
Backup information successfully deleted.
Backup information successfully added.
```

Figure 193. High Availability Status from aixncf157

This High Availability status window from the remote primary Dispatcher confirms that it was in standby state:



Current statistics	
State:	Standby
Sub-state:	Synchronized
Preferred target:	9.24.104.157
Port number:	34756
Role:	Primary
Recovery strategy:	Manual

Source	Destination
9.24.104.128	9.24.104.157

```
The manager has been started.
Heartbeat information successfully added.
Backup information successfully added.
```

Figure 194. High Availability Status from rs600023

The first test of the takeover that we performed was to verify that aixncf157 was responding to the third IP address. To do this we simply did a telnet to 9.24.104.247:

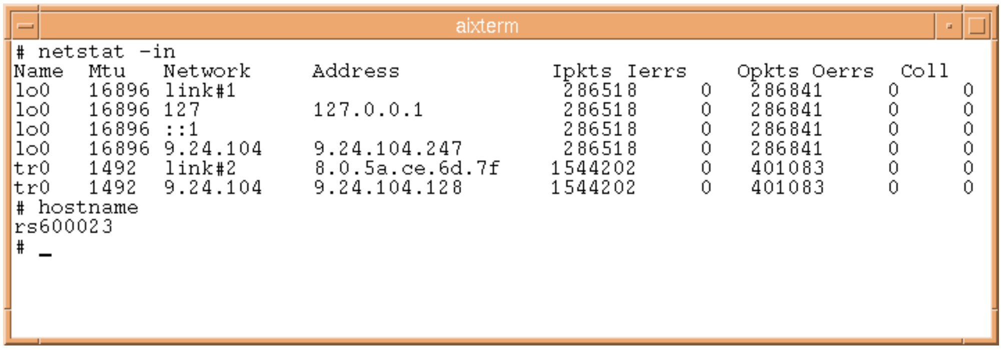
```
telnet 9.24.104.247
```

We saw that the response came from the backup Dispatcher, aixncf157, rather than the primary Dispatcher, rs600023.

Further confirmation of this was output from the command:

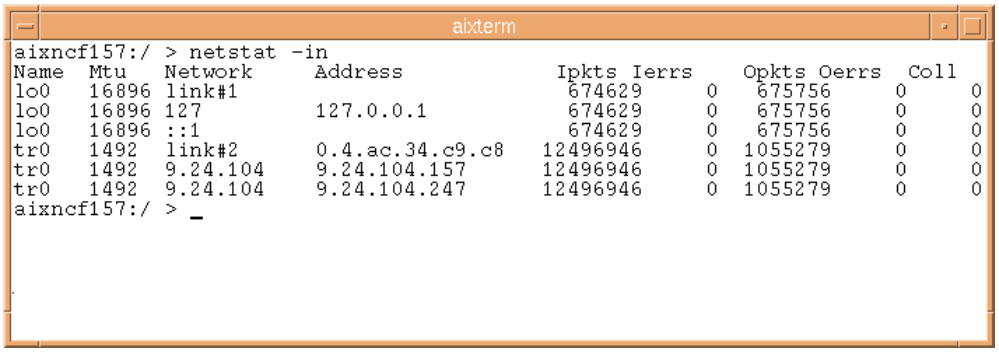
```
netstat -in
```

This command was executed on both machines. The output showed that the third IP address had been aliased to tr0 on the backup Dispatcher (currently, the active Dispatcher) and to lo0 on the primary Dispatcher (currently, the standby Dispatcher). This is shown in the following two figures:



```
# netstat -in
Name   Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
lo0    16896 link#1                          286518     0   286841     0     0
lo0    16896 127         127.0.0.1           286518     0   286841     0     0
lo0    16896 ::1                             286518     0   286841     0     0
lo0    16896 9.24.104    9.24.104.247        286518     0   286841     0     0
tr0    1492  link#2      8.0.5a.ce.6d.7f    1544202     0   401083     0     0
tr0    1492  9.24.104    9.24.104.128       1544202     0   401083     0     0
# hostname
rs600023
# _
```

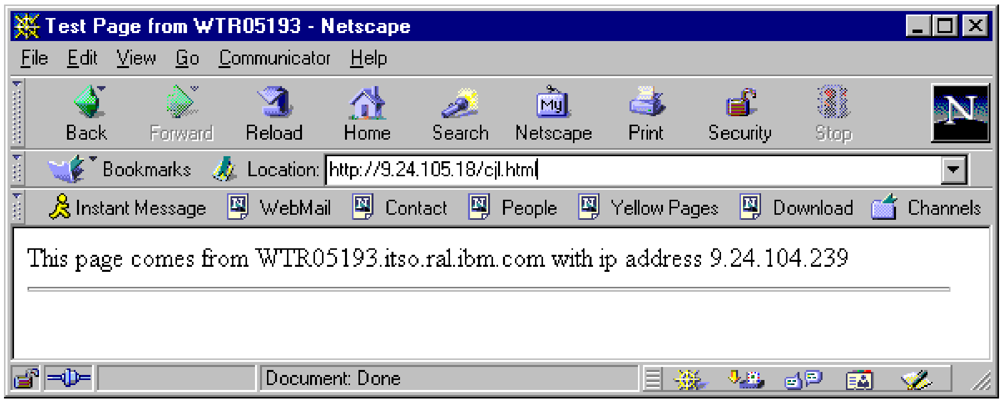
Figure 195. netstat -in Command Output from the Primary Dispatcher



```
aixncf157:/ > netstat -in
Name   Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
lo0    16896 link#1                          674629     0   675756     0     0
lo0    16896 127         127.0.0.1           674629     0   675756     0     0
lo0    16896 ::1                             674629     0   675756     0     0
tr0    1492  link#2      0.4.ac.34.c9.c8   12496946     0  1055279     0     0
tr0    1492  9.24.104    9.24.104.157      12496946     0  1055279     0     0
tr0    1492  9.24.104    9.24.104.247      12496946     0  1055279     0     0
aixncf157:/ > _
```

Figure 196. netstat -in Command Output from the Backup Dispatcher

Following this, to verify that the cluster was operational, we opened a browser on our Web client machine and requested the HTML file cjl.html from the cluster address, repeatedly. The page was returned successfully from the local server as well as the two remote servers, one of which appeared as follows:


Test Page from WTR05193 - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://9.24.105.18/cjl.html
Instant Message WebMail Contact People Yellow Pages Download Channels
Document: Done

This page comes from WTR05193.itso.ral.ibm.com with ip address 9.24.104.239

Figure 197. HTML Page Served by the Only Web Server in the Cluster

8.4.1.5 Configuration Summary

The following table summarizes the IP addresses configured on each of the machines in our scenario:

Table 17. WAN Dispatcher Support with Remote HA Configuration Summary

Role	IP Address – Non-Forwarding Address	Executor IP Address	Defined Cluster Address	Aliases
Entry Dispatcher	9.24.105.244	9.24.105.244	9.24.105.18	9.24.105.18 on tr0
Remote primary Dispatcher	9.24.104.128	9.24.104.247		9.24.104.247 on tr0 or lo0 – aliasing done by scripts
Remote backup Dispatcher	9.24.104.157			
Web server	9.24.105.181			9.24.105.18 on lo0
Web server	9.24.104.157			
Web server	9.24.104.239			

Chapter 9. Server Directed Affinity API

The load-balancing function in the Dispatcher component of IBM SecureWay Network Dispatcher (ND) distributes client requests to the servers that belong to a cluster of servers. If one client's transactions happen to use multiple TCP connections, it is very likely that those connections will be distributed to and handled by various servers. For most applications, such as serving Web pages, this is a very desirable behavior.

However, some protocols or applications would work more efficiently if the Dispatcher were to send all of the connections from a single client to the same server for a period of time. For example, a *shopping-cart* server application that does not have the ability to share state information between all of the server machines would require that the client come back to the same server for all of the TCP connections during a purchase. This mapping of client IP addresses to a selected server is the purpose of the classical Dispatcher affinity feature.

The ability to bind a client to a particular server has existed for several releases in the Dispatcher. This was accomplished by setting a *sticky time* value on the port. When the first request comes in from a client to the port, the Dispatcher selects a server to receive the request and then makes a record of this connection in one of its affinity tables. Each affinity record lives for as many seconds as specified by the sticky time value. Subsequent requests received from the same client within the sticky time value will be directed to the same server. Notice that any subsequent requests cause the sticky time to be reset back to its original value. For example, if sticky time is set to one hour, and 58 minutes into that hour the client does a request, that client now has another hour before the sticky time will expire, not just two minutes.

In response to customer demand, the *Server Directed Affinity* (SDA) application programming interface (API) has been introduced in this version of the Dispatcher. The purpose of the API is to supply a method to the customer's applications to allow the applications to select which server their client requests will be sent to, and also to control the length of time that the individual client requests will be sent to this specific server for. This has been accomplished by implementing a new socket listener within the Dispatcher to accept and handle requests to query and modify the server's affinity tables.

9.1 Server Directed Affinity Scenario

The example that we describe in this section shows how the SDA sample code can be modified to create an affinity table entry so that all requests from a particular client to port 80 at our cluster address would be handled by only one of our clustered Web servers. We also show the effect of creating this affinity from the client perspective.

Although we demonstrate creating this affinity entry from the command line, it would be more likely that the affinity entry would be created by an application program.

The application may base its decision about which server to connect the client to, on whatever information is appropriate to the application. The implementation of this application may take many forms, the simplest of which may be a CGI-BIN application or a Java servlet.

It is important to note that, with this introductory implementation of the SDA API, an affinity that already exists cannot be recreated. This implies:

- The affinity may be created by the application before the Dispatcher has had an opportunity to select a server to receive the request. This may be done by an application running on a server that creates affinity records for clients so that subsequent connections from the clients go to a different server, or the same server but a different port (thereby using a different affinity table).
- Another possibility is to use another program that is listening for requests to create affinity table entries from applications, and have it delay the creation of the affinity table entry until the Dispatcher-created entry has been cleared. In this way an application running on a server that already has an affinity table entry can make a request to have an affinity table entry created for it, when its Dispatcher-created affinity table entry expires.

We begin by showing an example situation where the use of SDA would be practical. We describe a servlet that we wrote for the purpose of counting the number of times a particular HTML file is accessed by a particular client. Instead of counting how many times the HTML file has been requested by all clients, as you've probably seen, it holds different counters for different clients.

We will show how to compile the servlet and place it on our three port 80 servers in our cluster environment. When we request the clustered Web page from our Web client repeatedly, we will see that our request is load balanced to all three servers. Without binding our client requests to one server, we see nonconsecutive results in our client-specific counter. This could be very confusing to the client, because from the client perspective he or she is making the same request repeatedly from what they believe to be the same server, when in fact it is a load-balanced clustered address. In the second part of our demonstration, we bind our client session to one server with the modified sample code, and see that the servlet client counter increments consecutively, as the end user would expect.

9.1.1 Servlet Client Counter Example

The servlet is called IncludedCounter. It is not supposed to be invoked directly by pointing to its URL, but it is invoked from within the `<SERVLET>` tag on HTML pages that have a .shtml extension. Its purpose is to produce a counter that can be dynamically embedded inside an HTML file that carries the .shtml extension. It counts how many times that HTML page has been accessed by a particular client machine. Instead of counting how many times the HTML file has been requested in general by all clients, it holds different counters for different clients. The output of this servlet is simply an integer, that is dynamically embedded inside the HTML page by using the servlet tag technique.

In addition, it is also possible that different HTML pages invoke this servlet, provided that they call it with different instance names, using the `Name` attribute of the `<SERVLET>` tag. In this way, there is an instance for each HTML page and each instance counts the number of times its related HTML page has been invoked.

Each instance of the IncludedCounter servlet uses a Hashtable object to register all the necessary information related to the HTML page invoking that instance. The java.util.Hashtable class implements a hash table, which maps keys to values.

In this case, when a client machine requests the HTML page related to a particular instance of the IncludedCounter servlet, its IP address is the key and the number of times that client machine invoked that HTML page is the value. IP addresses are held as string objects, while the counter is held by a support class, named StoredData.

The IncludedCounter servlet also provides an interesting example of how to write custom init() and destroy() methods. It is required, in fact, that the Hashtable objects where the servlet instances hold their data are saved in permanent storage, so that they are not lost when the servlet is destroyed and can be retrieved when it is reinitialized. For that reason, this servlet, when it is invoked from within the `<SERVLET>` tag of an HTML page, requires an initialization parameter, filename. This is the name of the file where the servlet instance stores the Hashtable object when the destroy() method for the servlet is called, typically when the server is shut down. Different HTML pages use different files to store their Hashtable objects, so that each HTML page invokes its own instance of the IncludedCounter servlet and each instance stores its Hashtable object in its own file.

It is not necessary for these files to be updated each time a client machine requests an HTML page that invokes that servlet, because this would create a slow mechanism for storing and retrieving data from the hard disk during the servlet life cycle. The IncludedCounter servlet presents custom init() and destroy() methods so that the Hashtable object is retrieved only when the servlet is initialized, and it is stored only when the servlet is destroyed. The destroy() method has been overridden in each instance to store the instance's Hashtable object in the file that has been specified as its initialization parameter.

In order to store an object, the Java 1.1 serialization mechanism is used. As we indicated, the Hashtable object used by one instance of the IncludedCounter servlet relates the IP address of a client machine to the number of times that the client machine has accessed that HTML page. IP addresses are held as string objects. Both the java.util.Hashtable and java.lang.String classes implement the java.io.Serializable interface. To store the integer counter number, a support class called StoredData is used. That class also implements java.io.Serializable, so that, in effect, serialization is possible.

The description that we have provided for the IncludedCounter servlet should help you understand the Java source code IncludedCounter.java, shown in the following two figures:

```
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class IncludedCounter extends HttpServlet
{
   String objFile;
   Hashtable table = new Hashtable();

   public void init(ServletConfig config) throws ServletException
   {
      super.init(config);
      log("init() method called");
      try
      {
         objFile = getInitParameter("Filename");
         if (objFile != null)
         {
            FileInputStream fis = new FileInputStream(objFile);
            ObjectInputStream ois = new ObjectInputStream(fis);
            table = (Hashtable)ois.readObject();
            fis.close();
         }
      }
      catch(ClassNotFoundException cnfe)
      {
         log("Class Not Found");
      }
      catch(FileNotFoundException fnfe)
      {
         log("File not found");
      }
      catch(IOException ioe)
      {
         log("Input/Output Error");
      }
   }

   public void service(HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException
   {
      log("service() method called");
      String remaddr = req.getRemoteAddr();
      int number;
```

Figure 198. (Part 1 of 2). Source of IncludedCounter.java

```
      synchronized(table)
      {
         StoredData counter = (StoredData)table.get(remaddr);
         if (counter != null)
            number = counter.increment();
         else
         {
            counter = new StoredData(1);
            table.put(remaddr, counter);
            number = 1;
         }
      }
      ServletOutputStream out = res.getOutputStream();
      out.println(number);
      out.close();
      log("service() method exit");
   }

   public void destroy()
   {
      log("destroy() method called");
      if (objFile != null)
      {
         try
         {
            FileOutputStream fos = new FileOutputStream(objFile);
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            synchronized(table)
            {
               oos.writeObject(table);
            }
            fos.close();
         }
         catch (Exception e)
         {
            log("Exception");
         }
      }
      super.destroy();
   }

   public String getServletInfo()
   {
      return "This servlet counts how many times each client accessed
         a specified .shtml page" ;
   }
}
```

Figure 199. (Part 2 of 2). IncludedCounter.java

The following figure shows the StoredData.java file used to wrap all the counters:

```
import java.io.*;
public class StoredData implements Serializable
{
   int number;
   public StoredData(int value)
   {
      number = value;
   }
   public int increment()
   {
      ++number;
      return number;
   }
}
```

Figure 200. StoredData.java

Finally, we want to show a simple HTML file that invokes the IncludedCounter servlet form within the `<SERVLET>` tag of an HTML page, for example, the file Page1.shtml, shown in the following figure:

```
<HTML>
   <HEAD>
      <TITLE>PAGE 1</TITLE>
   </HEAD>
   <BODY>
      <H1>PAGE 1</H1>
      This is the example of a SHTML page invoking the IncludedCounter
      servlet on server <B>9.24.104.157</B>
      <HR>
      The file containing the stored data is <B>Page1.obj</B>.
      <HR>
      <CENTER>
         <H2>
            You have invoked this page
               <H1>
                  <SERVLET
                     Code=IncludedCounter.class
                     Name=IncludedCounter1
                     Filename="/tmp/Page1.obj">
                  </SERVLET>
               </H1>
            times
         </H2>
      </CENTER>
   </BODY>
</HTML>
```

Figure 201. Page1.shtml

When the client’s browser requests the Page1.shtml file, the .shtml extension forces the Web server to search the pair of tags `<SERVLET>` and `</SERVLET>` within the HTML code. Everything between these two tags, plus the tags themselves, is replaced by the dynamic output of the IncludedCounter servlet.

Notice that this HTML page invokes the IncludedCounter servlet, so that the Web server automatically creates an instance of that servlet. This instance is called IncludedCounter1, according to the value of the `Name` attribute within the `<SERVLET>` tag. This instance is related to the Page1.shtml file. The Filename initialization parameter is passed to the servlet with the value /tmp/Page1.obj. This means that the Hashtable object related to the Page1.shtml file (used to register all the clients that have access to the Page1.shtml page and the number of their accesses) will be saved in the Page1.obj file.

In order for the Web server to be able to create the IncludedCounter1 servlet instance, the init() method is called. This method is used by the servlet to discover the value of the filename initialization parameter and to retrieve the Hashtable object, named table, from the file. Notice that the first time the servlet is run, the Page1.obj file will be empty and no Hashtable object will be retrieved. This situation is handled by simply catching an IOException.

The service() method works according to a very simple mechanism. First of all it retrieves the IP address of the client machine from where the Page1.shtml page was requested. This function is accomplished by the getRemoteAddr() method for the HttpServletRequest object, called req, and passed to the service() method. Then the service() method creates a StoredData object named counter, used to retrieve the number of accesses to the Page1.shtml file from the client machine. Also, the counter.increment() method is invoked to update the counter field.

If that client machine is accessing the Page1.shtml file for the first time, no counter object can be created starting from the table, so the counter's constructor is invoked, passing it the int value 1. In this case the table object must be updated with the IP address of the new client that accessed the Page1.shtml file and its relative StoredData object.

Finally the service() method produces the dynamic HTML portion to embed inside the Page1.shtml file.

The destroy() method is automatically invoked by the Web server only when the Web server itself is shut down. It is used to store the table object inside the Page1.obj file.

Notice that accesses to the table objects need to be synchronized in order to avoid multiple threads from multiple client requests causing problems when accessing the servlet concurrently.

Now that we have described how the IncludedCounter servlet works, we can experiment with it.

9.1.2 Experimenting with the Servlet

Before experimenting with the IncludedCounter servlet, it is a good idea to open the Preferences window for the Netscape browser to set the number of days after which pages in history expire to `0`. The Preferences window can be accessed from the Edit menu. You can also select **Advanced** and then **Cache** in the Preferences window and set Memory Cache and Disk Cache to `0 Kbytes`. In this way, you do not have to clear history, memory cache and disk cache for your browser each time you request the Page1.shtml file to experiment with the counter.

We compiled the IncludedCounter servlet and the StoredData class using the Java compiler that comes with the Java Development Kit (JDK) 1.1.6. The commands to issue are:

```
javac StoredData.java
javac IncludedCounter.java
```

The StoredData.java file must be compiled before compiling the IncludedCounter servlet, because IncludedCounter uses the StoredData class. Only if StoredData.java and IncludedCounter.java are in the same directory can you compile IncludedCounter directly, and the Java complier will automatically compile Notice that the full JDK 1.1.6 comes with the CD-ROM of WebSphere Performance Pack Version 2.

To compile the servlet the CLASSPATH environment variable must be set correctly, in order for the Java compiler to find the javax.servlet and javax.servlet.http packages. To do this, we exported the CLASSPATH environment variable on AIX as follows:

```
export CLASSPATH=/usr/lpp/IBMWebAS/lib/jsdk.jar:/usr/jdk_base/lib/classes.zip:.
```

CLASSPATH on Windows NT and Solaris

To set the CLASSPATH environment variable to contain the servlet library JAR file, jsdk.jar, use this command on Windows NT:

```
set CLASSPATH=C:\WebSphere\AppServer\lib\jsdk.jar;C:\jdk1.1.6\lib\classes.zip;.
```

Alternately, you can set the value above for the CLASSPATH system environment from the Control Panel, by double-clicking on the **System** icon, selecting the **Environment** tab, and editing the value for CLASSPATH.

Use this command on Solaris:

```
export CLASSPATH=/opt/IBMWebAS/lib/jsdk.jar:/opt/jdk_base/lib/classes.zip:.
```

Then we installed the class files StoredData.class and IncludedCounter.class, along with the associated Page1.shtml file, on the server with IP address 9.24.104.157, which was our aixncf157 host. We placed the two class files inside the servlet directory of WebSphere Application Server on our AIX system, /usr/lpp/IBMWebAS/servlets.

Servlet Directory on Windows NT

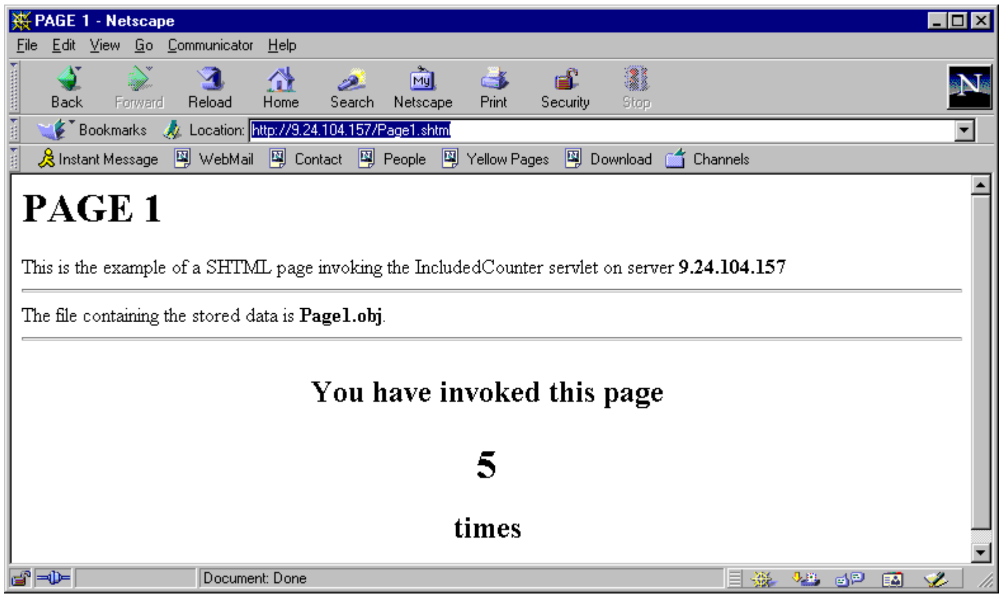
The default WebSphere Application Server servlet directory on Windows NT is C:\WebSphere\AppServer\servlets.

We placed the Page1.shtml file in our AIX Web server's document root directory, /usr/lpp/HTTPServer/share/htdocs.

Web Server Document root on Windows NT

The default IBM HTTP Server document root directory on Windows NT is C:\Program Files\IBM HTTP Server\htdocs.

We then used our browser to point to the Page1.shtml file and then clicked the **Reload** button of the Netscape browser window several times. We saw immediately how the IncludedCounter works. The dynamic portion embedded inside the Page1.shtml file was automatically updated each time, and each time the counter incremented consecutively. As an example, the fifth invocation is shown in the following figure:


PAGE 1 - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://9.24.104.157/Page1.shtml
Instant Message WebMail Contact People Yellow Pages Download Channels
Document: Done

PAGE 1

This is the example of a SHTML page invoking the IncludedCounter servlet on server **9.24.104.157**

The file containing the stored data is **Page1.obj**.

You have invoked this page

5

times

Figure 202. The Page1.shtml File Has Been Loaded Five Times

You can see that the counter, which is the dynamic portion of the HTML page, was automatically increased each time we reloaded the page.

In this case, we achieved the desired results above by directing our browser to the specific server (9.24.104.157) where we installed the servlet and its associated Page1.shtml file. If, however, we place the servlet class files and the SHTML file on multiple Web servers and access them through a Dispatcher that is load balancing our requests for this page across the servers, the results are less predictable. We discuss this issue in the next section.

9.1.3 Using the Servlet on a Cluster of Web Servers

The Dispatcher cluster environment in this scenario is shown in the following window:

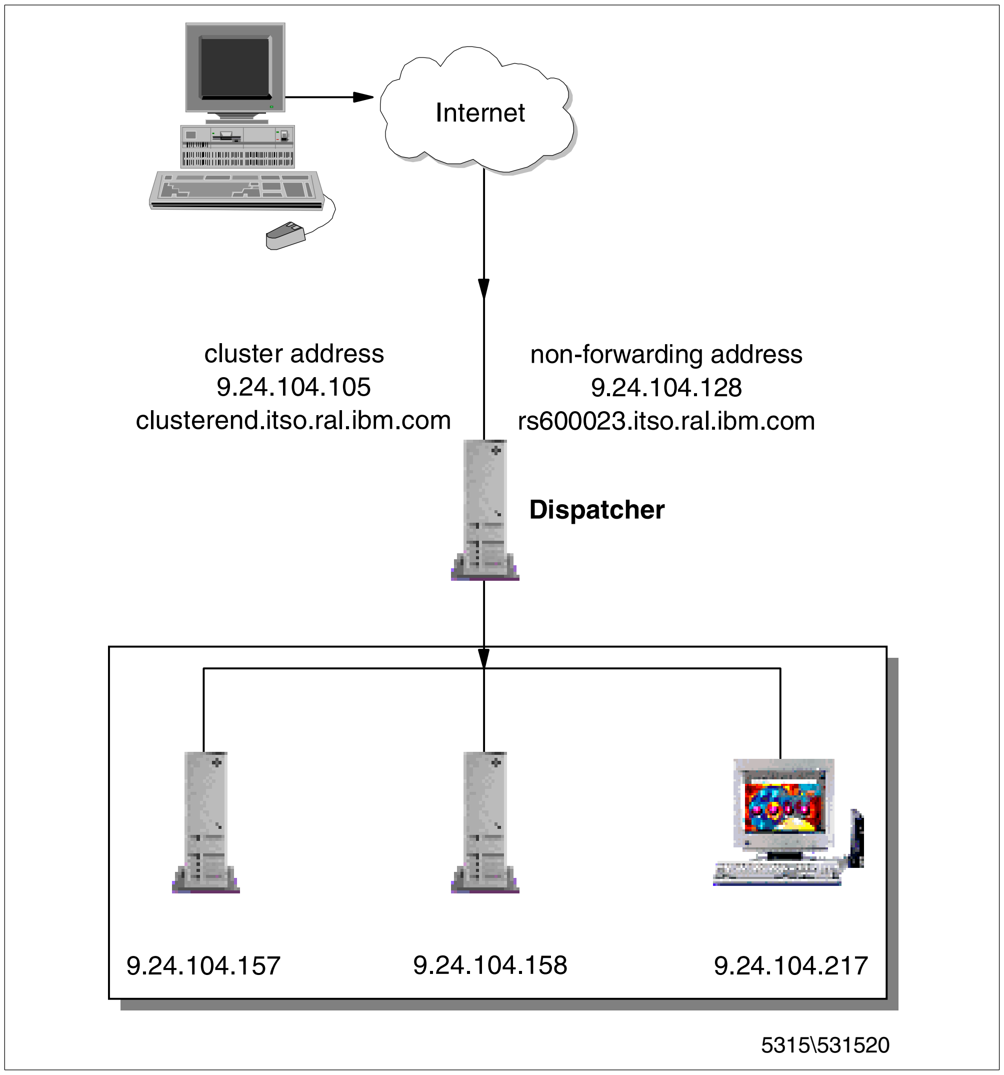


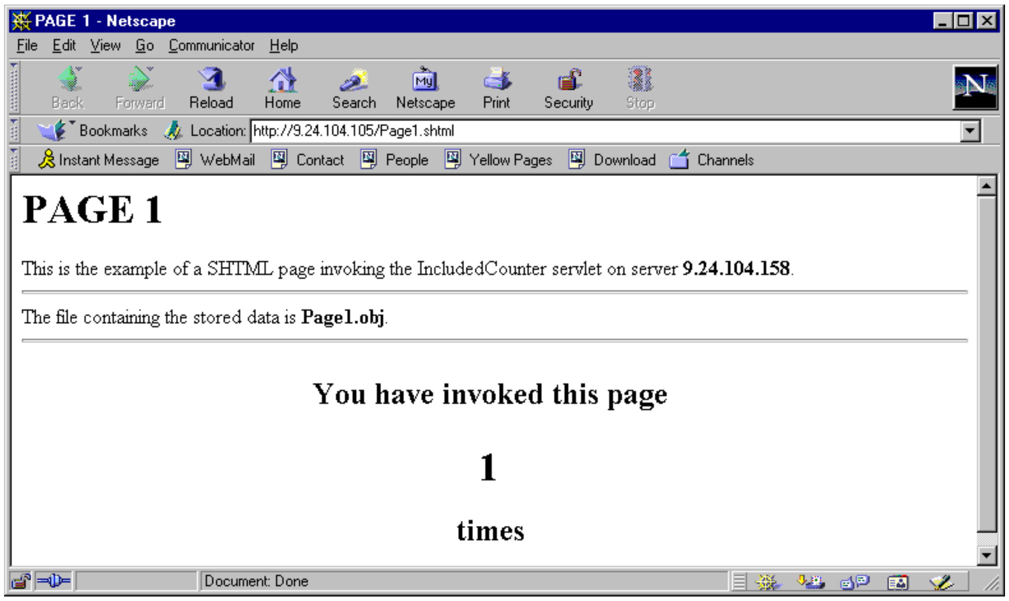
Figure 203. Dispatcher Cluster Environment for SDA Scenario

We placed a copy of the servlet and its associated Page1.shtml file on each of the three Web server machines in our scenario.

On our Windows NT Web server machine we modified the Page1.shtml file to point to the Page1.obj file as follows:

```
Filename="C:\\Objects\\Page1.obj”
```

Following this, we accessed the Page1.shtml file by using the cluster address 9.24.104.105 in our browser and saw the following sequence of pages, when we clicked the **Reload** button:


PAGE 1 - Netscape

http://9.24.104.105/Page1.shtml

PAGE 1

This is the example of a SHTML page invoking the IncludedCounter servlet on server **9.24.104.158**.

The file containing the stored data is **Page1.obj**.

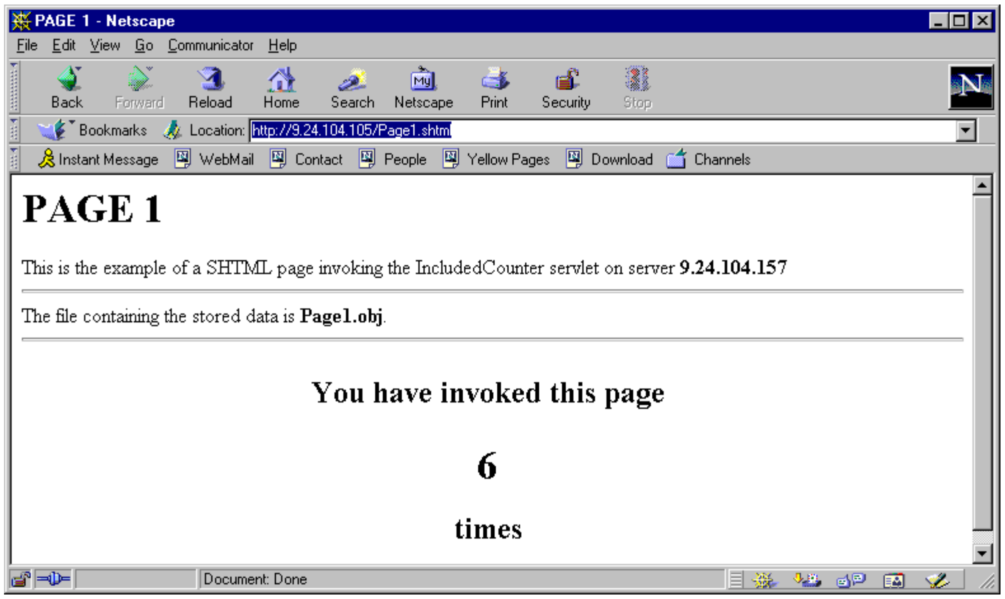
You have invoked this page

1

times

Document: Done

Figure 204. First Access of 9.24.104.105/Page1.shtml


PAGE 1 - Netscape

http://9.24.104.105/Page1.shtml

PAGE 1

This is the example of a SHTML page invoking the IncludedCounter servlet on server **9.24.104.157**

The file containing the stored data is **Page1.obj**.

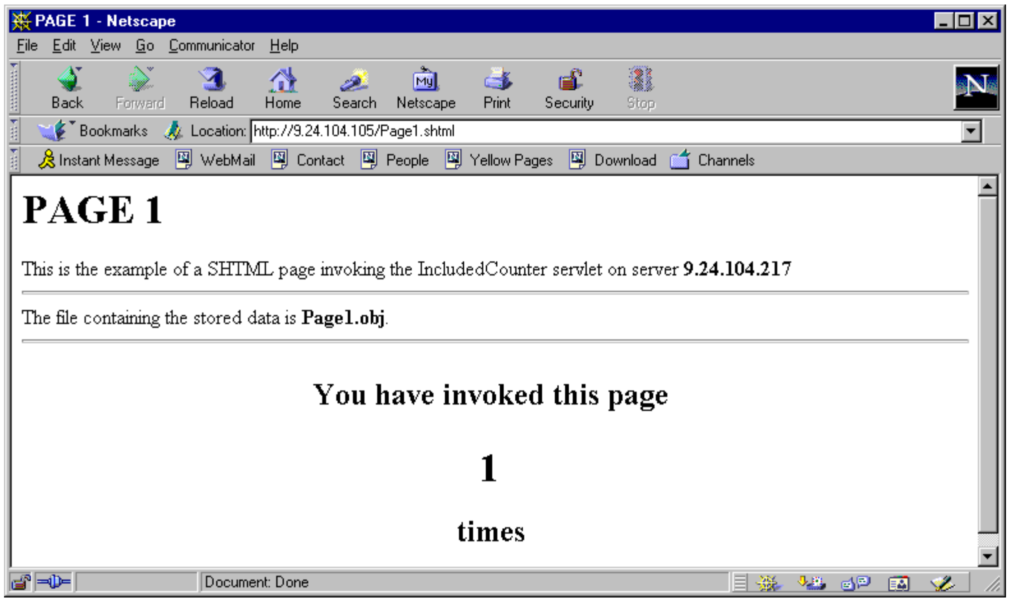
You have invoked this page

6

times

Document: Done

Figure 205. Second Access of 9.24.104.105/Page1.shtml


PAGE 1 - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://9.24.104.105/Page1.shtml
Instant Message WebMail Contact People Yellow Pages Download Channels
Document: Done

PAGE 1

This is the example of a SHTML page invoking the IncludedCounter servlet on server **9.24.104.217**

The file containing the stored data is **Page1.obj**.

You have invoked this page

1

times

Figure 206. Third Access of 9.24.104.105/Page1.shtml

The client counter number does not increase consecutively because each time we make a request for Page1.shtml, our request is load balanced by the Dispatcher and sent to a different Web server. This scenario would be very confusing to a customer accessing your Web site. Imagine what would happen if this application, instead of a counter, were a bank application.

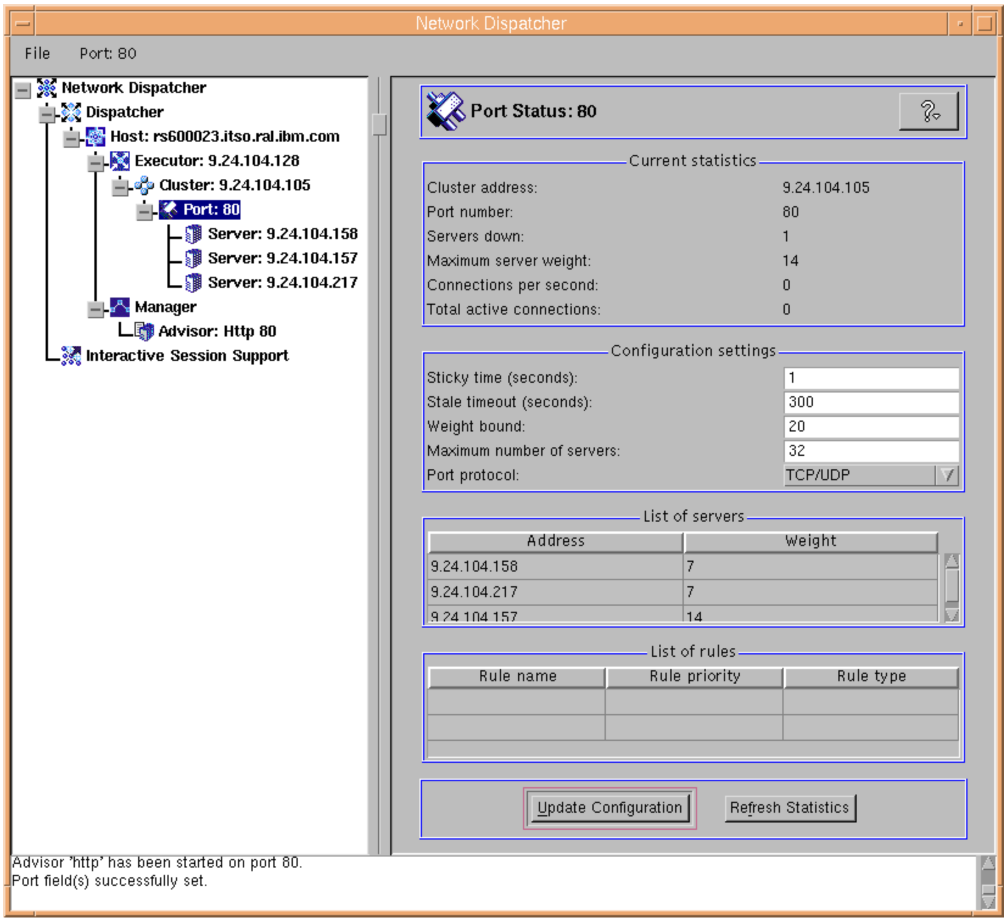
To prevent the client from seeing a not-consecutive counter, we could use the SDA API to bind the client to one of our servers, so subsequent requests by the client would result in a counter that increments as we would expect.

In the next section we show how to instruct the Dispatcher to listen for requests from SDA agents. The SDA agents will either query or modify the contents of the Dispatcher's affinity table for a particular port.

9.1.4 Enabling SDA

Each cluster/port must be individually configured to allow SDA access. To enable SDA access, you *must* define the port sticky time to be exactly one second. Anything different from that, and you will not have SDA.

In our scenario, we used the ND configuration GUI to change the port sticky time (seconds) value by typing `1` over the default value of `0` (representing no affinity) as shown in the following figure:



Current statistics

Cluster address:	9.24.104.105
Port number:	80
Servers down:	1
Maximum server weight:	14
Connections per second:	0
Total active connections:	0

Configuration settings

Sticky time (seconds):	1
Stale timeout (seconds):	300
Weight bound:	20
Maximum number of servers:	32
Port protocol:	TCP/UDP

List of servers

Address	Weight
9.24.104.158	7
9.24.104.217	7
9.24.104.157	14

List of rules

Rule name	Rule priority	Rule type

```
Advisor 'http' has been started on port 80.
Port field(s) successfully set.
```

Figure 207. Enabling the Dispatcher to Listen for SDA Requests

After changing the value, we clicked the **Update Configuration** button at the bottom of the GUI.

Command-Line Version

We could have done the same thing from the command line with the following command:

```
ndcontrol port set 9.24.104.105:80 stickytime 1
```

To verify that the sticky setting took effect, we used the following command:

```
ndcontrol port status 9.24.104.105:80
```

We got the following output:

```
Port Status:
------------
Port number ................... 80
Cluster address ............... 9.24.104.105
Number of servers ............. 3
Stale timeout ................. 300
Weight bound .................. 20
Port type ..................... tcp/udp
Maximum number of servers ...... 32
Sticky time ................... 1
```

The sticky time value of `1` confirms that we changed the setting successfully from the GUI.

9.1.5 Modifying the Sample SDA Client Code

The sample SDA client code is distributed as a Java source program in this directory *installbase*/lib/SDA, where *installbase* is for the Dispatcher component and varies by operating system (see Table 1 on page 69 for a list of *installbase* locations).

The SDA directory contains a file called SDA_API.htm and in it is a very good description of how the API works.

There are three other files in the directory as well:

- SDA_SampleAgent.java – is the main program.
- SDA_Info.class – holds affinity information in the format for transfer to the Dispatcher.
- SDA_Utils.class – implements miscellaneous utility routines to communicate with the Dispatcher.

SDA_Info.class and SDA_Utils.class were designed so that you could use them unchanged and intact, while replacing SDA_SampleAgent.java with your own code. The full code of the Java file SDA_SampleAgent is shown in Appendix B, “SDA_SampleAgent Code” on page 389.

In our case, we simply modified the supplied SDA_SampleAgent.java file to prepare it for use in our scenario:

- We changed the IP addresses in the variables used by the constructor to build the object used to communicate with the Dispatcher.

```
//----------------------------------------------------------------------------
  // Constructor.
  //----------------------------------------------------------------------------
  public SDA_Bind() throws Exception {

    // Note: These values are samples only.
    // You must modify them to work in your environment.

    // Define the address and port to communicate with ND.
    inaNetworkDispatcher      = InetAddress.getByName("9.24.104.128");
    // inaNetworkDispatcher      = InetAddress.getByName("10.0.0.3");
    iNetworkDispatcherPort    = 10005;

    // Define variables used as entries in the ND affinity table.
    inaSdaCluster             = InetAddress.getByName("9.24.104.105");
    iSdaPort                  = 80;
    inaSdaClient              = InetAddress.getByName("9.24.104.218");
    // inaSdaClient              = InetAddress.getByName("10.0.0.3");
    inaSdaServer              = InetAddress.getByName("9.24.104.157");
  }
```

Figure 208. Our Modified SDA_SampleAgent Constructor

- We also changed the name of the copied Java file to SDA_bnd218157.java to reflect the fact that it is used to bind the client machine with IP address ending in 218 to the Web server with IP address ending in 157. We changed all the references to `SDA_SampleAgent` in SDA_bnd218157.java, to `SDA_bnd218157`.
- We commented out the line in the main() method that performed the subsequent delete of the added entry, because we wanted to leave the entry in the affinity table for our test:

  ```
  // sa.deleteRecord()
  ```

As well, we made another version of the SDA_SampleAgent, which performed only the query function so we could verify what was contained in the affinity table. To do this we again copied SDA_SampleAgent.java to SDA_Query.java and:

- We also changed the name of the copied Java file to SDA_Query.java and changed all the references to `SDA_SampleAgent` in SDA_Query.java to `SDA_Query`.
- We commented out all of the lines in the main() method that performed operations on the affinity table, except for a single query.

We ensured that the CLASSPATH system environment variable contained the current directory:

- On AIX and Solaris:

  ```
  export CLASSPATH=$CLASSPATH:.
  ```

- On Windows NT

  ```
  set CLASSPATH=%CLASSPATH%;.
  ```

Then we compiled SDA_bnd218157.java and SDA_Query.java:

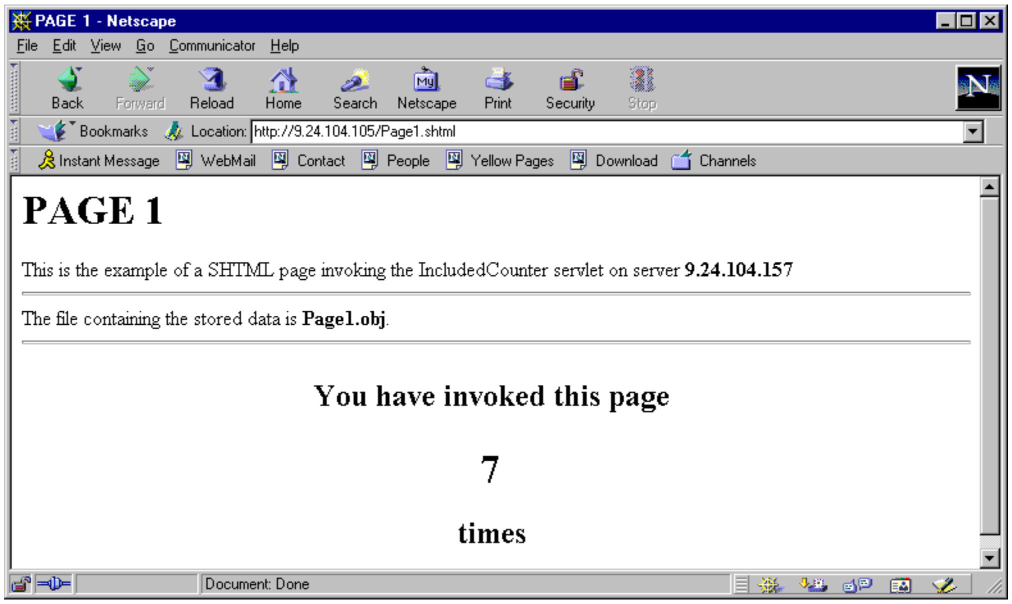
```
javac SDA_bnd218157.java
javac SDA_Query.java
```

These programs, can be launched from a command line by using the `java` command. After running SDA_bnd218157, the output from SDA_Query appeared as follows:

```
SDA_Sample> Program sdacjl , version 1.0, March 14, 1998.
SDA_Sample> Opening connection with Network Dispatcher:
            Address ........ rs600023.itso.ral.ibm.com/9.24.104.128
            Port ........... 10005
SDA_Sample> Opened connection successfully.
SDA_Sample> About to send an auth string to ND...
SDA_Sample> About to send a CIB string to ND...
SDA_Sample> Waiting for an auth string from ND...
SDA_Sample> Successfully received an auth string from ND...
SDA_Sample> About to query the contents of the ND affinity table:
SDA_Info object:
------------------------------------------------
MessageVersion .... 1
Command ........... 4 (SDA_CMD_QUERY)
Response .......... 0 (SDA_RSP_SUCCESS)
ClusterAddr ....... 9.24.104.105
Port .............. 80
NumAffinities ..... 0

SDA_Sample> About to send an auth string to ND...
SDA_Sample> Waiting for an auth string from ND...
SDA_Sample> Successfully received an auth string from ND...
SDA_Sample> Received response from ND:
SDA_Info object:
------------------------------------------------
MessageVersion .... 1
Command ........... 4 (SDA_CMD_QUERY)
Response .......... 0 (SDA_RSP_SUCCESS)
ClusterAddr ....... 9.24.104.105
Port .............. 80
NumAffinities ..... 1
Record 0:
  Response ........ 0 (SDA_RSP_SUCCESS)
  ClientAddr ...... 9.24.104.218
  ServerAddr ...... 9.24.104.157

SDA_Sample> Closing socket in routine main.
```

This verified that we had successfully bound our client IP address 9.24.104.218 to the server's IP address 9.24.104.157 in the affinity table for our cluster 9.24.104.105 port 80 on our 9.24.104.128 Dispatcher machine.

Subsequent requests by our client machine were as follows:


PAGE 1 - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://9.24.104.105/Page1.shtml
Instant Message WebMail Contact People Yellow Pages Download Channels
Document: Done

PAGE 1

This is the example of a SHTML page invoking the IncludedCounter servlet on server **9.24.104.157**

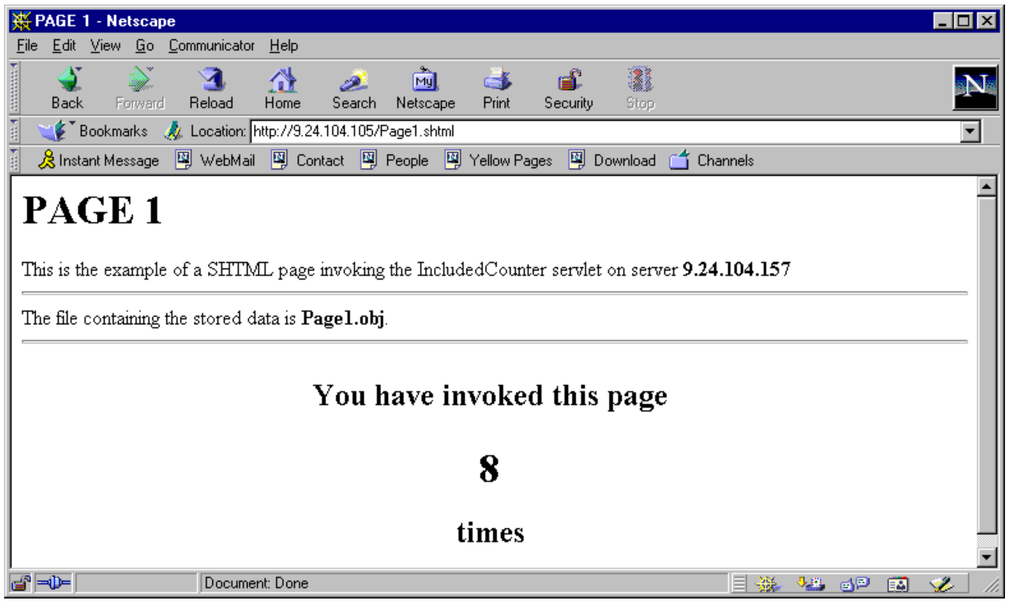
The file containing the stored data is **Page1.obj**.

You have invoked this page

7

times

Figure 209. First Request from the Client Machine with SDA Enabled


PAGE 1 - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://9.24.104.105/Page1.shtml
Instant Message WebMail Contact People Yellow Pages Download Channels
Document: Done

PAGE 1

This is the example of a SHTML page invoking the IncludedCounter servlet on server **9.24.104.157**

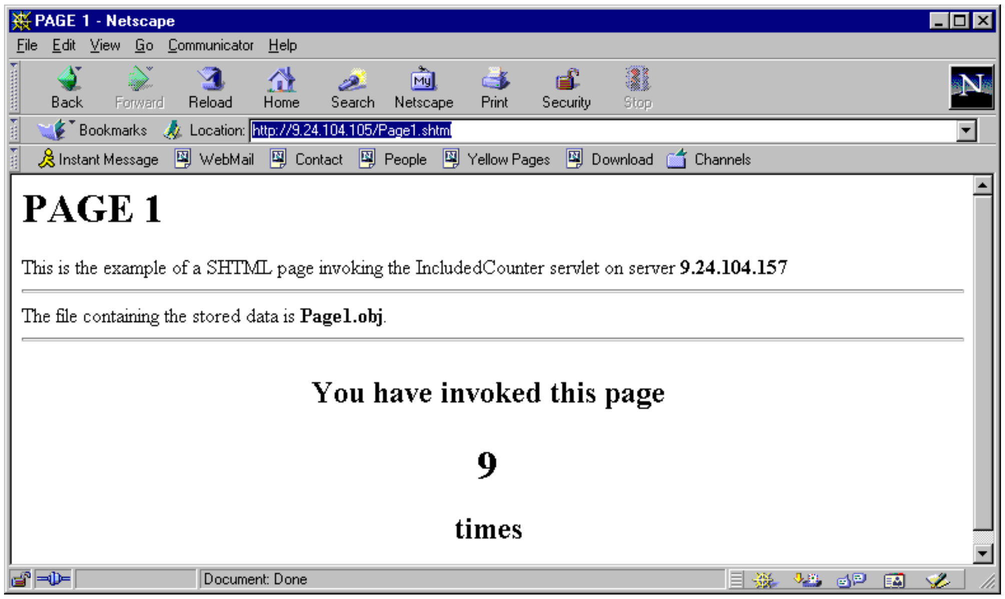
The file containing the stored data is **Page1.obj**.

You have invoked this page

8

times

Figure 210. Second Request from the Client Machine with SDA Enabled


PAGE 1 - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://9.24.104.105/Page1.shtml
Instant Message WebMail Contact People Yellow Pages Download Channels
Document: Done

PAGE 1

This is the example of a SHTML page invoking the IncludedCounter servlet on server **9.24.104.157**

The file containing the stored data is **Page1.obj**.

You have invoked this page

9

times

Figure 211. Third Request from the Client Machine with SDA Enabled

Any subsequent client requests for port 80 from this particular client machine would continue to be routed to 9.24.104.157 until we explicitly remove the affinity table entry.

9.1.6 A Different Scenario Implementation

Another possibility would have been for us to modify our IncludedCounter servlet so that for each new client request to Page1.shtml, an entry would be added to the Dispatcher's affinity table to bind requests from our client machine to whichever Web server machine was first selected to serve it. We could have changed the of SDA_SampleAgent.java as follows:

- Copy the SDA_SampleAgent.java file to a new file and change all references to the distributed name `SDA_SampleAgent` to the application's new name, such as `SDA_Bind`, for example.
- In your SDA_Bind.java file, change the constructor to allow another type of instantiation of the object to accept the client IP address and the Web server IP address as parameters. In this constructor, you could hardcode the nonforwarding address of your Dispatcher and cluster IP address, and dynamically obtain the client IP address and Web server IP address, as shown in the following figure:

```
 //----------------------------------------------------------------------------
  // Constructor.
  //----------------------------------------------------------------------------
  public SDA_Bind() throws Exception {

    // Note: These values are samples only.
    // You must modify them to work in your environment.

    // Define the address and port to communicate with ND.
    inaNetworkDispatcher      = InetAddress.getByName("9.37.52.219");
    // inaNetworkDispatcher      = InetAddress.getByName("10.0.0.3");
    iNetworkDispatcherPort    = 10005;

    // Define variables used as entries in the ND affinity table.
    inaSdaCluster             = InetAddress.getByName("9.37.61.44");
    iSdaPort                  = 80;
    inaSdaClient              = InetAddress.getByName("9.37.52.219");
    // inaSdaClient              = InetAddress.getByName("10.0.0.3");
    inaSdaServer              = InetAddress.getByName("9.37.52.114");
  }

    public SDA_Bind(String clientIP, String serverIP) throws Exception {

    // We added this section of the constuctor to accept two parameters
    // and hardcoded our Dispatcher's non forwarding address and cluster address

    // Define the address and port to communicate with ND.
    inaNetworkDispatcher      = InetAddress.getByName("9.24.104.128");
    iNetworkDispatcherPort    = 10005;

    // Define variables used as entries in the ND affinity table.
    inaSdaCluster             = InetAddress.getByName("9.24.104.105");
    iSdaPort                  = 80;
    inaSdaClient              = InetAddress.getByName(clientIP);
    inaSdaServer              = InetAddress.getByName(serverIP);
         System.out.println("ND=" +  inaNetworkDispatcher +";ND_Port=" +   iNetw
orkDispatcherPort  + ";SdaCluster=" + inaSdaCluster  + ";inaSdaClient=" + inaSda
Client + ";inaSdaServer=" + inaSdaServer);
   }
```

Figure 212. Possible Modified SDA_SampleClient Constructor

- You could also remove the main() method of the program.
- Place your modified code along with the two other distributed SDA class files (SDA_Utils and SDA_Info) in your WebSphere Application Server servlet directory and perform the Java compile.

To make use of the above, you could modify the service() method of the IncludedCounter.java program shown in Figure 198 on page 246 and Figure 199 on page 247 so that for each new client request to Page1.shtml, an entry would be added to the Dispatcher's affinity table. The part to add to the service() method of the IncludedCounter.java program follows:

```
public void service(HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException
{
   log("service() method called");
   String remaddr = req.getRemoteAddr();
   String serverName = (InetAddress.getLocalHost()).getHostAddress();
   int number;
   synchronized(table)
   {
      StoredData counter = (StoredData)table.get(remaddr);
      if (counter != null)
         number = counter.increment();
      else
      {
         counter = new StoredData(1);
         table.put(remaddr, counter);
         number = 1;
         try
         {
            // Instantiate SDA_Bind class
            SDA_Bind sa = new SDA_Bind(remaddr, serverName);

            // Open a connection with ND and identify ourself.
            sa.openND();

            // Add the new affinity record.
            sa.addRecord();

            // Query the contents of the ND affinity table.
            sa.queryTable();
         }
         catch(Exception e)
         {
            log("Excecption while executing SDA_Bind");
         }
      }
   }
```

Figure 213. Modified Service Function of IncludedCounter.java

You would then distribute your three SDA class files (the modified SDA_SampleAgent, SDA_Utils and SDA_Info) and the modified servlet class file(IncludedCounter) to your Web server machines. Repeated client requests for the same Page1.shtml as before should result in the entry being added to the Dispatcher's affinity table, binding the client machine to whichever server the Dispatcher sends it to the first time. The client counter numbers should increment sequentially.

We could have put a timer inside of our IncludedCounter servlet to purge the entry from the affinity table. Your application may have some other criteria on which to base the decision as to when to remove the affinity entry, such as when a particular transaction is completed.

Chapter 10. Custom Advisors

When Advisors are enabled within the Dispatcher or CBR components of IBM SecureWay Network Dispatcher (ND), they measure actual client response time for a particular protocol by periodically initiating a client-like exchange with the back-end servers. These results are then fed to the Manager to adjust the load-balancing weights. Currently, there are Advisors available for HTTP, FTP, SSL, SMTP, NNTP, POP3 and Telnet, and three new Advisors have been added in this version, namely ping, WTE[1], and WLM[2]. Using the Advisors is optional, but recommended. You also have the option of writing your own custom Advisors, which are Java programs and can be compiled using the Java Development Kit (JDK) 1.1.6, shipped with the product.

The Advisor component has been implemented so that most of the work is done by the Advisor base code ADV_Base. This is true for both the supplied Advisors and custom Advisors. The supplied and custom Advisors are small pieces of Java code that extend the functionality of the ADV_Base base code simply by sending and receiving user-defined data that is appropriate to the particular protocol service whose health it is attempting to assess.

The Advisor base code ADV_Base provides these administrative services to both supplied and custom Advisors:

- Socket connect and close operations
- Send and receive methods
- Methods to start and stop the Advisor
- Methods that provide status and report workload information back to the Manager
- Methods that record history information to the log file

When a custom Advisor is created, a mode setting in its constructor defines how the Advisor base code will determine the workload value to pass back to the Manager component on behalf of the custom Advisor.

The Advisor base code determines the load value by using one of these two methods:

- The Advisor base code monitors the duration of the custom Advisor's data exchange with the server and converts this to a load value. This is referred to as *normal mode*.
- The Advisor base code simply passes on the load value returned from the custom Advisor. This is referred to as *replace mode*.

With this feature, you have the flexibility of writing a very simple Advisor that makes a basic connection to a port and allows the Advisor base code to time the exchange. Alternatively, you can implement a more complex Advisor that executes some code that will more accurately assess the load value of the servers.

[1] The WTE advisor is named after IBM Web Traffic Express (WTE), the Caching and Filtering component of IBM WebSphere Performance Pack Version 2. This Advisor is specific for WTE (see 1.5.5.2, "Peak Load Management" on page 16).
[2] The Workload Manager (WLM) Advisor will be discussed in 10.2, "Workload Manager Advisor" on page 271

10.1 Custom Advisor Scenarios

In this section, we show an example of a normal mode custom Advisor and a replace mode custom Advisor.

10.1.1 Normal Mode Custom Advisor

A sample custom Advisor is provided with the Dispatcher and CBR components of ND. The sample code is located in the directory *installbase*/lib/CustomAdvisor/ADV_sample.java, where *installbase* varies by component (Dispatcher or CBR) and operating system. See Table 1 on page 69 for a list of the *installbase* locations.

The provided ADV_sample.java file is essentially the same as the HTTP Advisor. You can view the ADV_sample.java code in Appendix A, “ADV_sample Custom Advisor” on page 383.

We use ADV_sample.java as our example of a normal mode custom Advisor.

10.1.2 Replace Mode Custom Advisor

For an example of a replace mode custom Advisor, we have chosen to implement a custom Advisor that assesses the load information on a TN3270E server. We chose to use this TCP server type in our example because an increasingly popular use of the Dispatcher is to load balance requests to the TN3270E server option of the IBM 2210 or 2216 routers.

10.1.2.1 TN3270E Background

Many companies today have consolidated their wide area network (WAN) traffic into a single IP-only backbone. At the same time, other companies are simplifying their workstation configuration and attempting to run only the TCP/IP protocol stack at the desktop. However, most of these companies still require access to System Network Architecture (SNA) hosts.

TN3270 meets these requirements by allowing you to run IP from the desktop over the network and attach to your SNA host through a TN3270 server. The clients connect to the server using a TCP connection. The server provides a gateway function for the downstream TN3270 clients by mapping the client sessions to SNA-dependent LU-LU sessions that the server maintains with the SNA host. The TN3270 server handles the conversion between the TN3270 data stream and a SNA 3270 data stream.

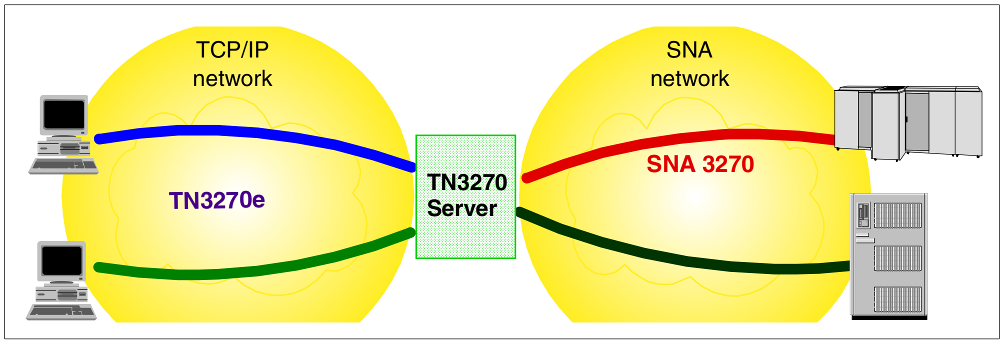


Figure 214. TN3270 Scenario

The Network Utility option of the IBM 2210 or 2216 routers implements a TN3270 server function with extended capabilities (hence the TN3270E nomenclature). It can be deployed in several configurations. For example, the TN3270E server function can be placed either in the remote branch or in the data center where the SNA host resides. It can attach to the host via a traditional SNA subarea connection or it can use Advanced Peer-to-Peer Networking (APPN). The solution can also scale to very large configurations while providing high availability and redundancy.

10.1.2.2 Custom Advisor Java Code

While a traditional Dispatcher cluster configuration can be used to load balance the TN3270e server requests, the ND code as distributed does not currently include an Advisor for this type of TCP server. The following code is offered as an example implementation of a custom Advisor that operates in replace mode to assess the performance capabilities of TN3270E servers:

```
//==================================================================================
// ADV_tn3r:
//
// This is a custom Advisor meant for use with IBM 2216 routers configured as TN3270 servers.
// It exploits a special control port available in the server to report on
// its health and loading.
//
// This is *NOT* a general-purpose TN3270 Advisor.
//
// Note: This Advisor runs on the software-platform Network Dispatcher.
// It has no interaction with the Network Dispatcher function in the 2216.
//================================================================================//
Implementation notes:
//
// This Advisor does not use the socket opened by ND on the protocol-specific port.
// Instead, it opens another java socket to the server on a 'control port'.
// The server immediately sends a number (an ascii string) which indicates
// the health of the server.
//
// The server returns number values in the range 1-100 and 9999, where:
//       1 means a fast server
//     100 means a slow server
//    9999 means a very slow server (do not mark down, but send no more connections.)
// Within this Advisor, the range of these values is checked, and optionally a
// normalization may be done.  These values are inserted into the Manager's
// port-specific table in the manager report as normal load values.
// In the constructor, super(), we specify 'true'.  This tells the
// ND Advisor code use our load number in 'replace' mode rather than
// using ND's timestamp value.
//==================================================================================
// We place this code in the archive as an example of an Advisor which opens its
// own socket to the to the server, extracts customer-proprietary data from
// the server, and returns it as a load value.
//==================================================================================
package CustomAdvisors;
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.Date;
import com.ibm.internet.nd.advisors.*;
import com.ibm.internet.nd.common.*;
import com.ibm.internet.nd.server.SRV_ConfigServer;

public class ADV_tn3r extends ADV_Base implements ADV_MethodInterface
{
  String COPYRIGHT = "(C) Copyright IBM Corporation 1997, All Rights Reserved.\n";

  static final String  ADV_NAME              = "TN3R";
  static final int     ADV_DEF_ADV_ON_PORT   = 23;
  static final int     ADV_DEF_CONTROL_PORT  = 10008;
  static final int     ADV_DEF_INTERVAL      = 7;
```

Figure 215. (Part 1 of 4). IBM 2216 TN3270E Custom Advisor

```
/**
  * Constructor.  Parms:  None;
  * but the constructor for ADV_Base has several parameters that must be passed to it.
  */
 public ADV_tn3r()
 {
   super( ADV_NAME,
          "2.0.0.0-12.08.98",
          ADV_DEF_ADV_ON_PORT,
          ADV_DEF_INTERVAL,
          "",     // not used
          true);  // replace mode
   super.setAdvisor( this );
 }

 /**
  * ADV_AdvisorInitialize*
  * Any Advisor-specific initialization that must take place after the Advisor base is started.
  * This method is called only once and is typically not used.
  */
 public void ADV_AdvisorInitialize()
 {
   return;
 }

 /**
  * normalizeLoad()
  * AD 12/21/1998 48102
  */
 int normalizeLoad(int iLoadOld) {
   int iLoadNew;
   // This routine implements a 'graph' with two straight lines.
   // For load values below a knee, the load value is returned unchanged.
   // This section of the 'graph' has a steady 1:1 slope.
   // For values above the knee, a value which rapidly grows up to 9999
   // is returned.  This section of the 'graph' has a very steep slope.
   // Define the load value at which the graph becomes very steep.
   int iKnee = 85;  // Don't use 100 (lest you divide by zero)!

   if (9999 == iLoadOld) {
     iLoadNew = iLoadOld;
   }
   else if (iKnee >= iLoadOld) {
     iLoadNew = iLoadOld;
   }
   else { // Between the knee and 100 inclusive.
     int iDelta = (9999 - 100) / (100 - iKnee);
     iLoadNew = iKnee + (iLoadOld - iKnee) * iDelta;
   }
   return iLoadNew;
 }
```

Figure 216. (Part 2 of 4). IBM 2216 TN3270E Custom Advisor

```
 /**
   * getLoad()
   */
  public int getLoad(int iConnectTime, ADV_Thread caller)
  {
   int iRc;
   int iLoad = ADV_HOST_INACCESSIBLE;  // -1
    boolean bLoadValid = false;
    BufferedReader brIn = null;
    Socket soServer = null;

    // Notice: This Advisor does not use the socket connection which has
    // been opened for us by the ND base classes.  We open a new java
    // socket connection with the server on a special port to fetch
    // health information from the server.

    // Get the name of the server on which we are advising.
    String sServer = caller.getCurrentServer();
    // System.out.println("ADV_tn3r: Entry.  Starting a new advise cycle for server " + sServer);

    try {
      // Open a new socket with the server machine's control port.
      // System.out.println("ADV_tn3r: Opening socket with server: " + soServer);
      soServer = new Socket(sServer, ADV_DEF_CONTROL_PORT);
      // System.out.println("ADV_tn3r: Socket opened successfully: " + soServer);

      // Extract a stream for reading.
      brIn = new BufferedReader(new InputStreamReader( soServer.getInputStream() ));

      // Set a max receive timeout so we do not wait forever.
      int iRecvTimeout = 10000;  // milliseconds.
      soServer.setSoTimeout(iRecvTimeout);

      // Read a load value from the server's control port.
      // System.out.println("ADV_tn3r: Reading response.");
      String sResponse = brIn.readLine();
      // System.out.println("ADV_tn3r: After reading.  Got response >>>" + sResponse + "<<<");

// Convert the response from a string to an int.
      if (null != sResponse) {
        if ((0 < sResponse.length()) && (6 > sResponse.length())) {
          String sResponseTrim = sResponse.substring(0, 4);
         // System.out.println("ADV_tn3r: After substring, sResponseTrim = >>>" + sResponseTrim +
"<<<");
          iLoad = new Integer(sResponseTrim).intValue();
          bLoadValid = true;
          // System.out.println("ADV_tn3r: Converted to an int.  iLoad=" + iLoad);
        }
        else {
          // System.out.println("ADV_tn3r: Error: length is " + sResponse.length());
        }
      }
```

Figure 217. (Part 3 of 4). IBM 2216 TN3270E Custom Advisor

```
// Range-check the load value.  9999 is ok.  Otherwise limit to 1-100.
      if (true == bLoadValid) {
        if (9999 != iLoad) {
          if (1 > iLoad) {
            iLoad = 1;
          }
          else if (100 < iLoad) {
            iLoad = 100;
          }
        }
      }

      // AD 12/22/1998 48102 - Optionally normalize the load value if desired.
      iLoad = normalizeLoad(iLoad);

      // System.out.println("ADV_tn3r: After range-check, iLoad=" + iLoad);
    }

    catch(Exception e) {
      // System.out.println("ADV_tn3r: Caught exception " + e );
    }

    // Always try to close the socket.
    try {soServer.close();}  catch ( Exception z ) {;}

    // System.out.println("ADV_tn3r: Exit. Returning " + iLoad);

  return iLoad;
  }

} // End - ADV_tn3r
```

Figure 218. (Part 4 of 4). IBM 2216 TN3270E Custom Advisor

This Advisor has the following unique features:

- It does not make use of the TCP socket that has been opened by the base ND classes on the client port number. Instead, it opens its own Java socket to the server on a special *control port number*. From there, it gets a number from the server indicating the load on the server machine.
- It operates in replace mode. Instead of relying upon the base ND Advisor classes to make a time measurement of the health of the server, it overtly returns a load number that is accepted by ND directly as a load number.

10.1.2.3 Custom Advisor Naming and Compiling

The custom Advisor filename must start with `ADV_` in uppercase and the rest of the name in lowercase. As per Java conventions, the class defined within the file must match the name of the file, and the file must carry a .java extension.

In order to compile the custom Advisor with the Java compiler `javac`, the CLASSPATH system environment variable must contain the ND base class file, the Java class file and the directory where the custom Advisor is located. We set the CLASSPATH as follows:

- On Windows NT:

```
set CLASSPATH=C:\WSPP\IBM\nd\dispatcher\lib\ibmnd.jar;C:\jdk1.1.6\lib\classes.zip;.
```

• On AIX:

```
export CLASSPATH=/usr/lpp/nd/dispatcher/lib/ibmnd.jar:/usr/jdk_base/lib/classes.zip:.
```

After setting the CLASSPATH, we compiled our two Advisors:

```
javac ADV_sample.java
javac ADV_tn3r
```

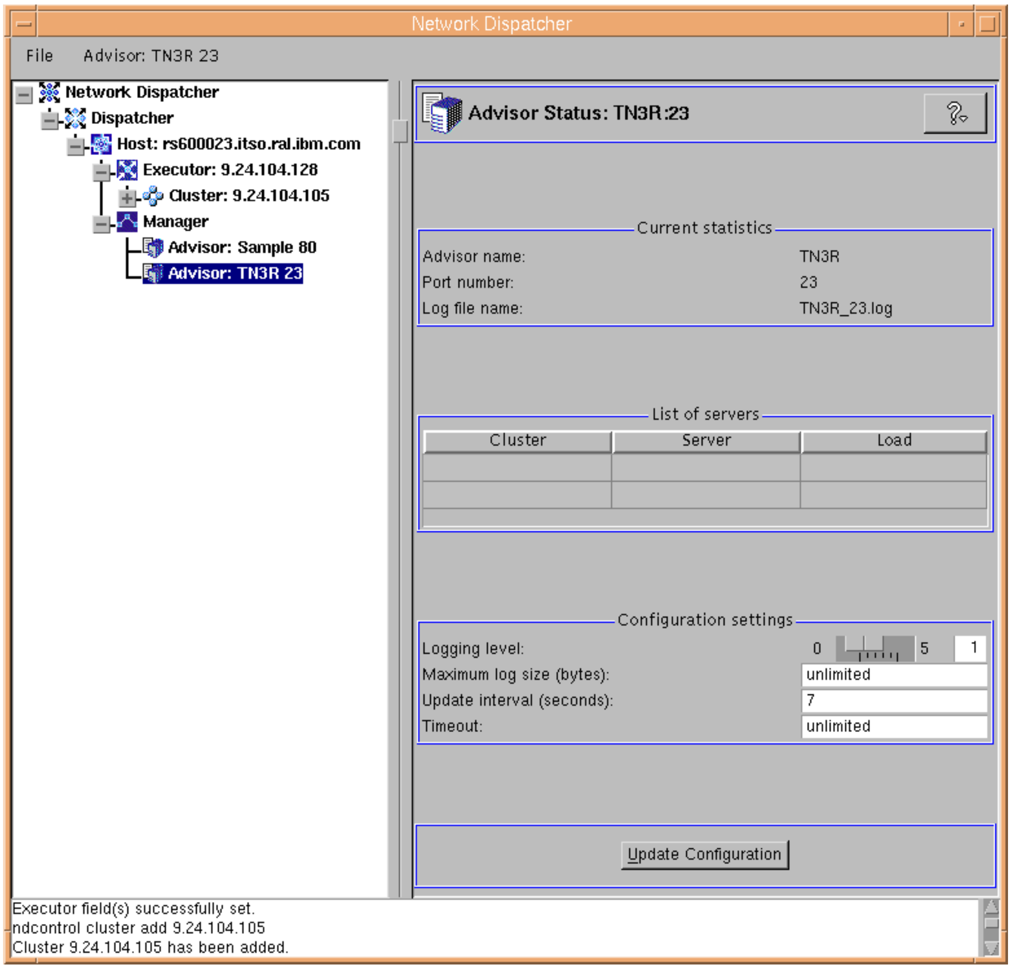
10.1.2.4 Using the Custom Advisor

Once compiled, we placed the custom Advisor class files in the directory *installbase*/lib/CustomAdvisor, where *installbase* varies by component (Dispatcher or CBR) and operating system. See Table 1 on page 69 to determine your *installbase* directory.

We started using the Advisors within the Dispatcher with these commands:

```
ndcontrol advisor start sample 80
ndcontrol advisor start tn3r 23
```

Although you cannot use the ND graphical user interface (GUI) to start your custom Advisors, once they are started from the command line, their use can be customized and stopped from the GUI:



Advisor name:	TN3R
Port number:	23
Log file name:	TN3R_23.log

Cluster	Server	Load

Logging level:	0 5 1
Maximum log size (bytes):	unlimited
Update interval (seconds):	7
Timeout:	unlimited

```
Executor field(s) successfully set.
ndcontrol cluster add 9.24.104.105
Cluster 9.24.104.105 has been added.
```

Figure 219. Custom Advisor Management through the GUI

For details on how to customize the Manager component to work with the Advisors, see 4.1.10, “Customization of the Manager for the Advisors” on page 130.

10.2 Workload Manager Advisor

The Workload Manager Advisor (WLM) is a Dispatcher Advisor that obtains capacity information from Multiple Virtual Storage (MVS) Workload Management software running on an OS/390 system.

There are several important differences between the WLM and the other Dispatcher Advisors:

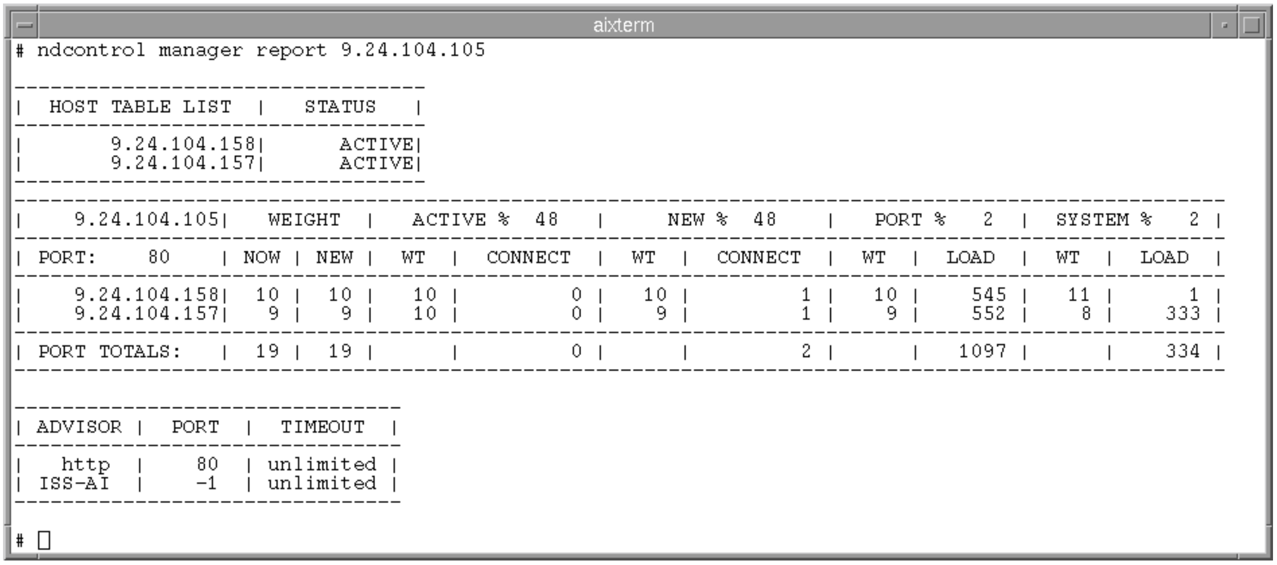
- Other Advisors open connections to the servers using the same port on which normal client traffic flows. The WLM Advisor opens connections to the servers using a port different from normal traffic. The WLM agent on each server machine must be configured to listen on the same port on which the Dispatcher WLM Advisor is started. The default WLM port is 10007.
- Other Advisors only assess those servers defined in the Dispatcher `cluster:port:server` configuration for which the server’s port matches the Advisor’s port. The WLM Advisor advises upon every server in the Dispatcher `cluster:port:server` configuration. Therefore, you must not define any non-WLM servers when using the WLM Advisor.
- Other Advisors place their load information in the Manager report under the Port column. The WLM Advisor places its load information in the Manager report under the System column.
- It is possible to use both protocol-specific Advisors along with the WLM Advisor. The protocol-specific Advisors will poll the servers on their normal traffic ports, and the WLM Advisor will poll the system load using the WLM port.

10.2.1 ISS Restriction

As with ISS, the WLM Advisor reports on server systems as a whole, rather than on individual protocol-specific services. Both ISS and WLM place their results in the System column of the manager report. As a consequence, running both the WLM Advisor and ISS at the same time is not supported.

Following is an example of output from the `ndcontrol` command:

```
ndcontrol manager report 9.24.104.105
```



```
# ndcontrol manager report 9.24.104.105

----------------------------------
|  HOST TABLE LIST  |    STATUS   |
----------------------------------
|        9.24.104.158|      ACTIVE|
|        9.24.104.157|      ACTIVE|
----------------------------------
----------------------------------------------------------------------------------------------------
|    9.24.104.105|   WEIGHT  |   ACTIVE %  48    |     NEW %  48     |    PORT %   2  |  SYSTEM %   2 |
----------------------------------------------------------------------------------------------------
| PORT:     80    | NOW | NEW |  WT  |  CONNECT   |  WT  |  CONNECT   |  WT  |  LOAD  |  WT  |  LOAD  |
----------------------------------------------------------------------------------------------------
|    9.24.104.158|  10 |  10 |  10  |          0 |  10  |          1 |  10  |   545  |  11  |      1 |
|    9.24.104.157|   9 |   9 |  10  |          0 |   9  |          1 |   9  |   552  |   8  |    333 |
----------------------------------------------------------------------------------------------------
| PORT TOTALS:   |  19 |  19 |      |          0 |      |          2 |      |  1097  |      |    334 |
----------------------------------------------------------------------------------------------------

--------------------------------
| ADVISOR |  PORT  |  TIMEOUT  |
--------------------------------
|   http  |    80  | unlimited |
| ISS-AI  |    -1  | unlimited |
--------------------------------

# 
```

Figure 220. ndmanager Report Showing HTTP Advisor and ISS Agent Data

In this figure, the data in the Port column has been provided by an HTTP Port 80 Advisor. The data in the System column has been provided by ISS agents running on the two TCP servers defined in the cluster.

Chapter 11. Firewall Load Balancing and High Availability

WebSphere Performance Pack can be successfully integrated with other IBM e-business offerings, such as IBM eNetwork Firewall. In particular, IBM SecureWay Network Dispatcher (ND), the load balancing component of WebSphere Performance Pack, can be used to achieve two goals:

- If you want to protect your site using a firewall, but at the same time you want to grant high availability for the firewall, you can use the Dispatcher function of ND. In this case, should the firewall machine unexpectedly fail, a backup firewall machine is ready to take over.

  The Dispatcher function should be installed on both the firewall machines, but it is not used to load balance the traffic. The purpose of the Dispatcher function is to provide high availability between the two firewall machines. This solution is easy to install and configure and is much less expensive than other products currently on the market.

- Another option is to use ND to load balance between two firewall machines. The advantage of this solution is that the workload of each firewall is effectively reduced.

  In addition, firewall high availability is still provided. In fact the ND software can detect if one of the two firewall machines has failed; in this case, the ND machine will mark the firewall machine that failed as not available and will no longer forward incoming requests to that machine.

In this chapter, we will examine both these solutions.

11.1 Firewall High Availability Using the Dispatcher

This section contains instructions on how to configure IBM eNetwork Firewall V3.3 and the ND component of IBM WebSphere Performance Pack V2.0 to provide a highly available firewall configuration. In this redbook, we do not discuss installation instructions for IBM eNetwork Firewall, as these topics are included in the product manual.

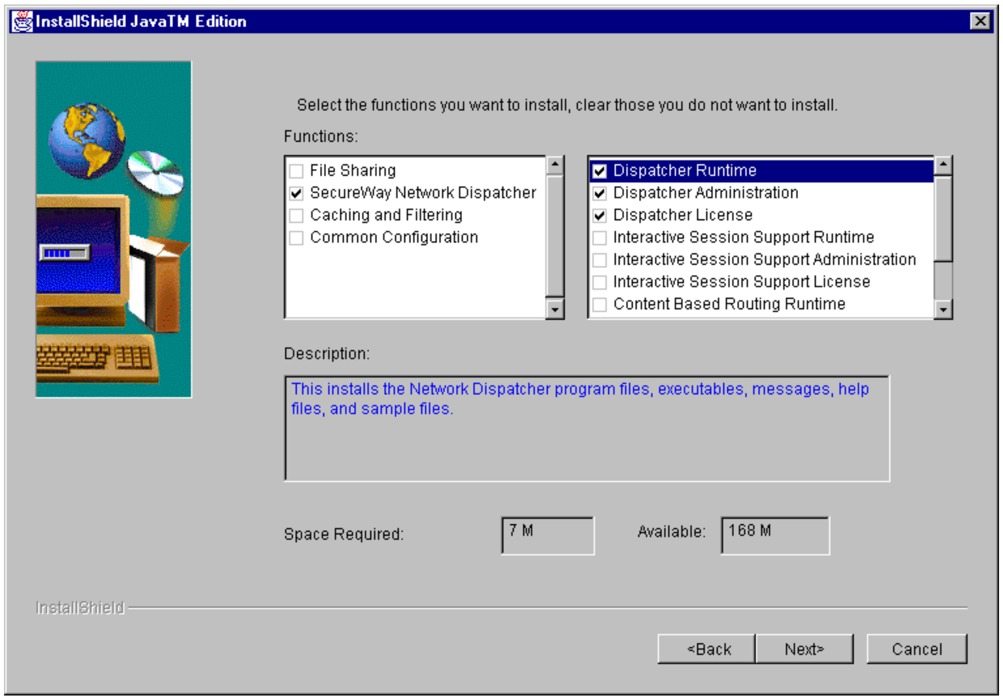
The scenario below makes use of the Dispatcher component of SecureWay Network Dispatcher Version 2.1 to provide firewall high availability.

11.1.1 Installation

Both the firewall machines must be installed in the following installation sequence:

1. Operating system (followed by the Service Pack 3 or greater if the operating system is Windows NT 4.0)
2. IBM eNetwork Firewall V3.3 (fixes are necessary with Windows NT 4.0 and Service Pack 3)
3. Dispatcher function of ND

Notice that this scenario does not require Interactive Session Support (ISS) and Content Based Routing (CBR), the two other functions of ND. On installing ND, you should select only the three subcomponents of ND that are associated with Dispatcher, as shown in the following figure:


InstallShield JavaTM Edition

Select the functions you want to install, clear those you do not want to install.

Functions:

- [ ] File Sharing
- [x] SecureWay Network Dispatcher
- [ ] Caching and Filtering
- [ ] Common Configuration

- [x] Dispatcher Runtime
- [x] Dispatcher Administration
- [x] Dispatcher License
- [ ] Interactive Session Support Runtime
- [ ] Interactive Session Support Administration
- [ ] Interactive Session Support License
- [ ] Content Based Routing Runtime

Description:

This installs the Network Dispatcher program files, executables, messages, help files, and sample files.

Space Required: 7 M   Available: 168 M

Figure 221. Selecting the Dispatcher Component for Installation

11.1.2 Basic Configuration Issues

The Dispatcher uses cluster addresses to identify the addresses over which it will perform its load balancing. In the scenario we are interested in, the Dispatcher will not be performing any load balancing, but we still need to use cluster addresses for the highly available portion of the setup. We need to configure two cluster addresses for this scenario: one for the secure interface and one for the nonsecure interface. The cluster addresses are the ones that will be transferred from one machine to the other in a failover situation. They are also the addresses that clients will point to on the secure and nonsecure networks. The configuration of these addresses is done in several scripts that are run by the Dispatcher.

Cluster Address and Loopback Interface

In a Windows NT firewall high availability scenario, you will not need to install the Microsoft Windows NT loopback adapter. The loopback adapter is only necessary for the TCP servers in a configuration and not on the ND machines. All loopback aliasing on the ND machines is taken care of in the high availability scripts.

Imagine the following scenario. There are two SecureWay Firewall machines: FW1 and FW2. FW1 will be the primary machine, and FW2 will act as a backup machine. Each machine also has a Dispatcher installed.

The following figure shows a graphical representation of this scenario:

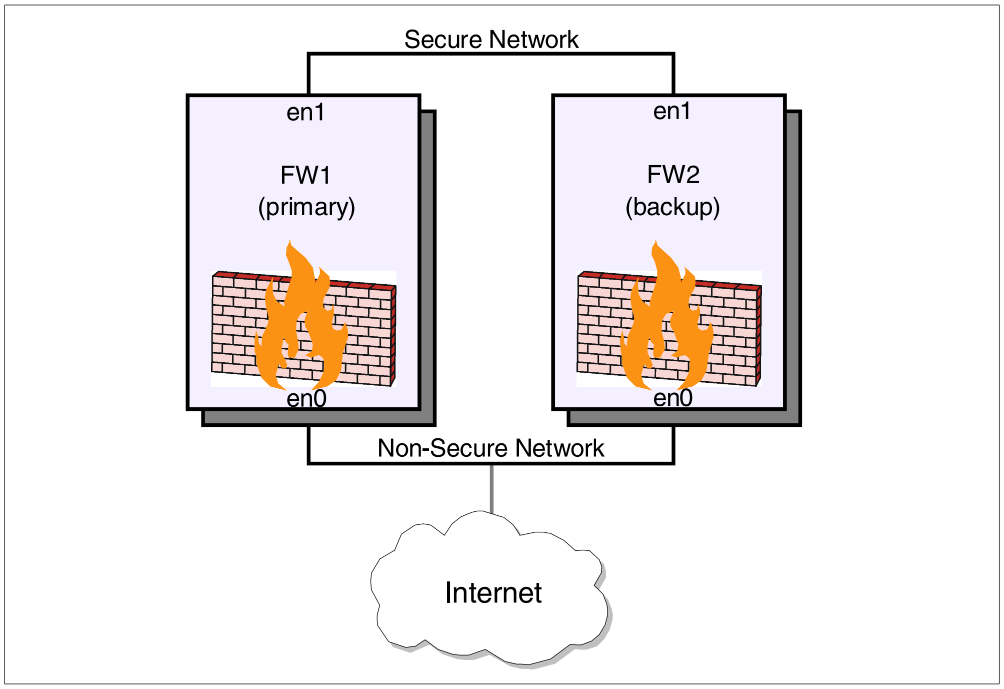


Figure 222. Graphical Representation of the Scenario

For this example, we assume that both networks are using Ethernet and that both machines are running AIX. The same scenario has also been successfully tested using token-ring networks and the Windows NT Server V4.0 operating system with Service Pack 4. Notice that the two machines must run the same operating system is a requirement when using high availability.

However, a known problem has been discovered when IBM eNetwork Firewall V3.3 for Windows NT and the ND component of IBM WebSphere Performance Pack are colocated on an IBM Netfinity Server running Windows NT Server 4.0 with Service Pack 3 (in this case fixes are required for IBM SecureWay Firewall) or Service Pack 4. The machine crashes when the ND Executor is started. The use of Netfinity machines is therefore not recommended in this particular scenario.[1]

There are three scripts that the Dispatcher uses to fail over the cluster addresses. They are goActive, goInOp and goStandby. Let's assume the two machines are using the following IP address scheme:

Table 18. Basic Configuration

Firewall Machine	**FW1**	**FW2**
IP Address for Nonsecure Network	9.67.123.4	9.67.123.5
IP Address for Secure Network	10.0.0.4	10.0.0.5
Cluster Address for Nonsecure Network	9.67.123.8	9.67.123.8
Cluster Address for Secure Network	10.0.0.8	10.0.0.8
Subnet Mask for Nonsecure Network	255.255.248.0	255.255.248.0

[1] The IBM WebSphere Performance Pack development team is working on a solution for the problem we have mentioned. By the time this redbook gets published, this problem will probably have been corrected. Check the IBM SecureWay Network Dispatcher Web site at http://www-4.ibm.com/software/network/dispatcher/.

Firewall Machine	**FW1**	**FW2**
Subnet Mask for Secure Network	255.255.255.0	255.255.255.0

If we assume the above, then the three scripts need to be configured as follows:

- goActive script

  This script deletes cluster loopback aliases and adds cluster device aliases. It will be executed when a Dispatcher machine, either the primary or backup in a high-availability configuration, goes into active state and begins routing packets.

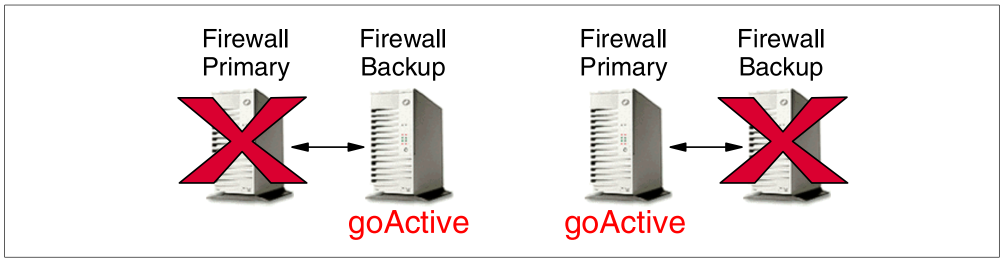


Figure 223. When the goActive Script Is Invoked

The goActive script for an ND machine running AIX is shown in the following figure:

```
#
# goActive script
#
# Configure this script when using the high availability feature of
# Network Dispatcher.
#
# This script is executed when Network Dispatcher goes into the
# 'Active' state and begins routing packets.
#
# This script must be placed in Network Dispatcher's bin directory
# (by default this is /usr/lpp/nd/dispatcher) and it needs to
# have root execute permission.
#
# Modify NETWORK, CLUSTER, INTERFACE and NETMASK to match your non-secure network environment.
#
# en0=first Ethernet adapter, tr0=first Token ring adapter, fi0=first FDDI adapter.
#
# NETMASK must be the netmask of your LAN.  It may be hexadecimal or octal notation.
#
NETWORK=9.67.123
INTERFACE=en0
NETMASK=255.255.248.0
#
 echo "Adding non-secure cluster alias(es)"
   for CLUSTER in 8; do
      ifconfig lo0 delete $NETWORK.$CLUSTER
      ifconfig $INTERFACE alias $NETWORK.$CLUSTER netmask $NETMASK
  done
#
# Modify SECNETWORK, SECCLUSTER, SECINTERFACE and SECNETMASK to match your secure network
# environment.
#
#
SECNETWORK=10.0.0
SECINTERFACE=en1
SECNETMASK=0xffffff00
#
 echo "Adding secure cluster alias(es)"
   for SECCLUSTER in 8; do
      ifconfig lo0 delete $SECNETWORK.$SECCLUSTER
      ifconfig $SECINTERFACE alias $SECNETWORK.$SECCLUSTER netmask $SECNETMASK
  done
```

Figure 224. goActive Script

- goStandby script

  This script deletes every cluster device alias and adds cluster loopback aliases. It will be executed when a Dispatcher machine in a high-availability configuration goes into standby state.

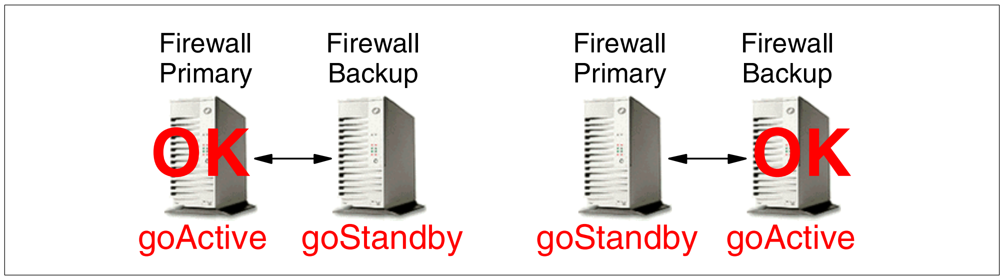


Figure 225. When the goStandby Script Is Invoked

The goStandBy script for an ND machine running AIX is shown in the following figure:

```
#!/bin/ksh
#
# goStandby script
#
# Configure this script when using the high availability feature of
# eNetwork Dispatcher.
#
# This script is executed when Network Dispatcher goes into the
# 'Standby' state.  Monitoring the health of the 'Active' machine
# but not routing packets.
#
# This script must be placed in Network Dispatcher's bin directory
# (by default this is /usr/lpp/nd/dispatcher) and it needs to
# have root execute permission.
#
# Modify NETWORK, CLUSTER, INTERFACE and NETMASK to match your non-secure environment.
#
# en0=first Ethernet adapter, tr0=first Token ring adapter, fi0=first FDDI adapter
#
# NETMASK must be the netmask of your LAN.  It may be hexadecimal or octal notation.
#
NETWORK=9.67.123
INTERFACE=en0
NETMASK=255.255.248.0
#
echo "Deleting the device aliases and adding the loopback aliases"
 for CLUSTER in 8 ; do
    ifconfig $INTERFACE delete $NETWORK.$CLUSTER
    ifconfig lo0 alias $NETWORK.$CLUSTER netmask $NETMASK
 done
#
# Modify SECNETWORK, SECCLUSTER, SECINTERFACE and SECNETMASK to match your secure environment.
#
SECNETWORK=10.0.0
SECINTERFACE=en1
SECNETMASK=0xffffff00
#
echo "Deleting the device aliases and adding the loopback aliases"
 for SECCLUSTER in 8 ; do
    ifconfig $SECINTERFACE delete $SECNETWORK.$SECCLUSTER
    ifconfig lo0 alias $SECNETWORK.$SECCLUSTER netmask $SECNETMASK
 done
```

Figure 226. goStandby Script

- goInOp script

  This script deletes all cluster device and loopback aliases. This script is executed when a Dispatcher Executor is stopped and before it is started for the first time.

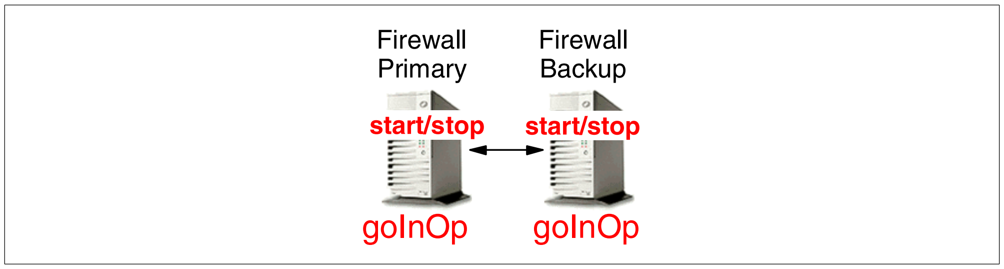


Figure 227. When the goInOp Script Is Invoked

The goInOp script for a ND machine running AIX is shown in the following figure:

```
#!/bin/ksh
#
# goInOp script
#
# Configure this script when using the high availability feature of
# Network Dispatcher and optionally when using Network Dispatcher in a
# standalone environment.
#
# This script is executed when the Network Dispatcher executor is stopped
# (and before the executor is initially started).
#
# This script must be placed in Network Dispatcher's bin directory
# (by default this is /usr/lpp/nd/dispatcher) and it needs to
# have root execute permission.
#
# Modify NETWORK, CLUSTER and INTERFACE to match your non-secure environment.
#
# en0=first Ethernet adapter, tr0=first Token ring adapter, fi0=first FDDI adapter
#
NETWORK=9.67.123
INTERFACE=en0
#
echo "Removing all loopback and device aliases"
for CLUSTER in 8; do
   ifconfig lo0 delete $NETWORK.$CLUSTER
   ifconfig $INTERFACE delete $NETWORK.$CLUSTER
 done

# Modify SECNETWORK, SECCLUSTER and SECINTERFACE to match your secure environment.
#
SECNETWORK=10.0.0
SECINTERFACE=en1
#
echo "Removing all loopback and device aliases"
for SECCLUSTER in 8; do
   ifconfig lo0 delete $SECNETWORK.$SECCLUSTER
   ifconfig $SECINTERFACE delete $SECNETWORK.$SECCLUSTER
 done
```

Figure 228. goInOp script

Notice that sample script files on AIX are located in the directory /usr/lpp/nd/dispatcher/samples. The sample scripts have the .sample extension. You should edit those scripts and save them in the parent directory /usr/lpp/nd/dispatcher without the .sample extension. The root user must have permission to execute those scripts.

On Windows NT, the location of the sample script files by default is C:\WSPP\IBM\nd\dispatcher\samples. On these platforms the script files must have a .cmd extension in order to be executed. The sample scripts present an additional .sample extension, which must be removed when the scripts are customized and copied in the final destination directory, where they will be executed. This directory is by default C:\WSPP\IBM\nd\dispatcher.

Configuration Scripts on Windows NT

The configuration scripts that we have shown in Figure 224, Figure 226, and Figure 228 would be different on the Windows NT platform. See “Script Modifications on Windows NT” on page 99 for more details.

11.1.3 Setting the Rules for IBM eNetwork Firewall

The next step in configuration is setting the rules for eNetwork Firewall to allow the Dispatcher to communicate with itself and the other machine.

Note: The name of the IBM eNetwork Firewall was changed to IBM SecureWay Firewall with Version 4.0.

The Dispatcher is a client/server application. The initial program that is started is ndserver. All the other commands are then issued using `ndcontrol`. When an `ndcontrol` command is issued, the ndcontrol program attempts to open a socket connection with the ndserver program. If it succeeds, the command is issued and the results returned. The source port for these TCP/IP packets will be a random port greater than 1023 and the destination port, by default, is 10003. This port can be changed by the user if deemed necessary; instructions for that are located in 2.5.4, “TCP Ports Used by the Dispatcher” on page 38. For example, if using the default, the socket might be opened using a TCP/IP connection from port 1048 to port 10003. The Dispatcher opens this socket on the default interface, which is where the host name of the machine is defined.

To discover the default interface, follow these simple rules:

1. Enter the `hostname` command on the firewall machine, as indicated in the following session screen:

```
# hostname
FW1
```

2. Issue the `ping` command to see on which network interface the host name of the machine is defined, as shown:

```
ping FW1
```

The output produced by this command will tell you the IP address of the network interface that replied to this Internet Control Message Protocol (ICMP) echo request.

The entire socket connection is taking place along the default interface so the rule can be explicit as to allow communication only along port 10003 from the interface back to itself.

Source and Destination Network Objects

IBM eNetwork Firewall V3.3 does not allow you to define a connection having the same object as both source and destination. For this reason, you will have to define two network objects having different names but representing the same network interface, that is, the default interface of the firewall. Then you should define a new connection having those two objects as source and destination, allowing a TCP connection to take place from a TCP port greater than 1023 to the TCP port 10003.

One Note of Caution

The default interface may be the nonsecure interface, and some firewall administrators may take offense at this. The risk is minimal, however, because the socket need be allowed only to port 10003 and only along the default interface.

The Dispatcher synchronizes the primary and backup machines using TCP/IP packets, called *heartbeats*. A rule has to be made to allow these heartbeats to be transmitted from one machine to the other. Heartbeat configuration is vitally important in order to ensure a high-availability solution. If the two Dispatcher machines lose contact with each other they will both assume that the other machine has failed and will assume routing duties for the cluster addresses. If this state occurs there will be network errors because there are multiple machines on the network answering to the same addresses. In order to prevent this, we recommend that you have multiple heartbeats, by having one heartbeat for each interface in the machine. If there are only two interfaces in the machine, one secure and one nonsecure, then one of the heartbeats needs to be sent along the nonsecure interface. Firewall administrators may not like opening up another port on the nonsecure interface, although the risk is small.

The port used by the heartbeat is defined by the user, and the TCP/IP packet will be constructed using this port as its source and destination. For example, if the user defines the heartbeat port as 12345, then the TCP/IP packet will be issued from port 12345 and to port 12345. This is convenient as it allows the rule on the firewall to be very precise. If one does not want to open this connection along the nonsecure interface, the solution is to have a dedicated network card on the nonsecure network side and exchange the heartbeat along it.

The last rule that needs to be opened on the firewall is for reachability. The Dispatcher has a feature called reachability that allows the user to define a series of IP addresses that the Dispatcher machines will attempt to ping; these machines are called reach targets. If the ping is successful, the Dispatcher assumes that it can reach those machines. For example, if the backup machine can reach more of the machines than the primary, it is likely that one of the network interfaces on the primary is no longer working and a failover will occur.

The firewall needs to be configured to allow each of these pings to reach their destination. The reachability ping is simply a standard ICMP ping from port 8 to port 0. Since the IP addresses are chosen by the user, the firewall configuration can be precise about allowing pings only to those specified machines if necessary. It is not necessary to allow inbound pings to the firewall.

Once the ports have been properly configured on the firewall, the Dispatcher can be configured. One last decision has to be made, and that is how recovery should be implemented. The Dispatcher has two modes, *automatic* and *manual*:

1. In automatic mode, if the primary machine fails, the backup takes over. When the primary recovers it will automatically take over from the backup machine.
2. In manual mode, if the primary machine fails, the backup takes over. When the primary recovers it will not take over unless the backup fails, or given an explicit command to do so.

The decision about which mode to use will depend on the individual customer situation.

The reasons for failover are as follows. If a machine is no longer receiving heartbeat information, it assumes the other machine has failed, and a failover will occur. If the backup machine is getting responses from more reach targets than the primary, a failover will occur. In all other situations the two machines should stay synchronized and be transparent to the firewall.

11.1.4 Scenario Implementation

We show now a sample configuration setup and give a concrete example of the steps mentioned in this chapter.

The following table gives more details about the two firewall machines used in this firewall high-availability scenario. In particular, it shows the IP addresses of the two firewall machines in the secure and nonsecure network respectively, the IP addresses of the reach target machines and the port selected for the firewall machines to exchange the heartbeat:

Table 19. High-Availability Settings

Firewall Machine	**FW1**	**FW2**
IP Address for Nonsecure Network	9.67.123.4	9.67.123.5
IP Address for Secure Network	10.0.0.4	10.0.0.5
Heartbeat Port	12345	12345
Reach Target Nonsecure	9.67.123.6	9.67.123.6
Reach Target Secure	10.0.0.6	10.0.0.6

The following list shows the configuration steps:

1. Configure FW1 and FW2 to allow communication from a port greater than 1023 to port 10003 along the default interface.
2. Allow communication between FW1 and FW2 using port 12345 as source and destination on the secure and nonsecure interfaces.

3. Allow ICMP pings outbound from the nonsecure interface to 9.67.123.6 and from the secure interface to 10.0.0.6.
4. Edit goActive, goInOp and goStandby scripts to alias the cluster addresses for the secure and nonsecure interfaces.
5. Start the ndserver program on FW1 and configure the Dispatcher to provide high availability.

   This is the sequence of commands to issue on FW1, which is the primary firewall machine:

   ```
   ndserver start
   ndcontrol executor start
   ndcontrol highavailability heartbeat add 9.67.123.4 9.67.123.5
   ndcontrol highavailability heartbeat add 10.0.0.4 10.0.0.5
   ndcontrol highavailability backup add primary auto 12345
   ndcontrol highavailability reach add 9.67.123.6
   ndcontrol highavailability reach add 10.0.0.6
   ndcontrol manager start
   ndcontrol file save HAFirewall
   ```

6. Start the ndserver program on FW2 and configure the Dispatcher to provide high availability.

   This is the sequence of commands to issue on FW2, which is the backup firewall machine:

   ```
   ndserver start
   ndcontrol executor start
   ndcontrol highavailability heartbeat add 9.67.123.5 9.67.123.4
   ndcontrol highavailability heartbeat add 10.0.0.5 10.0.0.4
   ndcontrol highavailability backup add backup auto 12345
   ndcontrol highavailability reach add 9.67.123.6
   ndcontrol highavailability reach add 10.0.0.6
   ndcontrol manager start
   ndcontrol file save HAFirewall
   ```

7. Direct all nonsecure clients to 9.67.123.8 and all secure clients to 10.0.0.8.

Note the following:

- The following command issued in steps 5 on page 284 and 6 on page 284 would fail if Step 1 on page 283 is not executed:

  ```
  ndcontrol executor start
  ```

- The following command also issued in Step 5 on page 284 and Step 6 on page 284, will allow the configuration to be reloaded quickly:

  ```
  ndcontrol file save HAFirewall
  ```

As you can see, we have shown how to issue the high availability configuration through the command line. Refer to Chapter 6, “ND High Availability Support” on page 177 to see how to perform the same steps by using the ND Graphical User Interface (GUI).

If either machine is stopped or rebooted, then the Dispatcher has to be reconfigured. It does not start automatically. In order to reconfigure it perform the following two steps, assuming you saved the configuration file as HAFirewall, as indicated in Step 5 on page 284 and Step 6 on page 284:

```
ndserver start
```

```
ndcontrol file load HAFirewall
```

11.2 Firewall Load Balancing

In this section we will demonstrate how the Dispatcher can be used to load balance traffic to two Firewall machines. To accomplish this, it is possible to make use of the new *wildcard cluster* and *wildcard port* features of the Dispatcher.

Wildcard Cluster and Wildcard Port in WebSphere Performance Pack V2

Wildcard cluster and wildcard port support is a new feature implemented for the first time in Version 2 of IBM WebSphere Performance Pack.

A wildcard cluster is configured as having IP address 0.0.0.0. The main application for wildcard clusters is to combine server configurations. That is, if you have many cluster addresses to load balance requests to, and some of them have the same port and server configurations, you can combine these clusters into one wildcard cluster. Other clusters with unique port and server configurations defined with actual cluster addresses can coexist with the wildcard cluster.

You must still explicitly configure each cluster address on one of the network adapters on your Dispatcher machine.

Wildcard ports are used when a set of servers in a specific cluster or in a wildcard cluster must respond to multiple ports, which could be known in advance or even unknown. Rather than adding each port in the Dispatcher configuration, you can add a wildcard port. The wildcard port number is 0.

Firewall load balancing is one example of how wildcard clusters and wildcard

The wildcard cluster can be used to load balance traffic to addresses that are not explicitly configured on any network adapter of the Dispatcher machine. Similarly, the wildcard port feature of the Dispatcher can be used to handle traffic that is not destined for any explicitly configured port. Load balancing firewalls is one practical use of these two features, since firewalls can process packets for any destination address and any destination port.

Although in general the use of wildcard clusters requires that each cluster address is explicitly configured on one of the network adapters on your Dispatcher machine, this is not necessary when the wildcard cluster is used in a firewall load-balancing scenario. In order for this to work, the Dispatcher must at least be able to see all the traffic that it is to load balance, and for this reason the Dispatcher must be set up as the default route for some set of traffic.[2]

As firewalls frequently are used to handle traffic going between secure and non-secure networks, a typical firewall load balancing scenario would contain two Dispatchers, one in the secure network and the other one in the nonsecure network. For high-availability reasons, it is advisable to put a backup Dispatcher in both the secure and nonsecure network. This configuration is the most reliable

[2] This is according to the documentation. However, we found that, at least on Windows NT, the cluster address must be explicitly configured on one of the network adapters of the Dispatcher machine. As we mention in 11.2.1.2, "Dispatcher Configuration" on page 288, it is our understanding that this problem does not arise on AIX and Solaris.

and requires a total number of four Dispatcher machines, as seen in the following figure:

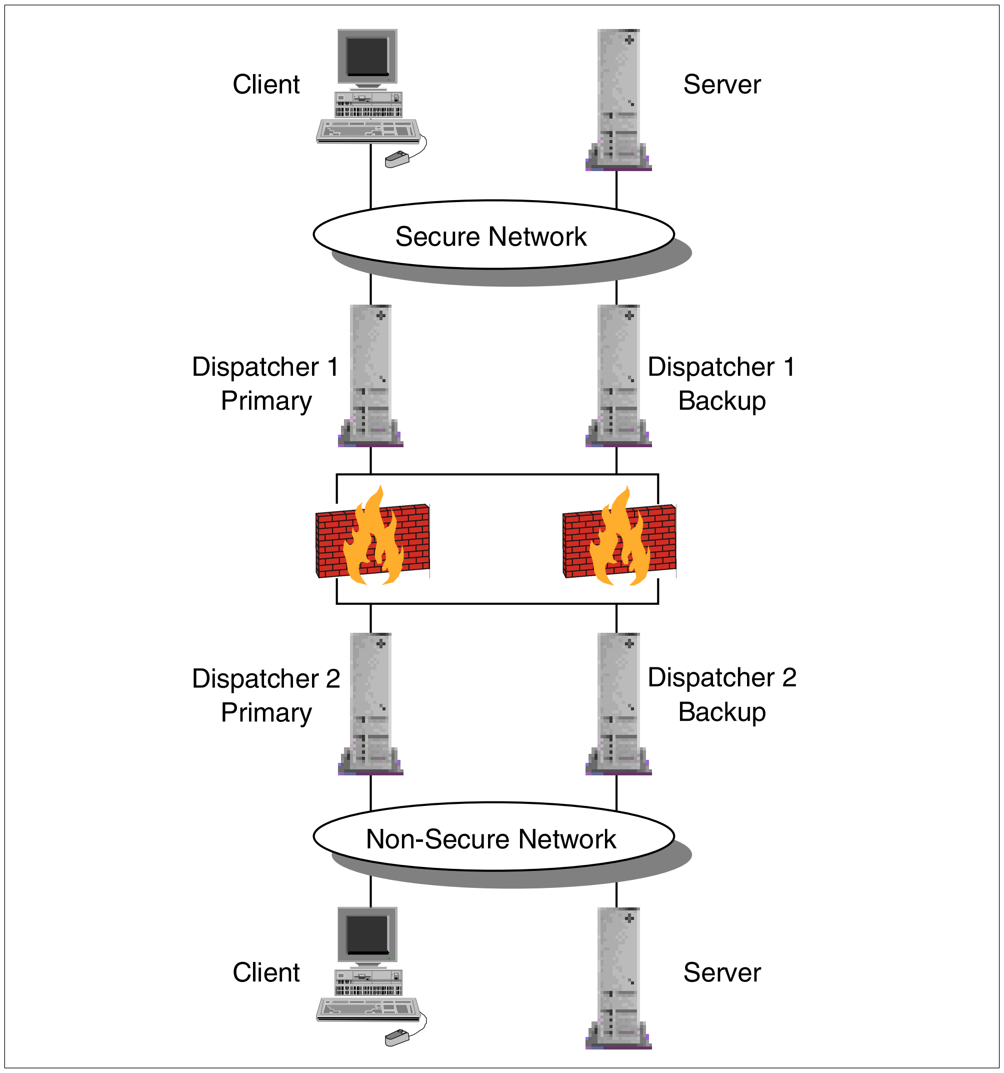


Figure 229. Typical Firewall Load-Balancing Scenario

Dispatcher 1 load balances traffic from the secure clients to the nonsecure servers. Dispatcher 2 load balances traffic from the nonsecure clients to the secure servers. As we said, the two backup Dispatcher machines are part of the architecture only for high-availability reasons, and each of them will take over only if the corresponding primary Dispatcher should unexpectedly fail.

If your security policy allows only requests originating in the secure network, and denies each request originating in the nonsecure network, then it is not necessary to place any Dispatcher machine in the non-secure network.

As is the case with other Dispatcher scenarios, the load-balanced servers (the firewalls in this case) do not send the response back to the client through the Dispatcher. Rather, the return traffic flows directly from the server to the client.

11.2.1 IP Filter Load-Balancing Scenario – Wildcard Cluster and Port

In our scenario we implemented the Dispatcher only on the secure side of the firewalls. This is a common configuration that occurs if your security policy allows only requests originating in the secure side to take place, and denies all the requests originating in the nonsecure side.

The firewalls in this case are running IBM eNetwork Firewall V3.3 on Windows NT 4.0 and Service Pack 3 and are configured simply to filter the traffic between the secure and the nonsecure networks. This firewall technology is known as IP filtering. In the rest of the sections in this chapter, we will also see how to load balance firewalls implementing other technologies.

We will use the secure client machine at IP address 192.168.10.6 to request a Web page from the Lotus Domino Go Webserver, which is located on the nonsecure side of the network on the Web server machine with IP address 172.16.0.1. The communication will go through the load-balanced firewall cluster.

11.2.1.1 Network Environment

The following figure shows a graphical representation of our firewall load-balancing network environment.

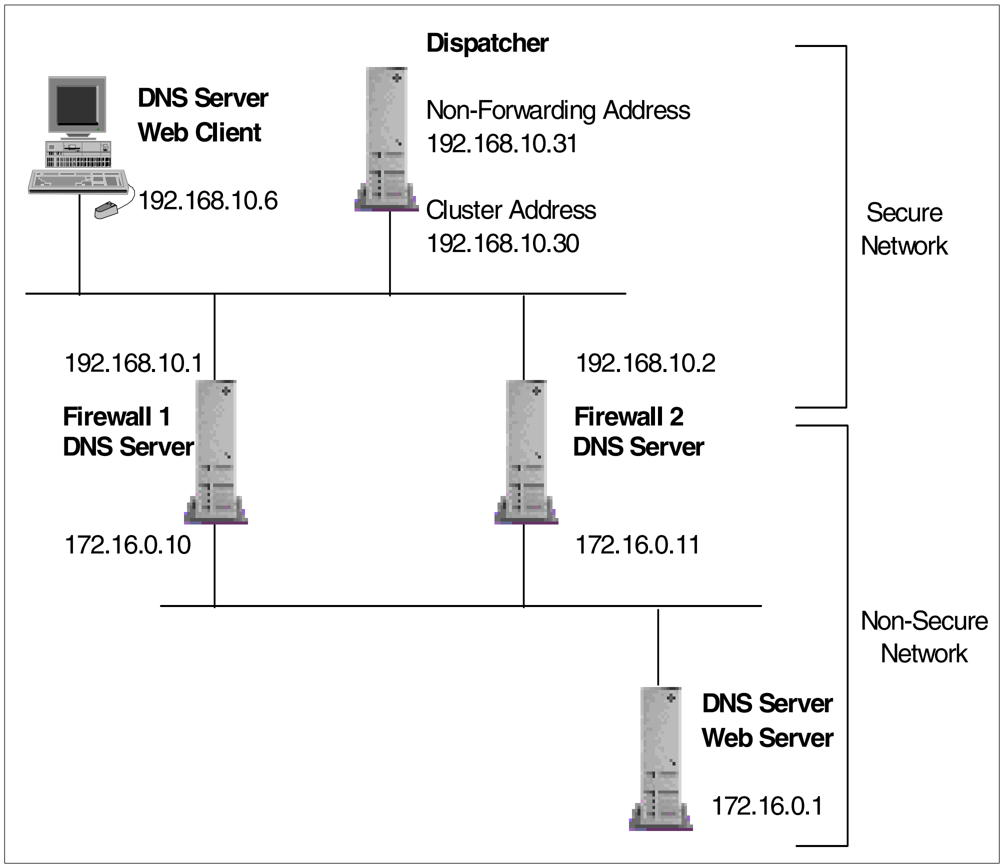


Figure 230. Firewall Load-Balancing Network Environment

The following table summarizes the machines we used in our scenario.

A summary of the hardware, software, and network configuration of the systems we used is shown in the following table:

Table 20. Basic Scenario - Hardware, Software and Network Configuration

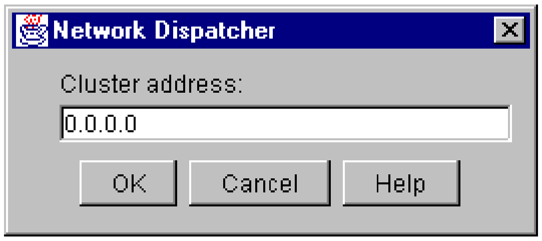
Workstation	Operating System	Secure LAN IP Address	Nonsecure LAN IP Address	Service
IBM PC 365	Windows NT Server 4.0	192.168.10.6		Secure Web client
IBM PC 365	Windows NT Server 4.0	192.168.10.31		Secure Dispatcher nonforwarding address
		192.168.10.30		Secure Dispatcher cluster address
IBM PC 365	Windows NT Server 4.0	192.168.10.1	172.16.0.10	Firewall 1
IBM PC 365	Windows NT Server 4.0	192.168.10.2	172.16.0.11	Firewall 2
IBM PC 365	Windows NT Server 4.0		172.16.0.1	Nonsecure Web server

Also note:

1. The load-balancing function was provided by the Dispatcher component of ND, installed from the CD-ROM of IBM WebSphere Performance Pack V2.0.
2. The firewall function was provided by IBM eNetwork Firewall Version 3.3.
3. Netscape Navigator 4.5 was the Web browser running on the secure Web client machine.
4. The Web server function on the nonsecure Web server was provided by Lotus Domino Go Webserver 4.6.2.5.
5. Both local area networks (LANs) were implemented on token-ring.

11.2.1.2 Dispatcher Configuration

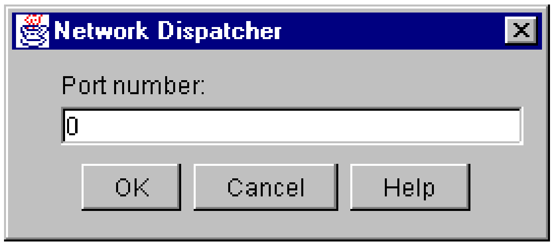
We began the Dispatcher wildcard cluster configuration by following the same steps as shown in 4.1.4, “Dispatcher Configuration” on page 83, up to the point where the cluster was added. To add the wildcard cluster, we right-clicked the **Executor** item and in the pop-up menu, selected **Add Cluster**. In the Add Cluster dialog, we entered the value `0.0.0.0` as our cluster address:


Network Dispatcher

Cluster address: 0.0.0.0

Figure 231. Entering the Wildcard Cluster

Next, we added the wildcard port. To do this we right-clicked the **Cluster:0.0.0.0** item and in the pop-up menu, selected **Add Port**. On the Add Port dialog, we entered a port number of `0` to represent the wildcard port.


Network Dispatcher

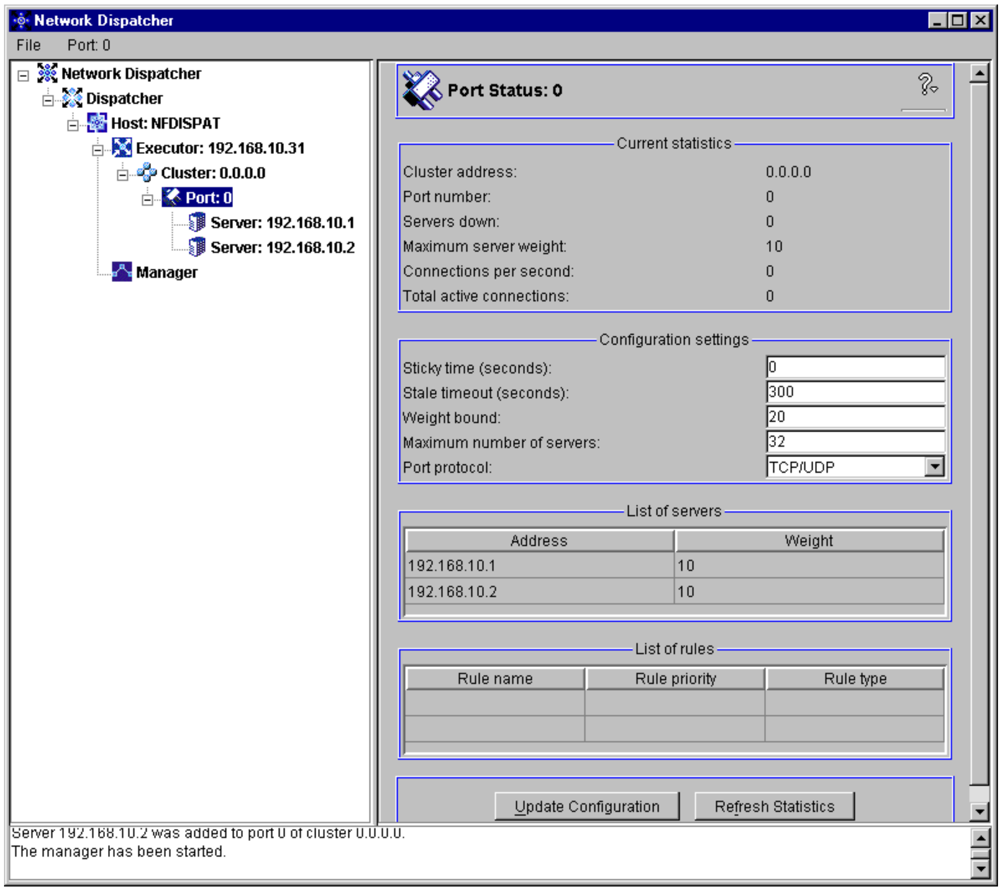
Port number:

0

Figure 232. Entering the Wildcard Port from the GUI

After adding the secure interface of our two IBM eNetwork Firewall V3.3 machines as the Port 0 servers and starting the Manager component, we could have enabled the ping advisor. This advisor does not open a TCP connection with the server, but instead reports whether the server responds to the ping. It is a new advisor, specifically intended for use with wildcard ports. If this advisor is used, the firewall machines would have to be configured to respond to pings.

The final port status from the GUI appeared as follows:



Current statistics

Cluster address:	0.0.0.0
Port number:	0
Servers down:	0
Maximum server weight:	10
Connections per second:	0
Total active connections:	0

Configuration settings

Sticky time (seconds):	0
Stale timeout (seconds):	300
Weight bound:	20
Maximum number of servers:	32
Port protocol:	TCP/UDP

List of servers

Address	Weight
192.168.10.1	10
192.168.10.2	10

List of rules

Rule name	Rule priority	Rule type

```
Server 192.168.10.2 was added to port 0 of cluster 0.0.0.0.
The manager has been started.
```

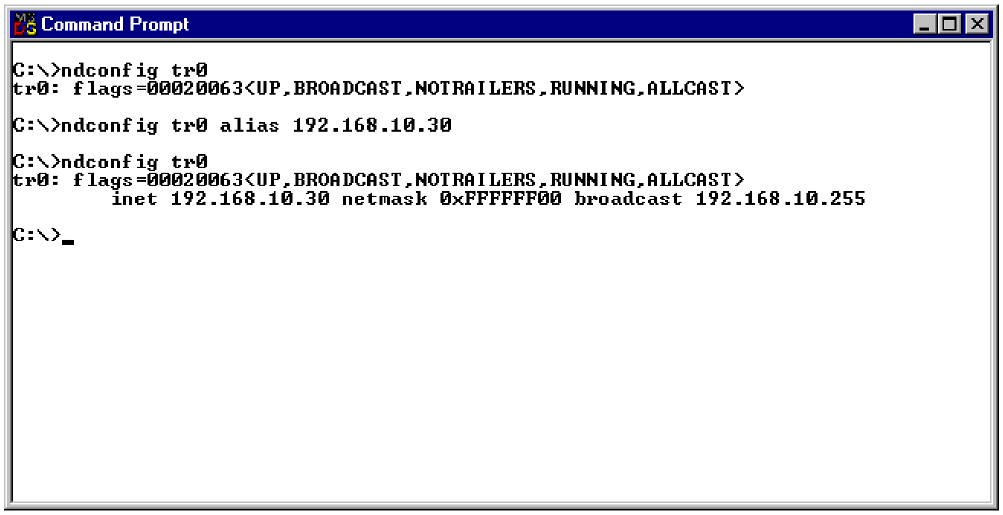
Figure 233. Wildcard Port Status

The final step on our Dispatcher machine was to alias a real cluster address (not the wildcard cluster address) to the tr0 network interface. We did this by entering the command:

```
ndconfig tr0 alias 192.168.10.30 netmask 255.255.255.0
```

We found the creation of this alias necessary for load balancing to occur on the Windows NT machine that we used as the Dispatcher in this scenario. It is our understanding that the creation of this alias is not necessary when the Dispatcher is running on either an AIX or Solaris system.

It is important to note that you should not use the GUI's Cluster:0.0.0.0 menu to configure the cluster address, as this will result in the wrong IP address being aliased to your network interface card (0.0.0.0 instead of 192.168.10.30). The following figure shows the `ndconfig` command output before and after we created the alias:



```
C:\>ndconfig tr0
tr0: flags=00020063<UP,BROADCAST,NOTRAILERS,RUNNING,ALLCAST>

C:\>ndconfig tr0 alias 192.168.10.30

C:\>ndconfig tr0
tr0: flags=00020063<UP,BROADCAST,NOTRAILERS,RUNNING,ALLCAST>
        inet 192.168.10.30 netmask 0xFFFFFF00 broadcast 192.168.10.255

C:\>_
```

Figure 234. ndconfig Command Output

11.2.1.3 Firewall Configuration

On each of the firewall machines, we defined one firewall connection for use in our scenario. This connection allowed requests from our secure client destined for HTTP port 80 to go out to Web server on the nonsecure network and the resulting responses (all having the ACK flag) from the server on the nonsecure network to be returned to the secure client. Consult the IBM redbook *Internet Security in the Network Computing Framework*, SG24-5220, for further information on how to configure the IBM eNetwork Firewall.

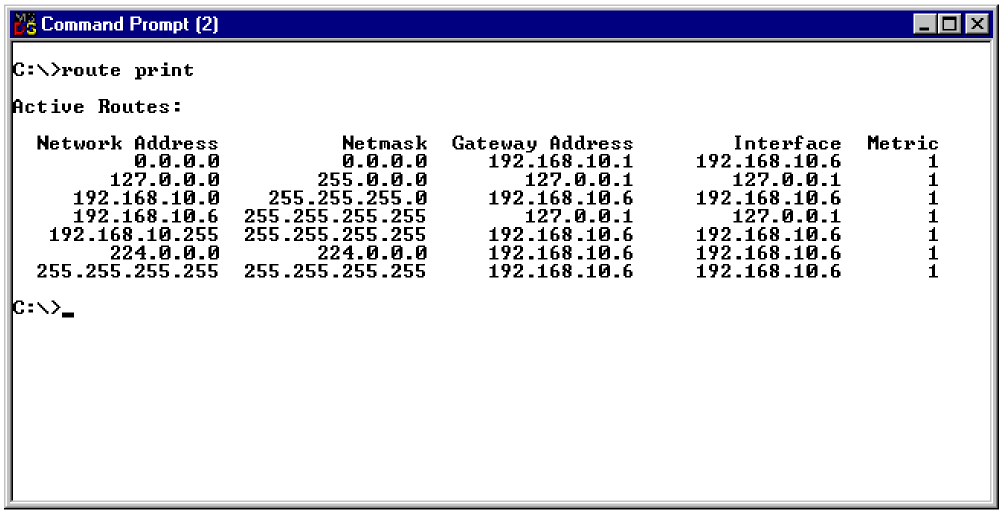
Because we allowed only port 80 traffic through the firewall, we could have used port 80 in our wildcard cluster on the Dispatcher, rather than a wildcard port. However, this example is useful for showing how to configure a wildcard port.

No other load-balancing specific configuration was done on either of the firewall machines.

11.2.1.4 Secure Client Configuration

The clients need to be configured so that *either* they use the ND workstation as the default gateway *or* their default gateway leads to a router that uses ND workstation as its default gateway.

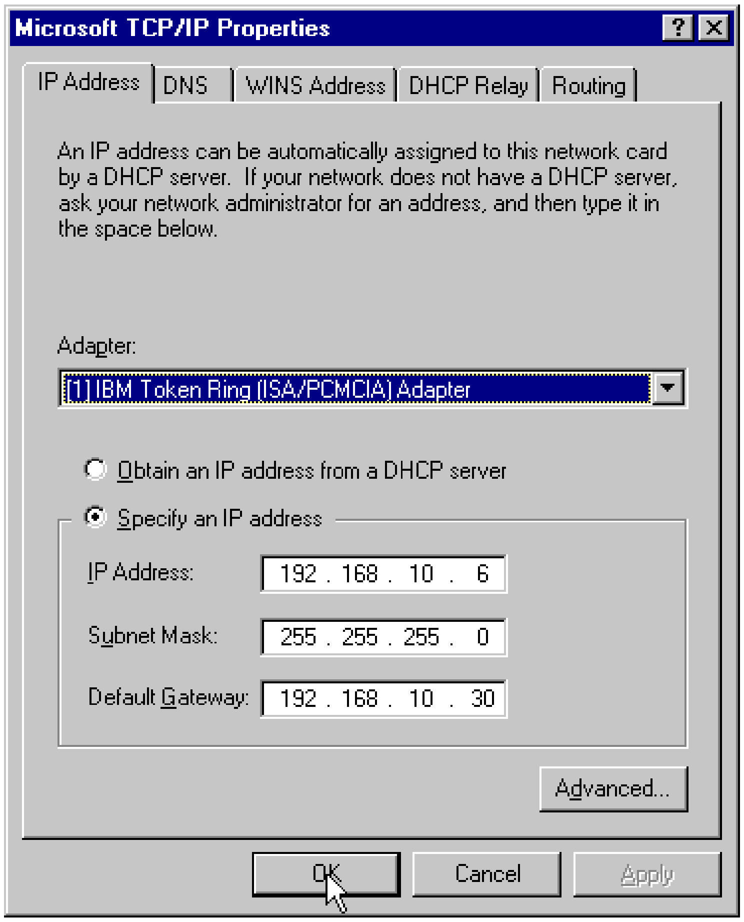
To accomplish this, we changed the client gateway associated with the network interface on the machine. This step is necessary in order for the Dispatcher to receive the network packets destined for any IP address that is not the Dispatcher's IP address. Before making the change, we noticed that our default route was set to the Firewall machine, 192.168.10.1. We used the `route print` command to see our default route. Following is the output showing the default route before changing the gateway in the first row of the table:



```
C:\>route print

Active Routes:

  Network Address          Netmask  Gateway Address        Interface  Metric
          0.0.0.0          0.0.0.0     192.168.10.1     192.168.10.6       1
        127.0.0.0        255.0.0.0        127.0.0.1        127.0.0.1       1
     192.168.10.0    255.255.255.0     192.168.10.6     192.168.10.6       1
     192.168.10.6  255.255.255.255        127.0.0.1        127.0.0.1       1
   192.168.10.255  255.255.255.255     192.168.10.6     192.168.10.6       1
        224.0.0.0        224.0.0.0     192.168.10.6     192.168.10.6       1
  255.255.255.255  255.255.255.255     192.168.10.6     192.168.10.6       1

C:\>_
```

Figure 235. Default Route Before Changing the Default Gateway Address (wildcfg03)

To change the gateway address in our default route, we changed the default gateway for our network interface. To do this we clicked **Start**, then **Settings,** and then **Control Panel**. We double-clicked the **Network** icon, and in the Network window selected the **Protocols** tab. On the Protocols page, we double-clicked the **TCP/IP** line and were presented with the Microsoft TCP/IP Properties window. In this window we selected the **IP Address** tab, and changed the Default Gateway field at the bottom of the window to contain the IP address of our cluster. This is shown in the following figure:



An IP address can be automatically assigned to this network card by a DHCP server. If your network does not have a DHCP server, ask your network administrator for an address, and then type it in the space below.

Adapter: [1] IBM Token Ring (ISA/PCMCIA) Adapter

Obtain an IP address from a DHCP server [✓ Specify an IP address]

IP Address: 192 . 168 . 10 . 6

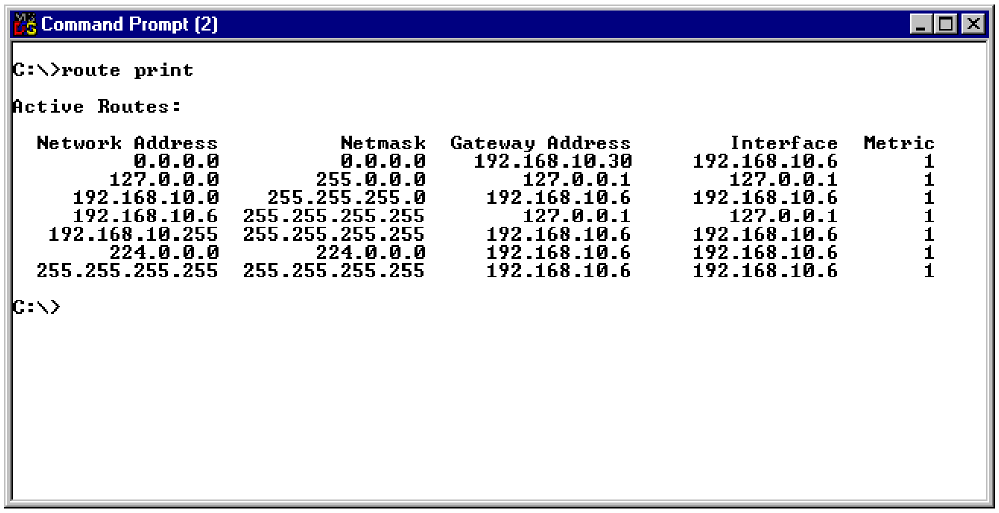
Subnet Mask: 255 . 255 . 255 . 0

Default Gateway: 192 . 168 . 10 . 30

Figure 236. Changing the Default Gateway on the Secure Client

After clicking **Apply** or **OK**, we entered the `route print` command again:

This way we could verify that the default route was changed to the cluster address 192.168.10.30, as shown in the following figure:



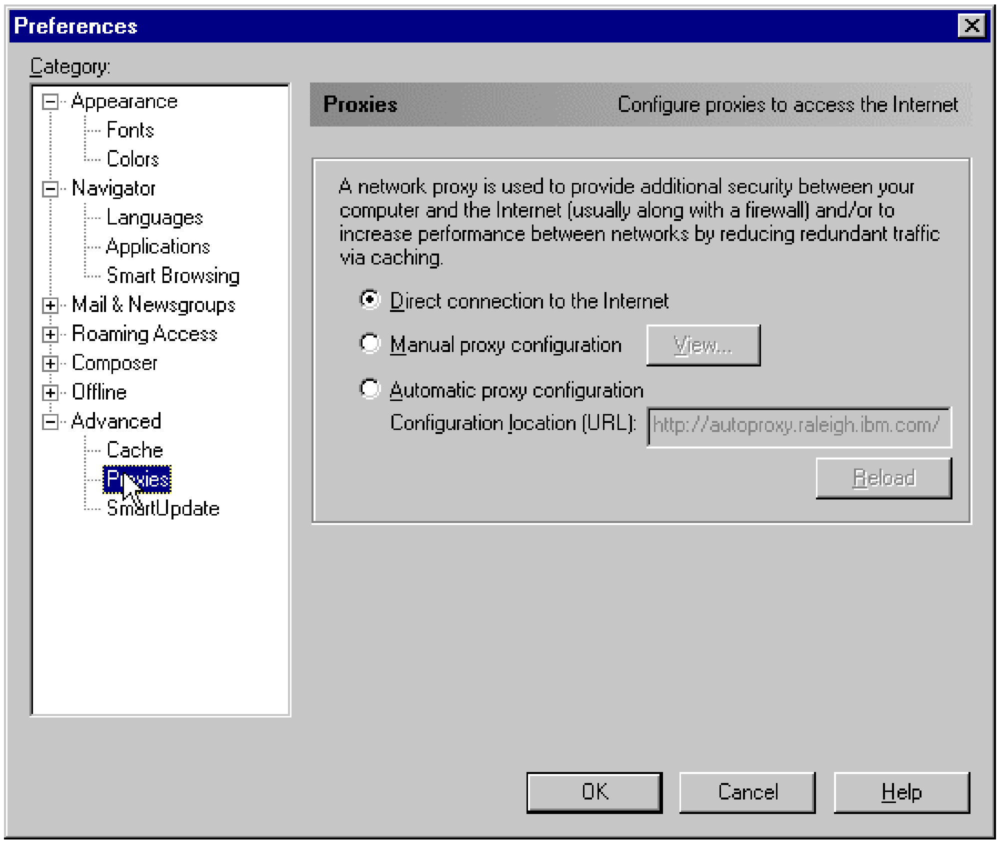
```
C:\>route print

Active Routes:

  Network Address          Netmask  Gateway Address        Interface  Metric
          0.0.0.0          0.0.0.0    192.168.10.30     192.168.10.6       1
        127.0.0.0        255.0.0.0        127.0.0.1        127.0.0.1       1
     192.168.10.0    255.255.255.0     192.168.10.6     192.168.10.6       1
     192.168.10.6  255.255.255.255        127.0.0.1        127.0.0.1       1
   192.168.10.255  255.255.255.255     192.168.10.6     192.168.10.6       1
        224.0.0.0        224.0.0.0     192.168.10.6     192.168.10.6       1
  255.255.255.255  255.255.255.255     192.168.10.6     192.168.10.6       1

C:\>
```

Figure 237. The New Default Route Pointing To The Cluster Address

We noticed as well that we could have used the nonforwarding address of the Dispatcher machine (192.168.10.31) as the default gateway on the secure client machine. *As long as an alias was set up on the network interface on the Dispatcher*, the client could successfully use either 192.168.10.30 or 192.168.10.31 as its gateway.

The next step was to change the Proxies configuration on the Web client machine's browser, so that it would use a direct connection to the Internet without passing through a proxy server.

To do this, we opened the Netscape Edit menu, and selected **Preferences...**. In the Preferences window we selected the **Advanced** twisty from the Category tree to expand the Advanced options and then we selected the **Proxy** item. In the Proxies section, we selected the radio button **Direct connection to the Internet** as follows:



A network proxy is used to provide additional security between your computer and the Internet (usually along with a firewall) and/or to increase performance between networks by reducing redundant traffic via caching.

[✓ Direct connection to the Internet]
Manual proxy configuration
Automatic proxy configuration
Configuration location (URL): http://autoproxy.raleigh.ibm.com/

Figure 238. Netscape Proxy Selection

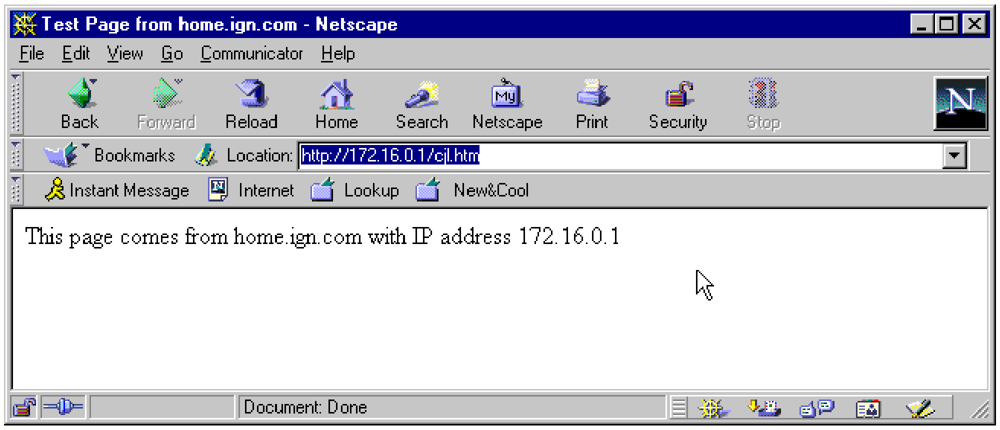
The last changes we made before generating the HTTP request were the following:

- We set to `0` the number of days after which pages in history expire. To do this we used the same Preferences window accessed from the Edit menu, and we clicked **Navigator**.
- We selected **Advanced** and then **Cache**, and set Memory Cache and Disk Cache to `0 Kbytes`.

In this way, we did not have to clear history, memory cache, and disk cache for the browser each time we made a request. In a real-life situation, it is not necessary to clear history, memory cache, and disk cache. This is necessary only in a test environment, to make sure that the Web pages are retrieved each time from the Web servers and not from the local cache.

11.2.1.5 Scenario Results

Following this, we made a request from the secure client browser for a specific page that we created and placed on our nonsecure Web server machine. The page contained text that uniquely identified the server that was serving the page. From the Netscape Navigator browser on the secure client, we obtained the page from the nonsecure Web server as follows:


Test Page from home.ign.com - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://172.16.0.1/cjl.htm
Instant Message Internet Lookup New&Cool
Document: Done

This page comes from home.ign.com with IP address 172.16.0.1

Figure 239. Nonsecure Request Results

This shows that the request was successfully transmitted through one of the firewall machines to the Web server on the nonsecure side of the network.

Following this, we made several more Web page requests and from the Dispatcher machine verified that the Dispatcher was load balancing these requests between the two firewall machines. We used the ND GUI port monitor to see these results:

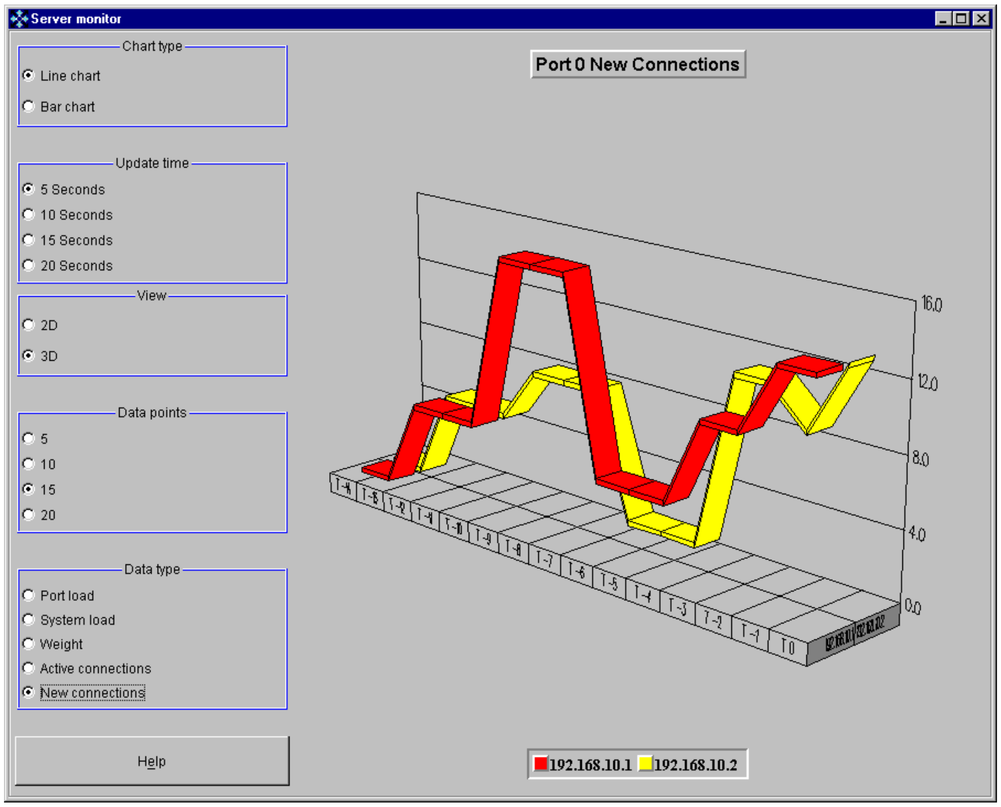


Chart type: [✓ Line chart] Bar chart

Update time: [✓ 5 Seconds] 10 Seconds 15 Seconds 20 Seconds

View: 2D [✓ 3D]

Data points: 5 10 [✓ 15] 20

Data type: Port load System load Weight Active connections [✓ New connections]

Figure 240. ND GUI Wildcard Port Requests Being Distributed by the Dispatcher

Firewall High Availability when Using Firewall Load Balancing

Firewall high availability is an inherent feature of firewall load balancing. If one of the firewall machines experiences a problem or needs to be removed from service for any reason, the Dispatcher will automatically detect this and distribute requests to the remaining Firewall machine as part of its normal load-balancing capabilities.

11.2.2 HTTP Proxy Server Load Balancing Scenario

In order to understand how to configure a Dispatcher to load balance requests to Firewall proxy servers, we need first to understand what a proxy server is.

11.2.2.1 Proxy Servers

Proxy servers are tools that run at the application level of the ISO/OSI network model. For this reason they are also known as *application level gateways*. When a firewall acts as a proxy server, it performs the necessary work on behalf of the secure user. The client must send its request to the proxy server and not to the server directly. The connection is broken at the proxy server, which sends the client's requests to the server without any computer on the nonsecure network knowing that the secure client even exists. Using this firewall technology, the client and the server do not speak directly to each other, but through the proxy server, so that the client located on the intranet remains hidden to the rest of the world.

Proxy servers do not involve packet routing at all. The IP address of the client machine on the intranet from which the IP session was established will never appear on the Internet, and attackers and intruders cannot use addresses of the protected network to gain information about the structure of the intranet.

Proxy servers also allow other security measures:

- Telnet and FTP proxy servers can be configured in order to perform authentication and authorization checks.
- Requests for Internet sites can be logged, along with addresses of the requesting machines.
- Requests for certain Web sites can be banned.

HTTP proxy servers support several protocols, such as FTP, HTTP, HTTPS (which is HTTP over SSL), Gopher and WAIS.

For more information on proxy servers, see the IBM redbook *Internet Security in the Network Computing Framework*, SG24-5220.

11.2.2.2 Scenario Implementation

In this scenario we implement a firewall HTTP proxy server load-balancing architecture by using a basic nonwildcard Dispatcher cluster. In order for a client to use a proxy server, the client must send its request to the proxy server and not to the server that it is making the request of directly. As a result, we do not require the use of the wildcard cluster and wildcard port for this configuration. In this case, we have a defined proxy server address that will be used as the destination for all of the packets coming out of the client machine. We configure this proxy server address to be the cluster address. In this way, the Dispatcher will first receive the requests destined to go the proxy server. The Dispatcher will then load balance these requests by forwarding them to the Firewall machine that it deems to be the best one to respond to the request.

11.2.2.3 Network Environment

The network environment that we used in this scenario was exactly the same as the one used in 11.2, "Firewall Load Balancing" on page 285. In particular, notice that also in this case the platform was completely based on Windows NT Server 4.0 with Service Pack 3. The necessary fixes for IBM eNetwork Firewall V3.3 for Windows NT, required with Service Pack 3, were also installed. A summary of the hardware, software and network configuration of the systems we used is shown in the following table:

Table 21. Basic Scenario - Hardware, Software and Network Configuration

Workstation	Operating System	Secure LAN IP Address	Nonsecure LAN IP Address	Service
IBM PC 365	Windows NT Server 4.0	192.168.10.6		Secure Web client
IBM PC 365	Windows NT Server 4.0	192.168.10.31		Secure Dispatcher nonforwarding address
		192.168.10.30		Secure Dispatcher cluster address
IBM PC 365	Windows NT Server 4.0	192.168.10.1	172.16.0.10	Firewall 1
IBM PC 365	Windows NT Server 4.0	192.168.10.2	172.16.0.11	Firewall 2

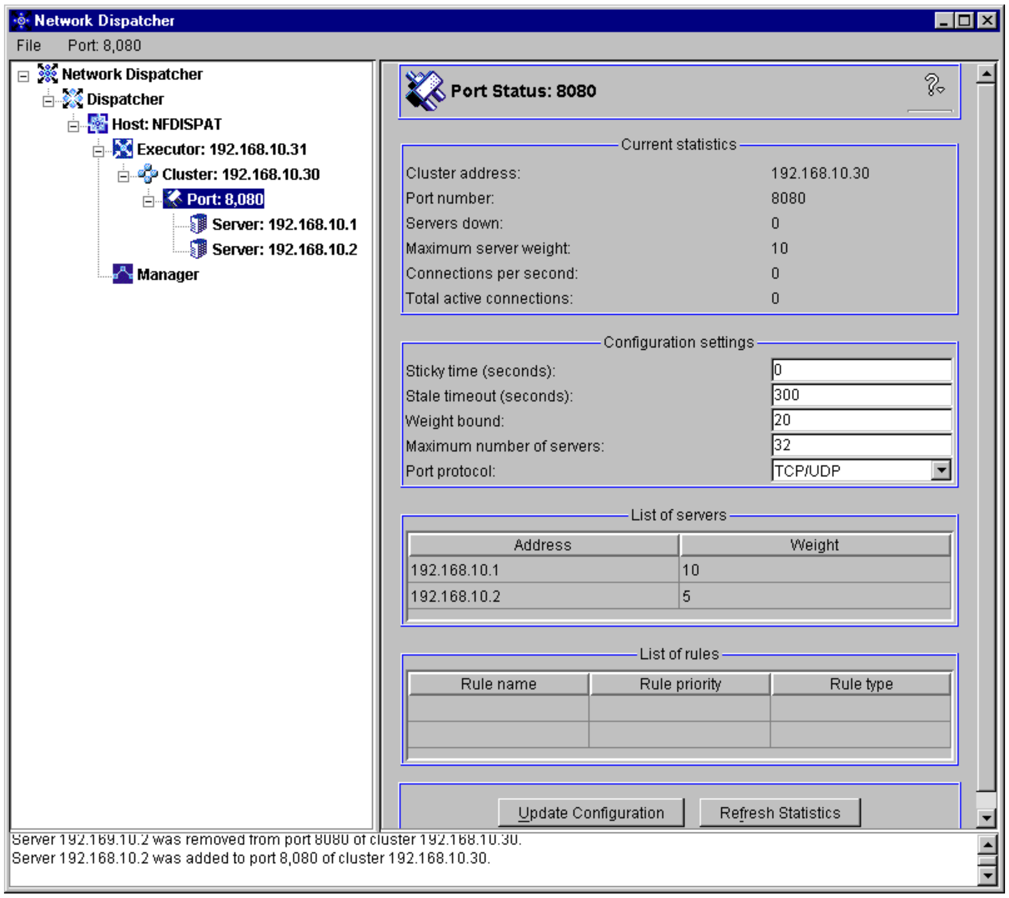
Workstation	Operating System	Secure LAN IP Address	Nonsecure LAN IP Address	Service
IBM PC 365	Windows NT Server 4.0		172.16.0.1	Nonsecure Web server

Also note:

1. The load-balancing function was provided by the Dispatcher component of SecureWay Network Dispatcher Version 2.1.
2. The firewall function was provided by IBM eNetwork Firewall Version 3.3.
3. Netscape Navigator 4.5 was the Web browser running on the secure Web client machine.
4. The Web server function on the non-secure Web server was provided by Lotus Domino Go Webserver 4.6.2.5.
5. Both LANs were implemented on token-ring.

11.2.2.4 Dispatcher Configuration

For this scenario, we configured the Dispatcher in a standard way. We added one cluster with the cluster address 192.168.10.30. We added one port to this cluster for port number 8080 (this is the port number that our firewalls were about to be configured to receive HTTP requests on, as seen in 11.2.2.5, “Firewall Proxy Configuration” on page 298) and the two firewall machines as servers. The following ND GUI display shows a summary of the Dispatcher configuration:



Current statistics

Cluster address:	192.168.10.30
Port number:	8080
Servers down:	0
Maximum server weight:	10
Connections per second:	0
Total active connections:	0

Configuration settings

Sticky time (seconds):	0
Stale timeout (seconds):	300
Weight bound:	20
Maximum number of servers:	32
Port protocol:	TCP/UDP

List of servers

Address	Weight
192.168.10.1	10
192.168.10.2	5

List of rules

Rule name	Rule priority	Rule type

```
Server 192.169.10.2 was removed from port 8080 of cluster 192.168.10.30.
Server 192.168.10.2 was added to port 8,080 of cluster 192.168.10.30.
```

Figure 241. Dispatcher Port Status

The final step on our Dispatcher machine was to alias the cluster address to the tr0 network interface. We did this by entering the command:

```
ndconfig tr0 alias 192.168.10.30 netmask 255.255.255.0
```

Other methods of configuring this alias are available, and are described in 4.1.4.8, “Methods of Aliasing the Cluster to the Network Interface” on page 94.

11.2.2.5 Firewall Proxy Configuration

Recall that the proxy server breaks the communication from the client to the server and sends out the request to the server on the clients’ behalf. For this reason we require two separate connections to be configured on each of our firewall machines.

The first connection allowed requests from our secure client destined for HTTP port 8080 to come to the secure interface on the firewall machine and go back to the secure client again. The second connection allowed HTTP port 80 requests to go out from the nonsecure interface of the firewall to the Web server on the nonsecure network and the resulting returned response from the Web server on the nonsecure network back to the nonsecure interface on the firewall. We do not show here the details of how to configure this firewall connection. Consult the IBM redbook *Internet Security in the Network Computing Framework*, SG24-5220 for further information on how to configure the FTP proxy server with IBM eNetwork Firewall.

The next step on the firewall machines was to alias the cluster address to the loopback interface on the machine. We followed the directions given in 4.1.5, “TCP Servers Configuration” on page 107 on both of our firewall proxy servers.

After adding the cluster alias and rebooting the firewall machines, we needed to make one more change on the firewalls in order for the communication to succeed. The IBM eNetwork Firewall V3.3 software detects that a new network interface has been added and it creates an associated firewall interface for the newly created interface. By default this new firewall interface is marked as belonging to the nonsecure network and it must be changed to secure. In order to change the designation of this interface to secure, we used the IBM eNetwork Firewall administration GUI. We double clicked the **System Administration** twisty to expand its options. From the expanded list, we selected **Interfaces** as follows:

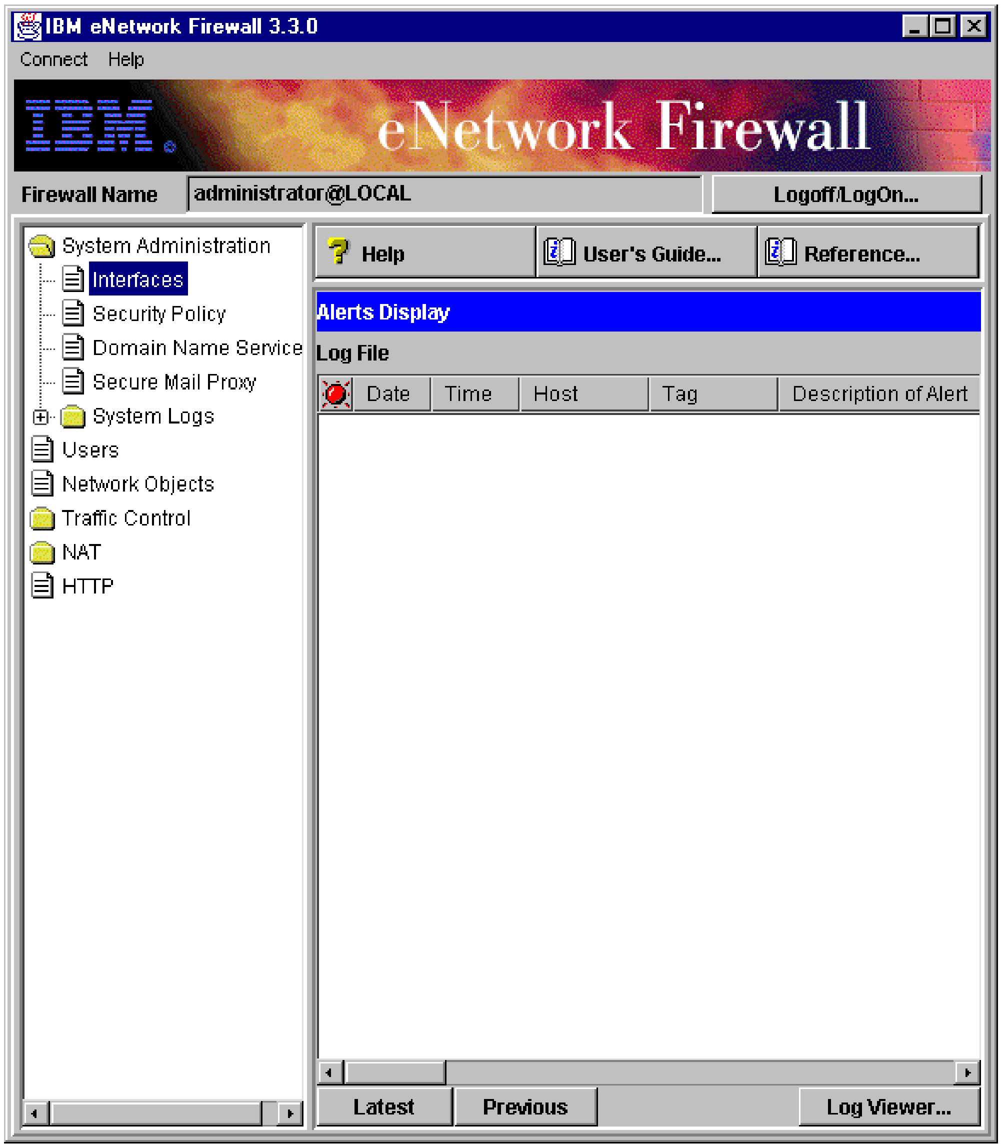
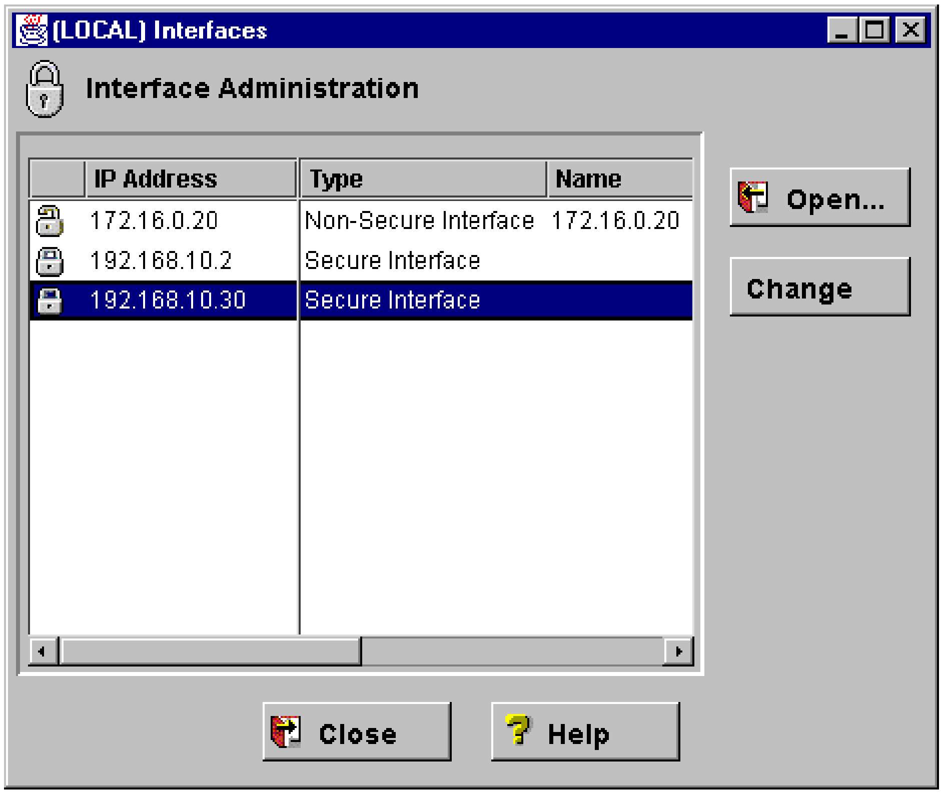


Figure 242. IBM eNetwork Firewall System Administration Window

In the resulting Network Interfaces dialog window, we selected the newly added interface and clicked **Change**. This changed the designation of the interface type from `Non-Secure Interface` to `Secure Interface` as follows:

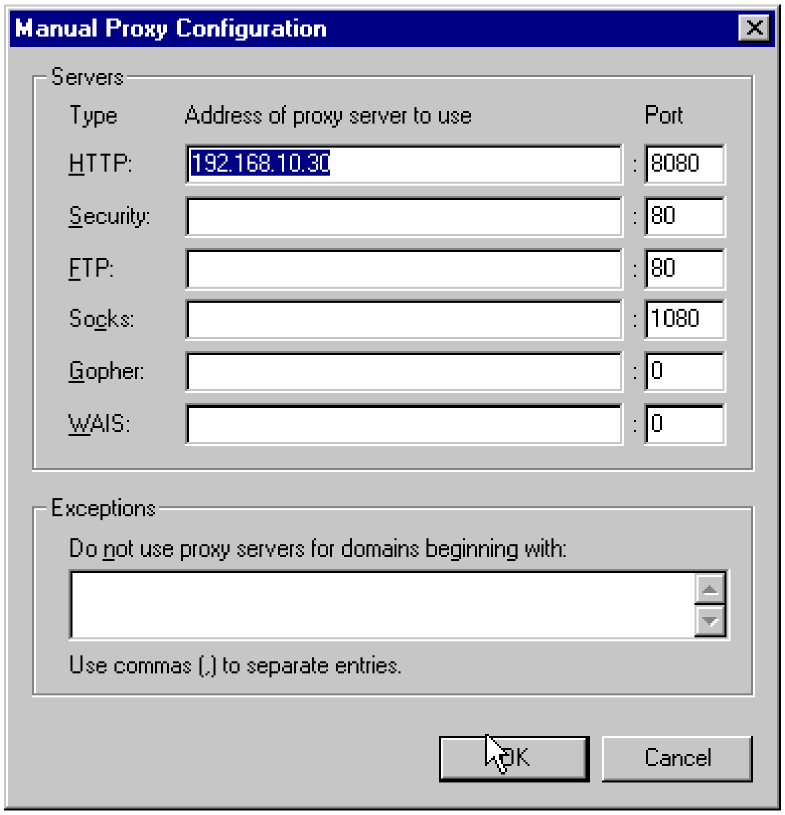


IP Address	Type	Name
172.16.0.20	Non-Secure Interface	172.16.0.20
192.168.10.2	Secure Interface	
192.168.10.30	Secure Interface	

Figure 243. Firewall Interface Administration

11.2.2.6 Secure Client Configuration

The client browser must be configured to forward each request to the HTTP proxy server, or it will try to establish a direct connection to the Web server machine located on the Internet. Netscape Navigator 4.5 can be configured through the Preferences window. We opened the Edit menu, and selected **Preferences...**. In the Preferences window we selected **Advanced** from the Category tree and then **Proxies**. In the Proxies section, we selected the radio box **Manual Proxy Configuration** and then clicked **View...**. Then, we filled in the HTTP field with the cluster IP address `192.168.10.30` and put the port number for HTTP protocol access that our proxy server was expecting to receive the requests on: `8080`. The final configuration is shown in the following figure:



Manual Proxy Configuration

Servers

Type	Address of proxy server to use	Port
HTTP:	192.168.10.30	8080
Security:		80
FTP:		80
Socks:		1080
Gopher:		0
WAIS:		0

Exceptions

Do not use proxy servers for domains beginning with:

Use commas (,) to separate entries.

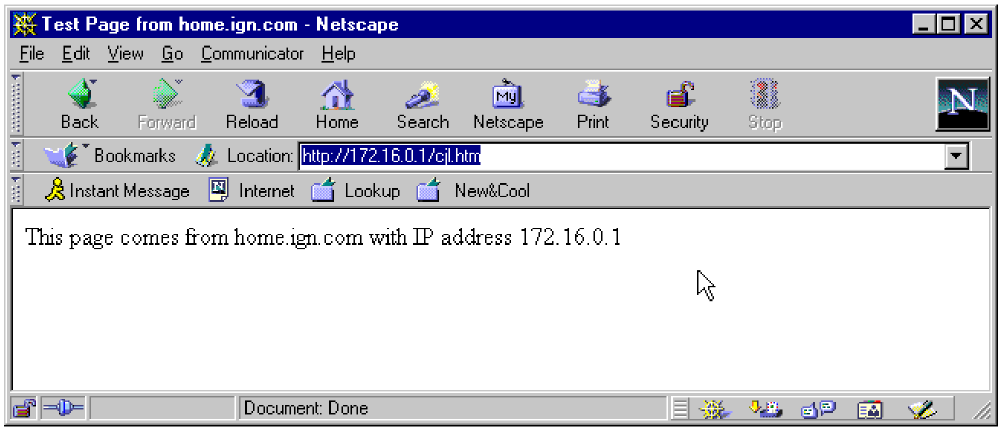
Figure 244. The Cluster Address Is Added to the HTTP Proxy Field (proxy03)

The other fields in this window could be used to enter the IP addresses or host names and ports of systems running proxy servers for the other protocols. However, using other protocols was not necessary for the purposes of this scenario.

Another change we made before generating the HTTP request was also done from the Advanced Preferences selection. We selected the **Cache** item and set the size of memory and disk cache to `0 KBytes`. Finally, we clicked **Navigator** and we set the expiration time of pages in the history to `0 days`. By doing this, we did not have to clear history, memory cache, and disk cache for the browser each time we made a request. Working with no pages in the history and cache is very important when working in a test environment, to make sure that pages are really served by the servicing Web server and not by the local cache.

11.2.2.7 Scenario Results

Once the above configuration was complete, we were able to use the Netscape browser on the secure client machine to request our test Web page from the machine running Lotus Domino Go Webserver on the nonsecure side of the firewall system through the HTTP proxy server:


Test Page from home.ign.com - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location:
Instant Message Internet Lookup New&Cool

http://172.16.0.1/cjl.htm

This page comes from home.ign.com with IP address 172.16.0.1

Document: Done

Figure 245. Nonsecure Request Results through the Proxy Firewall

Following this, we made several more Web page requests. From the Dispatcher machine we verified that the Dispatcher was load balancing these requests to the two Firewall machines. We used the ND GUI port monitor to see these results:

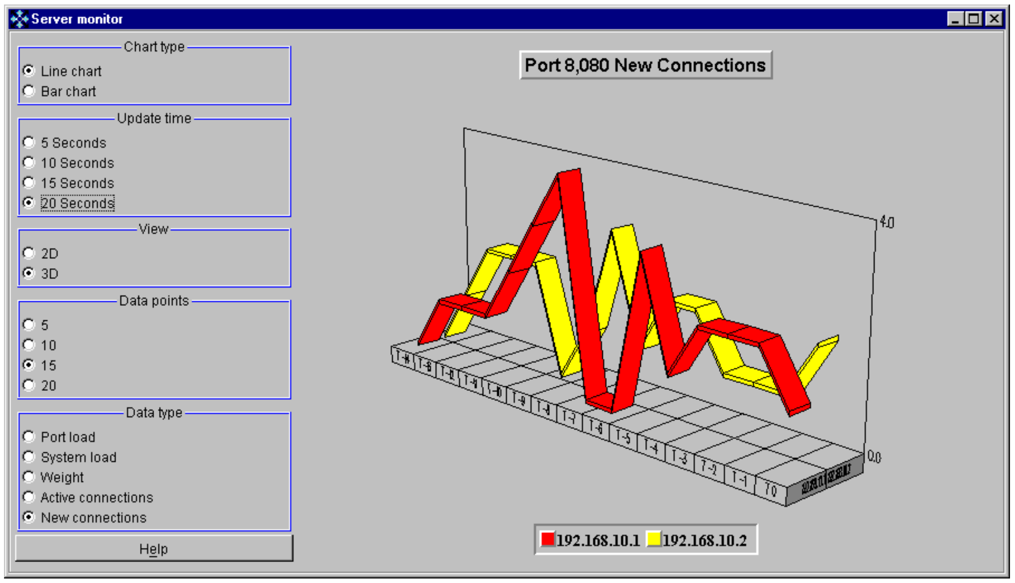


Chart type
- [✓ Line chart]
- Bar chart

Update time
- 5 Seconds
- 10 Seconds
- 15 Seconds
- [✓ 20 Seconds]

View
- 2D
- [✓ 3D]

Data points
- 5
- 10
- [✓ 15]
- 20

Data type
- Port load
- System load
- Weight
- Active connections
- [✓ New connections]

Port 8,080 New Connections

🟥 192.168.10.1 🟨 192.168.10.2

Figure 246. ND GUI Proxy Port Requests Being Distributed by the Dispatcher

Note that we could have used the wildcard port and cluster in this implementation but it is really not necessary, as the client packets in this case are always destined for a specific address.

Once again, firewall high availability is an inherent feature of firewall load balancing. If one of the firewall machines experiences a problem or needs to be removed from service for any reason, a Dispatcher advisor can automatically detect this and distribute requests to the remaining firewall machine as part of its normal load-balancing duties.

IP Forwarding on the Dispatcher Machine

IP forwarding should be turned off on the Dispatcher machine. When IP forwarding was turned on on the Dispatcher machine, we noticed a significant increase in the network traffic originating from the Dispatcher. And, possibly as a result of the increased network traffic, there appeared to be a significant increase in the length of time required to receive the response to our Web page request, relative to when IP forwarding was turned off.

11.2.3 FTP Proxy Server Load-Balancing Scenario

By adding on to the scenario described in 11.2.2.2, “Scenario Implementation” on page 296, we also implemented a firewall FTP proxy server load-balancing architecture. The configuration required on the Dispatcher machine consisted of simply adding another port to the already existing cluster. We added the FTP port number 21 (the FTP *control* port) and then added the same two firewall secure IP addresses that had been used for port 8080 as the servers for this port. We could also have implemented this by using a new cluster.

Of course, both the firewalls had to be configured to act as FTP proxy servers. Consult the IBM redbook *Internet Security in the Network Computing Framework*, SG24-5220 for further information on how to configure the FTP proxy server with IBM eNetwork Firewall.

The method we used on the secure client machine to initiate the FTP request to the nonsecure server was to initiate the FTP request to the FTP proxy server from the command line:

```
ftp 192.168.10.30
```

After logging into the firewall machine with a user ID that had been previously set up, we entered the following command in response to the FTP prompt:

```
ftp> quote site 172.16.0.0
```

We were then presented with an FTP login prompt from the nonsecure server. We logged in and were able to initiate the FTP file transfer with FTP commands.

Once again, monitoring the load-balancing activities with the ND GUI we had the confirmation that the Dispatcher was effectively load balancing the two FTP proxy servers, too.

11.2.4 Firewall SOCKS Server Load-Balancing Scenario

In addition to extending the scenario described in 11.2.2.2, “Scenario Implementation” on page 296 for use with an FTP proxy server port, we also extended the HTTP proxy server load-balancing scenario to allow our Dispatcher to load balance requests to two SOCKS servers running on the same two firewall machines.

11.2.4.1 SOCKS Servers

SOCKS technology involves a SOCKS server running on the firewall machine. The client uses a TCP protocol named SOCKS to communicate with the SOCKS server. SOCKS technology provides security by encapsulating any TCP protocol in the SOCKS protocol. This starts on the client machine where each TCP packet is encapsulated within a SOCKS packet and then transmitted to the SOCKS server on the firewall. The SOCKS server extracts the required information from each packet, such as destination address and port number, and then sends the data. As was also the case with proxy server technologies, the session is broken by the firewall and once again the secure user is hidden to the Internet, so nonsecure hosts cannot know that a secure user even exists. The source IP address that is exposed to the Internet is that of the SOCKS server itself. The IP address or host name of the machine from which the request originated is not exposed to the nonsecure network. Intruders and hackers cannot access internal information from your intranet.

When a non-secure host sends a response back to the secure client, it sends the response back to the SOCKS server, which then encapsulates the packets in the SOCKS protocol and sends them back to the client machine in the secure network.

SOCKS technology has become very popular because it enables the firewall to simply allow any TCP/IP connection (any TCP protocol and any port number) between the firewall itself and the nonsecure network, protecting the secure network by denying all the connections to the secure network that have been initiated by the Internet.

Modifications are required on the client in order for the client machine to be able to use the SOCKS protocol to communicate with the SOCKS server on the firewall machine. There are two possible ways to do this:

1. Recompile the client software to link the network client code with the SOCKS libraries to obtain *SOCKSified* client code.

   This option will not work for you unless you have the client source code. Fortunately, several client applications, such as Netscape Communicator, Sun HotJava, and Microsoft Internet Explorer, offer built in support for communicating with a SOCKS server using the SOCKS protocol. The client application must be configured by specifying the SOCKS server's IP address (the cluster address in this case) and the TCP port used on the SOCKS server (this is usually 1080).

2. Replace the dynamically linked libraries that implement the TCP calls with a SOCKSified version, named *SOCKSified TCP stack.*

   The SOCKSified TCP stack can be used with any client application, without the need to modify the code, because the SOCKSification is performed at the operating system level. In this case it is still necessary for the client program to be aware of the SOCKS server's IP address or host name.

11.2.4.2 Scenario Implementation

The scenario we implemented to demonstrate SOCKS server load balancing was completely based on the Windows NT 4.0 platform. On all the machines, Service Pack 3 had been applied, together with the necessary fixes on the firewall machines.

The configuration of the machines we used in this scenario is shown in the following table:

Table 22. Basic Scenario - Hardware, Software and Network Configuration

Workstation	Operating System	Secure LAN IP Address	Nonsecure LAN IP Address	Service
IBM PC 365	Windows NT Server 4.0	9.24.104.7		Secure Web client
IBM PC 365	Windows NT Server 4.0	9.24.104.31		Secure Dispatcher nonforwarding address
		9.24.104.30		Secure Dispatcher cluster address
IBM PC 365	Windows NT Server 4.0	9.24.104.56	150.24.104.59	Firewall 1
IBM PC 365	Windows NT Server 4.0	9.24.104.57	150.24.104.58	Firewall 2
IBM PC 365	Windows NT Server 4.0		150.24.104.5	Nonsecure Web server

Implementing a SOCKS server load-balancing scenario is no different from implementing a proxy server load-balancing scenario, because with both technologies the firewall breaks the communication between client and server. For this reason, you can safely repeat the same steps described in 11.2.2, “HTTP Proxy Server Load Balancing Scenario” on page 295 or 11.2.3, “FTP Proxy Server Load-Balancing Scenario” on page 303.

However, chances exist that after configuring the SOCKS server load-balancing scenario, it simply does not work on the first try. To understand what is wrong in the configuration, you should first of all enable the SOCKS server debug option, which traces the SOCKS proxy actions. On a Windows NT 4.0 system, you should follow these steps:

1. Run the Registry Editor, by entering the `regedit` command on a Command Prompt window.
2. From the HKEY_LOCAL_MACHINE subtree, select the **\SOFTWARE\IBM\IBMFirewall\3.3\Socks\Server** key.
3. From the Edit menu, select **Add String Value.**
4. Add a new string called:

   ```
   Debug Log File
   ```

   and set it to your desired file name and path, for example `C:\TEMP\SOCKS.LOG`.
5. Click **OK.**
6. Quit the Registry Editor.

The debug becomes available without any further action. However, it is advisable to remove this registry entry as soon as you finish debugging, as the log file grows quickly.

After another try to connect from the client via the cluster of SOCKS servers, we checked the created log file and found the following entries:

```
[175] Accept: Dispatching thread 1 of 64...
[175] Accept: Waiting on accept or a signal
[195] Accept: Thread beginning...
[195] Check: Checking host address (00681809 == 00683596)
[195] Route: Line 61: Destination host didn't match
[195] Check: Checking host address (00681809 == 00681809)
[195] Check: Checking port range   (0 <= 3562 <= 65535)?
[195] Route: Line62: Matched
[195] Proxy: Received connection via wrong route
[195] Proxy: closing monitor handle
[195] Proxy: cleaning input io context
[195] Proxy: done cleaning up
[195] Accept: Thread exiting...
```

Notice that, in the output above, bracketed numbers are thread numbers.

Notice that the log files mention in particular lines 61 and 62 in the SOCKS configuration file. A closer look at this file showed the following:

```
[61] route 150.24.104.0/255.255.255.0 - 150.24.104.59
[62] route 9.24.104.0/255.255.255.0 - 9.24.104.56
[63] route 9.24.104.0/255.255.255.0 - 9.24.104.30
```

These entries are related to some kind of security check. In this case, bracketed numbers are the line numbers. Line 61 indicates that incoming traffic for the nonsecure network will come over 150.24.104.59, while lines 62 and 63 indicate that incoming traffic for the secure network will come over 9.24.104.56 and 9.24.104.30 respectively. The rules will be checked from top to bottom, and because packets in our scenario will come from 9.24.104.30 (the cluster address), line 62 will deny our requests, and line 63 will never be reached.

We removed line 62 from the SOCKS configuration file with a text editor, and then forced the SOCKS server to reload the configuration by entering the command:

```
socks5 -config
```

Of course, this has to be done *on all of the load-balanced firewalls*.

Our new try with the Web browser on the client pointing to the Web server on the nonsecure network, was now successful. Your experience may vary since the order can be different between lines 62 and 63.[3]

SOCKS Server Configuration

There is actually no way to change the order of the rules with the firewall GUI by manually editing the SOCKS5.CONF configuration file. Since regenerating the firewall rules will also create a new SOCKS5.CONF configuration file, we have to fix it and then tell the SOCKS server to load this new configuration.

[3] It looks like the order of these entries depends on the order of the installation of the network adapters.

11.2.5 DNS Proxy Server Load-Balancing Scenario

By adding to a scenario similar to the one described in 11.2.2.2, “Scenario Implementation” on page 296, we also implemented a firewall Domain Name System (DNS) proxy server load-balancing scenario. The configuration required on the Dispatcher machine consisted of simply adding another port to the already existing cluster. We added the TCP port number 53 (the DNS port) and then added the same two firewall secure IP addresses that had been used for port 8080 as the servers for this port. We could also have implemented this by using a new cluster.

Both the firewall machines had to be configured to permit DNS queries. Consult the IBM redbook *Internet Security in the Network Computing Framework*, SG24-5220 for further information on how to configure the proxy server with IBM eNetwork Firewall.

In this scenario, we configured either secure client machine to use a DNS server that was also located on the secure LAN side of the firewalls. This DNS server was configured to forward DNS requests to the Dispatcher cluster address. The Dispatcher then load balanced the request to one of the firewall machines. Both firewalls had a DNS server configured and both contained the same forward rule to route the DNS requests for addresses on the nonsecure LAN out through the nonsecure firewall interface to a DNS server on the nonsecure network.

The client initiated the request by performing the command:

```
nslookup hostname
```

where *hostname* was the name of a host on the nonsecure network (served by the nonsecure DNS server).

As DNS is TCP/UDP protocol application, and as UDP is a connectionless protocol, each packet contains a bit (known as the FIN bit) indicating that this is the end of the synchronization (because no further synchronization is done). The effect of this on the Dispatcher is that it load balances each packet as if it were a new request. Our `nslookup` command was successful; however, we noticed the following pattern of server usage from our Port 53 GUI monitor:

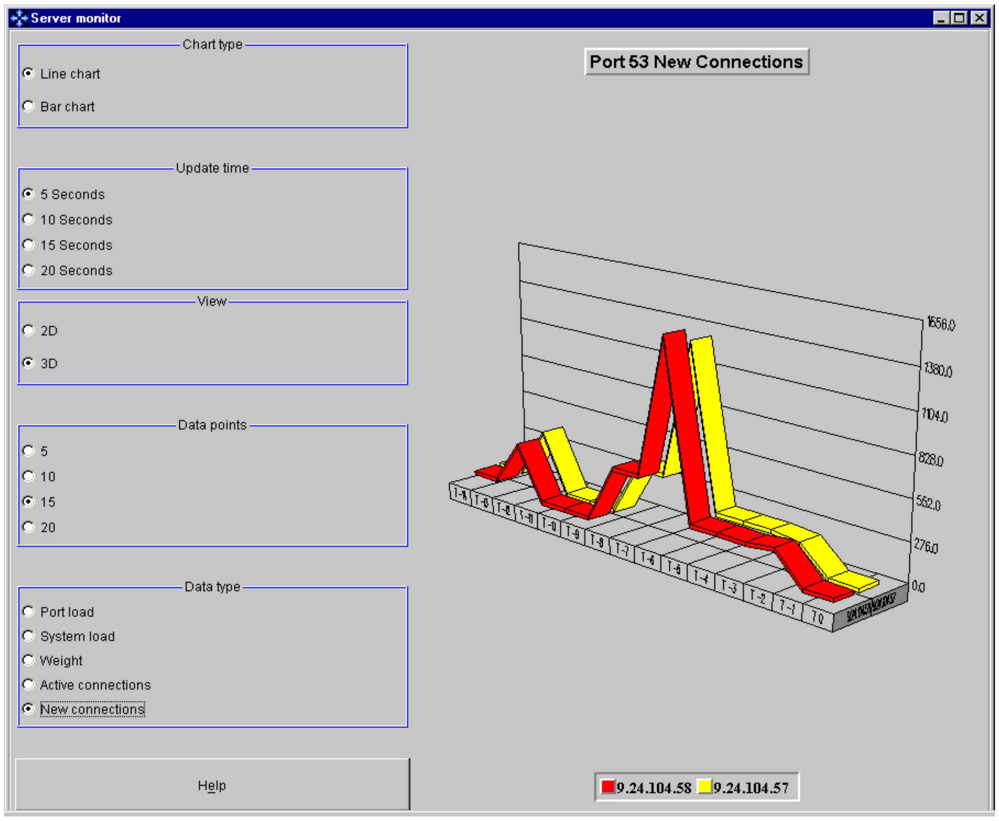

Server monitor

Chart type
[✓ Line chart]
Bar chart

Update time
[✓ 5 Seconds]
10 Seconds
15 Seconds
20 Seconds

View
2D
[✓ 3D]

Data points
5
10
[✓ 15]
20

Data type
Port load
System load
Weight
Active connections
[✓ New connections]

Help

Figure 247. DNS UDP Load-Balancing Monitor

This visually confirms that the Dispatcher is distributing the requests to both of the servers on an individual packet basis according to the weights of each of the servers.

Chapter 12. Automatic ND Startup on Windows NT

The benefits of being able to start the Dispatcher service along with a fully operational cluster environment in unattended mode are many. The most notable among these is that human intervention is not required to restart the Dispatcher service after a hardware outage. By adding a second backup Dispatcher machine and configuring Dispatcher high availability, you have a very robust highly available Dispatcher environment that can operate almost entirely without having someone immediately available to help restore service.

While on AIX and Solaris systems, the IBM SecureWay Network Dispatcher (ND) processes start by default without the need for a user to physically log in, this is not the case on Windows NT. On this platform, by default, a system administrator user must log in for the ND processes to start. In this section, we demonstrate the procedures we followed to automate on Windows NT the startup of a highly available Dispatcher service including the configuration of the complete cluster environment on both the Primary and Backup Dispatcher machines. With simple modifications, the procedure described in this chapter can also be applied to a Dispatcher scenario where high availability is not implemented.

12.1 High Availability Dispatcher Autostart Scenario

In this scenario we implement a standard Dispatcher high-availability environment with two Dispatcher machines and two Web servers.

12.1.1 Network Environment

The network environment that we used in this scenario was very similar to the one used in Chapter 6, “ND High Availability Support” on page 177 (see 6.1.1, “Network Architecture” on page 178), except that in this case both of our Dispatcher machines were running Windows NT Server. A summary of the hardware, software, and network configuration of the systems we used is shown in the following table:

Table 23. HA Dispatcher Autostart Scenario - Hardware, Software, and Network Configuration

Workstation	Operating System	Host Name	IP Address	Service
IBM PC 365	Windows NT Server 4.0	WTR05331	9.24.104.245	Primary Dispatcher
		clusterend	9.24.104.105	
IBM PC 365	Windows NT Server 4.0	WTR05084	9.24.104.79	Backup Dispatcher
		clusterend	9.24.104.105	
IBM RS/6000 43P	AIX 4.3.1	aixncf157	9.24.104.157	Web server
IBM RS/6000 43P	AIX 4.3.1	aixafs	9.24.104.158	Web server

As you can see from the table above, the network interface on each Dispatcher machine is configured to respond to two IP addresses and host names: the non-forwarding address and the cluster address respectively.

The following figure shows a graphical representation of our network environment:

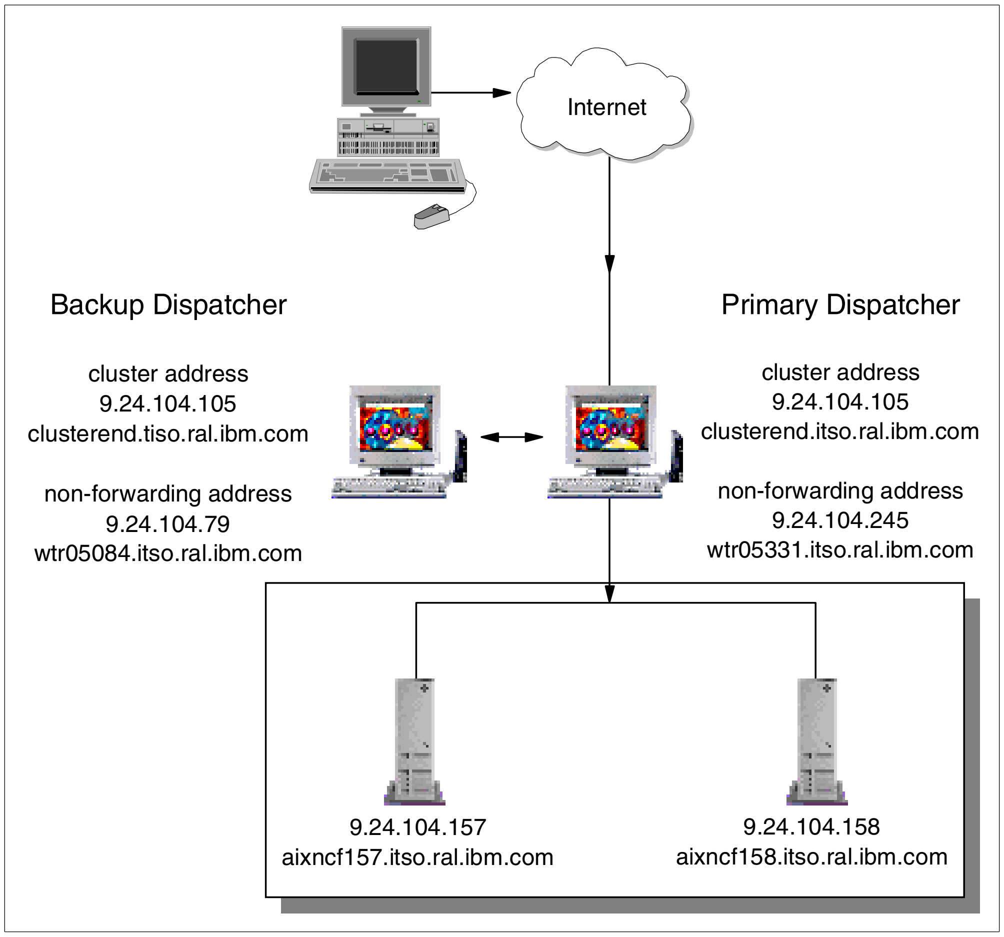


Figure 248. HA Dispatcher Autostart Scenario Environment

12.1.2 Configuration Steps

To configure the highly available cluster environment, we did not use either the ND graphic user interface (GUI) or `ndcontrol` commands from the command line. Instead we placed all of the `ndcontrol` commands, including the command to start the Executor, in a file. We examined several options for launching the command file on both our primary and backup Dispatcher machines and were successful with the following methods:

1. Place the command file in the Startup directory of a user with Administrator authority. This has the obvious disadvantage that the user must be present to log in to the Windows NT machine to launch the server.
2. The second alternative was to use an automated login. The main drawback with this option was the obvious security exposure in having an Administrator ID logged in to an unattended machine.
3. The third and best option we found was to make use of the Microsoft Windows NT Server Resource Kit, to launch our command file as a Windows NT service automatically at boot.

It is the third option that we demonstrate in this scenario.

12.1.3 The AUTOEXNT.BAT File

After installing the Microsoft Windows NT Server Resource Kit, we followed the instructions provided with the Resource Kit for enabling the AutoExNT Service. To enable the AutoExNT Service to launch our command file as a service at boot time, we again followed the instructions in the Autoexnt.txt file, and placed our commands into a file called AUTOEXNT.BAT, which we located in the directory C:\Winnt\system32.

Our AUTOEXNT.BAT file contained the following:

```
@echo off
rem
rem    file: C:\Winnt\system32\AUTOEXNT.BAT
rem    by: C. Letilley and M. Pistoia March 29, 1999
rem purpose: ND High Availability configuration
rem          <primary machine>
rem =======================================
rem

rem  use the net utility to wait for IBM Network Dispatcher service to start
call net start "IBM Network Dispatcher"

date /t
time /t
echo "computer is" %COMPUTERNAME%

rem -- set High Availability option
rem --  "NOHA" = no High Availability,  "HA" = use High Availability
rem set OPTION="NOHA"
set OPTION="HA"

rem -- set High Availability recovery mode
rem set MODE=manual
set MODE=auto

rem -- set Dispatcher operational role (Primary or Backup)
rem
rem -- set NFAs, netmask and gateway for Dispatcher machines

set WTR05331IP=9.24.104.245
set WTR05084IP=9.24.104.79
set NETMASK1=255.255.255.0
set GATEWAYADDR=9.24.104.1

rem assign role (Primary or Backup) based on computer name
GOTO %COMPUTERNAME%

:WTR05331
rem following lines for WTR05331
set ROLE=primary
set SRCNFA1=%WTR05331IP%
set DSTNFA1=%WTR05084IP%

GOTO COMMON

:WTR05084
rem following lines for WTR05084
set ROLE=backup
set SRCNFA1=%WTR05084IP%
set DSTNFA1=%WTR05331IP%
```

Figure 249. (Part 1 of 3). AUTOEXNT.BAT

```
:COMMON
rem
rem -- if role = backup, then sleep
rem    until primary comes up
rem
if %ROLE% == "backup" sleep 60

rem --set cluster address
set CLUSTER=9.24.104.105

rem Optionally set servers to be monitored for outages
rem   1)web servers
rem     AIXNCF157 = 9.24.104.157
rem     AIXAFS = 9.24.104.158
rem   2)main frame gateways
rem
set WEBSERVER1=9.24.104.157
set WEBSERVER2=9.24.104.158

rem  -- Start the executor
call ndcontrol executor start
rem sleep 4

rem Alias the SRCNFA address to the network adapter
rem if  heartbeat is on alternate network, then alias it
call ndconfig tr0 alias %SRCNFA1% netmask %NETMASK1%

rem Set the non-forwarding address (NFA)
call ndcontrol executor set nfa %SRCNFA1%

rem Create the cluster
call ndcontrol cluster add %CLUSTER%

rem Add a port to the cluster.  This causes ND to listen on port 80.
call ndcontrol port add %CLUSTER%:80

rem Optionally set the port sticky time for this website to 20 minutes-
rem to allow for HTTP session state management completion
rem call ndcontrol port set %CLUSTER%:80 stickytime 1200

rem Add HTTP servers to the cluster
call ndcontrol server add %CLUSTER%:80:%WEBSERVER1%+%WEBSERVER2%

rem Start the manager and log to manager.log
rem set logsize to 1 megabyte
call ndcontrol manager start manager.log
call ndcontrol manager logsize 1400000
```

Figure 250. (Part 2 of 3). AUTOEXNT.BAT

```
rem Start the HTTP advisor on port 80 and log to Http_80.log
rem set advisor HTTP/HEAD request to 5 second interval
rem set logsize to 1 megabyte
call ndcontrol advisor start http 80 Http_80.log
call ndcontrol advisor interval http 80 5
call ndcontrol advisor logsize http 80 1400000

rem Set the manager's proportions such that it uses:
rem active(58) new(40)  advisor(2) ISS(0)
rem to determine inactivity (or downed) webservers
call ndcontrol manager proportions 58 40 2 0

if %OPTION%=="HA" GOTO START-HA
rem Alias the cluster address to the network adapter
rem we are now ready to start accepting packets
call ndconfig tr0 alias %CLUSTER% netmask %NETMASK1%
exit

:START-HA
rem --establish heartbeat connection between dispatchers
call ndcontrol highavailability heartbeat add %SRCNFA1%  %DSTNFA1%

rem --establish reach connections for advisors
call ndcontrol highavailability reach add %GATEWAYADDR%

rem -- setup highavailability role mode and port
call ndcontrol highavailability backup add %ROLE% %MODE% 12345
time /t
call ndcontrol high stat
rem executor will run goActive or goStandby batch file to activate routing
```

Figure 251. (Part 3 of 3). AUTOEXNT.BAT

As you can see from the AUTOEXNT.BAT file, we were able to use the same file on both the primary and backup Dispatcher machines.

12.1.4 High-Availability Script Files

We also modified the three scripts goActive.cmd.sample, goStandby.cmd.sample, and goInOp.cmd.sample, provided in the *installbase*/samples directory for our cluster environment. We placed the three modified go scripts in the *installbase*/bin directory. We renamed them removing the .sample extension, and we made sure that they were executable. Then we customized them as follows:

```
@echo off
rem
rem goActive script
rem
rem Configure this script when using the high availability feature of
rem Network Dispatcher.
rem
rem This script is executed when Network Dispatcher goes into the
rem 'Active' state and begins routing packets.
rem
rem This script must be placed in Network Dispatcher's bin directory
rem (by default this is C:\Program Files\nd\dispatcher) and it
rem must have the extension .cmd in order to be executable.
rem
rem Modify CLUSTER, INTERFACE and NETMASK to match your environment.
rem
rem tr0=first Token ring adapter, en0=first Ethernet adapter
rem
rem NETMASK must be the netmask of your LAN.  It may be hexadecimal or octal
notation.
rem
set CLUSTER=9.24.104.105
set INTERFACE=tr0
set NETMASK=255.255.255.0
rem
echo "Deleting loopback alias(es)
call ndconfig lo0 delete %CLUSTER%
rem
echo "Adding device alias(es)"
call ndconfig %INTERFACE% alias %CLUSTER% netmask %NETMASK%
```

Figure 252. Modified goActive.cmd Script

```
@echo off
rem
rem goStandby script
rem
rem Configure this script when using the high availability feature of
rem Network Dispatcher.
rem
rem This script is executed when Network Dispatcher goes into the
rem 'Standby' state.  Monitoring the health of the 'Active' machine
rem but not routing packets.
rem
rem This script must be placed in Network Dispatcher's bin directory
rem (by default this is C:\Program Files\nd\dispatcher) and it
rem must have the extension .cmd in order to be executable.
rem
rem Modify CLUSTER, INTERFACE and NETMASK to match your environment.
rem
rem tr0=first Token ring adapter, en0=first Ethernet adapter
rem
set CLUSTER=9.24.104.105
set INTERFACE=tr0
set NETMASK=255.255.255.0
rem
echo "Deleting the device alias(es)"
call ndconfig %INTERFACE% delete %CLUSTER%
rem
echo "Adding loopback alias(es)"
call ndconfig lo0 alias %CLUSTER% netmask %NETMASK%
```

Figure 253. Modified goStandby.cmd Script

```
@echo off
rem
rem goInOp script
rem
rem Configure this script when using the high availability feature of
rem Network Dispatcher and optionally when using Network Dispatcher in a
rem standalone environment.
rem
rem This script is executed when the Network Dispatcher executor is stopped
rem (and before the executor is initially started).
rem
rem This script must be placed in Network Dispatcher's bin directory
rem (by default this is C:\Program Files\nd\dispatcher) and it
rem must have the extension .cmd in order to be executable.
rem
rem Modify CLUSTER and INTERFACE to match your environment.
rem
rem tr0=first Token ring adapter, en0=first Ethernet adapter
rem
set CLUSTER=9.24.104.105
set INTERFACE=tr0
rem
echo "Removing device(s)"
call ndconfig lo0 delete %CLUSTER%
rem
echo "Removing loopback alias(es)"
call ndconfig %INTERFACE% delete %CLUSTER%
```

Figure 254. Modified goInOp.cmd Script

12.1.5 Testing the Configuration

After completing the above configuration, we first tested the AUTOEXNT.BAT script by invoking it from the command line on our two machines. Following is the output from it on the backup Dispatcher machine:

```
The IBM Network Dispatcher service is starting.
The IBM Network Dispatcher service was started successfully.



Mon 02/01/1999
11:36p
"computer is" WTR05084
Loaded kernel successfully.
Executor field(s) successfully set.

Cluster 9.24.104.105 has been added.

Port 80 successfully added to cluster 9.24.104.105.

Server 9.24.104.157 was added to port 80 of cluster 9.24.104.105.
Server 9.24.104.158 was added to port 80 of cluster 9.24.104.105.

The manager has been started.

The log size for the manager was set to 1,400,000.

Advisor 'http' has been started on port 80.
The interval for the advisor was set to 5.
The log size for the advisor was set to 1,400,000.
The proportions of the manager were set to: 58 40 2 0
Heartbeat information successfully added.
Reach information successfully added.
Backup information successfully added.
11:37p

High Availability Status:
-------------------------
Role ................. Backup
Recovery strategy .... Auto
State ................ Standby
Sub-state ............ Synchronized
Port ................. 12345
Preferred target ..... 9.24.104.245

Heartbeat Status:
-----------------
Count ................ 1
Source/destination ... 9.24.104.79/9.24.104.245

Reachability Status:
--------------------
Count ................ 1
Address .............. 9.24.104.1
```

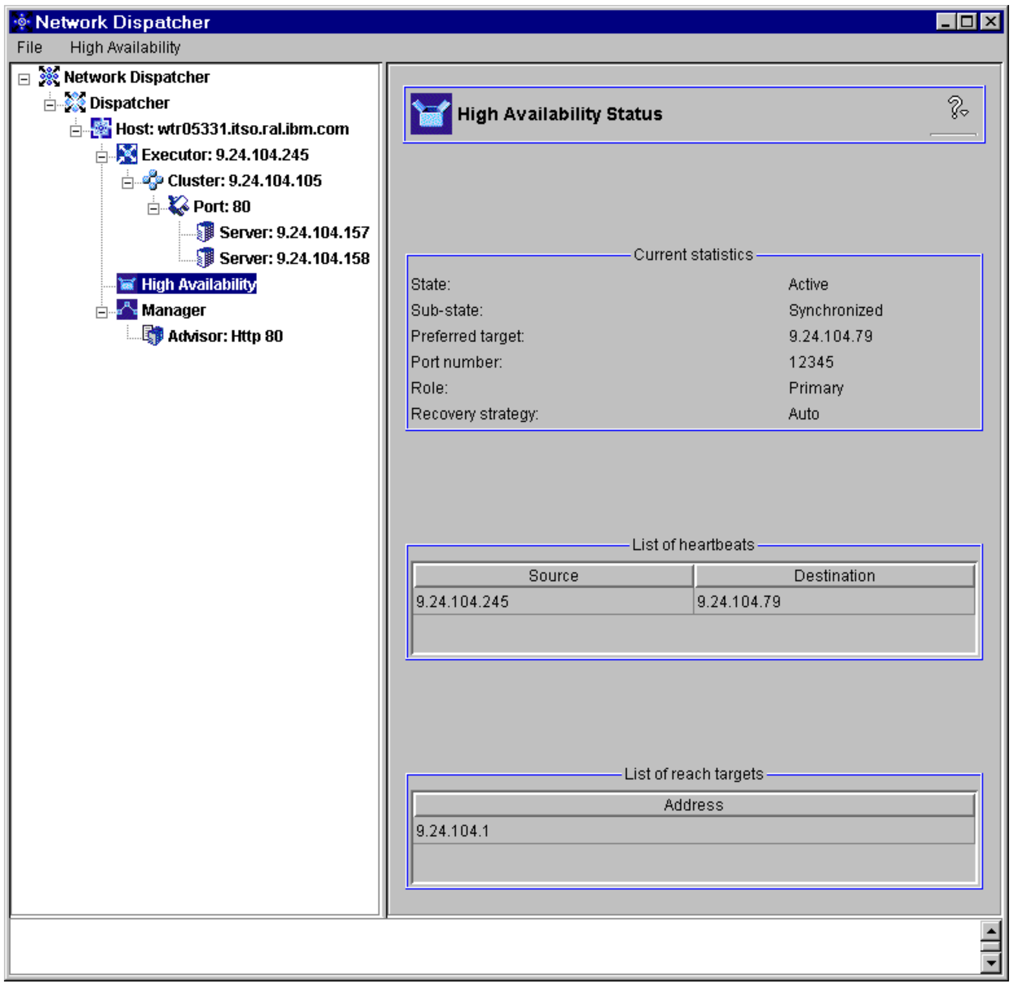
Figure 255. Script Output when Executed from the Command Line

After rebooting both machines, without logging on, we made a request from a browser on another machine connected to the same LAN. We had placed HTML files named cjl.html (each of them uniquely identifying the machine it was located on) on both of our servers. After disabling local caching by the browser and selecting Direct Connection to the Internet rather than using a Proxy server, we requested the page cjl.html at the cluster address and verified that the page was served regularly. This verified that our cluster was operational, as we were successful in our request for the page at the cluster address.

Following this, we physically disconnected the active primary Dispatcher machine from the LAN, simulating a machine failure. The standby backup Dispatcher machine immediately became active. We requested the Web page repeatedly during the transition and did not experience a significant delay in receiving the requested page.

Because the high availability takeover mode we specified in the AUTOEXNT.BAT script was `auto`, when **we** reconnected the LAN token-ring cable to the standby primary Dispatcher machine, it immediately became the active dispatcher once again.

At this time we logged on to the two Dispatcher machines. From the Start menu, we selected **Programs**, and then **SecureWay Network Dispatcher**. This started the ND GUI. As soon as we connected to each of the respective hosts, we saw the following GUI windows confirming the high availability status of both machines:



Current statistics

State:	Active
Sub-state:	Synchronized
Preferred target:	9.24.104.79
Port number:	12345
Role:	Primary
Recovery strategy:	Auto

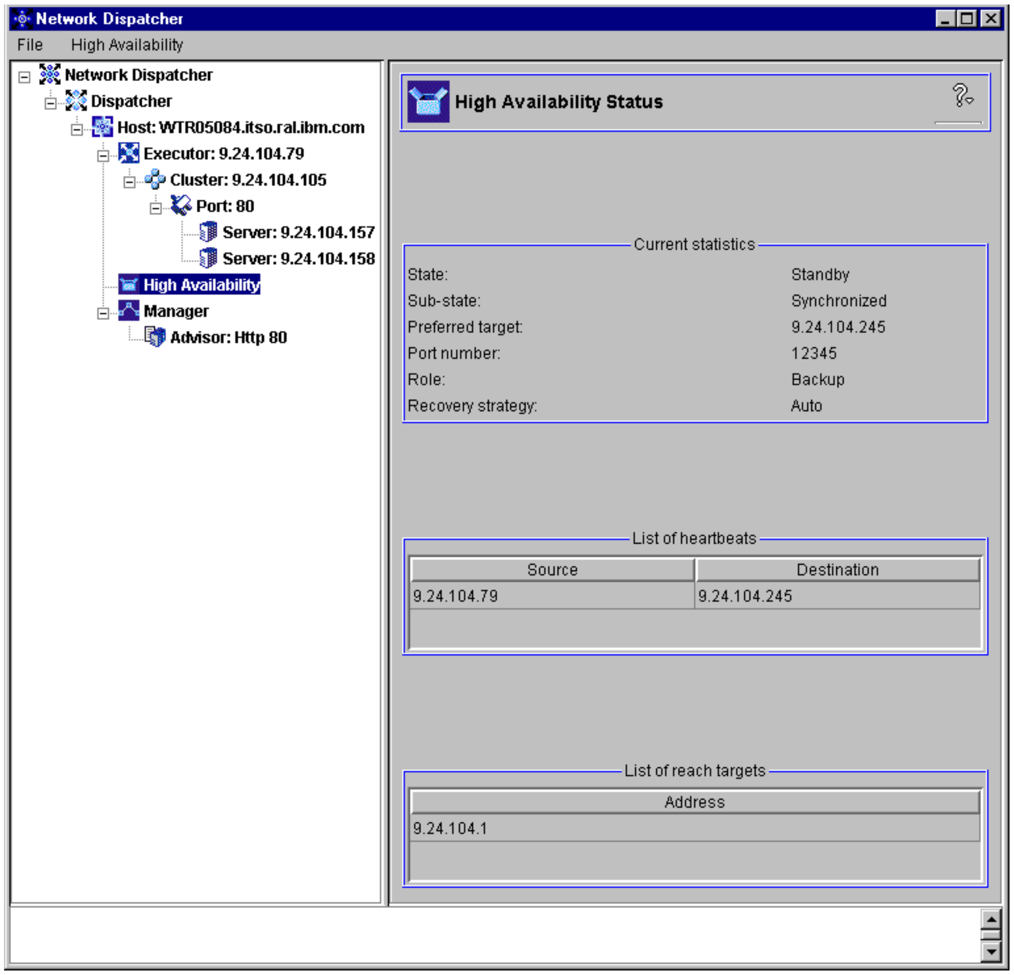
List of heartbeats

Source	Destination
9.24.104.245	9.24.104.79

List of reach targets

Address
9.24.104.1

Figure 256. Active Primary Dispatcher HA Status



Current statistics

State:	Standby
Sub-state:	Synchronized
Preferred target:	9.24.104.245
Port number:	12345
Role:	Backup
Recovery strategy:	Auto

List of heartbeats

Source	Destination
9.24.104.79	9.24.104.245

List of reach targets

Address
9.24.104.1

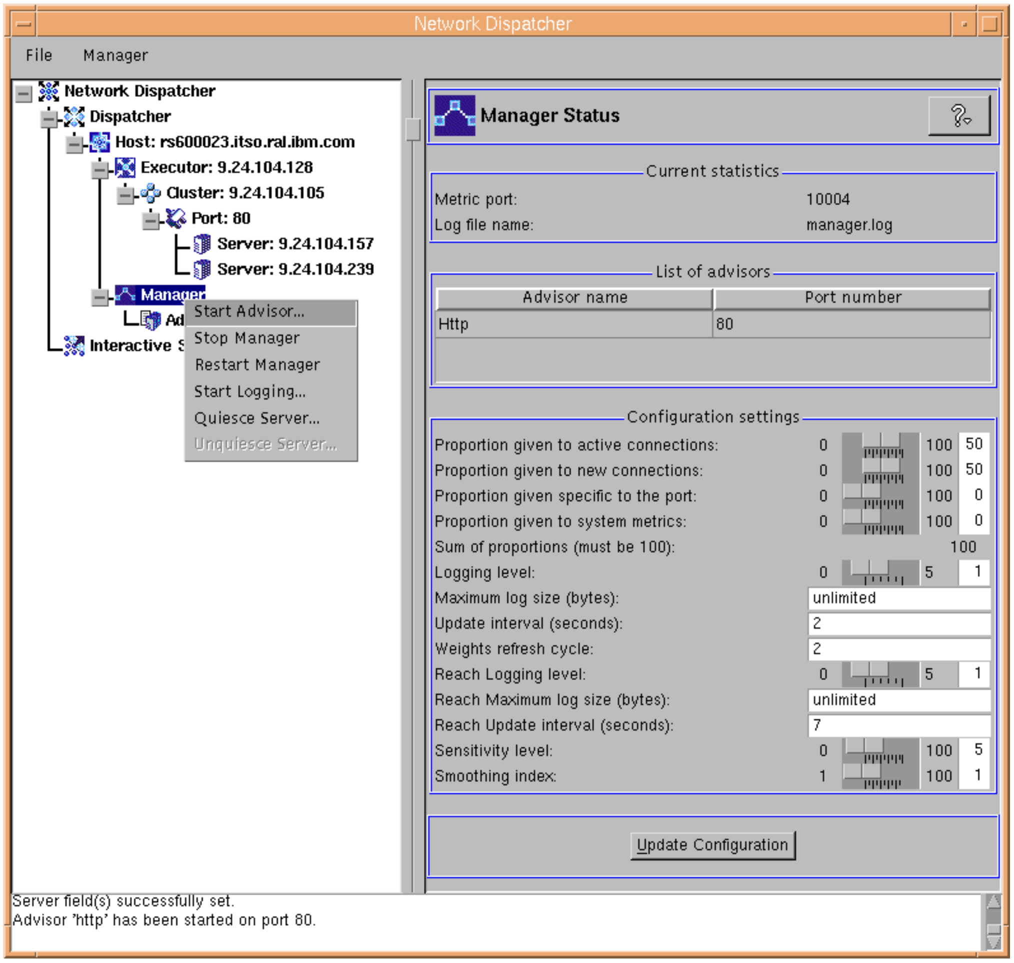
Figure 257. Standby Backup Dispatcher HA Status

Chapter 13. Binary Logging and Statistics

The Dispatcher and CBR components of IBM SecureWay Network Dispatcher (ND) Version 2.1 now provide an optional binary logging facility. The purpose of this facility is to allow you to analyze server usage and trends. The Manager component must be running in order for information to be logged to the binary logs.

13.1 Starting the Logging Facility

We turned on the logging facility with the ND graphical user interface (GUI) for a small cluster environment that was a subset of the one we used in 4.1, "Load Balancing Basic Scenario Using the Dispatcher" on page 81. To start capturing data to the log files, we right-clicked the **Manager** item and from the pop-up menu selected **Start Logging...** as shown in the following figure:



Metric port: 10004
Log file name: manager.log

Advisor name	Port number
Http	80

Proportion given to active connections:	0	100	50
Proportion given to new connections:	0	100	50
Proportion given specific to the port:	0	100	0
Proportion given to system metrics:	0	100	0
Sum of proportions (must be 100):			100
Logging level:	0	5	1
Maximum log size (bytes):			unlimited
Update interval (seconds):			2
Weights refresh cycle:			2
Reach Logging level:	0	5	1
Reach Maximum log size (bytes):			unlimited
Reach Update interval (seconds):			7
Sensitivity level:	0	100	5
Smoothing index:	1	100	1

```
Server field(s) successfully set.
Advisor 'http' has been started on port 80.
```

Figure 258. Start the Binary Logging Facility

Using the Command Line

We also could have turned on logging with this command:

```
ndcontrol log start
```

Several other log commands are also available:

```
ndcontrol log stop
ndcontrol log status
ndcontrol log set interval seconds
ndcontrol log set retention hours
```

Statistics will be placed in the logs at a user-specified interval. The interval can be changed via the GUI or the command line.

13.2 Examining the Log Files

When the logging facility is turned on, one log is created at the start of every hour with the date and time as the name of the file. The logs are placed in the directory *installbase*/logs, where *installbase* varies by component (Dispatcher or CBR) and by operating system. See Table 1 on page 69 for a list of the *installbase* locations.

13.2.1 Using the LOG_SampleReader Sample Java Program

The data is recorded in the log in binary format and therefore users are not able to read the raw log files directly. For this reason, a sample program is provided to be used primarily as a reference for you to use when writing your own program to read the binary logs. It can also be used as supplied to extract one type of log entry. When the Dispatcher is installed, the sample program, called LOG_SampleReader.java, is placed in the location *installbase*/lib/BinaryLog. A copy of this sample code can be found in Appendix C, “LOG_SampleReader Program” on page 407.

In this section we explain in general how the LOG_SampleReader sample Java program extracts the data from the binary logs. We also show how to invoke LOG_SampleReader and interpret the output from it.

13.2.1.1 Log Data Format

The installbase/lib/ibmnd.jar file for the Dispatcher component and the installbase/lib/ibmcbr.jar file for the CBR component provide the utilities (objects and their associated methods) that LOG_SampleReader uses to read the data from the log. The comments embedded in LOG_SampleReader.java inform us that the following types of records can be retrieved from the log:

- LOG_TimestampRecord
- LOG_ExecutorIDRecord
- LOG_ClusterIDRecord
- LOG_PortIDRecord
- LOG_ServerReportRecord

Each of these types has associated with it one or more methods that can be used to determine its value from the LOG_Record object. For example, the LOG_ServerReportRecord implements the following eight methods:

1. getIPAddress() returns the server IP address.
2. getWeight() returns the server weight of the server when the record was written.
3. getTotalConnections() returns the total number of server connections when the record was written.
4. getActiveConnections() returns the number of active connections.
5. getPortLoad() returns the port load of the server when the record was written.
6. getHasPortLoad() returns a Boolean value indicating whether or not port loads were being provided for the server when the record was written to the binary log (port loads are obtained through an advisor on the port).
7. getSystemLoad() returns system load of the server when the record was written.
8. getHasSystemLoad() returns a Boolean value indicating whether or not system loads were being provided for the server when the record was written to the binary log (system loads can be provided, for example, using ISS).

LOG_SampleReader.java extracts ServerReport data from the logs by using the LOG_Reader and LOG_Record objects and their associated methods. These methods scan through the log records searching for a record of type LOG_ServerReportRecord. When it finds one, it is printed.

13.2.1.2 Invoking the Sample Program

The Dispatcher and CBR components' *installbase*/lib/BinaryLog directories each contain a small shell script that serves as a wrapper for LOG_SampleReader. It sets up the required CLASSPATH elements and location of the log directory before invoking LOG_SampleReader through the `java` command.

Currently, the CBR wrapper file, cbrlogreport, contains a small mistake in the definition of the CLASSPATH system environment variable that prevents it from finding the *installbase*/lib/ibmcbr.jar file when it is executed. Until the problem is fixed in the next update, simply change the reference from `ibmnd.jar` to `imbcbr.jar` on the CLASSPATH definition in cbrlogreport.

Following is a copy of the Dispatcher wrapper file, ndlogreport, from our AIX system. The file on Windows NT or Solaris is very similar.

```
#!/bin/ksh
export CLASSPATH=\
/usr/lpp/nd/dispatcher/lib/BinaryLog:\
/usr/lpp/nd/dispatcher/lib/ibmnd.jar
java -DEND_LOG_DIRECTORY=/usr/lpp/nd/dispatcher/logs LOG_SampleReader $1 $2 $3 $4
```

Figure 259. ndlogreport Shell Script on AIX

We invoked this script with four arguments representing the start date and time and end date and time of the server records we wanted to see from the log. Following is the command as we entered on the AIX platform:

```
/usr/lpp/nd/dispatcher/lib/BinaryLog/ndlogreport 1999/03/08 19:05 1999/03/08 19:06
```

13.2.1.3 Interpreting LOG_SampleReader Output

The output from the above command is one or more lines with the following format:

```
1999/03/08-19:05:21.072,9.24.104.128,9.24.104.105,80,9.24.104.157,10,12,0,0,false,0,false
```

There are 13 fields on this line:

- A – date
- B – time
- C – Dispatcher nonforwarding address
- D – cluster IP address
- E – port number
- F – server IP address
- G – server weight
- H – total connections
- I – active connections
- J – port load of the server
- K – is the Dispatcher receiving port load information?
- L – server load
- M – is the Dispatcher receiving server load information?

The following figure shows each of the fields in a graphical fashion:

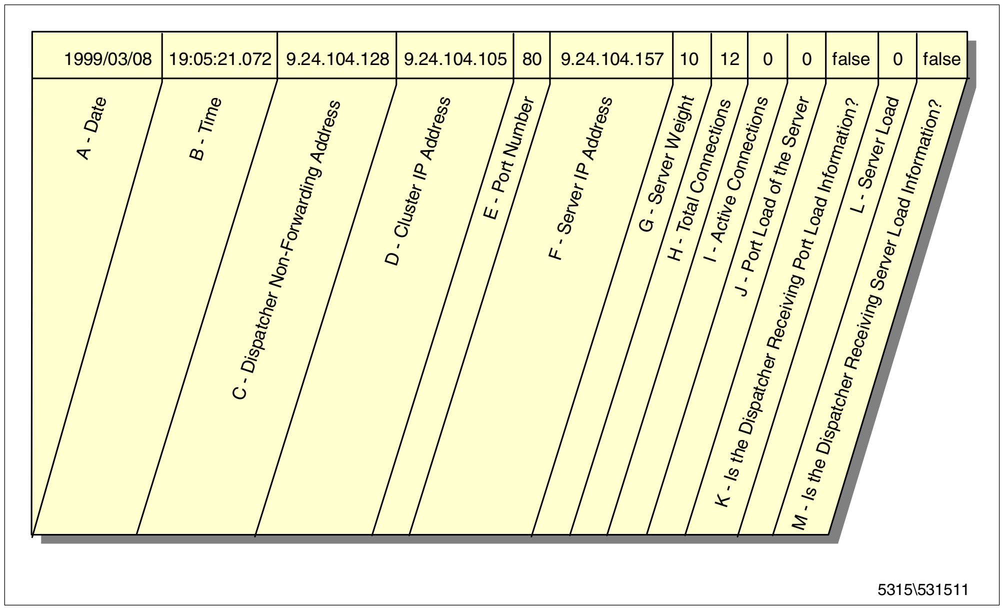
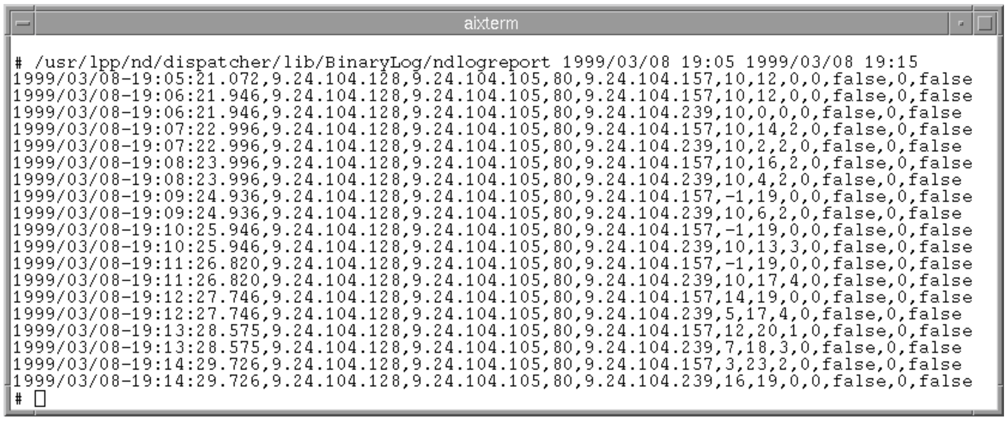


Figure 260. Format of ndlogreport Output

The last eight fields on the line correspond to the values returned by the eight methods that the LOG_ServerReportRecord implements as listed in Point 1 through Point 8 on page 323.

The following figure shows `ndlogreport` output for an 11-minute period:



```
# /usr/lpp/nd/dispatcher/lib/BinaryLog/ndlogreport 1999/03/08 19:05 1999/03/08 19:15
1999/03/08-19:05:21.072,9.24.104.128,9.24.104.105,80,9.24.104.157,10,12,0,0,false,0,false
1999/03/08-19:06:21.946,9.24.104.128,9.24.104.105,80,9.24.104.157,10,12,0,0,false,0,false
1999/03/08-19:06:21.946,9.24.104.128,9.24.104.105,80,9.24.104.239,10,0,0,0,false,0,false
1999/03/08-19:07:22.996,9.24.104.128,9.24.104.105,80,9.24.104.157,10,14,2,0,false,0,false
1999/03/08-19:07:22.996,9.24.104.128,9.24.104.105,80,9.24.104.239,10,2,2,0,false,0,false
1999/03/08-19:08:23.996,9.24.104.128,9.24.104.105,80,9.24.104.157,10,16,2,0,false,0,false
1999/03/08-19:08:23.996,9.24.104.128,9.24.104.105,80,9.24.104.239,10,4,2,0,false,0,false
1999/03/08-19:09:24.936,9.24.104.128,9.24.104.105,80,9.24.104.157,-1,19,0,0,false,0,false
1999/03/08-19:09:24.936,9.24.104.128,9.24.104.105,80,9.24.104.239,10,6,2,0,false,0,false
1999/03/08-19:10:25.946,9.24.104.128,9.24.104.105,80,9.24.104.157,-1,19,0,0,false,0,false
1999/03/08-19:10:25.946,9.24.104.128,9.24.104.105,80,9.24.104.239,10,13,3,0,false,0,false
1999/03/08-19:11:26.820,9.24.104.128,9.24.104.105,80,9.24.104.157,-1,19,0,0,false,0,false
1999/03/08-19:11:26.820,9.24.104.128,9.24.104.105,80,9.24.104.239,10,17,4,0,false,0,false
1999/03/08-19:12:27.746,9.24.104.128,9.24.104.105,80,9.24.104.157,14,19,0,0,false,0,false
1999/03/08-19:12:27.746,9.24.104.128,9.24.104.105,80,9.24.104.239,5,17,4,0,false,0,false
1999/03/08-19:13:28.575,9.24.104.128,9.24.104.105,80,9.24.104.157,12,20,1,0,false,0,false
1999/03/08-19:13:28.575,9.24.104.128,9.24.104.105,80,9.24.104.239,7,18,3,0,false,0,false
1999/03/08-19:14:29.726,9.24.104.128,9.24.104.105,80,9.24.104.157,3,23,2,0,false,0,false
1999/03/08-19:14:29.726,9.24.104.128,9.24.104.105,80,9.24.104.239,16,19,0,0,false,0,false
# 
```

Figure 261. LOG_SampleReader Output

We can determine from the above output that at the start of this 11-minute period, the server with IP address 9.24.104.157 was the only port 80 server in the cluster and one minute later another port 80 server, the one with IP address 9.24.104.239, was added to the cluster. Three minutes later, the server having IP address 9.24.104.157 (the original server) was marked down (see `-1` in the 8th row) and at this time, the number of active connections on the other server began to increase significantly. Two minutes later, the server with IP address 9.24.104.157 was back online (with a server weight of 14) and the two server nodes began sharing the load once again.

This program can be modified to process the binary log information to do any kind of analysis that you require.

Part 2. WebSphere Performance Pack Component Integration

Chapter 14. Content Based Routing

Content Based Routing (CBR) is an entirely new component within this version of SecureWay Network Dispatcher (ND). The purpose of CBR is to route client requests to specific servers based on the content of the URL request.

In this chapter we will show some scenarios using CBR. See *IBM WebSphere Performance Pack - Load Balancing with IBM SecureWay Network Dispatcher,* SG24-5858 for more information on how CBR works.

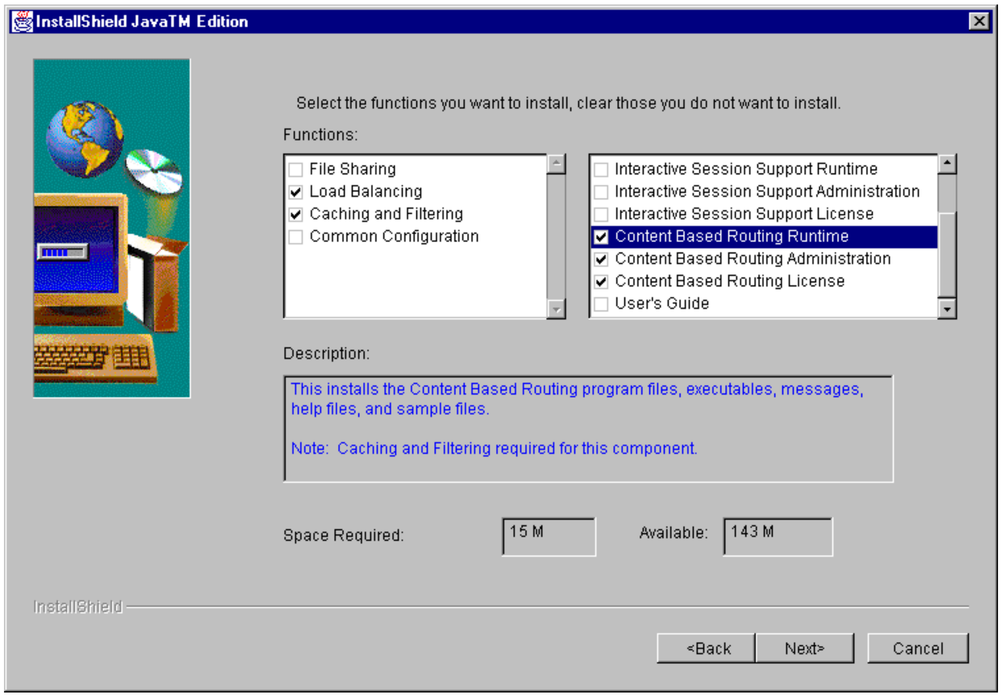
14.1 Installation of the CBR Function

The CBR component of ND Version 2.1 is supported on three operating systems: IBM AIX 4.2.1 or later, Microsoft Windows NT 4.0 and Sun Solaris 2.6 or later. Refer to the appropriate platform section of *IBM WebSphere Performance Pack - Load Balancing with IBM SecureWay Network Dispatcher,* SG24-5858 for details on how to use the Java InstallShield on your respective platform to install ND.

When you reach the point where you choose which ND component to install, select these three to install CBR on your machine:

- **Content Based Routing Runtime**
- **Content Based Routing Administration**
- **Content Based Routing License**

The following figure shows these three items as being selected:


InstallShield JavaTM Edition

Select the functions you want to install, clear those you do not want to install.

Functions:

- [ ] File Sharing
- [✓] Load Balancing
- [✓] Caching and Filtering
- [ ] Common Configuration

- [ ] Interactive Session Support Runtime
- [ ] Interactive Session Support Administration
- [ ] Interactive Session Support License
- [✓] Content Based Routing Runtime
- [✓] Content Based Routing Administration
- [✓] Content Based Routing License
- [ ] User's Guide

Description:

This installs the Content Based Routing program files, executables, messages, help files, and sample files.

Note: Caching and Filtering required for this component.

Space Required: 15 M

Available: 143 M

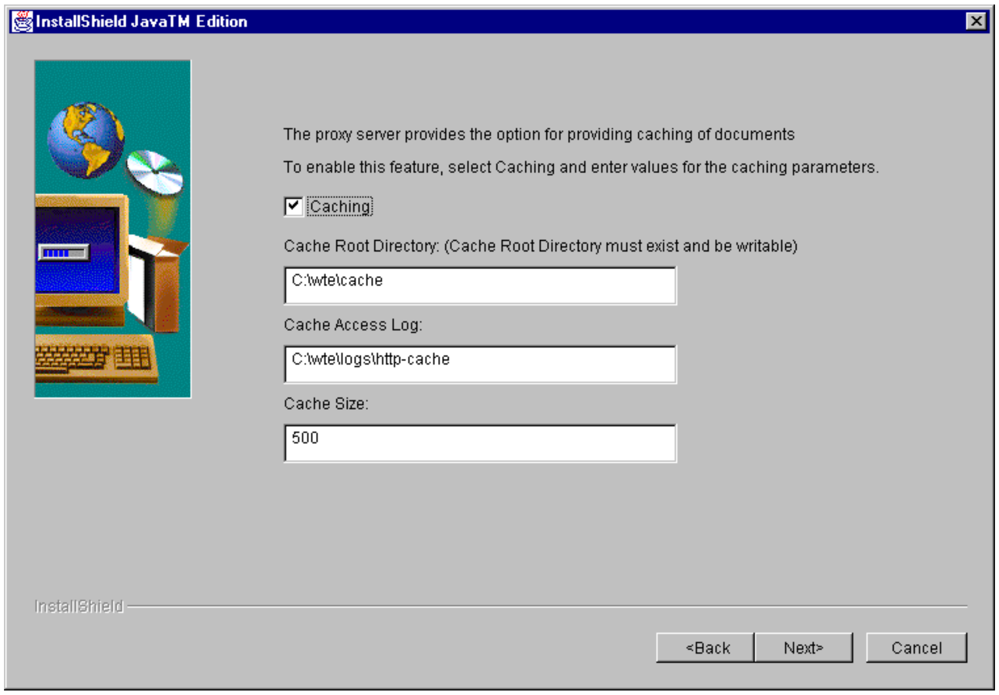
Figure 262. Java InstallShield WebSphere Performance Pack Component Selection

When you select the **Content Based Routing Runtime** ND component, the **Caching and Filtering** WebSphere Performance Pack Version 2 component is automatically selected to be installed, as CBR cannot be installed without it.

The next screen asks if you would like to use the WTE proxy server to cache documents. If you click on the **Caching** checkbox, you will be asked to fill in three blanks:

- Cache Root Directory
- Cache Access Log
- Cache Size

The following figure shows this on the Windows NT platform:



The proxy server provides the option for providing caching of documents

To enable this feature, select Caching and enter values for the caching parameters.

[✓] Caching

Cache Root Directory: (Cache Root Directory must exist and be writable)

C:\wte\cache

Cache Access Log:

C:\wte\logs\http-cache

Cache Size:

500

Figure 263. CBR Install Process Asks for Caching Details

In our case, we wanted to use the caching function of the WTE proxy server, so we checked the **Caching** item in the dialog window, and entered the values shown in Figure 263 on page 330. 14.3.1, "WTE Configuration Overview" on page 332 explains how we selected these values. WTE and CBR can be installed with caching disabled for simplicity and then caching can be enabled and configured at a later time.

After we clicked on the **Next** button, CBR and WTE were installed.

14.1.1 Installation Locations

If CBR is installed from the WebSphere Performance Pack Version 2 InstallShield as demonstrated in 14.1, "Installation of the CBR Function" on page 329, the WTE component is also automatically installed. Alternatively, if CBR is installed from the SecureWay Network Dispatcher InstallShield, then WTE is not automatically installed and must be installed manually.

Note that in each case, the default base install directories will be different on Windows NT as shown in the following table:

Table 24. CBR and WTE Default Installation Base Directories

Platform	**WebSphere Performance Pack components**	
	WTE	**CBR**
AIX	/usr/lpp/internet/server_root	/usr/lpp/nd/cbr
Solaris	/opt/internet/server_root	/opt/nd/cbr
Windows NT WSPP InstallShield	C:\WSPP\WWW\	C:\WSPP\IBM\nd\cbr
Windows NT WTE or CBR InstallShield	C:\WWW	C:\Program Files\IBM\nd\cbr

14.2 Configuration of the CBR Function

In order to perform the configuration you must be the root user on AIX and Solaris or, if the ND server is installed on Windows NT, a member of the Administrators group.

Configuration of CBR cannot take place until WTE has been configured to start the CBR subprocess. We performed CBR configuration in three steps:

1. Because the CBR server is a subprocess of WTE, the first step we performed was to confirm that the WTE server was functional. Optionally, during this step, the WTE server can be configured with values different from the defaults set at installation time (for example, caching can be enabled).
2. Once this is done, four lines must be added to the WTE configuration file to enable CBR functionality. The WTE configuration file is:
   - C:\WINNT\ibmproxy.conf on Windows NT
   - /etc/ibmproxy.conf on AIX and Solaris

   Notice that the WTE server must be stopped and restarted for the changes to take effect.
3. Once the modifications have been made to the WTE configuration file, and the WTE server has been restarted using the modified configuration file, only then can CBR configuration be done. CBR configuration is very similar in style to Dispatcher configuration, in that one or more clusters are usually configured with their associated ports and servers. Similar to rule-based load balancing, rules are then defined, but in the case of CBR the type of the rule is Content. There are three methods of configuring CBR available:
   - `cbrcontrol` commands can be issued from the command line. As well, you can make use of the `cbrcontrol` *persistent session*. The persistent session is a limited shell like utility that can be used to enter `cbrcontrol` commands. Use the `cbrcontrol` command without any parameters to start the persistent session. You will receive the

     ```
     cbrcontrol >>
     ```

     prompt to which you can respond by entering `cbrcontrol` commands without the `cbrcontrol` keyword.

     See *SecureWay Network Dispatcher User's Guide Version 2.1 for Solaris, Windows NT and AIX*, GC31-8496 for `cbrcontrol` command usage information.

- The `cbrcontrol` configuration commands can be placed in a script or command file that can be executed as a batch job. When the file is run the commands are executed in sequence. A CBR configuration file can be loaded into the execution environment with the command:

  ```
  cbrcontrol file load configfilename
  ```

  See Figure 270 on page 344 for an example of a file containing `cbrcontrol` commands.
- The ND graphical user interface (GUI) can be used to perform all of the configuration steps that can be performed with `cbrcontrol` commands.

In addition, all three methods of configuration can be performed by a user on a machine other than the one that is being configured. This new feature is referred to as Remote Authenticated Administration and is explained further in *IBM WebSphere Performance Pack - Load Balancing with IBM SecureWay Network Dispatcher,* SG24-5858.

The three configuration steps above are described in the next section.

14.3 CBR Scenario

In this scenario, we demonstrate how to use CBR by configuring a basic environment. In 14.3.1, “WTE Configuration Overview” on page 332 we give an overview of how to configure WTE. In 14.3.2, “WTE Configuration File CBR Modifications” on page 333 we give details on how to change the WTE configuration file to enable the CBR subprocess. Then in 14.3.3, “CBR Configuration” on page 335 we use the ND GUI to perform CBR configuration of our cluster environment and add the Content type rules. Following this, we show how the rules influence which server is selected to service our request.

14.3.1 WTE Configuration Overview

The WTE component of WebSphere Performance Pack Version 2 contains functionality to work with the CBR component of ND Version 2.1. The steps required to start the WTE server are different on AIX and Solaris than on Windows NT. However, on all these platforms, WTE can be started with all of the default values without making any modifications to the configuration file.

We chose to modify the installed default configuration of WTE to make two changes. The first change was to modify the log used by the WTE proxy server (the proxy server is the basic function of the WTE server that is started if no user configuration is done after WTE installation) and the second change was to enable caching. *IBM WebSphere Performance Pack - Load Balancing with IBM SecureWay Network Dispatcher,* SG24-5858, contains information on WTE including details on how to configure its different functions.

Following a summary list of tasks that we performed to configure WTE on our server:

1. We created a WTE admin user ID and password for use in the following steps.
2. We created a separate file system or directory (/wte on UNIX or C:\wte in the case of Windows NT) to contain the WTE cache root directory and log files. On AIX, we found it necessary to change its ownership with the following command:

```
chown nobody.nobody /wte
```

3. We used the WTE Configuration and Administration Forms, accessible by directing a Web browser to the machine where WTE is installed, to change the location of the proxy log file. The only item we changed was the proxy server log file.
4. In the Configuration and Administration Forms, clicking **Submit** caused the ibmproxy.conf file to be updated with our new log file, but **Submit** or even **Restart Server** (the bar icon at the top of the window) did not result in these changes being used by the server. So we had to stop and restart the WTE server.
5. At this point, we were able to start the Netscape Navigator browser on our client machine. We selected **Preferences** from the Edit menu. Then, we clicked **Advanced** and **Proxies** and in the Proxies panel, we selected the **Manual Proxy Configuration** radio button. In the HTTP field of the Manual Proxy Configuration window, we entered the IP address of the machine where we had installed and configured the WTE server, with the corresponding port field set to `80`.
6. We made a request for a Web page that was available in another subnet that we had connectivity to. At this point we were presented with the Web page and verified from the Server Activity Monitor, accessible from the WTE Configuration and Administration Forms, that the request had been proxied by our server.
7. The process of configuring the caching component is very similar to this and was done by selecting the **Cache Configuration** item in the WTE Configuration and Administration Forms window.

14.3.2 WTE Configuration File CBR Modifications

When WTE is installed, the WTE configuration file, ibmproxy.conf, contains many lines of preconfigured WTE directives. It is necessary to add four lines to this configuration file to enable WTE to start the CBR server subprocess. In the configuration file, we searched for the word `ServerInit` and discovered that `ServerInit` was only contained in the default ibmproxy.conf file as a comment. At the end of the `ServerInit` comment section, we added the four lines.

It is important to note that the first of the four lines is very long and therefore wraps over the right side of the line to the next line. The parameters on the first line are as follows:

- Client keys directory
- Server key directory
- Install path
- Class path
- RMI port
- Log directory
- Save directory

The four lines are shown in the following screens. In each of the screens, it appears that the `CBR_CLIENT_KEYS_DIRECTORY` item is on a new line, but this is due to a formatting problem in this publication. In fact, the `CBR_CLIENT_KEYS_DIRECTORY` item is on the same line as the `ServerInit` directive. On the `ServerInit` line there is one space after the `ServerInit` keyword and one space before

CBR_CLIENT_KEYS_DIRECTORY. Do not include the new line character or any spaces within the parameter string on this line.

Some of the fields have variable-like names included on the line, and others do not. The fields are comma delimited. Following are the four lines that are necessary to configure CBR. The first figure contains the lines we used on our Windows NT system in C:\WINNT\ibmproxy.conf:

```
ServerInit C:\WSPP\IBM\nd\cbr\lib\libndcbr.dll:ndServerInit
CBR_CLIENT_KEYS_DIRECTORY=C:\WSPP\IBM\nd\admin\keys\cbr,CBR_SERVER_KEYS_DIRECTORY=C:\WSPP\IBM\nd\cbr\key,
END_INSTALL_PATH=C:\WSPP\IBM\nd,C:\WSPP\IBM\nd\cbr\lib;C:\WSPP\IBM\nd\cbr\lib\ibmcbr.jar;C:\WSPP\IBM\nd\a
dmin\lib\ChartRuntime.jar,11099,C:\WSPP\IBM\nd\cbr\logs\,C:\WSPP\IBM\nd\cbr\configurations\

PreExit C:\WSPP\IBM\nd\cbr\lib\libndcbr.dll:ndPreExit

PostExit C:\WSPP\IBM\nd\cbr\lib\libndcbr.dll:ndPostExit

ServerTerm C:\WSPP\IBM\nd\cbr\lib\libndcbr.dll:ndServerTerm
```

Note that if you installed CBR with the ND InstallShield rather than the WebSphere Performance Pack InstallShield, the path contained in these lines will have to be changed from `\WSPP\IBM` to `\Progra~1\IBM`. You are required to type the DOS format of the Program Files directory because there can be no spaces within the path elements on these lines.

The changes to the WTE configuration file /etc/ibmproxy.conf on AIX are shown in the following screen:

```
ServerInit /usr/lpp/nd/cbr/lib/libndcbr.so:ndServerInit
CBR_CLIENT_KEYS_DIRECTORY=/usr/lpp/nd/admin/keys/cbr,CBR_SERVER_KEYS_DIRECTORY=/usr/lpp/nd/cbr/key,END_I
NSTALL_PATH=/usr/lpp/nd,/usr/lpp/nd/cbr/lib:/usr/lpp/nd/cbr/lib/ibmcbr.jar:/usr/lpp/nd/admin/lib/ChartRun
time.jar,11099,/usr/lpp/nd/cbr/logs/,/usr/lpp/nd/cbr/configurations/

PreExit /usr/lpp/nd/cbr/lib/libndcbr.so:ndPreExit

Postexit /usr/lpp/nd/cbr/lib/libndcbr.so:ndPostExit

ServerTerm /usr/lpp/nd/cbr/lib/libndcbr.so:ndServerTerm
```

The changes to the WTE configuration file /etc/ibmproxy.conf on Solaris are shown in the following screen:

```
ServerInit /opt/nd/cbr/lib/libndcbr.so:ndServerInit
CBR_CLIENT_KEYS_DIRECTORY=/opt/nd/admin/keys/cbr,CBR_SERVER_KEYS_DIRECTORY=/opt/nd/cbr/key,END_INSTALL_PA
TH=/opt/nd,/opt/nd/cbr/lib:/opt/nd/cbr/lib/ibmcbr.jar:/opt/nd/admin/lib/ChartRuntime.jar,11099,/opt/nd/cb
r/logs/,/opt/nd/cbr/configurations/

PreExit /opt/nd/cbr/lib/libndcbr.so:ndPreExit

Postexit /opt/nd/cbr/lib/libndcbr.so:ndPostExit

ServerTerm /opt/nd/cbr/lib/libndcbr.so:ndServerTerm
```

After adding these lines to the WTE configuration file and saving the file, there is another configuration issue you should take care of:

- On Windows NT, add the following to the Path system environment variable:

  ```
  C:\jdk1.1.7\bin;C:\Program Files\nd\cbr\lib
  ```

  Notice here that we are assuming that Version 1.1.7 of the Java Development Kit (JDK) was installed in the default installation directory C:\jdk1.1.7.
- On AIX, add the following to the LIBPATH system environment variable:

  ```
  /usr/jdk_base/lib:/usr/jdk_base/lib/aix/native_threads:/usr/lpp/nd/cbr/lib
  ```
- On Solaris, add the following to the LD_LIBRARY_PATH system environment variable:

  ```
  /opt/jre1.1.7/lib/sparc/native_threads:/opt/nd/cbr/lib
  ```

Now, you can restart WTE.

Starting the WTE Server on AIX Systems

On AIX, it is advisable to use the `httpd` command to start the WTE server in a CBR environment. The System Resource Control (SRC) command:

```
startsrc -s httpd
```

does not start WTE correctly in the CBR case because WTE cannot find the library files necessary to configure and start CBR. This can be verified by looking at the WTE error log file.

14.3.3 CBR Configuration

Once the ibmproxy.conf file has been modified with the four lines to enable CBR, and the WTE proxy server has been restarted, then CBR configuration can begin.

In this scenario we defined two clusters, A and B, each with two servers. The two servers in cluster A were AIX systems and the two servers in cluster B were Solaris systems. Each cluster already had one rule defined by us, which load balanced requests for a particular page to the two servers. In this section we describe the steps necessary to define two additional CBR rules.

14.3.3.1 Network Environment

The following figure is a graphical representation of our scenario environment:

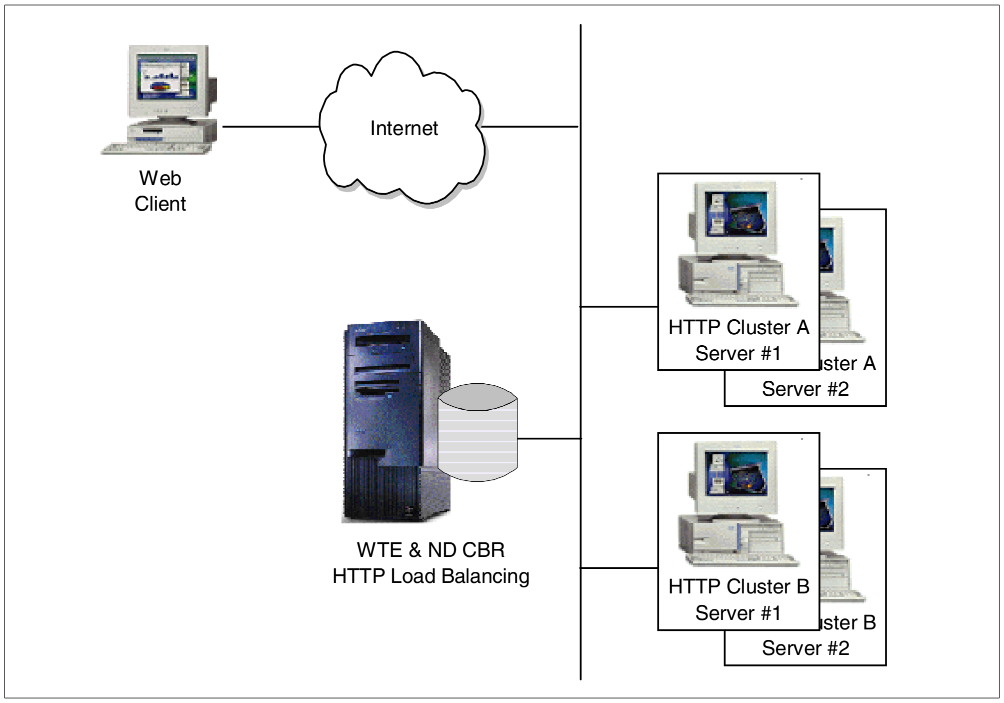


Figure 264. CBR Basic Scenario Environment

A summary of the software and network configuration of the environment where we performed our test is reported in the following table:

Table 25. CBR Basic Scenario - Hardware, Software and Network Configuration

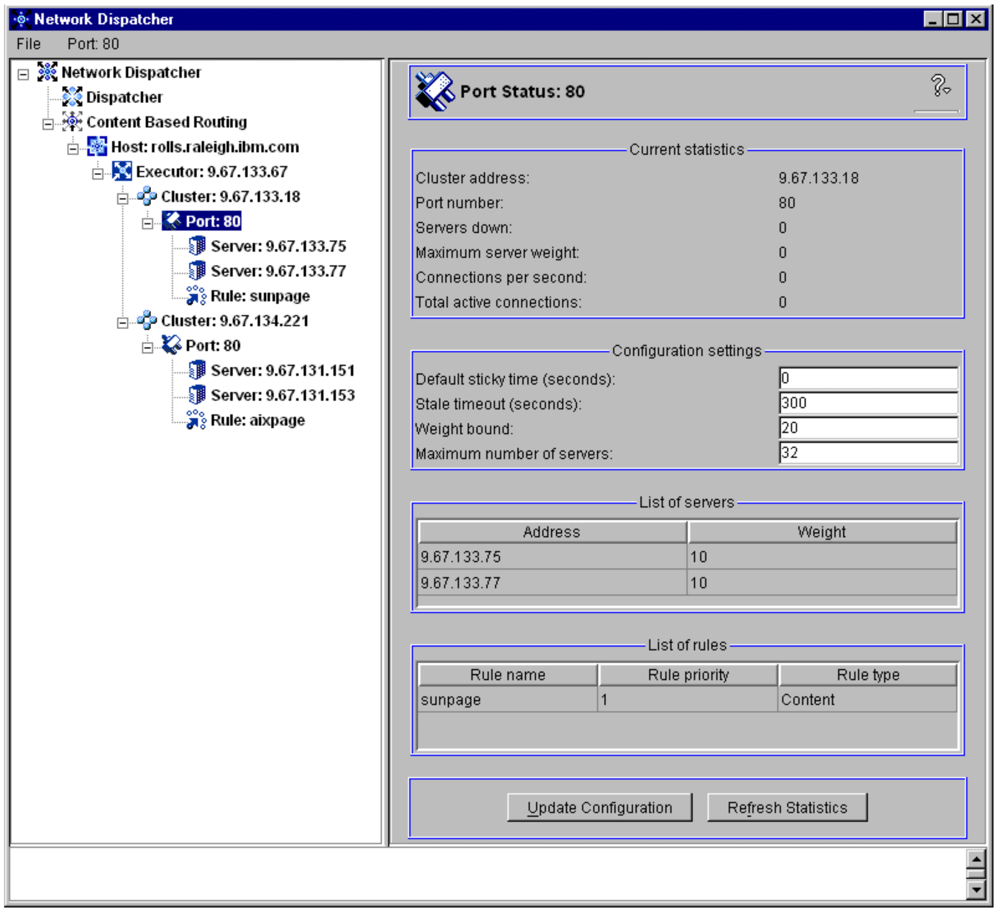
Service		IP Address	Operating System
WTE & CBR server		9.67.133.67	Windows NT Server 4.0
Cluster A IP address 9.67.133.18	Web server 1	9.67.133.75	Solaris 2.6
	Web server 2	9.67.133.77	Solaris 2.6
Cluster B IP address 9.67.134.221	Web server 1	9.67.131.151	AIX 4.3.1
	Web server 2	9.67.131.153	AIX 4.3.1

14.3.3.2 Cluster, Port, Server and Rule Configuration

Once the CBR-enabled proxy server was running, we used the ND GUI to configure our CBR cluster. We used the `ndadmin` command to start the GUI and after right-clicking the **Content Based Routing** component, we selected **Connect to Host** from the pop-up menu. This is an effect of the new Remote Authenticated Administration feature which enables the ND component configuration to be done on a remote client machine. In this case, we were performing this configuration on the Dispatcher machine itself, however the connection still had to take place.

If, at the point where you try to connect to the host to do configuration either through the ND GUI or with the `cbrcontrol` commands, you receive a key related error message, then it is possible that there is an error in the four lines that were added to your ibmproxy.conf file.

We then started the Executor, added our clusters, and to each cluster a port and two servers. As the following figure demonstrates, for our testing purposes we had already added a rule named sunpage for cluster A and a rule named aixpage for cluster B. Following this configuration, the ND GUI appeared as follows:



- Network Dispatcher
  - Dispatcher
  - Content Based Routing
    - Host: rolls.raleigh.ibm.com
      - Executor: 9.67.133.67
        - Cluster: 9.67.133.18
          - Port: 80
            - Server: 9.67.133.75
            - Server: 9.67.133.77
            - Rule: sunpage
        - Cluster: 9.67.134.221
          - Port: 80
            - Server: 9.67.131.151
            - Server: 9.67.131.153
            - Rule: aixpage

Port Status: 80

Current statistics

Cluster address:	9.67.133.18
Port number:	80
Servers down:	0
Maximum server weight:	0
Connections per second:	0
Total active connections:	0

Configuration settings

Default sticky time (seconds):	0
Stale timeout (seconds):	300
Weight bound:	20
Maximum number of servers:	32

List of servers

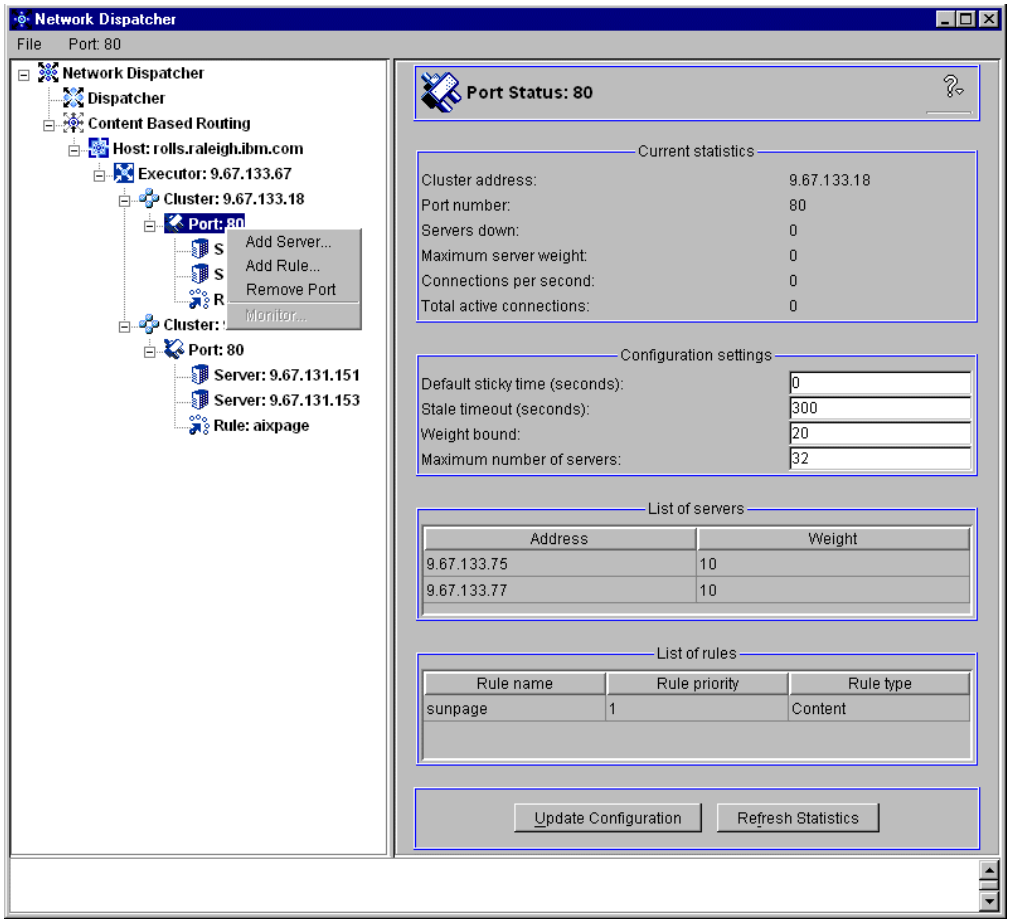
Address	Weight
9.67.133.75	10
9.67.133.77	10

List of rules

Rule name	Rule priority	Rule type
sunpage	1	Content

Figure 265. ND GUI Showing CBR Cluster, Port and Server Information

We will show you now details of how to add two more rules to the 9.67.133.18 cluster. To configure the content rules that CBR uses to determine which servers to load balance the request amongst, we right-clicked the **Port:80** item in the navigation portion of the GUI and selected **Add Rule...** from the pop-up menu, as shown in the following window:



Current statistics

Cluster address:	9.67.133.18
Port number:	80
Servers down:	0
Maximum server weight:	0
Connections per second:	0
Total active connections:	0

Configuration settings

Default sticky time (seconds):	0
Stale timeout (seconds):	300
Weight bound:	20
Maximum number of servers:	32

List of servers

Address	Weight
9.67.133.75	10
9.67.133.77	10

List of rules

Rule name	Rule priority	Rule type
sunpage	1	Content

Figure 266. ND GUI Showing Add Rule Menu Item

Of course you should plan the logic that you want CBR to follow before you start adding rules to your configuration.

Notice that the network interface card on the CBR machine must be aliased to all the cluster addresses defined in the configuration above. This is the same thing you would do in a Dispatcher scenario. However, unlike a Dispatcher scenario, the loopback adapters on the TCP server machines must not be aliased to any cluster address.

Cluster Aliasing Note

The network interface card of the CBR machine must be aliased to all the cluster addresses used in the configuration. However, the loopback interface on the TCP server machines does not need to be aliased; for those who are familiar with this kind of aliasing operations in a Dispatcher scenario, it may seem unusual that similar aliasing is not done with CBR.

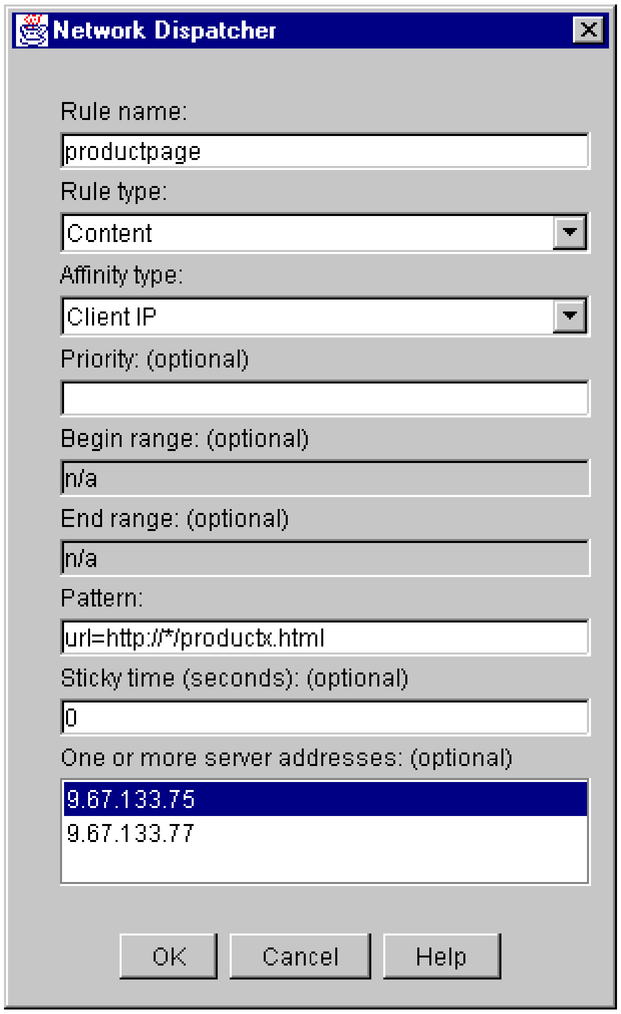
The reason for this is that when the packet is sent from the client machine, its destination IP address is the IP address of one of the clusters. When the packet arrives at the CBR machine, the packet is accepted because the network interface card on the CBR machine has been aliased to that cluster address. WTE receives the request and offers CBR the opportunity to examine its clusters and rules for a match. If a match is found, URL name translation is done.

If and when the proxy server sends the request to a clustered TCP server (the selection of which was done by CBR), WTE proxies a new request to the TCP server with the modified URL, and a destination IP address that is the IP address of the TCP server machine. When the TCP server replies, its response goes back to the CBR server, and is cached by WTE if caching is enabled and if WTE caching algorithms require caching of that particular page. For this reason, there is no requirement to alias the cluster address to the loopback interface on the TCP server machine.

On the contrary, Dispatcher keeps the cluster address as the destination address of the packet, and identifies the TCP servers through their Media Access Control (MAC) addresses. For this to work, the TCP servers need to have the cluster address aliased on their loopback interface. An advantage of this is that the server's response can flow directly to the client, without any need for it to be proxied by the Dispatcher. This would not be a good solution with CBR though, because it would not allow caching.

Another positive effect of this is that a CBR cluster does not have the same restriction as a Dispatcher cluster with regard to the location of the TCP servers relative to the Dispatcher or CBR server. With CBR, because WTE proxies the request to the servers, there is no requirement for the servers to be located on the same LAN as the CBR machine.

Our next step was to create a content rule to direct all requests to cluster 9.67.133.18 that contained the string `productx.html` in the URL to server 9.67.133.75. To accomplish this, in the **Add Rule** dialog window, we filled in the fields as follows:


Network Dispatcher

Rule name:
productpage

Rule type:
Content

Affinity type:
Client IP

Priority: (optional)

Begin range: (optional)
n/a

End range: (optional)
n/a

Pattern:
url=http://*/productx.html

Sticky time (seconds): (optional)
0

One or more server addresses: (optional)
9.67.133.75
9.67.133.77

Figure 267. Add Rule Dialog Window for the productpage Rule

- We assigned the name `productpage` to this rule.
- We selected the type of the rule to be **Content**. This is the new rule type added for use by the CBR component.
- The Affinity type field contains one of two possible values: **Client IP** or **Cookie**. Client IP affinity as used in the Dispatcher component can also be used with CBR. The cookie affinity feature applies only to the CBR component and provides a new way to make clients *sticky* to a particular server. This function is enabled by setting the sticky time of a rule to a positive number, and setting the affinity to **Cookie**.

  We left the default value of **Client IP** in the Affinity type field. In the next rule we demonstrate setting cookie affinity and sticky time.
- We also did not put a value in the priority field for the rule. Priorities establish the order in which rules will be reviewed. This parameter accepts integer values. If you do not specify the priority of the first rule you add, CBR will set it by default to `1`. When a subsequent rule is added, by default its priority is calculated to be 10 plus the current lowest priority of any existing rule. For example, assume you have an existing rule whose priority is 30. You add a new rule and set its priority at `25` (which is a higher priority than 30). Then you

add a third rule without setting a priority. The priority of the third rule is calculated to be as 30 + 10 = 40, and so on.

- The Begin range parameter and the End range parameters are not used on Content type rules.
- The Pattern field is used to define the pattern of characters that CBR will match against each client request.

  The pattern must not contain any spaces and can make use of the special characters listed in the following table:

Table 26. Special Characters Allowed in the Pattern Field

Character	Function
`*`	Matches 0 to *x* of any character
`(`	Used for logic grouping
`)`	Used for logic grouping
`&`	Logical AND
`|`	Logical OR
`!`	Logical NOT

The reserved keywords shown in the following table must always be followed by the equal (=) sign:

Table 27. Keywords Followed by the Equal (=) Sign

Keyword	Value
`client`	Client IP address
`url`	URL in request
`protocol`	Protocol section of URL
`path`	Path section of URL
`refer`	Referred URL (quality of service)
`user`	User ID section of URL

In our case, we made use of the `url` reserved word and the wildcard (*) character. We specified the pattern as:

```
url=http://*/productx.html
```

This rule specifies that any URL that is a request for the page productx.html will be a match.

Other examples of valid rule patterns are:

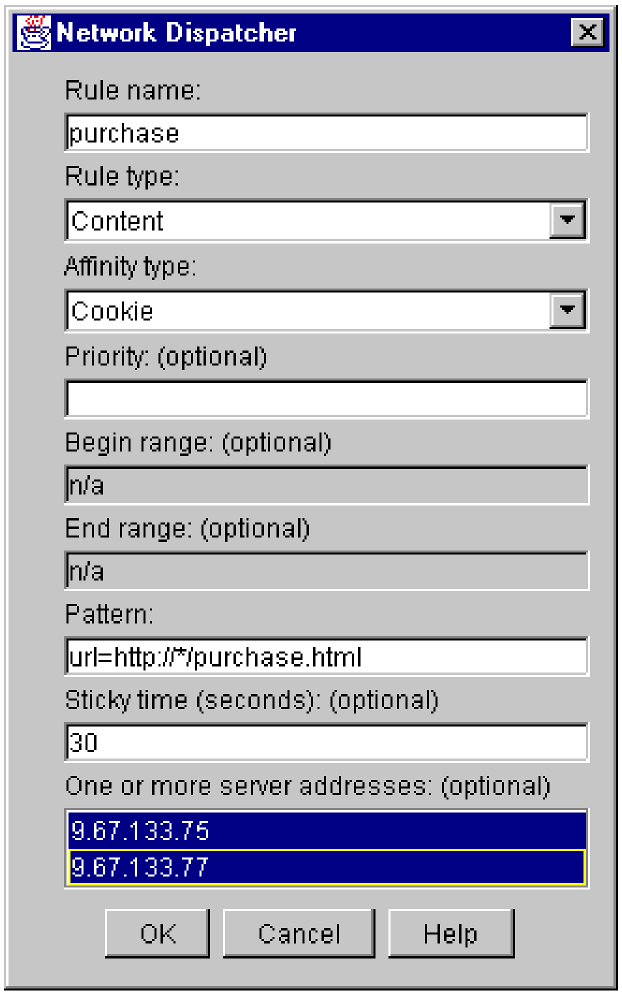
- `url=http://*/*.gif`
- `client=9.32.*`
- `(path=index/*.gif&protocol=http)|(client=9.1.2.3)`
- `!(path=*.jpeg)`

- The Sticky time field is used along with the Affinity type field. We demonstrate the use of this in our next rule.
- The last item on the dialog window shown in Figure 267 on page 340 is a scrolled list of server addresses to optionally choose from. In this case we chose the server whose IP address was 9.67.133.75, meaning that we wanted

this server to serve the client requests containing the string productx.html in the URL.

We then clicked the **OK** button in the dialog window shown in Figure 267 on page 340.

The next rule was added to send requests to both servers in the 9.67.133.18 cluster for client requests for the page purchase.html. In this case, however, we specified an Affinity type of **Cookie** and a Sticky time of `30` seconds. This means that when a client request to the cluster is made and the URL matches the rule (that is, contains the string `purchase.html`), CBR would load balance the request to the best server of the two and then all subsequent HTTP requests from that client to this cluster address would also go to that server for a period of 30 seconds. We called this rule `purchase`. The dialog window in this case appeared as follows:



Rule name: purchase

Rule type: Content

Affinity type: Cookie

Priority: (optional)

Begin range: (optional) n/a

End range: (optional) n/a

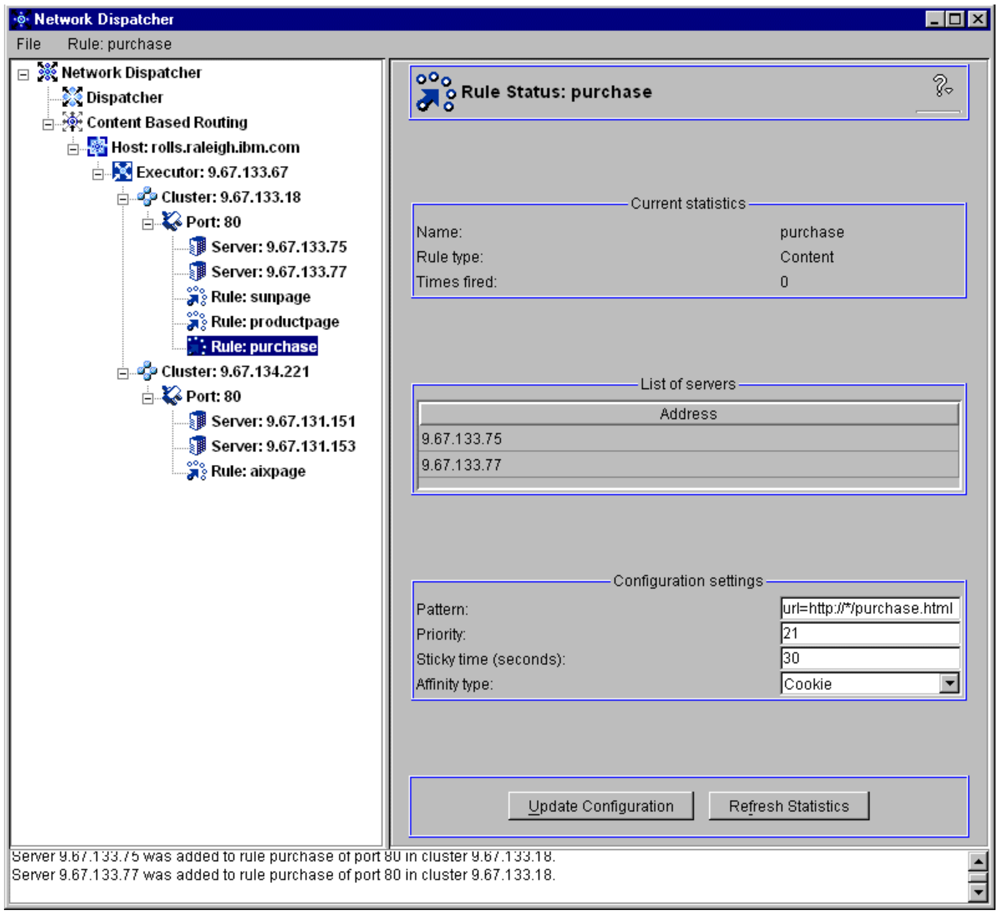
Pattern: url=http://*/purchase.html

Sticky time (seconds): (optional) 30

One or more server addresses: (optional)
9.67.133.75
9.67.133.77

Figure 268. Add Rule Dialog Window for the productpage Rule

We then clicked the **OK** button in the dialog window above. The GUI reported the configuration was updated with the added rules, as shown in the following figure:



Current statistics

Name:	purchase
Rule type:	Content
Times fired:	0

List of servers

Address
9.67.133.75
9.67.133.77

Configuration settings

Pattern:	url=http://*/purchase.html
Priority:	21
Sticky time (seconds):	30
Affinity type:	Cookie

```
Server 9.67.133.75 was added to rule purchase of port 80 in cluster 9.67.133.18.
Server 9.67.133.77 was added to rule purchase of port 80 in cluster 9.67.133.18.
```

Figure 269. ND GUI Showing the Newly Added productpage Rule

14.3.4 CBR Manager and Advisors

As with the Dispatcher, starting the Manager and using the Advisors is optional. The Manager can be activated with the command:

```
cbrcontrol manager start
```

The typical advisor that is used with CBR is the HTTP advisor, which can be launched by entering the following command:

```
cbrcontrol advisor start http port
```

where *port* is the port configured for the cluster.

If you have configured any Advisors, you must change the Manager proportions to allow the Advisor information to be included in the load balancing decisions. To do this, you should use the command:

```
cbrcontrol manager proportions
```

For simplicity, in the scenarios described in this chapter, we did not make use of the CBR Manager or Advisors functions.

14.3.5 Saving the Configuration

Once we finished configuring CBR, we saved the configuration to a file. To do this from the ND GUI, we right-clicked **Host** and selected **Save Configuration File As...**.

In the resulting pop-up window, we were prompted to enter the name of the configuration file where we wanted to save this information. We entered the file name landon.cfg in the Save Configuration pop-up window. By default this file is placed in the directory configurations under *installbase*, where *installbase* is for the CBR component and varies by operating system. See Table 24 on page 331 for a list of the installbase locations.

The configuration file is saved in ASCII format and contains the list of commands that would be necessary to reconfigure your environment. Following is the content of the landon.cfg file:

```
cbrcontrol cluster add 9.67.133.18

cbrcontrol port add 9.67.133.18:80

cbrcontrol server add 9.67.133.18:80:9.67.133.77

cbrcontrol server add 9.67.133.18:80:9.67.133.75
cbrcontrol rule add 9.67.133.18:80:sunpage type content pattern url=http://*/sunpage.html priority 1 beginrange 0
endrange 0
cbrcontrol rule useserver 9.67.133.18:80:sunpage 9.67.133.75
cbrcontrol rule useserver 9.67.133.18:80:sunpage 9.67.133.77
cbrcontrol rule set 9.67.133.18:80:sunpage stickytime 30

cbrcontrol rule add 9.67.133.18:80:productpage type content pattern url=http://*/productx.html priority 11
beginrange 0 endrange 0
cbrcontrol rule useserver 9.67.133.18:80:productpage 9.67.133.75

cbrcontrol rule add 9.67.133.18:80:purchase type content pattern url=http://*/purchase.html priority 21 beginrange 0
endrange 0
cbrcontrol rule useserver 9.67.133.18:80:purchase 9.67.133.75
cbrcontrol rule useserver 9.67.133.18:80:purchase 9.67.133.77
cbrcontrol rule set 9.67.133.18:80:purchase stickytime 30
cbrcontrol rule set 9.67.133.18:80:purchase affinity cookie

cbrcontrol cluster add 9.67.134.221

cbrcontrol port add 9.67.134.221:80

cbrcontrol server add 9.67.134.221:80:9.67.131.153

cbrcontrol server add 9.67.134.221:80:9.67.131.151

cbrcontrol rule add 9.67.134.221:80:aixpage type content pattern url=http://*/aixpage.html priority 1 beginrange 0
endrange 0
cbrcontrol rule useserver 9.67.134.221:80:aixpage 9.67.131.151
cbrcontrol rule useserver 9.67.134.221:80:aixpage 9.67.131.153
```

Figure 270. Configuration File landon.cfg

Examination of the saved configuration file is interesting because it contains each of the `cbrcontrol` commands that could also be entered from the command line to configure the same environment. If you chose to configure CBR with commands, you can save your configured environment with the command:

`cbrcontrol file save` *configurationfilename*

You can then subsequently reload the configuration file with the command:

`cbrcontrol file load` *configurationfilename*

14.3.6 Scenario Results

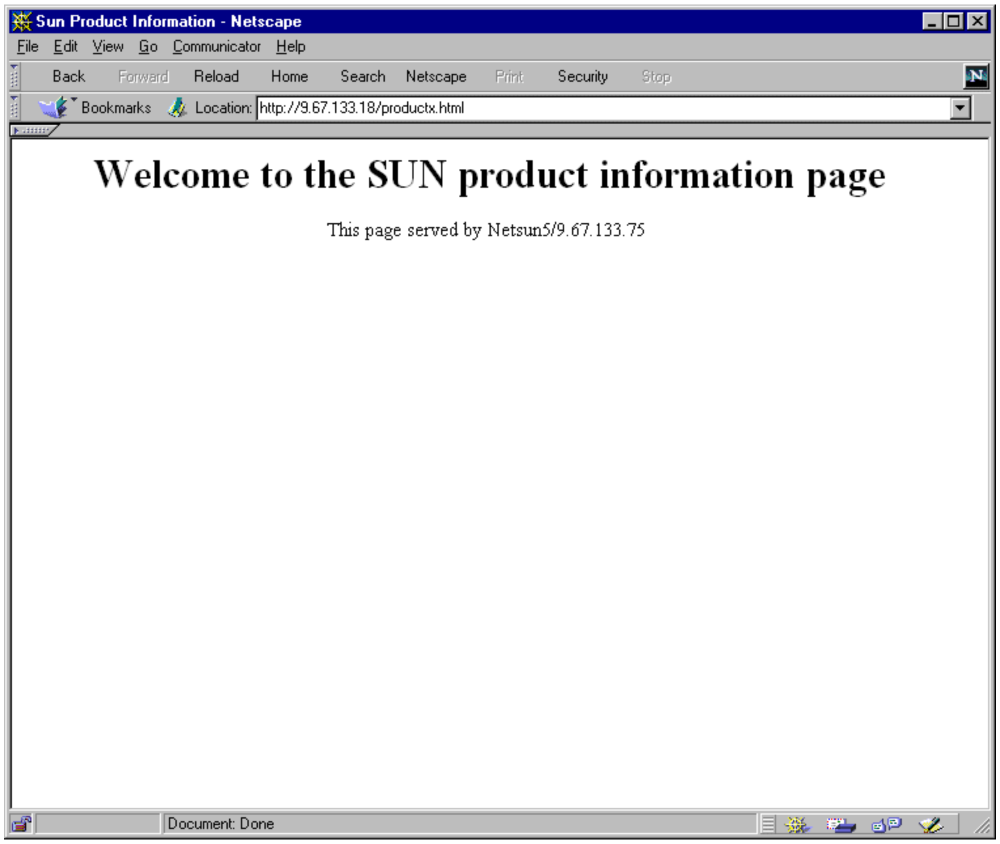
We placed simple HTML files in the document root directories on each of our Web servers. On the Web server with IP address 9.67.133.75 we placed a file named productx.html and another called purchase.html. On the Web server with IP address 9.67.133.77 we placed a different file also called productx.html. Recall that we had defined a rule specifying that client requests containing the string `productx.html` in the URL be load balanced between both of these servers. Each of the files uniquely identified the server that it was located on and the name of the page.

Client Configuration

In a CBR scenario, the client machine does not need any special configuration. In particular, client browsers must not be set to redirect all the requests to the CBR machine. For example, in a real-life situation, it would not be appropriate to require end users to reconfigure their Web browsers before accessing a Web site that uses CBR. The use of CBR in a Web site is completely transparent to end users.

14.3.6.1 Client IP Affinity Demonstration

For our first request, with our browser we requested the productx.html page from our cluster address and received the following page:


Sun Product Information - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://9.67.133.18/productx.html
Document: Done

Welcome to the SUN product information page

This page served by Netsun5/9.67.133.75

Figure 271. Sun Product Request

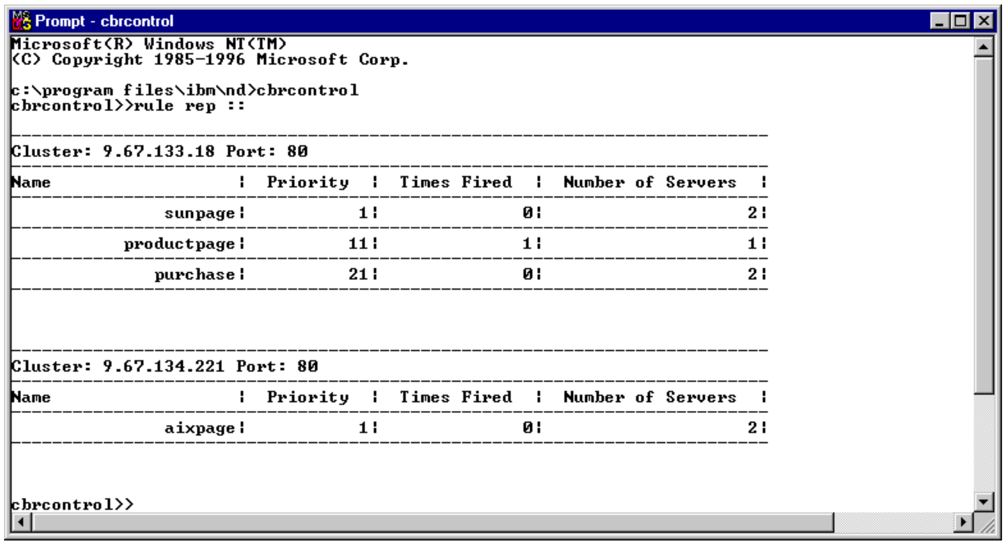
To verify that our productpage rule was matched by this request, we used the `cbrcontrol` persistent session command

```
rule rep ::
```

The command line version of this is:

```
cbrcontrol rule report cluster:port:rule
```

but as with other `cbrcontrol` commands, short forms of the keywords can be used. In order to specify that you would like the report to include all clusters, ports and rules, use only the two colon (`::`) delimiters on the command line. The command and its output are shown in the following figure:



```
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

c:\program files\ibm\nd>cbrcontrol
cbrcontrol>>rule rep ::

------------------------------------------------------------------------
Cluster: 9.67.133.18 Port: 80
------------------------------------------------------------------------
Name                 |  Priority  |  Times Fired  |  Number of Servers  |
------------------------------------------------------------------------
              sunpage|           1|              0|                    2|
------------------------------------------------------------------------
          productpage|          11|              1|                    1|
------------------------------------------------------------------------
             purchase|          21|              0|                    2|
------------------------------------------------------------------------


------------------------------------------------------------------------
Cluster: 9.67.134.221 Port: 80
------------------------------------------------------------------------
Name                 |  Priority  |  Times Fired  |  Number of Servers  |
------------------------------------------------------------------------
              aixpage|           1|              0|                    2|
------------------------------------------------------------------------


cbrcontrol>>
```

Figure 272. Rule Report Showing productpage Rule Fired One Time

Since the sticky time is set to 0 seconds, Web requests from the same client should not stick to the same Web server, and subsequent requests should normally load balance between the two Web servers defined in cluster A. If the sticky time were set to a positive number of seconds, CBR would choose one of the two servers the first time a client request arrives that matches the productpage rule, and then would redirect all the client requests coming from the same IP address to the same server until the sticky time expires.

In this case, however, we forced the CBR server to redirect all the requests to the Web server having IP address 9.67.133.75, because this was the only server available in the productpage rule.

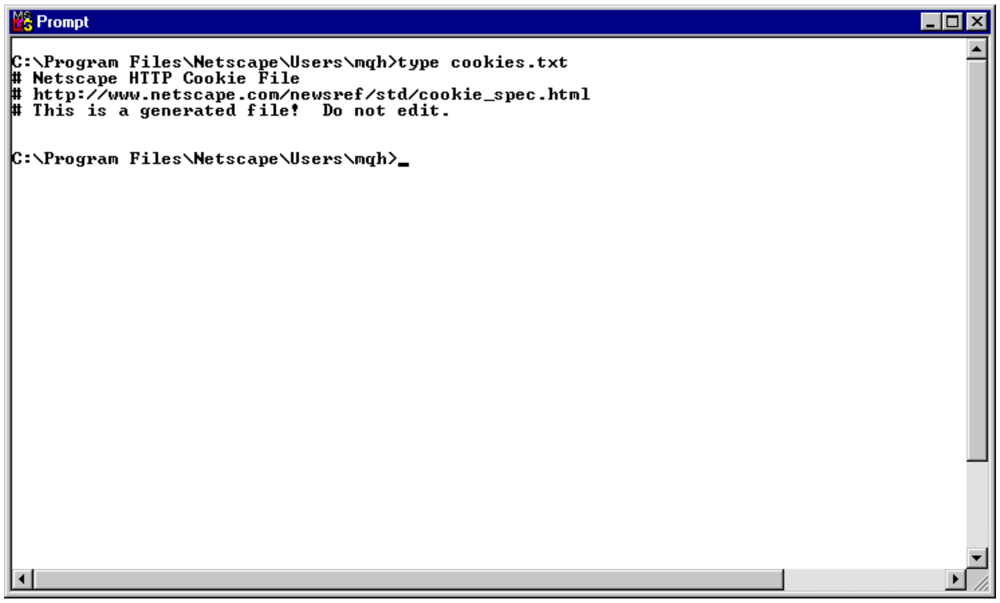
14.3.6.2 Cookie Affinity Demonstration

The next test we made was to demonstrate the use of Cookie affinity.

Web cookies are simple pieces of information passed between the client Web browser and the Web server during an HTTP transaction. Cookies do not contain any information about the client that the server does not already know and they cannot do anything on the client machine that the client itself cannot already do, provided the browser is within the specifications. Cookies were introduced as an answer to a fundamental problem of the Web's underlying HTTP 1.0 protocol: the lack of a state, or a persistent connection.

The first time a client accesses a Web site that serves cookies, the chosen server sends a cookie to the client browser, identifying the server and some information about which URLs the cookie is good for. The next time the client visits one of those URLs, the browser includes the cookie in its request. As long as the client's future requests contain the cookie, and each request arrives within the sticky time interval, the client will maintain affinity with the initial server.

Prior to making a request for the purchase page that will trigger the 30 second affinity rule, we examined the contents of the local cookie file (cookies.txt) in the browser directory on our client machine and it was empty:



```
C:\Program Files\Netscape\Users\mqh>type cookies.txt
# Netscape HTTP Cookie File
# http://www.netscape.com/newsref/std/cookie_spec.html
# This is a generated file!  Do not edit.


C:\Program Files\Netscape\Users\mqh>_
```

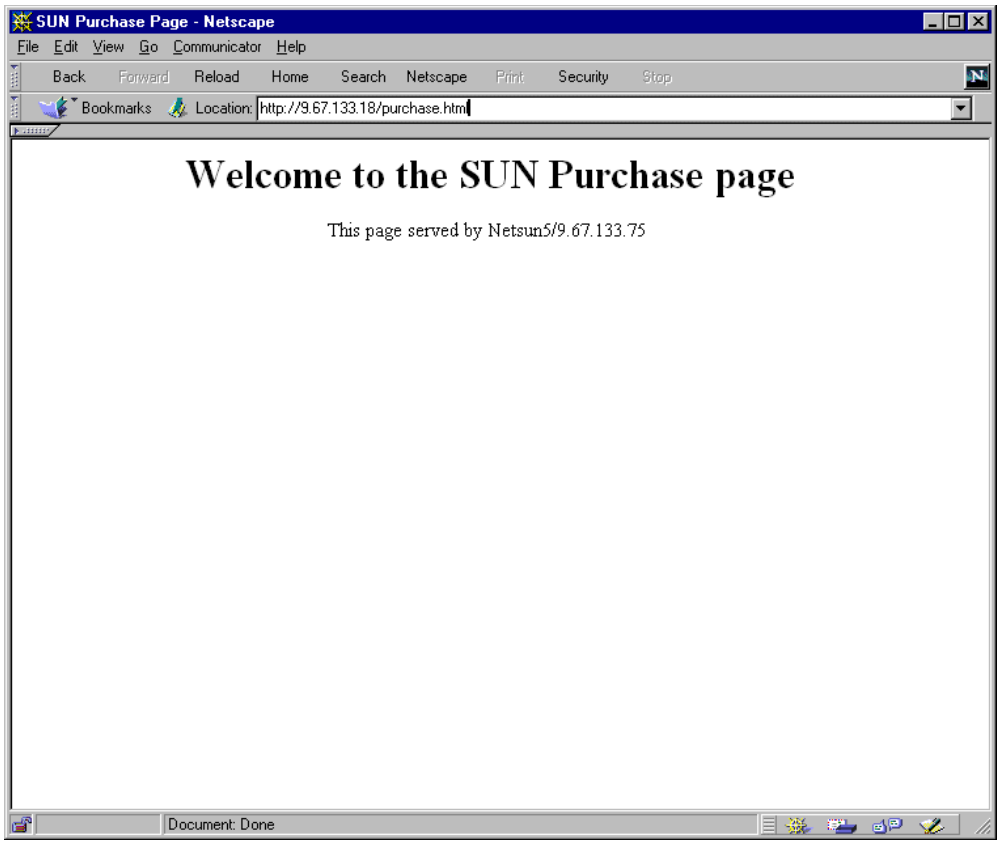
Figure 273. Empty Cookie File Before Request Was Made

Recall that we had enabled a 30 second cookie affinity on our purchase rule by setting the sticky time to `30`, and setting the affinity to **Cookie** when we created the rule. This also could have been set with the command:

```
cbrcontrol rule set
```

Once a server is selected by CBR to respond to our request, subsequent requests were also responded to by the same server.

In the 30 second period we made three requests for `http://9.67.133.18/purchase.html`. Each request was responded to by the same server:

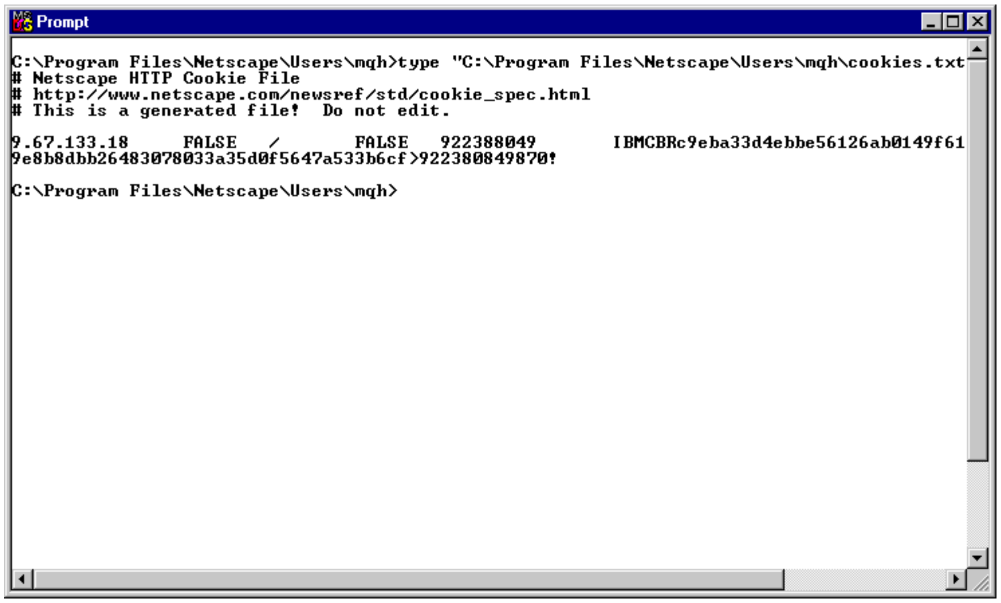

SUN Purchase Page - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://9.67.133.18/purchase.html
Document: Done

Welcome to the SUN Purchase page

This page served by Netsun5/9.67.133.75

Figure 274. Sun Purchase Using a Cookie

We verified that the cookie was set on our browser machine by examining the cookie.txt file:



```
C:\Program Files\Netscape\Users\mqh>type "C:\Program Files\Netscape\Users\mqh\cookies.txt
# Netscape HTTP Cookie File
# http://www.netscape.com/newsref/std/cookie_spec.html
# This is a generated file!  Do not edit.

9.67.133.18     FALSE   /       FALSE   922388049       IBMCBRc9eba33d4ebbe56126ab0149f61
9e8b8dbb26483078033a35d0f5647a533b6cf>922380849870!

C:\Program Files\Netscape\Users\mqh>
```

Figure 275. Cookie File Containing one Cookie after Request Was Made

Even though the contents of the cookie are not understandable by us, they are meaningful to the Web server and browser.

After the 30 second sticky time expired, we saw that the CBR server was allowed to choose the other Web server to satisfy the request from the client.

14.4 WTE CacheByIncomingUrl Directive

A new WTE directive has been added for use with CBR in the ibmproxy.conf configuration file. This directive, `CacheByIncomingUrl`, specifies whether to use the incoming URL or the outgoing URL as the basis for generating cache file names. The values of this directive can be `on` or `off`:

- If `CacheByIncomingUrl` is set to `on`, the incoming URL will be used to generate the cache file name. In other words, when this directive is set to `on`, WTE keeps the original URL and uses it to cache the page that it gets back from the ND-changed URL.
- On the other hand, if `off` is specified, CBR rule matching and load balancing will be done on the incoming URL and the resulting URL will be used to generate the cache name. In this case, WTE uses the ND-changed URL to cache the page.

The `CacheByIncomingUrl` directive’s default value is `off` in the ibmproxy.conf file.

Notice that `CacheByIncomingUrl` is a *hidden directive* of WTE. In other words, it is possible to alter its value only by manually editing the configuration file ibmproxy.conf; there is no way to change the value of this directive through the Configuration and Administration Forms.

Notice also that a Web page is cached only if WTE decides it should be. WTE does this by looking at the `Expires` tag. If there is not one, it estimates an expiration time by the `Last-Modified` parameter. You should keep this in mind if you are testing CBR caching with sample pages you have just created and which do not contain `Expires` header information. In this case, recently created pages would not be cached, since they would be interpreted by WTE as frequently changed pages, therefore resulting in an apparent failure of the CBR caching function.

Chapter 15. Remote Cache Access

It is not uncommon for a proxy server to receive more traffic than it can handle. One solution is to distribute the traffic between multiple, load balanced proxy servers. However, in this situation, the content of one cache is likely to overlap with the contents of the other caches. Besides unnecessary redundancy, this situation also requires additional bandwidth, because each copy in each server is fetched fresh from the origin. This problem can be minimized by chaining a hierarchy of proxies together, but this still results in additional traffic passing through a given server, and each additional link in the chain adds latency and increases the possibility of a failure. Though useful, proxy chaining does not solve the problem as it does not allow sharing the cache across multiple systems.

IBM Web Traffic Express (WTE) Version 2, the Caching and Filtering component of IBM WebSphere Performance Pack Version 2, has a feature called Remote Cache Access (RCA), which permits a number of WTE servers to share their cache. Eliminating duplicate objects results in bandwidth savings because:

- Objects are not fetched multiple times.
- A larger, combined logical cache yields a higher hit rate.

RCA, which was not available in the previous release of IBM WebSphere Performance Pack, allows multiple proxy servers to cooperate to form a cache array. RCA uses the Cache Array Routing Protocol (CARP) to determine which peer in the array should process the incoming request. If the requested file is not in the proxy's cache, it will query the other peers in the array to determine which proxy might have the object cached. Therefore, using RCA, multiple proxy servers can distribute the cache contents across their combined, logical cache to improve hit rates and reduce redundancy of cached content.

To run a RCA, WTE demands a shared file system. This can be IBM AFS Enterprise File System (AFS), DFS, NFS, the Windows NT file system, or any other, but the cache directories must have read and write permissions for all the cache array members.

15.1 How RCA Works

In the WTE cache array, all the machines know the existence of the others, and all run a special hash function that maps every URL to an address in the cache. An address in the cache is defined as the pair:

```
(array_member, cache_file_name)
```

This way, the cache space is the combination of all members (WTE servers) and all cache directories. Each URL has a unique location in the cache, and the hash function allows each member to compute the member that owns any given URL.

If a request is made to one proxy, this proxy will first check its local cache. If the object is not found, then it will compute the owner of the URL and ask the owner if it has the object already cached. If so, the first proxy will directly access the cache space of the other proxy, retrieve the requested object and serve it to the requesting client. If not, the requested object will be retrieved directly from the remote content server.

The following diagram shows a graphical representation of an RCA flow:

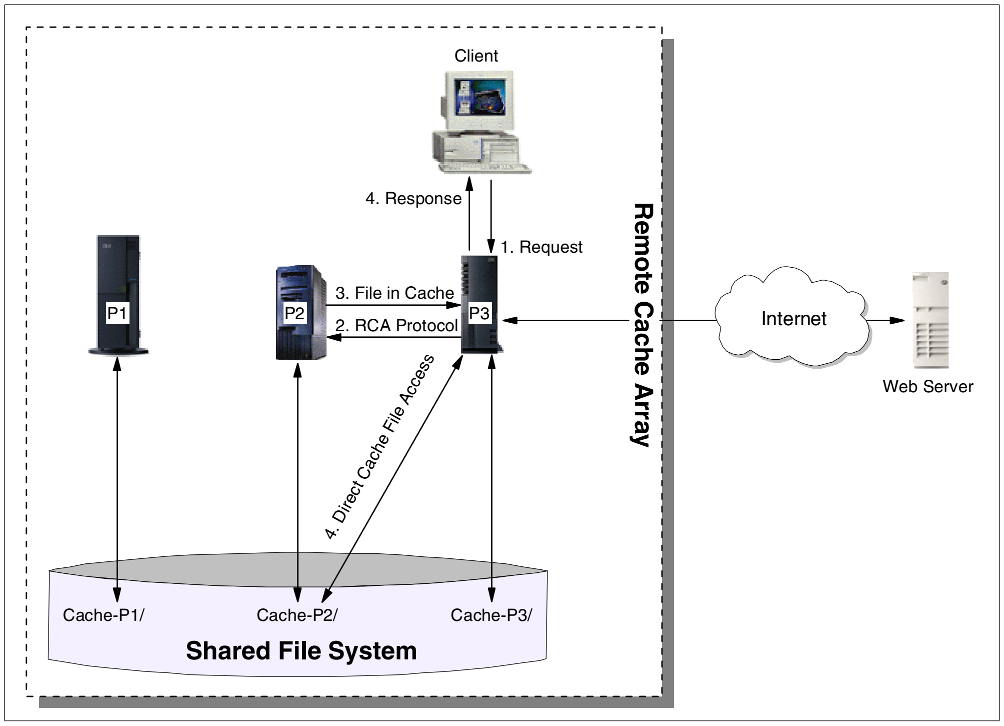


Figure 276. RCA Information Flow

The information flow on RCA is as follows:

1. The client makes a URL request to one of the proxies. In our example, this proxy is P3. Of course, the configuration of the figure above would make much more sense if a load balancing machine were placed between the client and the WTE proxy servers, to load balance the traffic between the proxy servers. Although this is not a mandatory requirement, it is a very good idea to distribute the workload between multiple WTE proxy servers using RCA by using IBM SecureWay Network Dispatcher (ND), the Load Balancing component of IBM WebSphere Performance Pack. In particular, ND offers the WTE Advisor to perform peak load management, as we will see in Chapter 16, “Peak Load Management” on page 357.
2. Using the hash function, P3 knows that this URL may be cached only in P2.
3. P3 makes an RCA request to P2 asking if it has already cached the requested object.
4. At this point, two situations can happen:
   - P2 responds that it has the URL in its cache:
     1. P3 directly accesses P2’s cache to pick up the file corresponding to the requested URL. P2 will not pass all the content of the cached object to P3. In fact, since P3 knows that P2 owns the URL, and since P2’s cache directory is accessible to P3 via the shared file system, P3 accesses P2’s cache directory directly.
     2. P3 returns the cached object to the requesting client
   - P2 responds that it does not have the requested object in its cache:

1. P3 will grab the URL in its original source, the Web or FTP server.
2. P3 will create the cache entry of the URL in P2's cache directory and use RCA to tell P2 about the modification.

Using this procedure, RCA is guaranteed to be an extremely fast protocol, and provides a way to build a scalable array.

15.2 Planning for RCA

When planning to use RCA, you should remember the following:

- The participating proxy servers should be as close as possible, with high-bandwidth connections. It is recommended that members are on the same LAN segment, or communicate over an SP switch, if at all possible. In addition, consider dedicating a network segment just to the inter-node communication (the RCA messages and the shared file system).
- The same file system must be used by all members of an array. You cannot use, for example, AFS on some nodes, NFS on others and the Windows NT file system on others. The recommended file system is AFS.
- Membership in the RCA array should be long-term. The configuration should be as stable as possible.
- Proxy servers should have similar capabilities (for example, CPU, memory, size, cache size).
- Network outages between members of the array must be infrequent.
- There should be less than 100 members in any array.
- All members of the array must run WTE Version 2. You cannot share a cache between WTE Version 2 and Distributed Web Traffic Express (DWTE) Version 1.1[1].
- The members of the array should be load-balanced using ND.

Conversely, RCA is not appropriate when:

- The participating proxy servers are not in close proximity.
- Frequent network outages are expected.
- Servers differ widely in capacity. For example, a PC server with a 500 MB cache and an RS/6000 server with a 20 GB cache should not be in the same array.
- Membership in the RCA array is short term.

15.3 RCA Scenario

Installing and configuring an RCA environment is a typical case where you can install, configure and integrate all of the WebSphere Performance Pack components. In fact, RCA nodes need to share the same file system, and by using AFS you can take advantage of the powerful file sharing features of WebSphere Performance Pack. Moreover, as we have already said, a scenario involving multiple proxy servers sharing the same cache makes more sense if the

[1] DWTE is another IBM caching and filtering proxy server. DWTE also implements RCA. Originally, IBM implemented RCA only in DWTE. Then, the RCA feature was ported also to WTE V2.0.

workload is distributed between the servers. For this purpose, you can use the ND component of IBM WebSphere Performance Pack.

In this section, we show you how we implemented an RCA scenario similar to the one represented below:

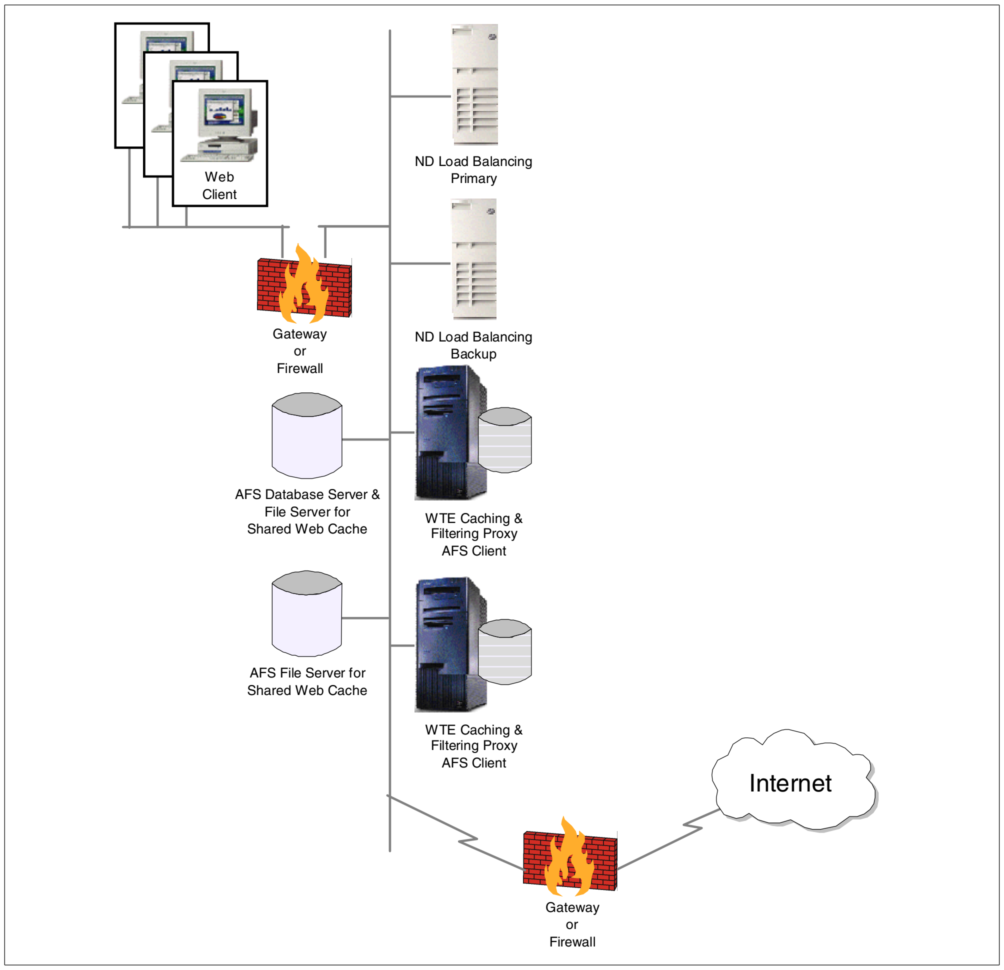

Web Client
ND Load Balancing Primary
Gateway or Firewall
ND Load Balancing Backup
AFS Database Server & File Server for Shared Web Cache
WTE Caching & Filtering Proxy AFS Client
AFS File Server for Shared Web Cache
WTE Caching & Filtering Proxy AFS Client
Internet
Gateway or Firewall

Figure 277. Graphical Representation of the RCA Scenario

15.3.1 Scenario Implementation

1. We constructed an AFS cell called cuzcuz.itso.ibm.com following the steps described in *IBM WebSphere Performance Pack: Web Content Management with IBM AFS Enterprise File System*, SG24-5857.

   One of the most useful characteristics of AFS is server ReadOnly volume replication, which means that we can have several copies of the same volume replicated across a number of file servers. This way, clients can access the closest server. Another advantage is that the workload will be balanced through all the servers. But in the case of RCA, the cache directory needs to be updated continuously, so the volumes must be accessed through the ReadWrite path. Even for a remote read access from another proxy, the ReadWrite volume must be available, because a ReadOnly volume is created and replicated manually and statically.

2. We created a separate volume for each proxy, but we didn't replicate those volumes.
3. We configured all the proxy servers as clients of the AFS cell.
4. We granted read and write permissions for all proxies when accessing any other proxy cache directory.

We set up two proxy AIX machines, pluto.itso.ral.ibm.com and rs600030.itso.ral.ibm.com. These machines had essentially the same configurations.

We also configured RCA. This can be done by either accessing the Configuration and Administration Forms or by including the following lines in the ibmproxy.conf configuration file of both the WTE machines:

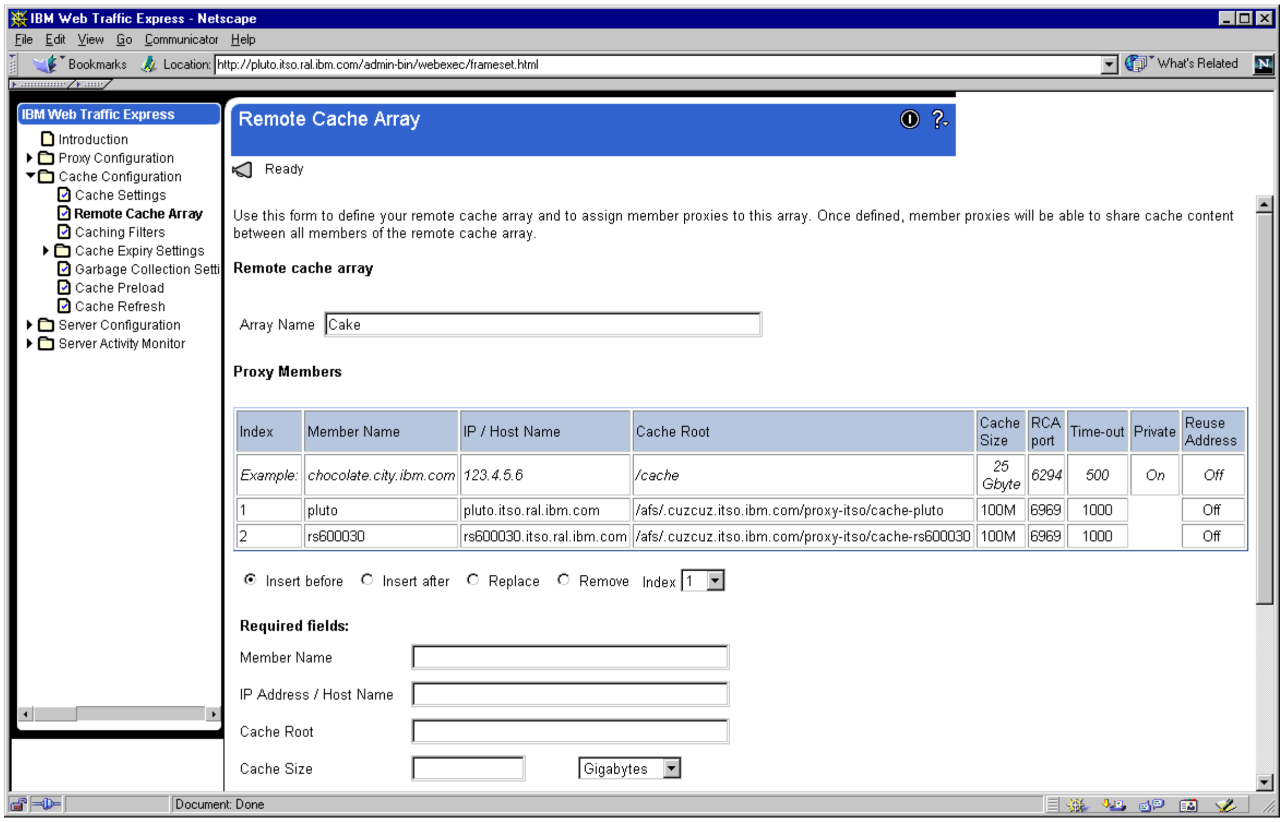
```
Version    RCA/1.0
ArrayName Cake
Member rs600030 {
        RCAAddr         rs600030.itso.ral.ibm.com
        RCAPort         6969
        CacheSize       100M
        CacheRoot       /afs/.cuzcuz.itso.ibm.com/proxy-itso/cache-rs600030
        Timeout         1000 milliseconds
        BindSpecific    On
        ReuseAddr       Off
}

Member pluto {
        RCAAddr         pluto.itso.ral.ibm.com
        RCAPort         6969
        CacheSize       100M
        CacheRoot       /afs/.cuzcuz.itso.ibm.com/proxy-itso/cache-pluto
        Timeout         1000 milliseconds
        ReuseAddr       Off
}
```

Figure 278. RCA Configuration in ibmproxy.conf

Figure 278 on page 355 shows the configuration for an array called Cake. We can see that the cache directory for each proxy is inside the same AFS cell, and is under the ReadWrite path, .cuzcuz.itso.ibm.com. Through these lines in the WTE configuration file, both proxies know what the cache directories are, and can access them directly. The `RCAPort` directive tells each proxy which port it should connect to, and how to communicate with the other members.

It is very important that the RCA block of the configuration be identical in all the array members. Even the order in which members appear in the RCA block should not differ. One way to make this easier is to store the RCA configuration in a separate file, located somewhere in the shared file system (even on a replicated volume, if using AFS), and then including this configuration using the `RCAConfigFile` directive in the WTE ibmproxy.conf configuration file.

After restarting the WTE servers, we accessed the WTE Configuration and Administration Forms of both the WTE servers, and we verified that the forms reflected the configuration we had issued:



Remote Cache Array

Ready

Use this form to define your remote cache array and to assign member proxies to this array. Once defined, member proxies will be able to share cache content between all members of the remote cache array.

Remote cache array

Array Name: Cake

Proxy Members

Index	Member Name	IP / Host Name	Cache Root	Cache Size	RCA port	Time-out	Private	Reuse Address
Example:	*chocolate.city.ibm.com*	*123.4.5.6*	*/cache*	*25 Gbyte*	*6294*	*500*	*On*	*Off*
1	pluto	pluto.itso.ral.ibm.com	/afs/.cuzcuz.itso.ibm.com/proxy-itso/cache-pluto	100M	6969	1000		Off
2	rs600030	rs600030.itso.ral.ibm.com	/afs/.cuzcuz.itso.ibm.com/proxy-itso/cache-rs600030	100M	6969	1000		Off

[✓ Insert before] Insert after Replace Remove Index 1

Required fields:

Member Name	
IP Address / Host Name	
Cache Root	
Cache Size	Gigabytes

Figure 279. RCA Configuration in the Web Interface

A simple test permitted us to demonstrate that RCA was functioning correctly.

Chapter 16. Peak Load Management

In WebSphere Performance Pack Version 2, a new Advisor, known as the WTE Advisor for SecureWay Network Dispatcher (ND), enhances load balancing management between multiple Web Traffic Express (WTE) nodes by preventing the ND server from sending new requests to a WTE node that is engaged in garbage collection or cache refresh. This is a new feature implemented for the first time in WebSphere Performance Pack Version 2.

Using the WTE Advisor, an ND server has the ability to detect and react to sudden increases in activity on a WTE node belonging to a cluster. The new WTE Advisor on ND opens a connection to the WTE node, sends a *WTE-specific* HTTP GET request, and interprets the response as a WTE load.

Notice that the WTE Advisor integrates the load balancing component with the caching and filtering component of WebSphere Performance Pack. This new feature is called *peak load management* and is part of the quality-of-service enhancements offered by IBM WebSphere Performance Pack Version 2.

16.1 How Peak Load Management Works

Peak load management is a new feature implemented for the first time in IBM WebSphere Performance Pack Version 2. You can perform peak load management by activating the WTE Advisor on a Dispatcher machine that is load balancing the workload between two or more WTE proxy servers.

The WTE Advisor is a process running on the Dispatcher machine, but is specific for WTE. This process will not work if the load balanced proxy server is not WTE. When the WTE Advisor runs on a Dispatcher machine, this machine sends a specific HTTP request to the load balanced WTE servers, and these reply back with four pieces of information:

- A boolean saying if the cache is reloading
- A boolean saying if garbage collection is running
- The number of active threads
- The maximum number of threads

The WTE Advisor is able to understand these pieces of information passed to it by the WTE machines, and feeds them back to the Dispatcher Manager. The Manager uses the data retrieved by the WTE Advisor to modify the weights associated with the WTE proxy servers, if necessary, in proportion to the importance given by the Dispatcher administrator to the Advisor’s input.

This way, the Dispatcher can detect when a WTE proxy server is performing some CPU-intense activities better than it could if the WTE Advisor were not running. The Dispatcher takes into account all these resource-intensive activities and makes its load balancing decision appropriately, as we demonstrate in the following section.

16.2 Peak Load Management Scenario

In this section, we show you a basic configuration for an ND machine load balancing multiple WTE proxy server nodes. We built the network with five workstations: one client, one Web server, two WTE proxy servers and one ND machine load balancing the traffic between the two WTE servers. We installed the components as shown in the following diagram:

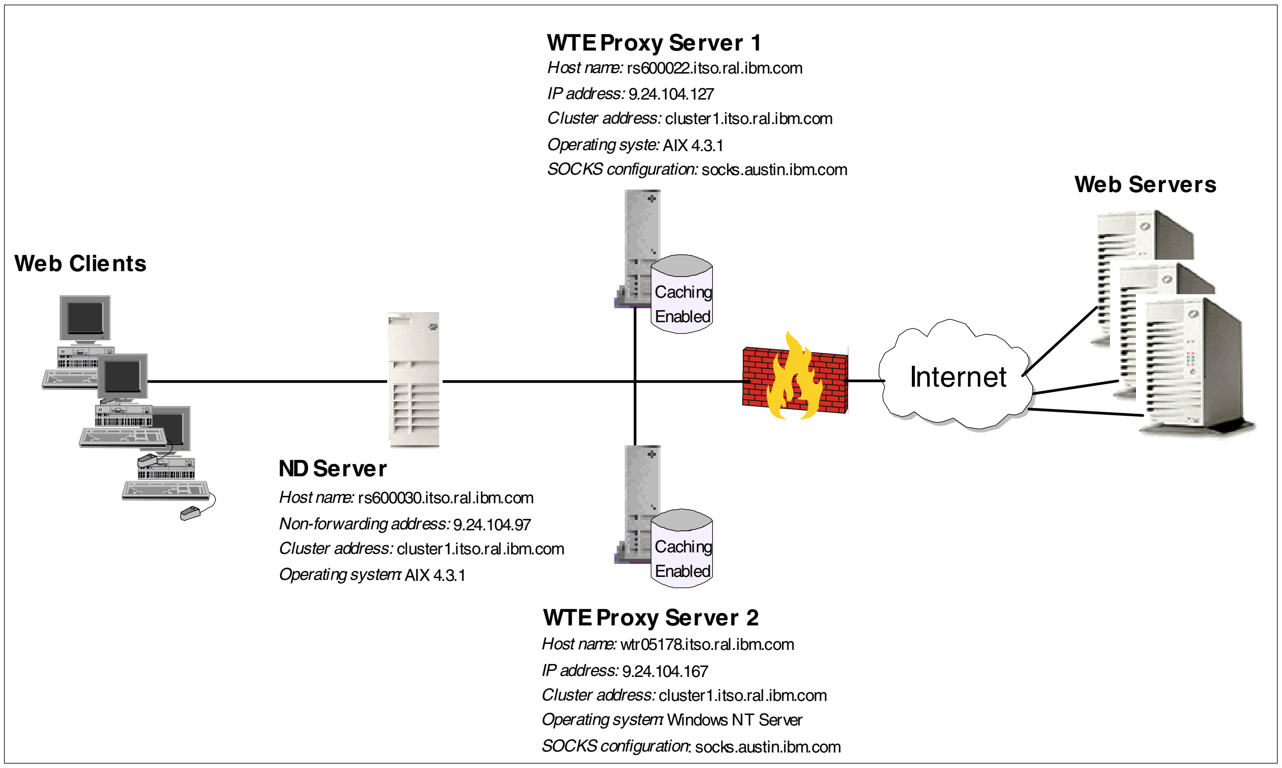

WTE Proxy Server 1
Host name: rs600022.itso.ral.ibm.com
IP address: 9.24.104.127
Cluster address: cluster1.itso.ral.ibm.com
Operating syste: AIX 4.3.1
SOCKS configuration: socks.austin.ibm.com
Web Servers
Web Clients
Caching Enabled
Internet
ND Server
Host name: rs600030.itso.ral.ibm.com
Non-forwarding address: 9.24.104.97
Cluster address: cluster1.itso.ral.ibm.com
Operating system: AIX 4.3.1
Caching Enabled
WTE Proxy Server 2
Host name: wtr05178.itso.ral.ibm.com
IP address: 9.24.104.167
Cluster address: cluster1.itso.ral.ibm.com
Operating system: Windows NT Server
SOCKS configuration: socks.austin.ibm.com

Figure 280. Peak Load Management Scenario

Notice that the diagram above tries to reflect a real-life situation, where multiple clients and multiple Web servers come into the scenario.

The following table provides more details on the WTE cluster configuration:

Table 28. WTE Cluster Configuration

Machine Role	Host Name	IP Address	Operating System
Dispatcher	rs600030	9.24.104.97	AIX 4.3.1
	cluster1	9.24.104.156	
WTE proxy server 1	rs600022	9.24.104.127	AIX 4.3.1
WTE proxy server 2	wtr05178	9.24.104.167	Windows NT Server 4.0

All the machines above belonged to the domain itso.ral.ibm.com. Notice that 9.24.104.156 was the cluster address in this configuration. The host name associated with this cluster address was cluster1.itso.ral.ibm.com.

All the machines shown in Figure 280 on page 358 were provided with a token-ring interface and connected to the same LAN.

The software we used in our scenario is described in the following list:

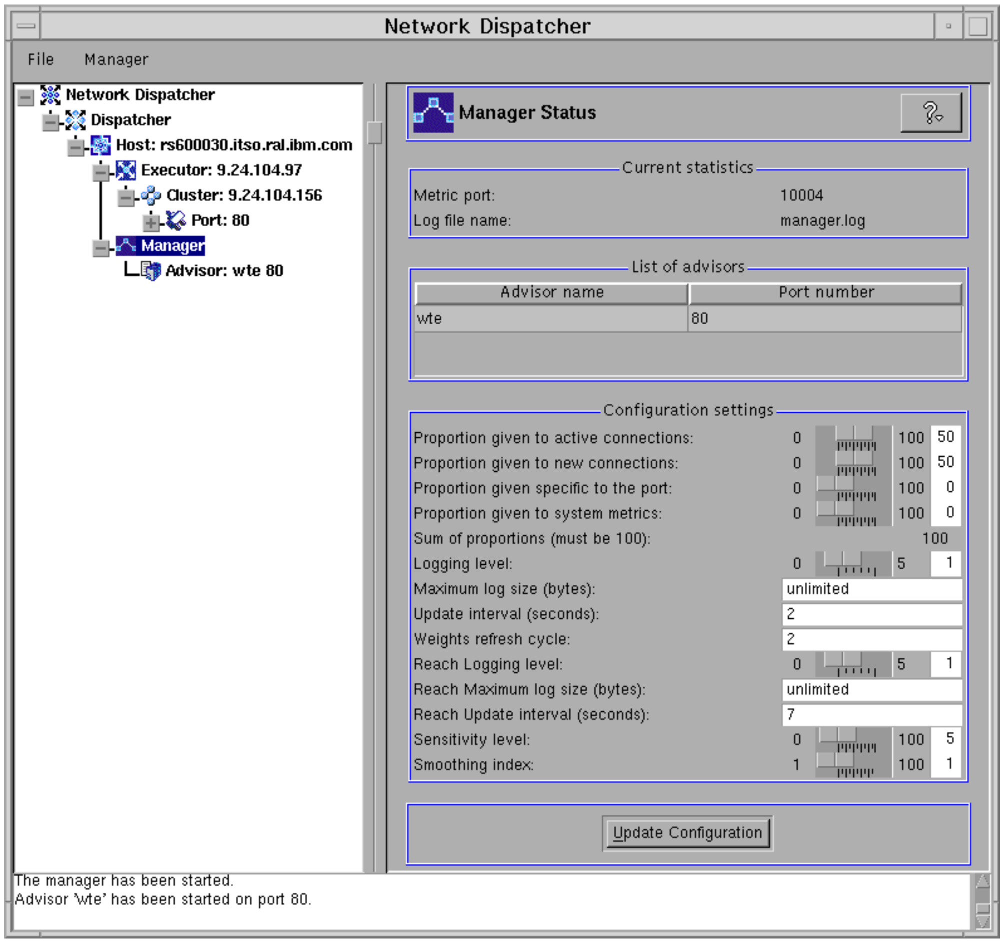
- Netscape Navigator 4.5 was the Web browser running on the Web client machine.

- The load balancing function was provided by the Dispatcher component of ND Version 2.1.
- The proxy server function was provided by WTE Version 2.0.
- The Web server function was provided by Lotus Domino Go Webserver V4.6.2.5.

16.2.1 Dispatcher Configuration

For the installation and configuration of the ND component of IBM WebSphere Performance Pack, refer to *IBM WebSphere Performance Pack - Load Balancing with IBM SecureWay Network Dispatcher,* SG24-5858. We configured the Dispatcher using the ND configuration graphical user interface (GUI).

The following diagram shows the Dispatcher GUI after the configuration was completed:



- Network Dispatcher
  - Dispatcher
    - Host: rs600030.itso.ral.ibm.com
      - Executor: 9.24.104.97
        - Cluster: 9.24.104.156
          - Port: 80
      - Manager
        - Advisor: wte 80

Manager Status

Current statistics

Metric port:	10004
Log file name:	manager.log

List of advisors

Advisor name	Port number
wte	80

Configuration settings

Proportion given to active connections:	0	100	50
Proportion given to new connections:	0	100	50
Proportion given specific to the port:	0	100	0
Proportion given to system metrics:	0	100	0
Sum of proportions (must be 100):			100
Logging level:	0	5	1
Maximum log size (bytes):			unlimited
Update interval (seconds):			2
Weights refresh cycle:			2
Reach Logging level:	0	5	1
Reach Maximum log size (bytes):			unlimited
Reach Update interval (seconds):			7
Sensitivity level:	0	100	5
Smoothing index:	1	100	1

```
The manager has been started.
Advisor 'wte' has been started on port 80.
```

Figure 281. Network Dispatcher GUI - WTE Cluster with WTE Advisors

As you can see from the figure above, we defined a cluster of two WTE proxy servers, load balanced by an ND machine. In this cluster, we also activated the WTE Advisor on port 80. Advisors can be activated as explained in *IBM WebSphere Performance Pack - Load Balancing with IBM SecureWay Network Dispatcher,* SG24-5858. In our configuration, you can see that the Dispatcher was configured to load balance the traffic to the two WTE servers giving 40% of importance to the active connections, 40% of importance to the new connections, and 20% of importance to the feedback coming from Advisors (in this case, from the WTE Advisor). In this scenario, no importance was given to the input coming from system monitoring tools, such as ISS, because no system monitoring tool was part of the configuration.

It would not be appropriate to give a high importance to the Advisors' inputs, but in this case we preferred to set the importance to 20% as we wanted to see the real effects of the WTE Advisors in the load balancing decisions made by the Dispatcher.

16.2.2 Configuration of the WTE Proxy Servers

No special configurations other than a basic one are required on the WTE servers when the WTE Advisor is running on the load balancing Dispatcher machine.

16.2.3 Testing the Peak Load Management Scenario

With the configuration described above, we performed some tests to verify how the Dispatcher reacts to a sudden activity increase on one of the WTE machines.

16.2.3.1 Load Balancing Basic WTE Activities

On the Web client machine, we configured the Web browser to access the Internet through the proxy server 9.24.104.156. This was the cluster address of our WTE proxy server cluster (see Table 28 on page 358). Then, we requested multiple URLs from the clients simultaneously. From the Dispatcher Server monitor GUI, we could see that the two WTE proxy servers were in use simultaneously as shown in the following figure:

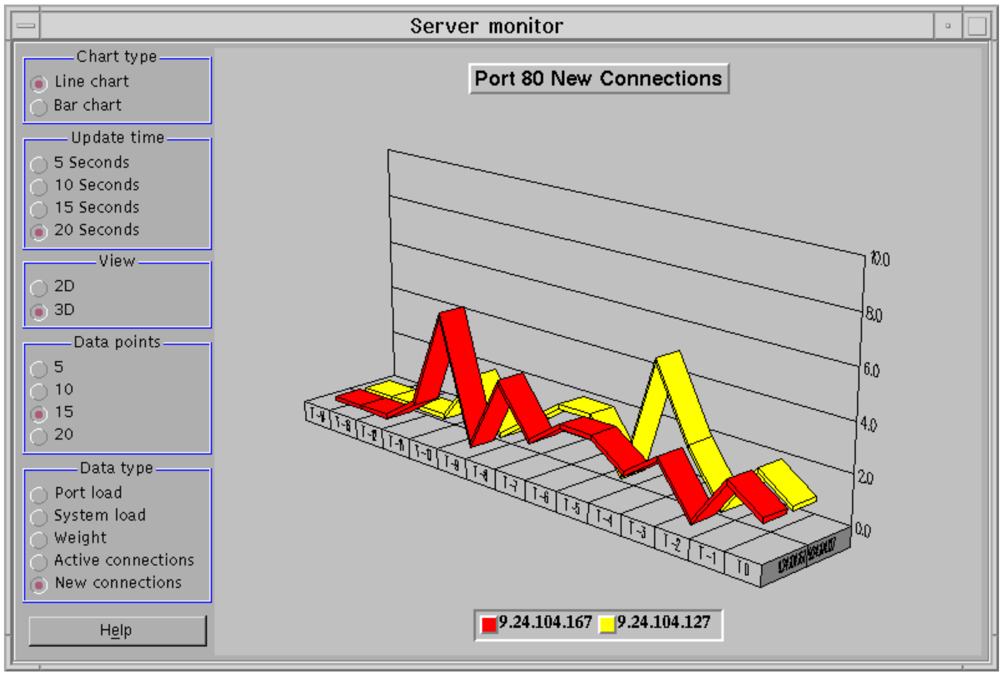

Server monitor

Chart type: [✓ Line chart] Bar chart
Update time: 5 Seconds, 10 Seconds, 15 Seconds, [✓ 20 Seconds]
View: 2D [✓ 3D]
Data points: 5, 10, [✓ 15], 20
Data type: Port load, System load, Weight, Active connections, [✓ New connections]

Port 80 New Connections

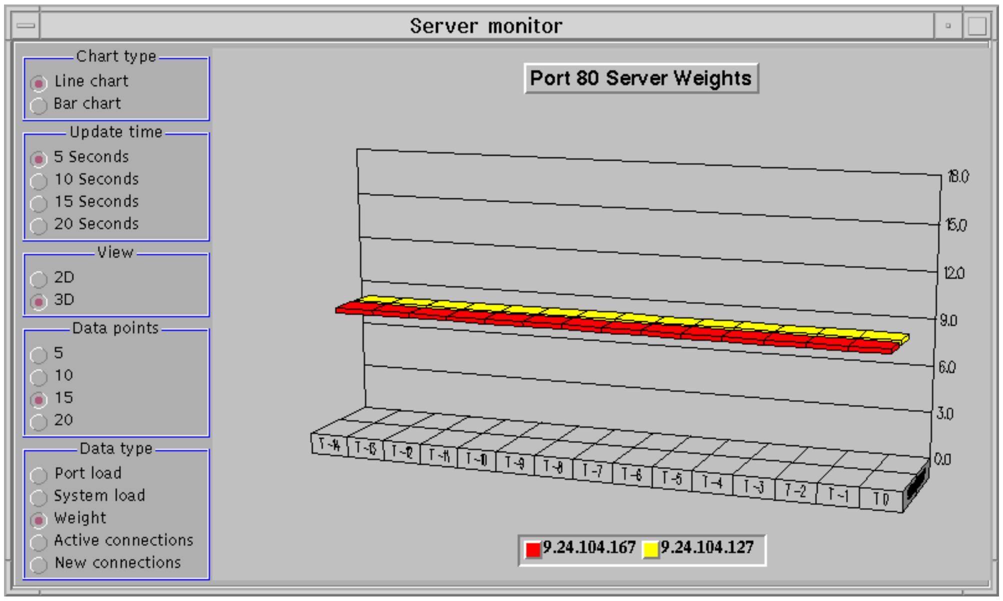
🟥 9.24.104.167 🟨 9.24.104.127

Figure 282. Network Dispatcher GUI - WTE Proxy Servers Running

16.2.3.2 Consequences of the WTE Advisor Activation

As mentioned earlier, the WTE Advisor for the ND enhances load management on multiple WTE servers by preventing the Dispatcher from sending new requests to a WTE node that is currently engaged in garbage collection or cache refresh. This function is very useful, since garbage collection and cache refresh are very CPU-intensive activities for a WTE server, and it is very important that the Dispatcher have a way to detect when a WTE server is using most of its resources for these resource-consuming activities.

The following figure shows that, at the beginning of our tests, when there was no particular activity on either of the WTE proxy servers, both WTE servers had the same weights, as shown:



Server monitor

Chart type: [✓ Line chart] Bar chart

Update time: [✓ 5 Seconds] 10 Seconds 15 Seconds 20 Seconds

View: 2D [✓ 3D]

Data points: 5 10 [✓ 15] 20

Data type: Port load System load [✓ Weight] Active connections New connections

Port 80 Server Weights

🟥 9.24.104.167 🟨 9.24.104.127

Figure 283. Network Dispatcher GUI - WTE Proxy Servers - No Load

We then activated the cache refresh process on the WTE proxy server 1 (see Table 28 on page 358), whose hostname was rs600022. Although cache refresh can start automatically at specific times, for this test we manually forced the WTE proxy server 1 to run the cache refresh process. We did this with the command:

```
cacheagt -r /etc/ibmproxy.conf
```

Through the WTE Advisor, the Dispatcher immediately registered that cache refresh was running on the WTE proxy server 1, and the Dispatcher Manager decreased the weight of this WTE server to reduce the number of connections that were going to be forwarded to it. The Dispatcher Server monitor immediately registered this weight variation:

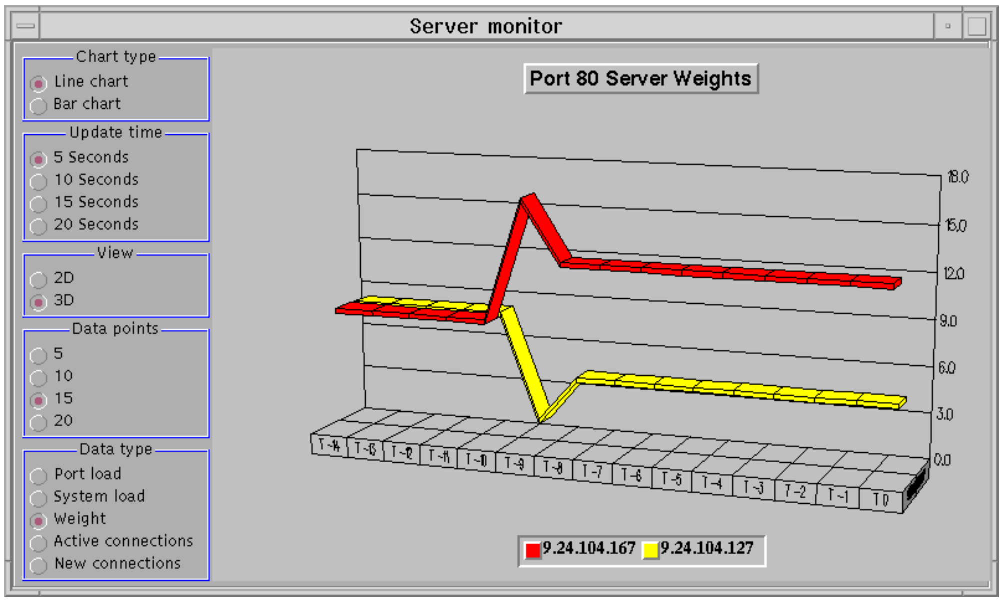
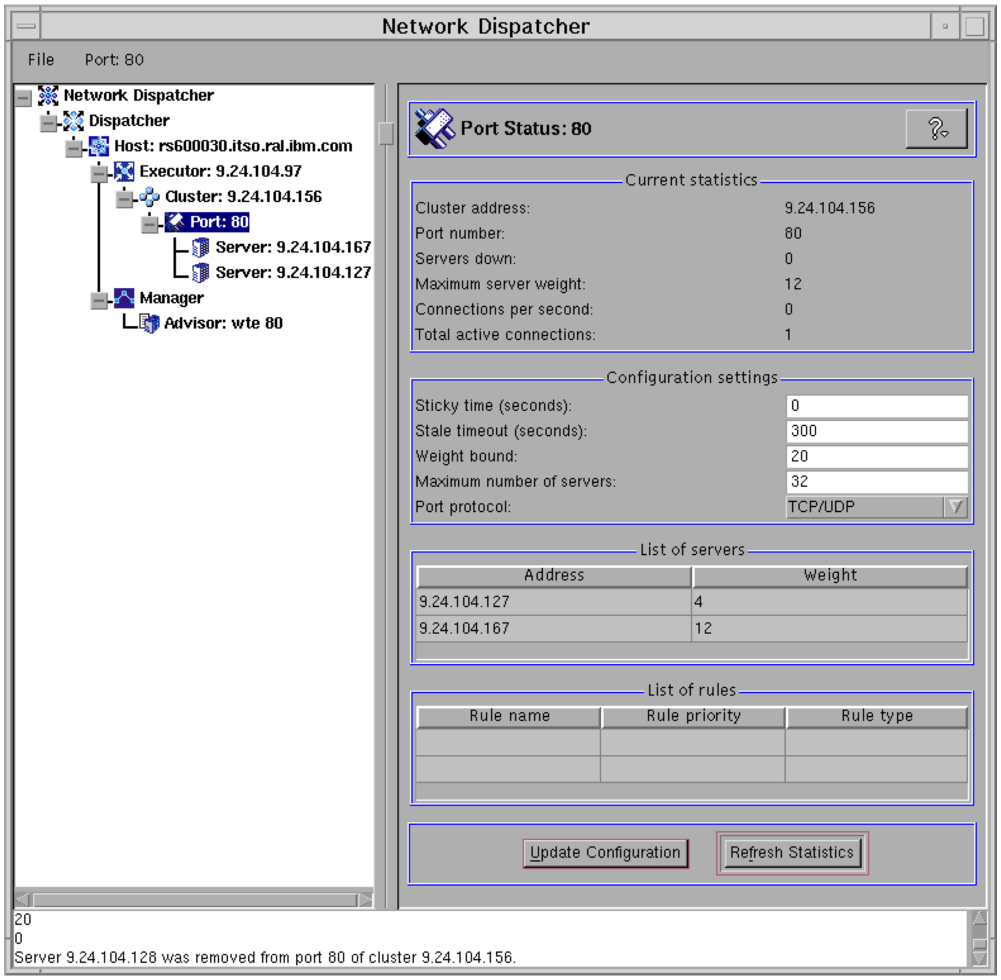


Chart type: [✓ Line chart] Bar chart
Update time: [✓ 5 Seconds] 10 Seconds 15 Seconds 20 Seconds
View: 2D [✓ 3D]
Data points: 5 10 [✓ 15] 20
Data type: Port load System load [✓ Weight] Active connections New connections

Figure 284. Network Dispatcher GUI - WTE Proxy Servers - Cache Refresh in Progress

We can see that the weight of the WTE proxy server 1, having IP address 9.24.104.127, is now much lower than the weight of the WTE proxy server 2, having IP address 9.24.104.167. The following diagram shows the Dispatcher configuration GUI and the weight of each of the WTE proxy servers when the cache refresh process was running:



Current statistics

Cluster address:	9.24.104.156
Port number:	80
Servers down:	0
Maximum server weight:	12
Connections per second:	0
Total active connections:	1

Configuration settings

Sticky time (seconds):	0
Stale timeout (seconds):	300
Weight bound:	20
Maximum number of servers:	32
Port protocol:	TCP/UDP

List of servers

Address	Weight
9.24.104.127	4
9.24.104.167	12

List of rules

Rule name	Rule priority	Rule type

```
20
0
Server 9.24.104.128 was removed from port 80 of cluster 9.24.104.156.
```

Figure 285. Network Dispatcher GUI - Cache Refresh in Progress

This demonstration completes our scenario on the peak load management feature of IBM WebSphere Performance Pack Version 2.

Chapter 17. Common Configuration

The Common Configuration utility is a new feature of IBM WebSphere Performance Pack Version 2. In Version 2, it provides access to seven browser-based wizards that can be used to configure specific aspects of the three components of the product:

- IBM AFS Enterprise File System (AFS):
  - Getting Started with AFS
  - Add an AFS Web volume
  - Replicate an AFS Web volume
  - Move an AFS Web volume
  - Delete an AFS Web volume
- IBM Web Traffic Express (WTE):
  - Platform for Internet Content Selection (PICS) filtering wizard
- IBM SecureWay Network Dispatcher (ND):
  - Add a Network Dispatcher cluster

The wizards do not provide access to all the functions of the individual component configuration programs. Use the GUIs provided with each of the components to perform additional product-specific configuration.

17.1 Installation

The Common Configuration utility must be installed and configured as a separate component. The installation and configuration are supported on the three platforms where IBM WebSphere Performance Pack itself is supported.

17.1.1 Planning for the Installation

The first step is to choose a machine that will perform the function of *configuration server* for your environment. The Common Configuration utility allows for remote configuration of the WebSphere Performance Pack components. In other words, it is not necessary to install the Common Configuration utility on the same ND, AFS, or WTE machine you will use the Common Configuration utility for.

Access to the Common Configuration utility machine is Web based.

The following diagram shows how the Common Configuration utility works in relation to all of the other WebSphere Performance Pack components:

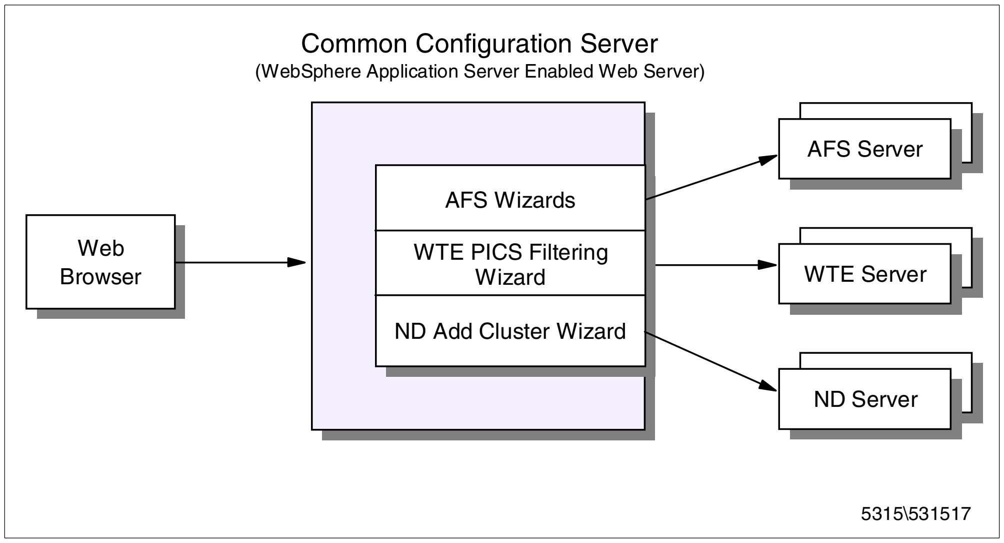

Common Configuration Server
(WebSphere Application Server Enabled Web Server)
Web Browser
AFS Wizards
WTE PICS Filtering Wizard
ND Add Cluster Wizard
AFS Server
WTE Server
ND Server
5315\531517

Figure 286. Common Configuration Server Environment

Because the Common Configuration utility is implemented by using IBM WebSphere Application Server and associated Java servlets (see 1.7.1, “IBM WebSphere Application Server” on page 23), it is necessary that your chosen configuration server machine meet these two prerequisites:

1. A Web server must be installed on the Common Configuration machine. This Web server must be supported by WebSphere Application Server on your platform. See 17.1.2, “Installing the Common Configuration Utility” on page 366 for a list of the supported Web servers on your platform.

   You need to know the document directory and the configuration file directory for whichever Web server you have installed.

2. The Web server must have the ability to serve files that are contained on one of the machine’s local file systems. It cannot be set up to serve files only from AFS.

Notice that it is not necessary to install WebSphere Application Server on the Common Configuration server machine. If WebSphere Application Server is not installed yet, the Common Configuration installation process installs the WebSphere Application Server filesets transparently during the installation routine.

Once these prerequisites are met, you can proceed to install the Common Configuration on your Common Configuration server machine.

17.1.2 Installing the Common Configuration Utility

The process of installing the Common Configuration utility with the Java InstallShield differs only slightly on the three platforms that it can be installed on. We will demonstrate the installation and note where the differences are.

We performed our installation of the Common Configuration utility on a uniprocessor IBM RS/6000 43P having 192 MB of RAM, 2.2 GB of hard disk and one token-ring interface. This RS/6000 had AIX Version 4.3.1, Java Runtime Environment (JRE) 1.1.6 and IBM HTTP Server Version 1.3.3 installed on it.

We also installed the Common Configuration utility on an IBM PC 365 running Windows NT Server4.0. This machine had Java Development Kit (JDK) 1.1.6 and IBM HTTP Server Version 1.3.3 installed on it.

One of the Web servers listed below that is supported by the WebSphere Application Server must be installed on your machine:

- On the AIX platform:
  - IBM HTTP Server V1.3.3
  - Lotus Domino Go Webserver V4.6.2
  - Netscape Enterprise V3.01
  - Netscape Enterprise V3.51
  - Apache HTTP Server V1.3.2
- On the Windows NT platform:
  - IBM HTTP Server V1.3.3
  - Lotus Domino Go Webserver V4.6.2
  - Netscape Enterprise v3.01
  - Netscape Enterprise v3.5
  - Apache v1.3.2
  - IIS 2/3.0
  - IIS 4.0
- On the Solaris platform:
  - IBM HTTP Server V1.3.3
  - Apache Server V1.3.2
  - Lotus Domino Go Webserver V4.6.2.5
  - Netscape Enterprise Server V3.01 and V3.51 (recommend V3.5.1)
  - Netscape FastTrack Server V3.01

The installation program for the Common Configuration utility makes use of Java InstallShield's setup class. For this reason you are required to pre-install the Java Virtual Machine (JVM). On AIX, only the Java Runtime Environment (JRE) is sufficient as it contains the JVM, the Java platform core classes, and supporting files. On Windows NT we found it necessary to install the entire JDK to perform the installation. We used JDK Version 1.1.6.

If Java is not installed on your machine or installed at a level lower than 1.1.6, you can find the install image for JDK 1.1.6:

- For AIX at `http://www.ibm.com/java/jdk/download/index.html` or on the IBM WebSphere Performance Pack Version 2 CD-ROM
- For Windows NT and Sun Solaris at `http://www.javasoft.com` or on the IBM WebSphere Performance Pack Version 2 CD-ROM

The installation of the JDK is described in the IBM redbook *Network Computing Framework Component Guide,* SG24-2119.

To prepare for the installation of the Common Configuration utility, follow the steps listed below:

1. Insert the IBM WebSphere Performance Pack Version 2 CD-ROM in the CD-ROM drive.
2. This step will differ on each platform:
   - On AIX, from a command line, enter the following commands:

```
mkdir /cdrom
mount -rv cdrfs /dev/cd0 /cdrom
cd /cdrom/aix
```

- On Windows NT, enter:

  ```
  E:
  cd nt
  ```

  where E is assigned to the CD-ROM drive.

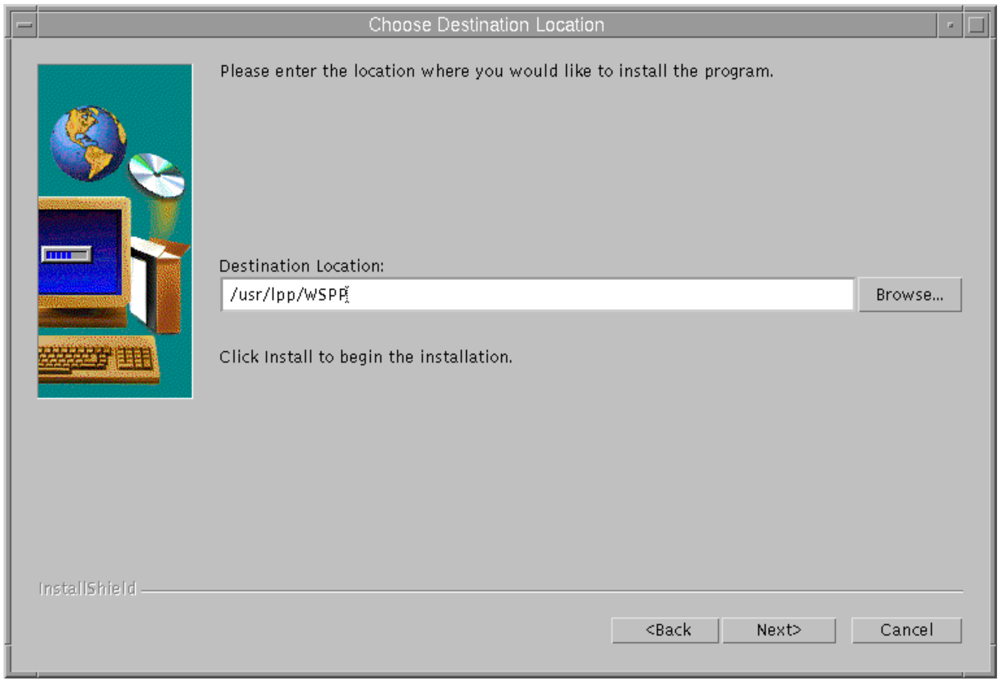
- On Solaris, insert the IBM Websphere Performance Pack CD-ROM in the CD-ROM drive. The system will automatically mount the WebSphere Performance Pack CD-ROM as /cdrom/websphere. Then, from a command prompt, enter the following command:

  ```
  cd /cdrom/websphere/sun
  ```

3. To start the Java InstallShield installation program, enter:

   ```
   java setup
   ```

The first screen you will see is the Welcome window. After clicking the **Next** button, you are prompted to enter the destination location. Note that this is a working directory for Websphere Performance Pack. The default location on Windows NT is C:\WSPP; on Solaris /opt/WSPP; and on AIX /usr/lpp/WSPP, as shown in the following window:


Choose Destination Location

Please enter the location where you would like to install the program.

Destination Location: /usr/lpp/WSPP

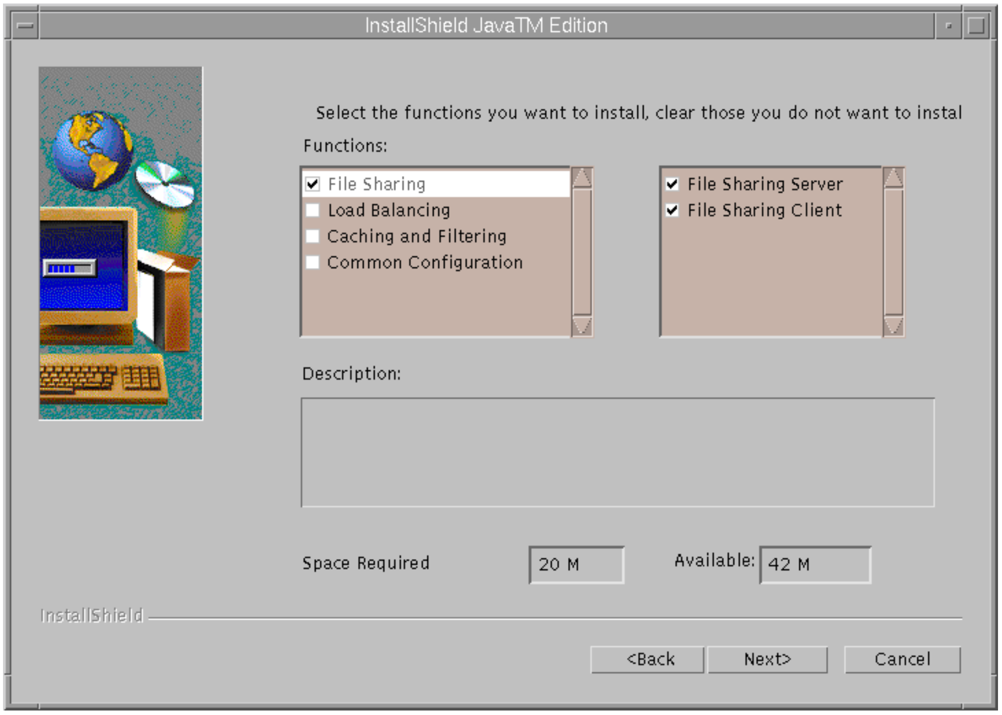
Click Install to begin the installation.

Figure 287. Choose Destination Location

In the above window, you are prompted to click **Install** to proceed. The button does not exist, due to a known InstallShield for Java problem. Click the button labeled **Next** to continue instead. If the destination directory does not exist on your system, you will be presented with a question dialog asking you if you would like the location to be created. Then, you will be presented with a window allowing you to choose which IBM WebSphere Performance Pack V2 components you want to install:

- File Sharing
- Load Balancing
- Caching and Filtering
- Common Configuration

This is shown in the following figure:


InstallShield JavaTM Edition

Select the functions you want to install, clear those you do not want to instal

Functions:

- [✓] File Sharing
- [ ] Load Balancing
- [ ] Caching and Filtering
- [ ] Common Configuration

- [✓] File Sharing Server
- [✓] File Sharing Client

Description:

Space Required 20 M

Available: 42 M

Figure 288. Choose the Component to Install (accinst01)

When we selected **Common Configuration**, a description of that component was displayed as well as the space required to install it. This window also shows you how much space is available in the destination location file system.

If you do not have enough free space in your destination location file system and you click **Next**, you will see a warning window letting you know that there is not enough space to perform the installation. You will then have to use the operating system utilities to increase the size of the file system so that there is enough space for the installation to be performed. After we had successfully increased the size of the /usr file system, we were not able to continue with the Java InstallShield process. In order for the available space field (as seen in Figure 288 on page 369) to be updated to reflect the new amount of available space in the target directory, it was necessary to backup one screen by clicking the **Back** button followed by the **Next** button. Then, with the Common Configuration component selected, we were able to click the **Next** button.

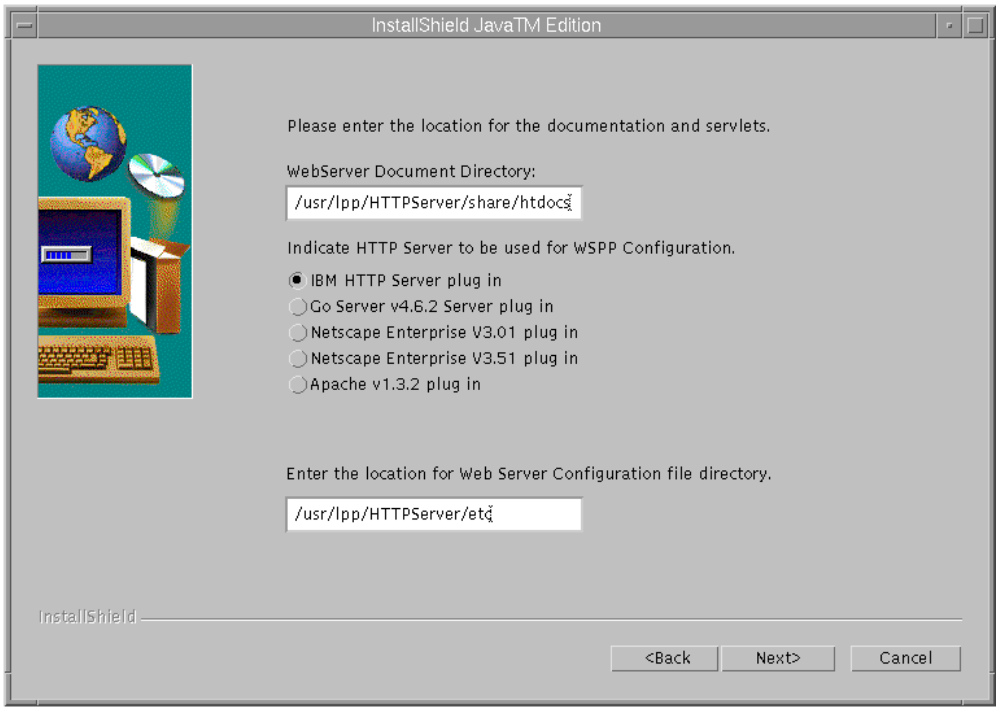
In the next step, we had to provide some information to the install process. The type of information that the user is prompted for next depends on whether or not IBM WebSphere Application Server is already installed on the machine. Furthermore, in both cases the information that the user is prompted for is different for each of the supported platforms and so in the next sections we will show all of the possibilities.

17.1.2.1 Providing Information on AIX and Solaris

The information you provide during the installation depends on whether or not WebSphere Application Server is already installed on your system.

1. **WebSphere Application Server is not already installed**

   When WebSphere Application Server was not preinstalled on our AIX machine, we saw the following window:



Please enter the location for the documentation and servlets.

WebServer Document Directory: /usr/lpp/HTTPServer/share/htdocs

Indicate HTTP Server to be used for WSPP Configuration.

[✓ IBM HTTP Server plug in]
Go Server v4.6.2 Server plug in
Netscape Enterprise V3.01 plug in
Netscape Enterprise V3.51 plug in
Apache v1.3.2 plug in

Enter the location for Web Server Configuration file directory.

/usr/lpp/HTTPServer/etc

Figure 289. Request to Supply Web Server Configuration Details

   The above window will be displayed if you do not have WebSphere Application Server already installed. In order for the correct Web server plug-in to be installed, you will need to select one of the radio buttons to inform the installation process which Web server is installed on the machine.

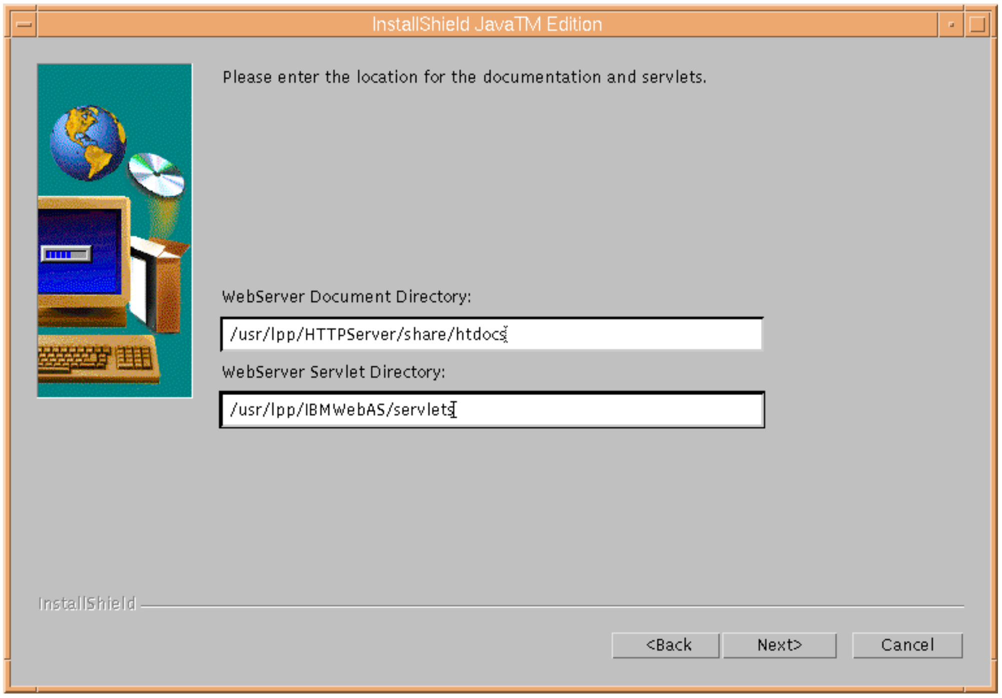
   In addition, you are asked to provide two directories:

   1. The first piece of information the installation process asks you for is the Web server document directory. The installation process requires this information as it places some files in this directory that are used when the Common Configuration utility is launched.
   2. The second piece of information the installation process asks you for is the Web server configuration file directory. Because the installation in this case is going to install WebSphere Application Server, it must make modifications to the Web server configuration file so that the Web server will be able to work together with WebSphere Application Server.

   You will receive warning messages if you try to continue without filling in either of the two fields or if you set the location for the Web server configuration file incorrectly. As we were using the IBM HTTP Server, we selected the radio button for it and filled in the associated directory information.

2. **WebSphere Application Server is already installed**

When WebSphere Application Server was already installed on our AIX machine, we saw the following window:



Please enter the location for the documentation and servlets.

WebServer Document Directory:

/usr/lpp/HTTPServer/share/htdocs

WebServer Servlet Directory:

/usr/lpp/IBMWebAS/servlets

Figure 290. Request to Supply Web Server and Servlet Configuration Details

In this case, you are again asked to supply two directory locations to the install process:

1. The first piece of information it asks you for is the same as in Point 1 on page 370: that is, the Web server document directory. The installation process requires this information as it places some files in this directory that are used when the Common Configuration utility is launched. We supplied the default document directory for IBM HTTP Server as that was the Web server we had preinstalled.
2. The second piece of information is different from Point 2 on page 370. In this case, it asks where the Web server servlet directory is. This directory is where the Common Configuration servlets will be placed during the installation process. We supplied the default servlet directory for WebSphere Application Server on AIX in the second field.

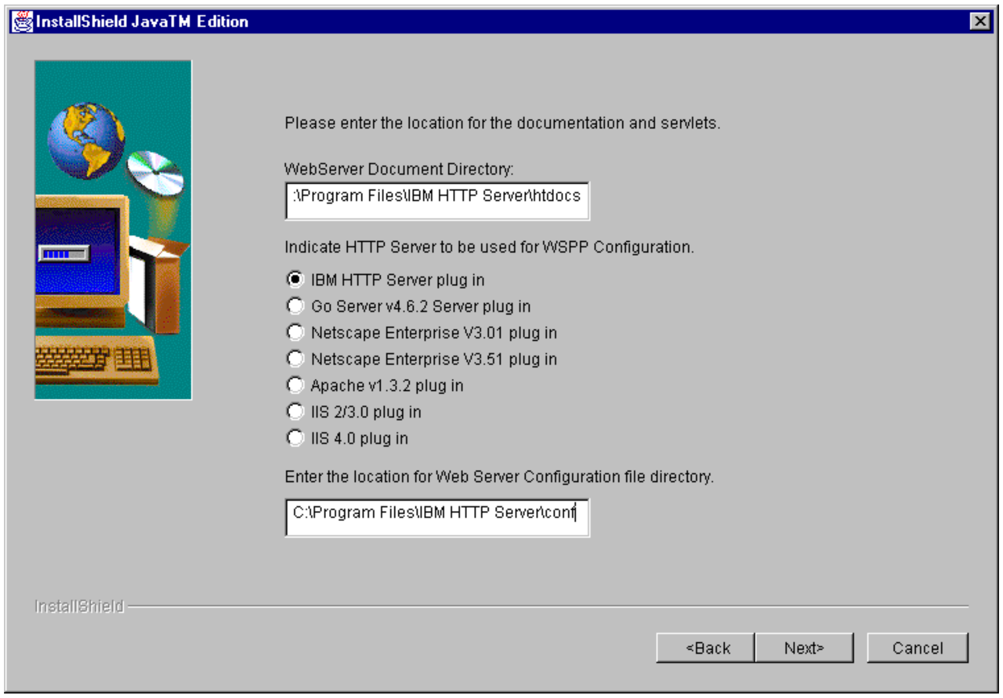
What we have just seen for AIX is very similar in the Solaris platform.

17.1.2.2 Providing Information on Windows NT

The information you provide during the installation depends on whether or not WebSphere Application Server is already installed on your system.

1. **WebSphere Application Server is not already installed**

   When WebSphere Application Server was not preinstalled on our Windows NT machine, we saw the following window:



Please enter the location for the documentation and servlets.

WebServer Document Directory:
:\Program Files\IBM HTTP Server\htdocs

Indicate HTTP Server to be used for WSPP Configuration.

● IBM HTTP Server plug in
○ Go Server v4.6.2 Server plug in
○ Netscape Enterprise V3.01 plug in
○ Netscape Enterprise V3.51 plug in
○ Apache v1.3.2 plug in
○ IIS 2/3.0 plug in
○ IIS 4.0 plug in

Enter the location for Web Server Configuration file directory.
C:\Program Files\IBM HTTP Server\conf

Figure 291. Request to Supply Web Server Configuration Details

The above window will be displayed if you do not have WebSphere Application Server already installed. In order for the correct Web server plug-in to be installed, you will need to select one of the radio buttons to inform the installation process which Web server is installed on the machine.

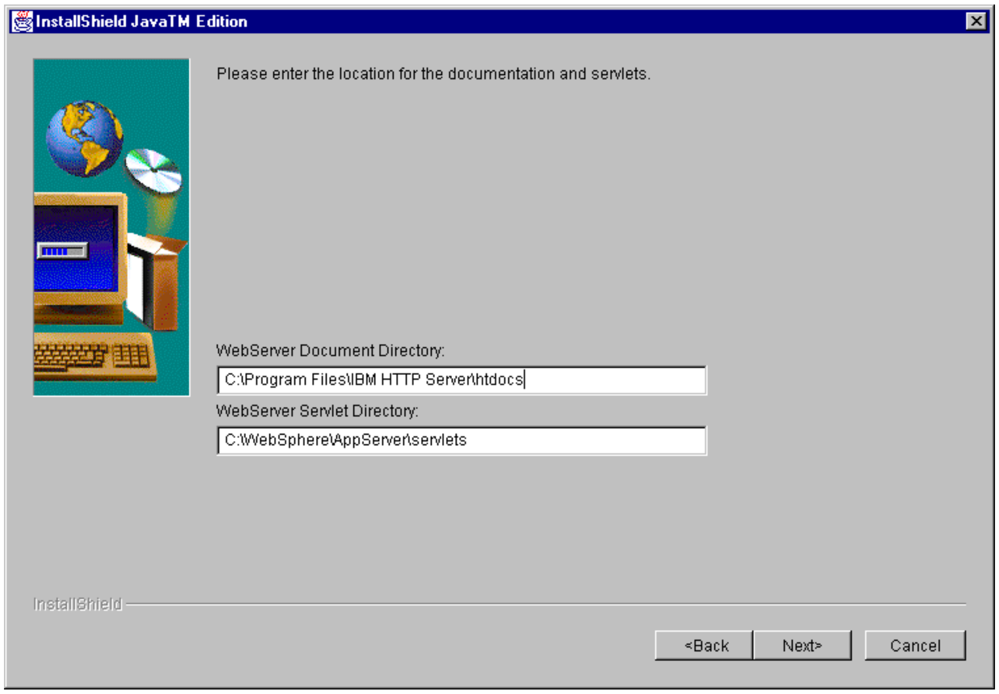
In addition, you are asked to provide two directories:

1. The first piece of information the installation process asks you for is the Web server document directory. The installation process requires this information as it places some files in this directory that are used when the Common Configuration utility is launched.
2. The second piece of information the installation process asks you for is the Web server configuration file directory. Because the installation in this case is going to install WebSphere Application Server, it must make modifications to the Web server configuration file so that the Web server will work together with WebSphere Application Server.

You will receive warning messages if you try to continue without filling in either of the two fields or if you set the location for the Web server configuration file incorrectly. As we were using the IBM HTTP Server, we selected the radio button for it and filled in the associated directory information.

2. WebSphere Application Server is already installed

When WebSphere Application Server was already installed on our Windows NT machine, we saw the following window:



Please enter the location for the documentation and servlets.

WebServer Document Directory:
C:\Program Files\IBM HTTP Server\htdocs

WebServer Servlet Directory:
C:\WebSphere\AppServer\servlets

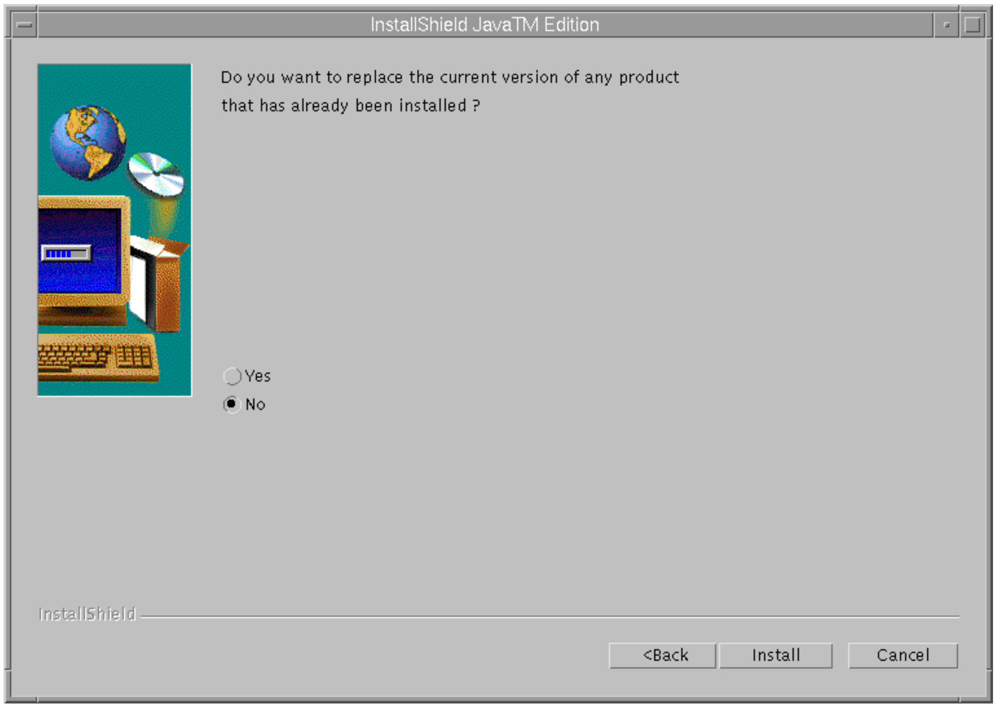
Figure 292. Request to Supply Web Server and Servlet Configuration Details

In this case, you are again asked to supply two directory locations to the install process:

1. The first piece of information the installation process asks you for is the same as in Point 1 on page 371: the Web server document directory. The installation process requires this information as it places in this directory some files that are used when the Common Configuration utility is launched. We supplied the default document directory for IBM HTTP Server as that was the Web server we had preinstalled.
2. The second piece of information is different from Point 2 on page 372. Now, in fact, the installation process asks you where the Web server servlet directory is. This directory is where the Common Configuration servlets will be placed during the installation process. We supplied the default servlet directory for WebSphere Application Server on Windows NT in the second field.

17.1.3 Remaining Installation Steps

After clicking **Next** we saw a window asking if we wanted the installation program to replace other programs already installed. We selected **No** in this case, since we had not installed any programs on our system yet:



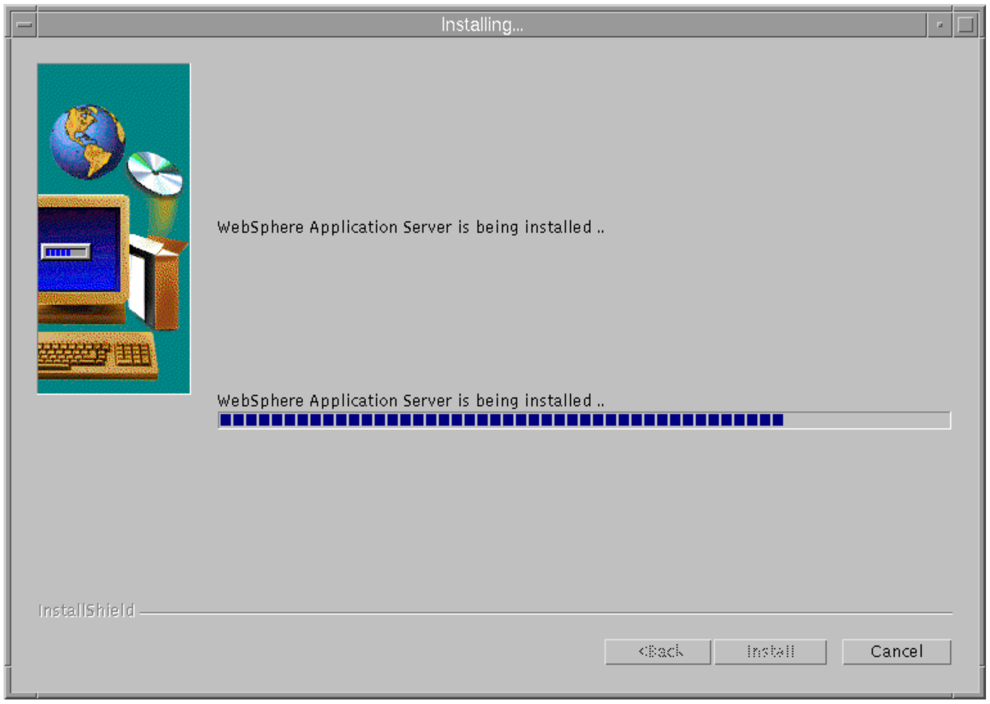
InstallShield JavaTM Edition

Do you want to replace the current version of any product that has already been installed ?

Yes [✓ No]

Figure 293. Choose to Replace Version

After clicking **Install**, the Common Configuration servlets were installed. When WebSphere Application Server was not already installed on our system, we first saw the following screen displayed, showing the installation progress for WebSphere Application Server:



Installing...

WebSphere Application Server is being installed ..

WebSphere Application Server is being installed ..

Figure 294. WebSphere Application Server Installation Progress Indicator

The following screen is always displayed during the installation of the Common Configuration utility, showing the installation progress for the Common Configuration utility itself:

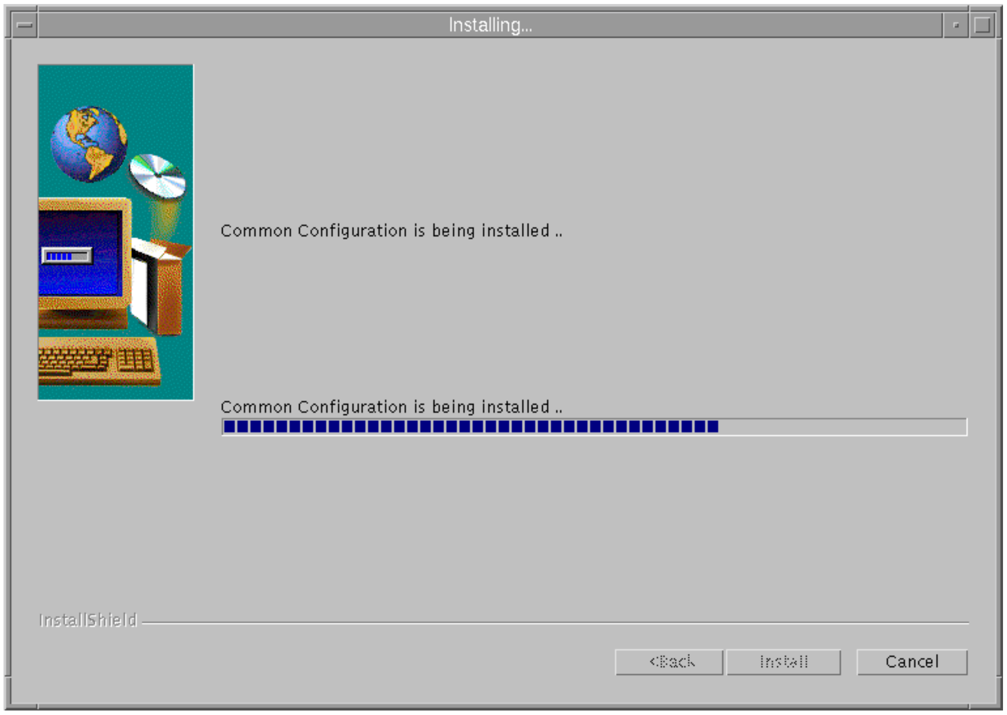
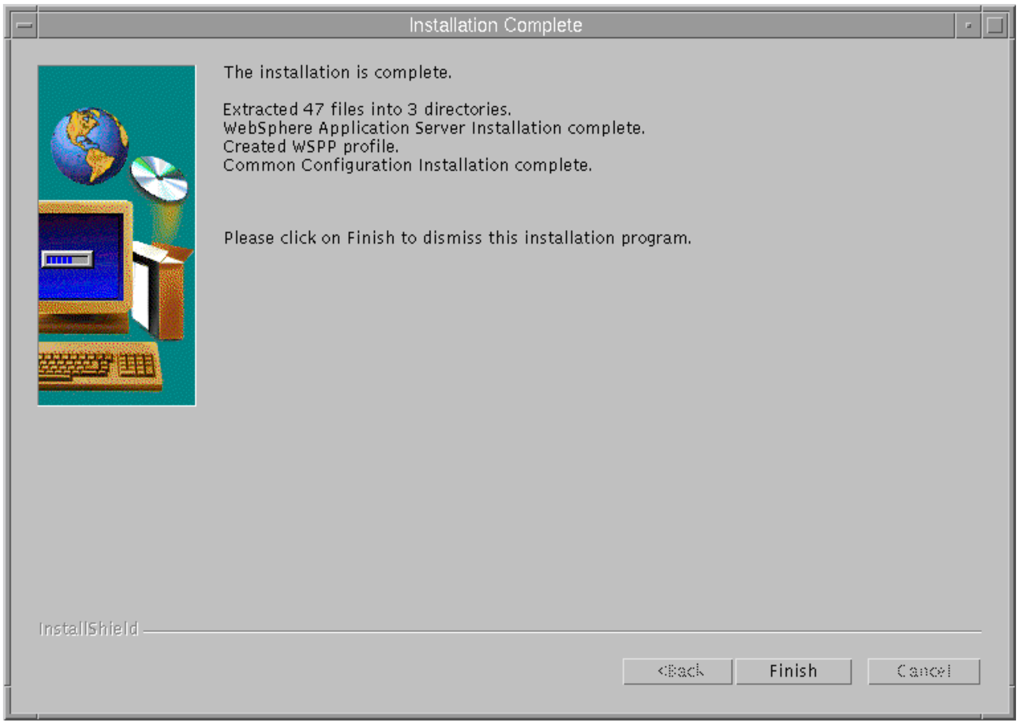

Installing...
Common Configuration is being installed ..
Common Configuration is being installed ..
InstallShield
<Back
Install
Cancel

Figure 295. Common Configuration Installation Progress Indicator

Then, you will be informed that the installation completed successfully:



The installation is complete.

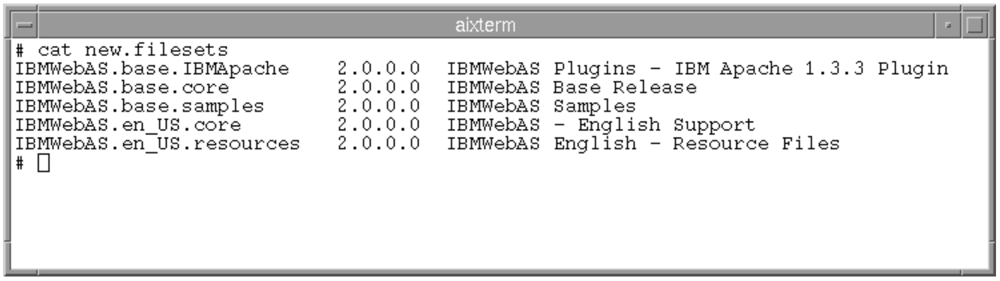
Extracted 47 files into 3 directories.
WebSphere Application Server Installation complete.
Created WSPP profile.
Common Configuration Installation complete.

Please click on Finish to dismiss this installation program.

Figure 296. Successful Completion Indication

We clicked on **Finish** to dismiss the installation process.

At this point, we could verify that the installation of the Common Configuration utility transparently installs WebSphere Application Server on the Common Configuration server. For example, on our AIX system, we saw that the filesets shown in the following figure were the only ones that were installed on our system as a result of the Common Configuration installation:



```
# cat new.filesets
IBMWebAS.base.IBMApache    2.0.0.0  IBMWebAS Plugins - IBM Apache 1.3.3 Plugin
IBMWebAS.base.core         2.0.0.0  IBMWebAS Base Release
IBMWebAS.base.samples      2.0.0.0  IBMWebAS Samples
IBMWebAS.en_US.core        2.0.0.0  IBMWebAS - English Support
IBMWebAS.en_US.resources   2.0.0.0  IBMWebAS English - Resource Files
# 
```

Figure 297. Installing the Common Configuration Utility Installs WebSphere Application Server

17.2 Preparing for the Common Configuration Utility

Communication between the Common Configuration server machine and the WebSphere Performance Pack Version 2 machine that the Common Configuration server is configuring is done via Remote Method Invocation (RMI).

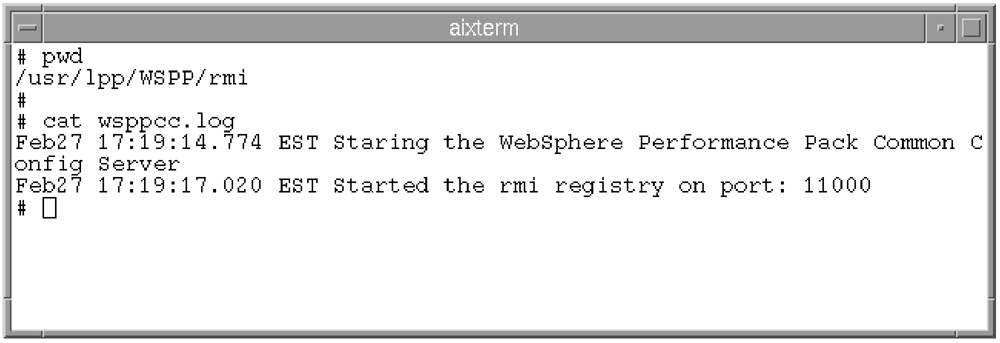
A machine where WebSphere Performance Pack has been installed, and that needs to be configured through the Common Configuration utility, is also known as *WebSphere Performance Pack server machine*. On a WebSphere Performance Pack server machine, the installation program creates a file that must be executed before you can use the Common Configuration to configure WebSphere Performance Pack on this machine. The file is called wsppserver on AIX and Solaris, and wsppserver.cmd on Windows NT. It is located in the rmi subdirectory of the installation directory that you instructed the installation program to use at installation time.

Notice that the wsppserver file must not be executed on the Common Configuration server machine, but on the WebSphere Performance Pack server machine that is going to be the target of the Common Configuration utility. In other words, the wsppserver script must be executed on the machine where one of the three components (AFS, ND, and WTE) of WebSphere Performance Pack has been installed, and needs to be configured (typically remotely) through the Common Configuration utility.

On AIX, we followed these steps to execute the wsppserver file:

```
cd /usr/lpp/WSPP/rmi
chmod +x wsppserver
./wsppserver
```

We confirmed the success of the wsppserver execution by looking at the wsppcc.log file that is also present in the same directory as wsppserver:



```
# pwd
/usr/lpp/WSPP/rmi
#
# cat wsppcc.log
Feb27 17:19:14.774 EST Staring the WebSphere Performance Pack Common C
onfig Server
Feb27 17:19:17.020 EST Started the rmi registry on port: 11000
# 
```

Figure 298. Confirming the Success of the wsppserver Execution

If you would like wsppserver to be run each time the WebSphere Performance Pack machine is rebooted, you could add it to an rc file on AIX or Solaris; on a Windows NT machine, add wsppserver to the Startup folder or registry.

To summarize, the steps that must be done before using the Common Configuration utility are:

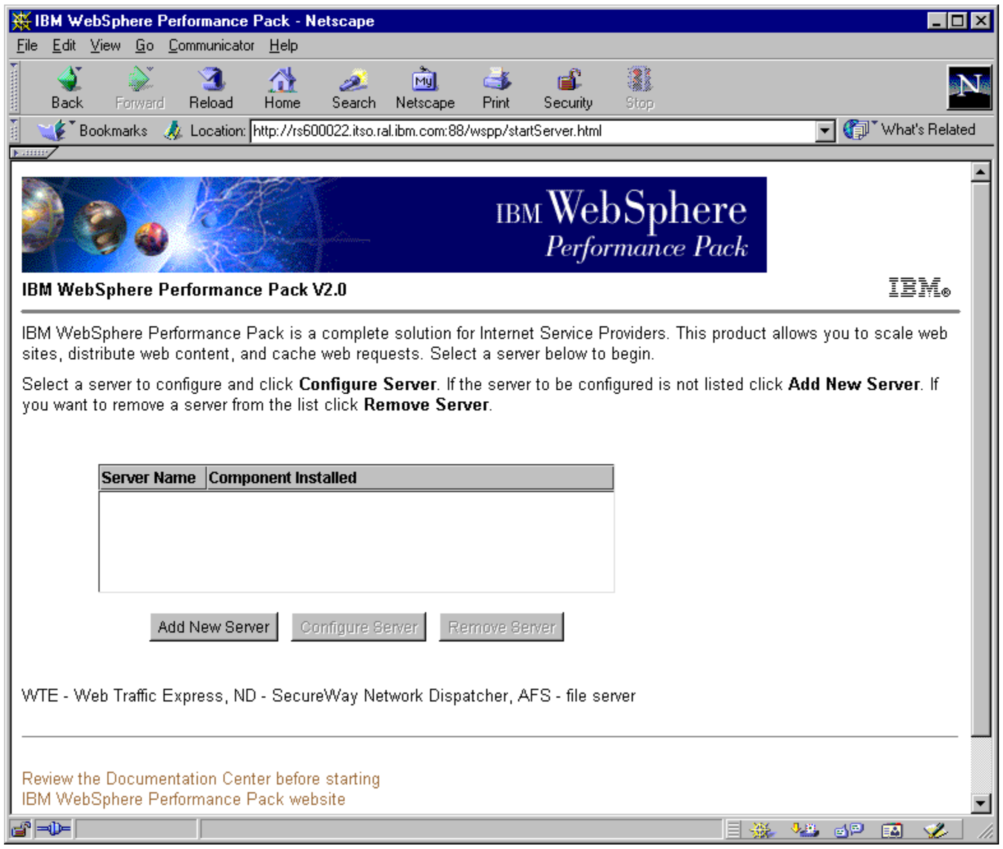
1. Identify a Websphere Performance Pack server machine to be the target of the Common Configuration utility and on this machine execute the wsppserver script file. This file carries a .cmd extension on Windows NT.
2. Identify a configuration server machine and on this machine:

- Install the Common Configuration utility.
- Start the WebSphere Application Server-compliant Web server if it is not already running.

The next step that you must perform is dependent upon which WebSphere Performance Pack component you plan to configure with the Common Configuration utility.

17.3 Launching the Common Configuration Utility

To launch the Common Configuration utility from a Web browser, go to `http://common.configuration.server.name/wspp/startServer.html`, where common.configuration.server.name is the host name of the Common Configuration server machine. Following is the Common Configuration home page:


IBM WebSphere Performance Pack - Netscape

http://rs600022.itso.ral.ibm.com:88/wspp/startServer.html

IBM WebSphere Performance Pack V2.0

IBM WebSphere Performance Pack is a complete solution for Internet Service Providers. This product allows you to scale web sites, distribute web content, and cache web requests. Select a server below to begin.

Select a server to configure and click **Configure Server**. If the server to be configured is not listed click **Add New Server**. If you want to remove a server from the list click **Remove Server**.

Server Name	Component Installed

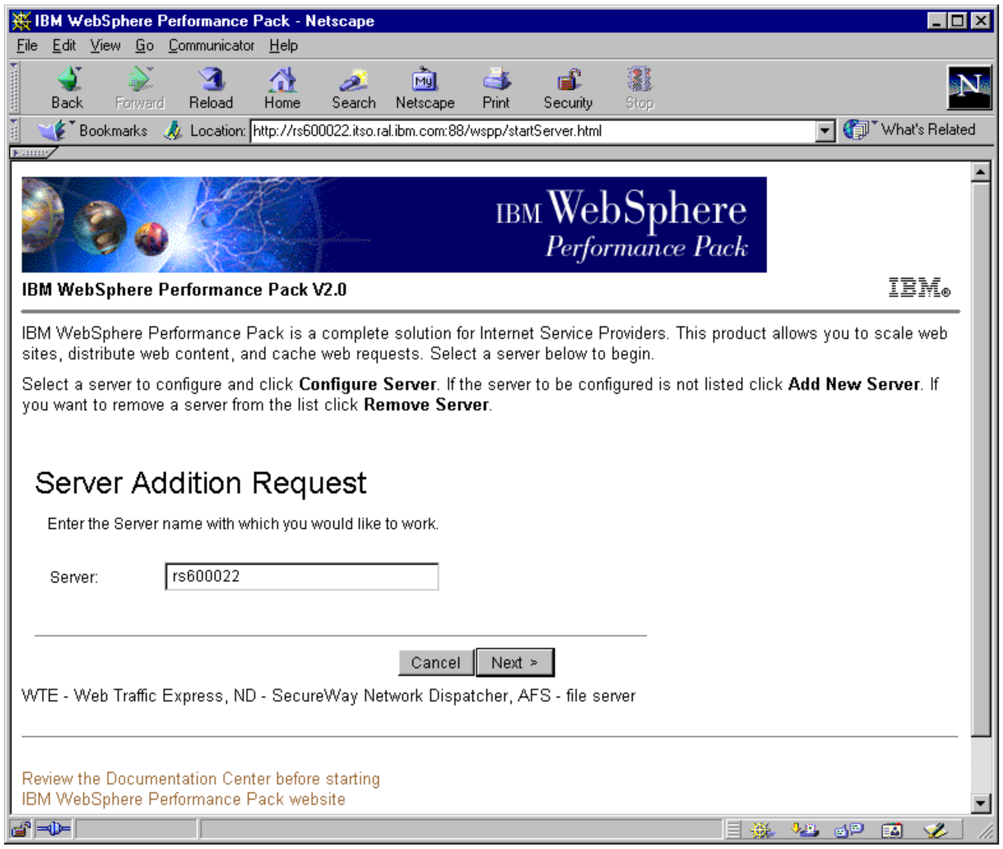
WTE - Web Traffic Express, ND - SecureWay Network Dispatcher, AFS - file server

Review the Documentation Center before starting
IBM WebSphere Performance Pack website

Figure 299. Common Configuration Main Page

From the server selection list on the page, you can either click **Add New Server** or select a server from the list (if the list is not empty) and then click **Configure Server** or **Remove Server**.

The first time we accessed the Common Configuration utility, we selected **Add New Server** and in the subsequent Server Addition Request flield, entered the name of our WebSphere Performance Pack server machine:


IBM WebSphere Performance Pack - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://rs600022.itso.ral.ibm.com:88/wspp/startServer.html
What's Related
IBM WebSphere Performance Pack

IBM WebSphere Performance Pack V2.0

IBM WebSphere Performance Pack is a complete solution for Internet Service Providers. This product allows you to scale web sites, distribute web content, and cache web requests. Select a server below to begin.

Select a server to configure and click **Configure Server**. If the server to be configured is not listed click **Add New Server**. If you want to remove a server from the list click **Remove Server**.

Server Addition Request

Enter the Server name with which you would like to work.

Server: rs600022

Cancel Next >

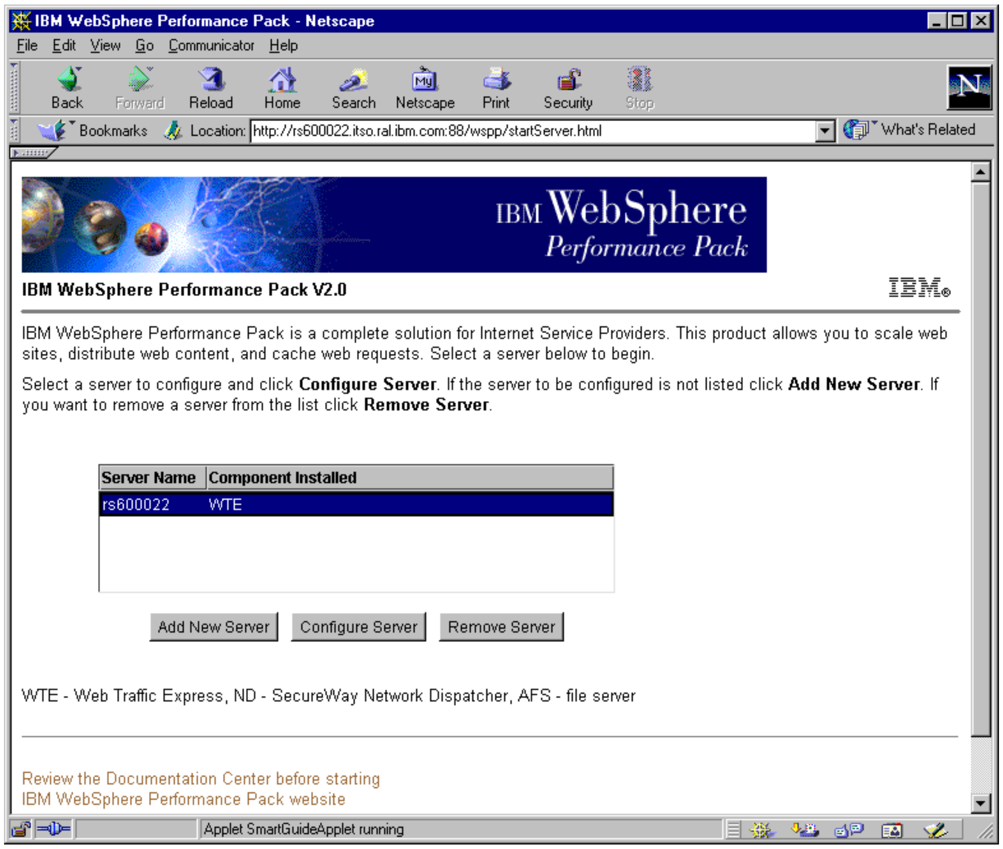
WTE - Web Traffic Express, ND - SecureWay Network Dispatcher, AFS - file server

Review the Documentation Center before starting
IBM WebSphere Performance Pack website

Figure 300. Server Addition Request

As you can see from the figure above, in this example, the Common Configuration server machine (see the host name in the URL) is the same as the WebSphere Performance Pack server machine (see the host name in the Server Addition Request field). However, there is no need for this to happen. The WebSphere Performance Pack server machines that a Common Configuration server can administer can be either remote or local.

After clicking the **Next** button, we saw a refreshed version of the Common Configuration main page, showing the name of the WebSphere Performance Pack server machine. The Common Configuration utility can immediately detect the WebSphere Performance Pack components that are installed on a WebSphere Performance Pack server and that, therefore, are subject to be configured through the Common Configuration utility. The following figure shows how the Common Configuration utility immediately detected the presence of WTE in the WebSphere Performance Pack server machine whose host name has been added in the Server Addition Request field:


IBM WebSphere Performance Pack - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://rs600022.itso.ral.ibm.com:88/wspp/startServer.html
What's Related
IBM WebSphere Performance Pack

IBM WebSphere Performance Pack V2.0

IBM WebSphere Performance Pack is a complete solution for Internet Service Providers. This product allows you to scale web sites, distribute web content, and cache web requests. Select a server below to begin.

Select a server to configure and click **Configure Server**. If the server to be configured is not listed click **Add New Server**. If you want to remove a server from the list click **Remove Server**.

Server Name	Component Installed
rs600022	WTE

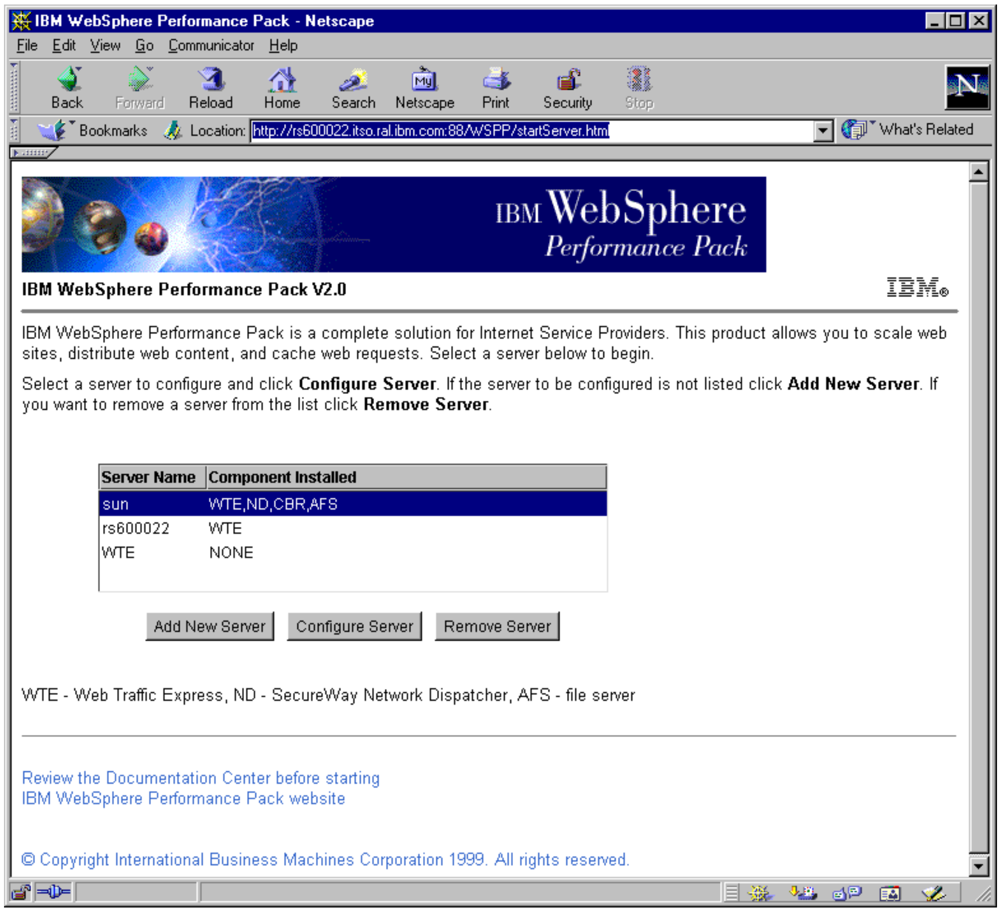
WTE - Web Traffic Express, ND - SecureWay Network Dispatcher, AFS - file server

Review the Documentation Center before starting
IBM WebSphere Performance Pack website

Applet SmartGuideApplet running

Figure 301. One Performance Pack Server Added to the Common Configuration Page

Multiple WebSphere Performance Pack server machines (the local machine as well as remote machines) can be added to the list. The Common Configuration server can automatically detect the WebSphere Performance Pack components that are installed on all the WebSphere Performance Pack server machines, so that these can be configured through the Common Configuration utility. The following figure shows an example of this:


IBM WebSphere Performance Pack - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://rs600022.itso.ral.ibm.com:88/WSPP/startServer.html
What's Related
IBM WebSphere Performance Pack
IBM

IBM WebSphere Performance Pack V2.0

IBM WebSphere Performance Pack is a complete solution for Internet Service Providers. This product allows you to scale web sites, distribute web content, and cache web requests. Select a server below to begin.

Select a server to configure and click **Configure Server**. If the server to be configured is not listed click **Add New Server**. If you want to remove a server from the list click **Remove Server**.

Server Name	Component Installed
sun	WTE,ND,CBR,AFS
rs600022	WTE
WTE	NONE

WTE - Web Traffic Express, ND - SecureWay Network Dispatcher, AFS - file server

Review the Documentation Center before starting
IBM WebSphere Performance Pack website

© Copyright International Business Machines Corporation 1999. All rights reserved.

Figure 302. Common Configuration with Multiple WebSphere Performance Pack Servers

With a server selected, click **Configure Server** to begin the configuration. Once activated, the Common Configuration does not differ too much from the graphical user interface (GUI) tools that can be used to configure AFS, ND and WTE respectively. The advantage of the Common Configuration utility is that it allows centralized, remote, and secure administration of entire WebSphere Performance Pack sites. Only authenticated users can administer a WebSphere Performance Pack server through a Common Configuration server.

Appendix A. ADV_sample Custom Advisor

This appendix shows the code of the HTTP sample advisor ADV_sample.java, shipped with the IBM SecureWay Network Dispatcher V2.1, the Load Balancing component of IBM WebSphere Performance Pack Version 2.

This sample code is located in the directory *installbase*/lib/CustomAdvisor/ADV_sample.java, where *installbase* varies by component (Dispatcher or CBR) and operating system. See Table 1 on page 69 for a list of the *installbase* locations.

The code is shown in the following figures:

```
/**
 * ADV_sample:  The Network Dispatcher HTTP advisor
 *
 *
 * This class defines a sample custom advisor for Network Dispatcher.  Like all
 * advisors, this custom advisor extends the function of the advisor base, called ADV_Base.
 * It is the advisor base that actually performs most of the advisor's functions,
 * such as reporting  loads back to the Network Dispatcher for use in the
 * Network Dispatcher's weight algorithm.  The advisor base also performs socket connect
 * and close operations and provides send and receive
 * methods for use by the advisor.  The advisor itself is used only for
 * sending and receiving data to and from the port on the server being advised.
 * The TCP methods within the advisor base
 * are timed to calculate the load.  A flag within the constructor in the ADV_base
 * overwrites the existing load with the new load returned from the advisor if desired.
 *
 * Note: Based on a value set in the constructor,  the advisor base supplies
 * the load to the weight algorithm at specified intervals.  If the actual
 * advisor has not completed so that it can return a valid load, the advisor base uses
 * the previous load.
 *
 * NAMING
 *
 * The naming convention is as follows:
 *
 *  - The file must be located in the Network Dispatcher base directory.  The defaults for this
 *     directory vary by operating system:
 *
 *    - NT  - \Program Files\nd\dispatcher
 *    - AIX - /usr/lpp/nd/dispatcher
 *    - Solaris - /opt/nd/dispatcher
 *      within the subdirectory of lib\CustomAdvisors.
 *
 * - The Advisor name must be preceded with "ADV_".  The advisor can
 *    be started with only the name, however; for instance, the "ADV_sample"
 *    advisor can be started with "sample".
 *
 * - The advisor name must be in lowercase.
 *
 *  With these rules in mind, therefore, this sample is referred to as:
 *
 *              <base directory>/lib/CustomAdvisors/ADV_sample.class.
 *
 *
 * Advisors, as with the rest of Network Dispatcher, must be compiled with Java 1.1.5.
 * To ensure access to Network Dispatcher classes, make sure that the ibmnd.jar
 * file (located in the lib subdirectory of the base directory) is included in the system's
CLASSPATH.
 *
 *
 * Methods provided by ADV_Base:
 *
 * - ADV_Base (Constructor):
 *
```

Figure 303. (Part 1 of 5). ADV_Sample.java

```
 *   - Parms
 *     - String sName = Name of the advisor
 *     - String sVersion = Version of the advisor
 *     - int iDefaultPort = Default port number to advise on
 *     - int iInterval = Interval on which to advise on the servers
 *     - String sDefaultLogFileName = Unused.  Must be passed in as "".
 *     - boolean replace = True - replace the load value being calculated by the advisor base
 *                         False - add to the load value being calculated by the advisor base
 *   - Return
 *     - Constructors do not have return values.
 *
 * Because the advisor base is thread based, it has several other methods available
 * for use by an advisor.  These methods can be referenced using the CALLER parameter passed in
 * getLoad().
 *
 * These methods are as follows:
 *
 * - send - Send a packet of information on the established socket connection to the server on
 *          the specified port.
 *   - Parms
 *     - String sDataString - The data to be sent is sent in the form of a string
 *   - Return
 *     - int RC - Whether the data was sucessfully sent or not: zero indicates data was
 *                sent; a negative integer indicates an error.
 *
 * - receive - Receive information from the socket connection.
 *   - Parms
 *     - StringBuffer sbDataBuffer - The data received during the receive call
 *   - Return
 *     - int RC - Whether the data was successfully received or not; zero indicates data was
 *                sent; a negative integer indicates an error.
 *
 * If the function provided by the advisor base is
 * not sufficient, you can create the appropriate function within the advisor and
 * the methods provided by the advisor base will then be ignored.
 *
 * An important question regarding
 * the load returned is whether to apply it to the load being generated
 * within the advisor base, or to replace it; there are valid instances of both situations.
 *
 * This sample is essentially the Network Dispatcher HTTP advisor.  It functions very simply:
 * a send request--an http head request--is issued.  Once a response is received, the
 * getLoad method terminates, flagging the advisor base to stop timing the request.  The method
 * is then complete.  The information returned is not parsed; the load is based on the time
required
 * to perform the send and receive operations.
 */

package CustomAdvisors;
import com.ibm.internet.nd.advisors.*;

public class ADV_sample extends ADV_Base implements ADV_MethodInterface
{
  String COPYRIGHT = "(C) Copyright IBM Corporation 1997, All Rights Reserved.\n";
```

Figure 304. (Part 2 of 5). ADV_Sample.java

```
  static final String  ADV_NAME              = "Sample";
  static final int     ADV_DEF_ADV_ON_PORT   = 80;
  static final int     ADV_DEF_INTERVAL      = 7;

  // Note: Most server protocols require a carriage return ("\r") and line feed ("\n")
  //       at the end of messages.  If so, include them in your string here.
  static final String  ADV_SEND_REQUEST      =
    "HEAD / HTTP/1.0\r\nAccept: */*\r\nUser-Agent: " +
    "IBM_Network_Dispatcher_HTTP_Advisor\r\n\r\n";

  /**
   * Constructor.
   *
   * Parms:  None; but the constructor for ADV_Base has several parameters that must be passed to
it.
   *
   */
  public ADV_sample()
  {
    super( ADV_NAME,
   "2.0.0.0-03.27.98",
           ADV_DEF_ADV_ON_PORT,
           ADV_DEF_INTERVAL,
           "",     // not used
           false);
    super.setAdvisor( this );
  }


  /**
   * ADV_AdvisorInitialize
   *
   * Any Advisor-specific initialization that must take place after the advisor base is started.
   * This method is called only once and is typically not used.
   */
  public void ADV_AdvisorInitialize()
  {
    return;
  }


  /**
   * getLoad()
   *
   * This method is called by the advisor base to complete the advisor's operation, based
   * on details specific to the protocol.  In this sample advisor, only a single send and receive
are
   * necessary; if more complex logic is necessary, multiple sends and receives
   * can be issued.  For example, a response might be received and parsed.  Based on the
information
   * learned thereby, another send and receive could be issued.
   *
   * Parameters:
   *
```

Figure 305. (Part 3 of 5). ADV_Sample.java

```
  * - iConnectTime - The current load as it refers to the length of time it took to
  *                  complete the connection to the server through the specified port.
  *
  * - caller - A reference to the advisor base class where the Network Dispatcher-supplied
  *            methods are to perform simple TCP requests, mainly send and receive.
  *
  * Results:
  *
  * - The load - A value, expressed in milliseconds, that can either be added to the existing
  * load, or that can replace the existing load, as determined by the constructor's "replace"
  * flag.
  *
  *   The larger the load, the longer it took the server to respond; therefore, the higher
  *   the weight will be within Network Dispatcher regarding load balancing.
  *
  *   If the value is negative, an error is assumed.  An error from an advisor indicates that
  *   the server the advisor is trying to reach is not accessible and has been identified as being
  *   down.
  *   Network Dispatcher will not attempt to load balance to a server that is down.
  *   Network Dispatcher will resume load balancing to the server when a positive value is
  *   received.
  *
  *   A value of zero is typically not returned; Network Dispatcher handles a load of zero in a
  *   special way.
  *   Zero is assumed to indicate an unknown status, and Network Dispatcher gives the server a
  *   high weight in
  *   response.
  */
 public int getLoad(int iConnectTime, ADV_Thread caller)
 {
  int iRc;
  int iLoad = ADV_HOST_INACCESSIBLE;  // -1

  // Send tcp request
  iRc = caller.send(ADV_SEND_REQUEST);
  if (iRc >= 0)
  {
     // Perform a receive
     StringBuffer sbReceiveData = new StringBuffer("");
     iRc = caller.receive(sbReceiveData);

     // If the receive is successful, a load of zero is returned.  This is because the "replace"
     // flag is set to false, indicating that the load built within the base advisor is to be
     // used.
     // Since nothing was done with the returned data, additional load is not necessary.

     // Note: it is known that the advisor base load will not be zero, therefore a zero load will
     // not be returned for use in calculating the weight.
     if (iRc >= 0)
     {
        iLoad = 0;
     }
  }
  return iLoad;
```

Figure 306. (Part 4 of 5). ADV_Sample.java

```
  }

} // End - ADV_sample
```

Figure 307. (Part 5 of 5). ADV_Sample.java

Appendix B. SDA_SampleAgent Code

In this appendix, we show the sample code SDA_SampleAgent.java, shipped with the IBM SecureWay Network Dispatcher (ND), the Load Balancing component of IBM WebSphere Performance Pack Version 2.

The sample SDA client code is distributed as a Java source program in this directory *installbase*/lib/SDA, where *installbase* is the Dispatcher component and varies by operating system (see Table 1 on page 69 for a list of *installbase* locations).

As explained in 9.1.5, “Modifying the Sample SDA Client Code” on page 256, this Java program implements a Server Directory Affinity (SDA) client, and is used to create entries in the Dispatcher affinity table to bind a specific client IP address to a specific server IP address in the cluster, so that subsequent requests from the same client will be bound to the same server.

```
//=============================================================================
// SDA_SampleAgent
//
// Used to simulate an intelligent agent which provides affinity information
// to Network Dispatcher's 'Server Directed Affinity' feature.
//
// Three classes are defined herein:
//  SDA_SampleAgent - main program which communicates with ND.
//  SDA_Info - holds affinity information in the format for transfer to ND.
//  SDA_Utils - miscellaneous utility routines to communicate with ND.
// These classes were designed so that you could use SDA_Info and SDA_Utils
// unchanged and intact, while replacing SDA_SampleAgent with your own code.
//
// This program opens a TCP connection with Network Dispatcher.
// It sends affinity records to ND, queries them, and deletes them.
//
//      Message Flows
// --------------------------
//  ND   <- connect -   Agent
//      <- Auth ----
//      <- CIB -----
//      -- Auth --->
//
//      <- Auth ----
//      <- Affin----
//      -- Auth --->
//      -- Resp --->
//
//      <- Auth ----
//      <- Affin----
//      -- Auth --->
//      -- Resp --->
//
//      <- Auth ----
//      <- Affin----
//      -- Auth --->
//      -- Resp --->
//
//         etc...
//
//
// AD 03/14/1998 - for Java 1.1.4 (and above)
//=============================================================================
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.Date;


//-----------------------------------------------------------------------------
// SDA_SampleAgent - A class to implement a simple SDA agent.
//-----------------------------------------------------------------------------
class SDA_SampleAgent
{
```

Figure 308. (Part 1 of 16). SDA_SampleAgent.java

```
//------------------------------------------------------------------------
// Version of this agent.
//------------------------------------------------------------------------
public static String sVersion = "Program SDA_SampleAgent, version 1.0, March 14, 1998.";


//------------------------------------------------------------------------
// Object variables.
//------------------------------------------------------------------------
// Variables used to communicate with ND.
InetAddress      inaNetworkDispatcher;    // The IP address of ND.
int              iNetworkDispatcherPort;  // Port number to connect to ND.
Socket           soNetworkDispatcher;     // Socket to talk to ND.
DataInputStream  disIn = null;            // Input stream from ND.
DataOutputStream dosOut = null;           // Output stream to ND.

// Variables used to define entries in the ND affinity table.
InetAddress      inaSdaCluster;           // Cluster address under test.
int              iSdaPort;                // Port number under test.
InetAddress      inaSdaClient;            // Client address under test.
InetAddress      inaSdaServer;            // Server address under test.


//------------------------------------------------------------------------
// Function - Display the contents of this object (for debug only).
//------------------------------------------------------------------------
public String toString() {
  String sRc = "SDA_SampleAgent object:\n" +
               "-----------------------------------------------\n" +
               "inaNetworkDispatcher ..... " + inaNetworkDispatcher   + "\n" +
               "iNetworkDispatcherPort ... " + iNetworkDispatcherPort + "\n" +
               "soNetworkDispatcher ...... " + soNetworkDispatcher    + "\n" +
               "disIn .................... " + disIn                  + "\n" +
               "dosOut ................... " + dosOut                 + "\n" +
               "inaSdaCluster ............ " + inaSdaCluster          + "\n" +
               "iSdaPort ................. " + iSdaPort               + "\n" +
               "inaSdaClient ............. " + inaSdaClient           + "\n" +
               "inaSdaServer ............. " + inaSdaServer           + "\n" ;
  return sRc;
}


//------------------------------------------------------------------------
// Constructor.
//------------------------------------------------------------------------
public SDA_SampleAgent() throws Exception {

  // Note: These values are samples only.
  // You must modify them to work in your environment.

  // Define the address and port to communicate with ND.
  inaNetworkDispatcher      = InetAddress.getByName("9.37.52.219");
  // inaNetworkDispatcher      = InetAddress.getByName("10.0.0.3");
  iNetworkDispatcherPort    = 10005;
```

Figure 309. (Part 2 of 16). SDA_SampleAgent.java

```
    // Define variables used as entries in the ND affinity table.
    inaSdaCluster             = InetAddress.getByName("9.37.61.44");
    iSdaPort                  = 80;
    inaSdaClient              = InetAddress.getByName("9.37.52.219");
    // inaSdaClient              = InetAddress.getByName("10.0.0.3");
    inaSdaServer              = InetAddress.getByName("9.37.52.114");
  }


  //------------------------------------------------------------------------
  // Function - Open an authorized connection with ND and identify ourself.
  //------------------------------------------------------------------------
  public void openND() throws Exception {

    // Open the socket.
    openSocket();

    // Send an auth string.
    SDA_Utils.sendAuth(dosOut);

    // Send our CIB string.
    SDA_Utils.sendCIB(dosOut);

    // Wait for an auth string.
    SDA_Utils.recvAuth(disIn);
  }


  //------------------------------------------------------------------------
  // Function - Open a socket to ND on its SDA listen port.
  //------------------------------------------------------------------------
  public void openSocket() throws Exception {

    // Tell the user...
    System.out.println( "SDA_Sample> Opening connection with Network Dispatcher:\n" +
                        "            Address ........ " + inaNetworkDispatcher + "\n" +
                        "            Port ........... " + iNetworkDispatcherPort );
    try
    {
      // Open the socket.
      soNetworkDispatcher = new Socket(inaNetworkDispatcher, iNetworkDispatcherPort);

      // "Cast" the socket into streams for use reading and writing data.
      disIn = new DataInputStream( soNetworkDispatcher.getInputStream() );
      dosOut = new DataOutputStream( soNetworkDispatcher.getOutputStream() );

      // Set the socket receive timeout (arbitrarily) in milliseconds.
      soNetworkDispatcher.setSoTimeout(20000);

      // Tell the user...
      System.out.println( "SDA_Sample> Opened connection successfully." );
    }

    catch( Exception e )
```

Figure 310. (Part 3 of 16). SDA_SampleAgent.java

```
    {
      // Tell the user...
      System.out.println( "SDA_Sample> Error: Caught a java exception while opening socket to
         ND:\n" + e );
      System.out.println( "SDA_Sample> Perhaps the ND's hostname is invalid?...");
      System.out.println( "SDA_Sample> Perhaps the ND's SDA listen port is invalid?...");
      throw e;
    }
  }


  //------------------------------------------------------------------------
  // Function - Query the contents of the ND affinity table.
  //------------------------------------------------------------------------
  public void queryTable() throws Exception {

    // Create a java object for use communicating with ND and define the command.
    SDA_Info sdai = new SDA_Info();
    sdai.iMessageVersion = 1;
    sdai.iCommand        = SDA_Info.SDA_CMD_QUERY;
    sdai.iResponse       = SDA_Info.SDA_RSP_SUCCESS;
    sdai.iClusterAddr    = SDA_Utils.inaToInt(inaSdaCluster);
    sdai.iPort           = iSdaPort;
    sdai.iNumAffinities  = 0;

    // Tell the user...
    System.out.println("SDA_Sample> About to query the contents of the ND affinity table:");
    System.out.println(sdai);

    try {
      // Send the command to ND and get a response.
      SDA_Utils.talkToNd(sdai,disIn,dosOut);
    }
    catch(Exception e) {
      throw e;
    }

    // Bomb on error.
    if (SDA_Info.SDA_RSP_SUCCESS != sdai.iResponse) {
      throw new Exception("Error from ND querying the contents of the ND affinity table...");
    }
  }


  //------------------------------------------------------------------------
  // Function - Query the contents of the ND affinity table for one client.
  //------------------------------------------------------------------------
  public void queryOneClient() throws Exception {

    // Create a java object for use communicating with ND and define the command.
    SDA_Info sdai = new SDA_Info();
    sdai.iMessageVersion = 1;
    sdai.iCommand        = SDA_Info.SDA_CMD_QUERY;
    sdai.iResponse       = SDA_Info.SDA_RSP_SUCCESS;
```

Figure 311. (Part 4 of 16). SDA_SampleAgent.java

```
    sdai.iClusterAddr    = SDA_Utils.inaToInt(inaSdaCluster);
    sdai.iPort           = iSdaPort;
    sdai.iNumAffinities  = 1;

    sdai.aiClientAddr[0] = SDA_Utils.inaToInt(inaSdaClient);
    sdai.aiServerAddr[0] = 0;
    sdai.aiResponse[0]   = SDA_Info.SDA_RSP_SUCCESS;


    // Tell the user...
    System.out.println("SDA_Sample> About to query the table for one client:");
    System.out.println(sdai);

    try {
      // Send the command to ND and get a response.
      SDA_Utils.talkToNd(sdai,disIn,dosOut);
    }
    catch(Exception e) {
      throw e;
    }

    // Bomb on error.
    if (SDA_Info.SDA_RSP_SUCCESS != sdai.iResponse) {
      throw new Exception("Error from ND querying one client in the ND affinity table...");
    }
  }


  //------------------------------------------------------------------------
  // Function - Query the contents of the ND affinity table for one server.
  //------------------------------------------------------------------------
  public void queryOneServer() throws Exception {

    // Create a java object for use communicating with ND and define the command.
    SDA_Info sdai = new SDA_Info();
    sdai.iMessageVersion = 1;
    sdai.iCommand        = SDA_Info.SDA_CMD_QUERY;
    sdai.iResponse       = SDA_Info.SDA_RSP_SUCCESS;
    sdai.iClusterAddr    = SDA_Utils.inaToInt(inaSdaCluster);
    sdai.iPort           = iSdaPort;
    sdai.iNumAffinities  = 1;

    sdai.aiClientAddr[0] = 0;
    sdai.aiServerAddr[0] = SDA_Utils.inaToInt(inaSdaServer);
    sdai.aiResponse[0]   = SDA_Info.SDA_RSP_SUCCESS;


    // Tell the user...
    System.out.println("SDA_Sample> About to query the table for one server:");
    System.out.println(sdai);

    try {
      // Send the command to ND and get a response.
      SDA_Utils.talkToNd(sdai,disIn,dosOut);
```

Figure 312. (Part 5 of 16). SDA_SampleAgent.java

```
    }
    catch(Exception e) {
      throw e;
    }

    // Bomb on error.
    if (SDA_Info.SDA_RSP_SUCCESS != sdai.iResponse) {
      throw new Exception("Error from ND querying one server in the ND affinity table...");
    }
  }


  //------------------------------------------------------------------------
  // Function - Add a record to the ND affinity table.
  //------------------------------------------------------------------------
  public void addRecord() throws Exception {

    // Create a java object for use communicating with ND and define the command.
    SDA_Info sdai = new SDA_Info();
    sdai.iMessageVersion = 1;
    sdai.iCommand        = SDA_Info.SDA_CMD_ADD;
    sdai.iResponse       = SDA_Info.SDA_RSP_SUCCESS;
    sdai.iClusterAddr    = SDA_Utils.inaToInt(inaSdaCluster);
    sdai.iPort           = iSdaPort;
    sdai.iNumAffinities  = 1;

    sdai.aiClientAddr[0] = SDA_Utils.inaToInt(inaSdaClient);
    sdai.aiServerAddr[0] = SDA_Utils.inaToInt(inaSdaServer);
    sdai.aiResponse[0]   = SDA_Info.SDA_RSP_SUCCESS;

    // Tell the user...
    System.out.println("SDA_Sample> About to add a record to the ND affinity table:");
    System.out.println(sdai);

    try {
      // Send the command to ND and get a response.
      SDA_Utils.talkToNd(sdai,disIn,dosOut);
    }
    catch(Exception e) {
      throw e;
    }

    // Bomb on error.
    if ((SDA_Info.SDA_RSP_SUCCESS != sdai.iResponse) ||
        (1 > sdai.iNumAffinities) ||
        (SDA_Info.SDA_RSP_SUCCESS != sdai.aiResponse[0])) {
      throw new Exception("Error from ND adding record to the ND affinity table...");
    }
  }


  //------------------------------------------------------------------------
  // Function - Delete a record from the ND affinity table.
  //------------------------------------------------------------------------
```

Table 29. (Part 6 of 16). SDA_SampleAgent.java

```
  public void deleteRecord() throws Exception {

    // Create a java object for use communicating with ND and define the command.
    SDA_Info sdai = new SDA_Info();
    sdai.iMessageVersion = 1;
    sdai.iCommand        = SDA_Info.SDA_CMD_DELETE;
    sdai.iResponse       = SDA_Info.SDA_RSP_SUCCESS;
    sdai.iClusterAddr    = SDA_Utils.inaToInt(inaSdaCluster);
    sdai.iPort           = iSdaPort;
    sdai.iNumAffinities  = 1;

    sdai.aiClientAddr[0] = SDA_Utils.inaToInt(inaSdaClient);
    sdai.aiServerAddr[0] = SDA_Utils.inaToInt(inaSdaServer);
    sdai.aiResponse[0]   = SDA_Info.SDA_RSP_SUCCESS;

    // Tell the user...
    System.out.println("SDA_Sample> About to delete a record from the ND affinity table:");
    System.out.println(sdai);

    try {
      // Send the command to ND and get a response.
      SDA_Utils.talkToNd(sdai,disIn,dosOut);
    }
    catch(Exception e) {
      throw e;
    }

    // Bomb on error.
    if ((SDA_Info.SDA_RSP_SUCCESS != sdai.iResponse) ||
        (1 > sdai.iNumAffinities) ||
        (SDA_Info.SDA_RSP_SUCCESS != sdai.aiResponse[0])) {
      throw new Exception("Error from ND deleting record from the ND affinity table...");
    }
  }


  //-----------------------------------------------------------------------
  // Function - Delete all records from the ND affinity table.
  //-----------------------------------------------------------------------
  public void deleteAllRecords() throws Exception {

    // Create a java object for use communicating with ND and define the command.
    SDA_Info sdai = new SDA_Info();
    sdai.iMessageVersion = 1;
    sdai.iCommand        = SDA_Info.SDA_CMD_DELETE_ALL;
    sdai.iResponse       = SDA_Info.SDA_RSP_SUCCESS;
    sdai.iClusterAddr    = SDA_Utils.inaToInt(inaSdaCluster);
    sdai.iPort           = iSdaPort;
    sdai.iNumAffinities  = 0;

    // Tell the user...
    System.out.println("SDA_Sample> About to delete all records from the ND affinity table:");
    System.out.println(sdai);
```

Figure 313. (Part 7 of 16). SDA_SampleAgent.java

```
    try {
      // Send the command to ND and get a response.
      SDA_Utils.talkToNd(sdai,disIn,dosOut);
    }
    catch(Exception e) {
      throw e;
    }

    // Bomb on error.
    if (SDA_Info.SDA_RSP_SUCCESS != sdai.iResponse) {
      throw new Exception("Error from ND deleting all records from the ND affinity table...");
    }
  }


  //-----------------------------------------------------------------------------
  // Function - Main.
  //-----------------------------------------------------------------------------
  public static void main( String[] sArgs )
  {
    SDA_SampleAgent sa = null;

    // Tell the user our version.
    System.out.println("SDA_Sample> " + sVersion);

    try {
      // Define the addresses and ports specifically for this sample program.
      sa = new SDA_SampleAgent();

      // Open a connection with ND and identify ourself.
      sa.openND();

      // Query the contents of the ND affinity table.
      sa.queryTable();

      // Delete all affinity records.
      sa.deleteAllRecords();

      // Query the contents of the ND affinity table.
      sa.queryTable();

      // Add a new affinity record.
      sa.addRecord();

      // Query the contents of the ND affinity table.
      sa.queryTable();

      // Query one client in the ND affinity table.
      sa.queryOneClient();

      // Query one server in the ND affinity table.
      sa.queryOneServer();

      // Delete the new affinity record.
```

Figure 314. (Part 8 of 16). SDA_SampleAgent.java

```
      sa.deleteRecord();

      // Query the contents of the ND affinity table.
      sa.queryTable();

    }

    // Error handling.
    catch(Exception e) {
      // Tell the user.
      System.out.println("SDA_Sample> Caught exception in routine main: " + e);
    }

    // Close socket.
    finally {
      if (null != sa.soNetworkDispatcher) {
        System.out.println("SDA_Sample> Closing socket in routine main.");
        try { sa.soNetworkDispatcher.close(); }  catch(Exception x){;}
      }
    }
  }


}  // End - class SDA_SampleAgent.





//------------------------------------------------------------------------------
// SDA_Utils - Miscellaneous utility routines to communicate with ND.
//------------------------------------------------------------------------------
class SDA_Utils
{
  //----------------------------------------------------------------------------
  // Definition of the auth string.
  //----------------------------------------------------------------------------
  public static final byte[] abAuth = {
    // "MANAGER Copyright (C) International Business Machines 1996"
    0x4d, 0x41, 0x4e, 0x41, 0x47, 0x45, 0x52, 0x20,
    0x43, 0x6f, 0x70, 0x79, 0x72, 0x69, 0x67, 0x68,
    0x74, 0x20, 0x28, 0x43, 0x29, 0x20, 0x49, 0x6e,
    0x74, 0x65, 0x72, 0x6e, 0x61, 0x74, 0x69, 0x6f,
    0x6e, 0x61, 0x6c, 0x20, 0x42, 0x75, 0x73, 0x69,
    0x6e, 0x65, 0x73, 0x73, 0x20, 0x4d, 0x61, 0x63,
    0x68, 0x69, 0x6e, 0x65, 0x73, 0x20, 0x31, 0x39,
    0x39, 0x36
  };


  //----------------------------------------------------------------------------
  // Definition of the ConnectionInitBlock (CIB) for this agent.
  //----------------------------------------------------------------------------
  public static final byte[] abCIB = {
```

Figure 315. (Part 9 of 16). SDA_SampleAgent.java

```
    // Interface protocol version number "01.00.00.00"
    0x30, 0x31, 0x2e, 0x30, 0x30, 0x2e, 0x30, 0x30, 0x2e, 0x30, 0x30, 0x00,
    // Name of this agent "SDA_Sample"
    0x53, 0x44, 0x41, 0x5F, 0x53, 0x61, 0x6d, 0x70, 0x6c, 0x65,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    // Reserved for future use. Must be zero.
    0x00, 0x00, 0x00, 0x00
  };


  //-----------------------------------------------------------------------
  // Function - Send an "Auth" string to ND.
  //-----------------------------------------------------------------------
  public static void sendAuth(DataOutputStream dosOut) throws Exception {

    // Tell the user...
    System.out.println( "SDA_Sample> About to send an auth string to ND...");

    try {
      // Send the "auth" string as an array of bytes.
      dosOut.write( abAuth, 0, abAuth.length );
      dosOut.flush();
    }

    catch( Exception e ) {
      // Tell the user...
      System.out.println( "SDA_Sample> Error 2 sending auth string to ND.");
      throw e;
    }
  }


  //-----------------------------------------------------------------------
  // Function - Receive an "Auth" string from ND.
  // This blocks until data is received or the socket timeout expires.
  //-----------------------------------------------------------------------
  public static void recvAuth(DataInputStream disIn) throws Exception {
    byte[] ab = new byte[abAuth.length];
    int iBytesRead = 0;

    // Tell the user...
    System.out.println( "SDA_Sample> Waiting for an auth string from ND...");

    try {
      // Read into temporary array ab.
      iBytesRead = disIn.read(ab);
```

Figure 316. (Part 10 of 16). SDA_SampleAgent.java

```
      // Verify we received an auth.  Check the number of bytes.
      if (abAuth.length == iBytesRead) {
        // Check each byte.
        for (int i=0; i<iBytesRead; i++) {
          if (ab[i] != abAuth[i]) {
            // Tell the user.
            System.out.println( "SDA_Sample> Error in content of auth string received from ND.");
            throw new Exception("invalid content auth");
          }
        }
      }
      else {
        // Tell the user.
        System.out.println( "SDA_Sample> Error in length of auth string received from ND.");
        throw new Exception("invalid length auth");
      }

      // Tell the user...
      System.out.println( "SDA_Sample> Successfully received an auth string from ND...");
    }

    catch( Exception e ) {
      System.out.println( "SDA_Sample> Error waiting for auth string from ND.");
      throw e;
    }
  }


  //-------------------------------------------------------------------------
  // Function - Send a "CIB" (ConnectionInitBlock) string to ND.
  //-------------------------------------------------------------------------
  public static void sendCIB(DataOutputStream dosOut) throws Exception {

    // Tell the user...
    System.out.println( "SDA_Sample> About to send a CIB string to ND...");

    try {
      // Send the "CIB" string as an array of bytes.
      dosOut.write( abCIB, 0, abCIB.length );
      dosOut.flush();
    }

    catch( Exception e ) {
      // Tell the user...
      System.out.println( "SDA_Sample> Error sending CIB string to ND.");
      throw e;
    }
  }


  //-------------------------------------------------------------------------
  // Function - Send an SDA_Info command to ND and get a response.
  //-------------------------------------------------------------------------
  public static void talkToNd(SDA_Info sdai,
```

Figure 317. (Part 11 of 16). SDA_SampleAgent.java

```
                               DataInputStream disIn,
                               DataOutputStream dosOut) throws Exception {

    try {
      // Send an Authorization string to ND.
      SDA_Utils.sendAuth(dosOut);

      // Send the affinity information command.
      sdai.sendCommand(dosOut);
    }
    catch(Exception e) {
      System.out.println("SDA_Sample> Error sending command to ND.");
      throw e;
    }

    try {
      // Wait for an auth string.
      SDA_Utils.recvAuth(disIn);

      // Receive the response.
      sdai.recvResponse(disIn);

      // Tell the user.
      System.out.println("SDA_Sample> Received response from ND:");
      System.out.println(sdai);
    }
    catch(Exception e) {
      System.out.println("SDA_Sample> Error receiving response from ND.");
      throw e;
    }
  }


  //------------------------------------------------------------------------
  // Function - Convert a java InetAddress to an int.
  //------------------------------------------------------------------------
  public static int inaToInt(InetAddress ina) {
    int i;
    int iRc;

    // Extract the address into an array of bytes.
    byte[] ab = ina.getAddress();

    // Move the bytes into an int.
    i = (ab[0] << 24) & 0xFF000000;  iRc  = i;
    i = (ab[1] << 16) & 0x00FF0000;  iRc += i;
    i = (ab[2] <<  8) & 0x0000FF00;  iRc += i;
    i = (ab[3]      ) & 0x000000FF;  iRc += i;
    return iRc;
  }


}  // End - class SDA_Utils.
```

Figure 318. (Part 12 of 16). SDA_SampleAgent.java

```
//--------------------------------------------------------------------------
// SDA_Info - A class to hold affinity information for transfer to and from ND
//--------------------------------------------------------------------------
class SDA_Info extends Thread    // Extends thread only to use the sleep method.
{
  //------------------------------------------------------------------------
  // Command and response constants.
  //------------------------------------------------------------------------
  public static final int SDA_CMD_ADD                        = 1;
  public static final int SDA_CMD_DELETE                     = 2;
  public static final int SDA_CMD_DELETE_ALL                 = 3;
  public static final int SDA_CMD_QUERY                      = 4;

  public static final int SDA_RSP_SUCCESS                    = 0;
  public static final int SDA_RSP_NO_SUCH_SERVER             = -11;
  public static final int SDA_RSP_NO_SUCH_RECORD             = -26;
  public static final int SDA_RSP_NO_MEMORY_FOR_RECORD       = -27;
  public static final int SDA_RSP_RECORD_ALREADY_EXISTS      = -28;
  public static final int SDA_RSP_TOO_MANY_RECORDS           = -101;
  public static final int SDA_RSP_NO_MEMORY_FOR_REQUEST      = -102;
  public static final int SDA_RSP_NO_SUCH_CLUSTER            = -103;
  public static final int SDA_RSP_NO_SUCH_PORT               = -104;
  public static final int SDA_RSP_PORT_NOT_STICKY            = -105;
  public static final int SDA_RSP_IS_NULL                    = -106;
  public static final int SDA_RSP_NO_SUCH_COMMAND            = -107;
  public static final int SDA_RSP_STICKYTIME_INVALID_FOR_SDA = -108;
  public static final int SDA_RSP_EXECUTOR_NOT_RUNNING       = -109;


  //------------------------------------------------------------------------
  // Misc Constants.
  //------------------------------------------------------------------------
  public static final int SDA_MAX_AFFINITIES = 3000;
  public static final int SDA_MESSAGE_VERSION = 1;


  //------------------------------------------------------------------------
  // Object Variables.
  //------------------------------------------------------------------------
  public int iMessageVersion;
  public int iCommand;
  public int iResponse;
  public int iClusterAddr;
  public int iPort;
  public int iNumAffinities;
  public int[] aiClientAddr  = new int[SDA_MAX_AFFINITIES];
  public int[] aiServerAddr  = new int[SDA_MAX_AFFINITIES];
  public int[] aiResponse    = new int[SDA_MAX_AFFINITIES];


  //------------------------------------------------------------------------
  // Constructor.
  //------------------------------------------------------------------------
  public SDA_Info() {
```

Figure 319. (Part 13 of 16). SDA_SampleAgent.java

```
    iMessageVersion = SDA_MESSAGE_VERSION;
    iResponse = SDA_RSP_SUCCESS;
  }


  //-----------------------------------------------------------------------
  // Function - Send an affinity message to an output stream.
  //-----------------------------------------------------------------------
  public void sendCommand(DataOutputStream dosOut) throws IOException
  {
    // Give ND a little time to do some other work.
    try { sleep(250); }  catch(Exception e) {;}

    // Write single values.
    dosOut.writeInt(iMessageVersion);
    dosOut.writeInt(iCommand);
    dosOut.writeInt(iResponse);
    dosOut.writeInt(iClusterAddr);
    dosOut.writeInt(iPort);
    dosOut.writeInt(iNumAffinities);

    // Write all Affinities.
    for (int i=0; i<iNumAffinities; i++) {
      // Get the values.
      dosOut.writeInt(aiResponse[i]);
      dosOut.writeInt(aiClientAddr[i]);
      dosOut.writeInt(aiServerAddr[i]);
    }
  }


  //-----------------------------------------------------------------------
  // Function - Read an affinity message from an input stream.
  //-----------------------------------------------------------------------
  public void recvResponse(DataInputStream disIn) throws IOException
  {
    // Read the message version.
    iMessageVersion = disIn.readInt();

    // Check the message version.
    if (SDA_MESSAGE_VERSION != iMessageVersion) {
      // We do not know how to interpret the received data.
      throw new IOException();
    }

    // Read single values from the message.
    iCommand        = disIn.readInt();
    iResponse       = disIn.readInt();
    iClusterAddr    = disIn.readInt();
    iPort           = disIn.readInt();
    iNumAffinities  = disIn.readInt();

    // Read all Affinities.
    for (int i=0; i<iNumAffinities; i++) {
```
Figure 320. (Part 14 of 16). SDA_SampleAgent.java

```
      // Get the values.
      aiResponse[i]   = disIn.readInt();
      aiClientAddr[i] = disIn.readInt();
      aiServerAddr[i] = disIn.readInt();
    }
  }


  //-------------------------------------------------------------------------
  // Function - Display the contents of this object (for debug only).
  //-------------------------------------------------------------------------
  public String toString() {
    String sRc = "SDA_Info object:\n" +
                 "----------------------------------------------\n" +
                 "MessageVersion .... " + iMessageVersion + "\n" +
                 "Command ........... " + iCommand + " (" + printSdaCommand(iCommand) + ")\n" +
                 "Response .......... " + iResponse + " (" + printSdaResponse(iResponse) + ")\n" +
                 "ClusterAddr ....... " + printAddress(iClusterAddr) + "\n" +
                 "Port .............. " + iPort + "\n" +
                 "NumAffinities ..... " + iNumAffinities + "\n";
    for (int i=0; i<iNumAffinities; i++) {
      sRc += "Record " + i + ":\n" +
          "  Response ........ " + aiResponse[i] + " (" + printSdaResponse(aiResponse[i]) + ")\n" +
           "  ClientAddr ...... " + printAddress(aiClientAddr[i]) + "\n" +
           "  ServerAddr ...... " + printAddress(aiServerAddr[i]) + "\n" ;
    }
    return sRc;
  }


  //-------------------------------------------------------------------------
  // Function - convert address from an int to a dotted "a.b.c.d" string.
  //-------------------------------------------------------------------------
  public static String printAddress(int i) {
    String sRc = "";
    sRc += (( i >>> 24 ) & 0xFF ) + "." +
           (( i >>> 16 ) & 0xFF ) + "." +
           (( i >>>  8 ) & 0xFF ) + "." +
           (( i >>>  0 ) & 0xFF );
    return sRc;
  }

  //-------------------------------------------------------------------------
  // Function - Convert a command number into a string.
  //-------------------------------------------------------------------------
  public static String printSdaCommand(int i) {
    String sRc;
    switch (i) {
      case SDA_CMD_ADD        : sRc =  "SDA_CMD_ADD"           ; break;
      case SDA_CMD_DELETE     : sRc =  "SDA_CMD_DELETE"        ; break;
      case SDA_CMD_DELETE_ALL : sRc =  "SDA_CMD_DELETE_ALL"    ; break;
      case SDA_CMD_QUERY      : sRc =  "SDA_CMD_QUERY"         ; break;
      default                 : sRc =  "Unknown"               ; break;
```

Figure 321. (Part 15 of 16). SDA_SampleAgent.java

```
    }
    return sRc;
  }



  //-----------------------------------------------------------------------
  // Function - Convert a response number into a string.
  //-----------------------------------------------------------------------
  public static String printSdaResponse(int i) {
    String sRc;
    switch (i) {
      case SDA_RSP_SUCCESS                  : sRc = "SDA_RSP_SUCCESS"                  ; break;
      case SDA_RSP_NO_SUCH_SERVER           : sRc = "SDA_RSP_NO_SUCH_SERVER"           ; break;
      case SDA_RSP_NO_SUCH_RECORD           : sRc = "SDA_RSP_NO_SUCH_RECORD"           ; break;
      case SDA_RSP_NO_MEMORY_FOR_RECORD     : sRc = "SDA_RSP_NO_MEMORY_FOR_RECORD"     ; break;
      case SDA_RSP_RECORD_ALREADY_EXISTS    : sRc = "SDA_RSP_RECORD_ALREADY_EXISTS"    ; break;
      case SDA_RSP_TOO_MANY_RECORDS         : sRc = "SDA_RSP_TOO_MANY_RECORDS"         ; break;
      case SDA_RSP_NO_MEMORY_FOR_REQUEST    : sRc = "SDA_RSP_NO_MEMORY_FOR_REQUEST"    ; break;
      case SDA_RSP_NO_SUCH_CLUSTER          : sRc = "SDA_RSP_NO_SUCH_CLUSTER"          ; break;
      case SDA_RSP_NO_SUCH_PORT             : sRc = "SDA_RSP_NO_SUCH_PORT"             ; break;
      case SDA_RSP_PORT_NOT_STICKY          : sRc = "SDA_RSP_PORT_NOT_STICKY"          ; break;
      case SDA_RSP_IS_NULL                  : sRc = "SDA_RSP_IS_NULL"                  ; break;
      case SDA_RSP_NO_SUCH_COMMAND          : sRc = "SDA_RSP_NO_SUCH_COMMAND"          ; break;
      case SDA_RSP_STICKYTIME_INVALID_FOR_SDA : sRc = "SDA_RSP_STICKYTIME_INVALID_FOR_SDA"; break;
      case SDA_RSP_EXECUTOR_NOT_RUNNING     : sRc = "SDA_RSP_EXECUTOR_NOT_RUNNING"     ; break;
      default                               : sRc = "Unknown"                          ; break;
    }
    return sRc;
  }

}  // End - class SDA_Info.
```

Figure 322. (Part 16 of 16). SDA_SampleAgent.java

Appendix C. LOG_SampleReader Program

This appendix shows the code of the program LOG_SampleReader.java. After the default installation of IBM SecureWay Network Dispatcher (ND) V2.1, the Load Balancing component of IBM WebSphere Performance Pack Version 2, the file LOG_SampleReader.java is located in the directory *installbase*/lib/BinaryLog, where *installbase* varies by component (Dispatcher or CBR) and by operating system. See Table 1 on page 69 for a list of the *installbase* locations.

Once compiled to a class file, this program can be used to examine binary logging and statistics, as explained in 13.2.1, “Using the LOG_SampleReader Sample Java Program” on page 322.

The code of this file is shown in the following figures:

```
// ==========================================================================
// LOG_SampleReader.java
//
// (C) COPYRIGHT International Business Machines Corp. 1999
// All Rights Reserved
//
// US Government Users Restricted Rights - Use, duplication or
// disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
//
// PURPOSE
// Sample program to read log records from the binary log files
// ==========================================================================
import com.ibm.internet.nd.log.*;

import java.io.*;
import java.util.*;
import java.text.*;

/**
 * The LOG_SampleReader class is an example of a main program to extract
 * information from an eNetwork Dispatcher component's binary log files.
 *
 * The program accepts two times as input arguments; and retrieves all
 * available log records that were logged between the two times.
 *
 * <p>The directory where the log files are stored is provided to the
 * program in the java environment variable "END_LOG_DIRECTORY", using
 * the -D flag of the java or jre command.
 *
 * <p>An example use of this class would be
 * <xmp>
 * jre -cp <classpath> -D<log_directory> LOG_SampleReader 1999/02/26 8:00 1999/02/26 17:00
 * </xmp>
 * The above command would display all the server information logged between
 * 8:00 AM and 5:00 PM on February 2nd, 1999.
 * <p>The &lt classpath&gt argument must point first to the directory containing the class
 * file for LOG_SampleReader.java, then to the ibmnd.jar file.
 * A sample command to invoke the logreader is provided in the lib/BinaryLog directory.
 */

public class LOG_SampleReader
{
  // ---------------------
  // Public Class variables
  // =====================

  /**
   * The SimpleDateFormat used to read dates from the command line.
   *
   * <p>This format is used to parse the dates entered on the command line.
   * Its value is "yyyy/MM/dd HH:mm".  This means that the two
   * dates provided on the command line would look like:
   * <xmp>
   * 1999/02/26 8:00 1999/02/26 17:00
```
Figure 323. (Part 1 of 5). LOG_SampleReader.java

```
    * </xmp>
    * You must edit the source code to change the format used.
    * <p>Since the format contains spaces between the date and time fields,
    * the main program reads them as args[0] through args[3] of the input
    * arguments.  If you modify the format to contain fewer tokens or
    * more tokens, you must modify the main program to read your arguments
    * from the correct input arg.
    */
   // Declared public so javadoc will print it by default
   public static SimpleDateFormat sdfInput = new SimpleDateFormat("yyyy/MM/dd HH:mm");

   // -----------
   // Constructor
   // ===========
   /**
    * The object is never actually constructed, since all the work is done
    * in the main program.
    */
   LOG_SampleReader(){}

   // --------------------
   // Main Program
   // ====================

   /**
    * The main program takes input form the command line and writes out the
    * requested log records from the specified log directory's binary log files.
    *
    * <p>The program instantiates a LOG_Reader object, initializes it with the
    * start and end times, then retrieves LOG_Record objects from the LOG_Reader
    * until a null record is returned.  The LOG_Record objects retrieved are cast
    * to the appropriate type, and the information from the LOG_Record is either
    * stored for later output, or written to the screen.
    *
    * <p>The LOG_Reader object's constructor takes as input the directory where
    * the log files are stored.
    * <p>The LOG_Reader object's initialize function takes as input two long values
    * that represent the start and end times for the records to be retrieved.  These
    * times are in Coordinated Universal Time (CUT).
    * The input arguments from the command line have been converted from strings to
    * the appropriate longs using the java.text.SimpleDateFormat and java.util.Date
    * classes.
    * <p>The LOG_Reader object's initialize method returns the first record in the
    * logs that falls within the time range specified. If no records exist between
    * the times specified, null is returned.
    * <p>The main program then starts a loop which processes the current LOG_Record,
    * then calls the LOG_Reader object's readRecord method to get the next LOG_Record.
    * This loop will continue until the LOG_Record retrieved is null.
    * <p>The following are the types of LOG_Record objects that can be retrieved:
    * <ul>
    * <li>LOG_TimeStampRecord
    * <br>The LOG_TimeStampRecord contains the following methods:
    * <ul>
    * <li>getTimestamp() - returns a long value representing the time in CUT when
```

Figure 324. (Part 2 of 5). LOG_SampleReader.java

```
 * the record was written to the binary log.
 * </ul>
 * <li>LOG_ExecutorIDRecord
 * <br>The LOG_ExecutorIDRecord contains the following methods:
 * <ul>
 * <li>getNFA() - returns a string representing the Non-Forwarding address
 * of the executor.
 * <li>getNumClusters() - returns an int representing the number of clusters defined
 * in the executor.
 * </ul>
 * <li>LOG_ClusterIDRecord
 * <br>The LOG_ClusterIDRecord contains the following methods:
 * <ul>
 * <li>getIPAddress() - returns a string representing the cluster address.
 * <li>getNumPorts() - returns an int representing the number of ports defined
 * for this cluster.
 * </ul>
 * <li>LOG_PortIDRecord
 * <br>Each LOG_PortIDRecord describes a port that has been defined under the cluster
 * address that was returned by the most recently retrieved LOG_ClusterIDRecord.
 * <br>The LOG_PortIDRecord contains the following methods:
 * <ul>
 * <li>getPortNumber() - returns an int representing the port number.
 * <li>getNumServers() - returns an int representing the number of servers defined
 * for this port.
 * </ul>
 * <li>LOG_ServerReportRecord
 * <br>Each LOG_ServerReportRecord describes a server that has been defined under
 * the port number that was returned by the most recently retrieved LOG_PortIDRecord.
 * <br>The LOG_ServerReportRecord contains the following methods:
 * <ul>
 * <li>getIPAddress() - returns a string representing the server address.
 * <li>getWeight()    - returns an int representing the weight
 * of the server when the record was written to the binary log
 * <li>getTotalConnections() - returns an int representing the total connections
 * of the server when the record was written to the binary log
 * <li>getActiveConnections() - returns an int representing the active connections
 * of the server when the record was written to the binary log
 * <li>getPortLoad() - returns an int representing the port load
 * of the server when the record was written to the binary log
 * <li>getSystemLoad() - returns an int representing the system load
 * of the server when the record was written to the binary log
 * <li>getHasPortLoad() - returns a boolean specifying whether port loads were
 * being provided for the server when the record was written to the binary log
 * <li>getHasSystemLoad() - returns a boolean specifying whether system loads were
 * being provided for the server when the record was written to the binary log
 * </ul>
 * </ul>
 * <p>LOG_Record objects of each type will be retrieved from the binary logs in
 * the following order.
 * <ol>
 * <li>LOG_TimeStampRecord
 * <li>LOG_ExecutorIDRecord
 * <li>LOG_ClusterIDRecord (if any clusters exist)
```

Figure 325. (Part 3 of 5). LOG_SampleReader.java

```
   * <li>LOG_PortIDRecord (if any ports exist for the previously read cluster)
   * <li>LOG_ServerReportRecord (one for each server defined on the previously read port)
   * <li>Additional LOG_PortIDRecords and their LOG_ServerReportRecords for other ports
   * defined on the previously read cluster
   * <li>Additional LOG_ClusterIDRecords and their LOG_PortIDRecords and their
LOG_ServerReportRecords
   * for other defined clusters.
   * <li>Next sequence in same order, starting with a LOG_TimeStampRecord.
   * </ol>
   */
  public static void main(String[] args) throws ParseException {
    // The directory where the logs are kept is communicated through a java environment variable.
    // This allows the main class to be called with just the times to report.  If the directory
    // name doesn't end with a path separator, add one on.
    String sLogDirectory = System.getProperty("END_LOG_DIRECTORY");
    if (!sLogDirectory.endsWith(File.separator)) sLogDirectory = sLogDirectory+File.separator;

    // Get the start and end time from the command line and convert to longs.
    Date date;
    date = sdfInput.parse(args[0]+" "+args[1]);
    long lStartTime = date.getTime();
    date = sdfInput.parse(args[2]+" "+args[3]);
    long lEndTime = date.getTime();

    // Initialize all the fields we will be retrieving and writing out
    SimpleDateFormat sdfTimestamp = new SimpleDateFormat("yyyy/MM/dd-HH:mm:ss.SSS");
    long lTimeStamp = 0;
    String sNFA = null;
    int iNumClusters =0;
    String sClusterIPAddress = null;
    int iNumPorts = 0;
    int iPortNumber = 0;
    int iNumServers = 0;
    String sServerIPAddress = null;
    int iWeight = 0;
    int iTotalConnections = 0;
    int iActiveConnections = 0;
    int iPortLoad = 0;
    int iSystemLoad = 0;
    boolean bHasPortLoad = false;
    boolean bHasSystemLoad = false;

    // Instantiate a LOG_Reader and get the first LOG_Record
    LOG_Reader logReader = new LOG_Reader(sLogDirectory);
    LOG_Record logRecord = logReader.initialize(lStartTime,lEndTime);

    // Process each LOG_Record, accumulate information until we get a
    // LOG_ServerReportRecord, then write out everything.
    while (null != logRecord) {
      if (logRecord instanceof LOG_TimeStampRecord) {
        // Got a TimeStamp
        lTimeStamp = ((LOG_TimeStampRecord)logRecord).getTimeStamp();
      }
      else if (logRecord instanceof LOG_ExecutorIDRecord) {
```

Figure 326. (Part 4 of 5). LOG_SampleReader.java

```
        // Got an Executor ID
        sNFA         = ((LOG_ExecutorIDRecord)logRecord).getNFA();
        iNumClusters = ((LOG_ExecutorIDRecord)logRecord).getNumClusters();
      }
      else if (logRecord instanceof LOG_ClusterIDRecord) {
        // Got a Cluster ID
        sClusterIPAddress = ((LOG_ClusterIDRecord)logRecord).getIPAddress();
        iNumPorts         = ((LOG_ClusterIDRecord)logRecord).getNumPorts();
      }
      else if (logRecord instanceof LOG_PortIDRecord) {
        // Got a Port ID
        iPortNumber = ((LOG_PortIDRecord)logRecord).getPortNumber();
        iNumServers = ((LOG_PortIDRecord)logRecord).getNumServers();
      }
      else if (logRecord instanceof LOG_ServerReportRecord) {
        // Got a Server Report
        sServerIPAddress   = ((LOG_ServerReportRecord)logRecord).getIPAddress();
        iWeight            = ((LOG_ServerReportRecord)logRecord).getWeight();
        iTotalConnections  = ((LOG_ServerReportRecord)logRecord).getTotalConnections();
        iActiveConnections = ((LOG_ServerReportRecord)logRecord).getActiveConnections();
        iPortLoad          = ((LOG_ServerReportRecord)logRecord).getPortLoad();
        iSystemLoad        = ((LOG_ServerReportRecord)logRecord).getSystemLoad();
        bHasPortLoad       = ((LOG_ServerReportRecord)logRecord).getHasPortLoad();
        bHasSystemLoad     = ((LOG_ServerReportRecord)logRecord).getHasSystemLoad();
        System.out.println(sdfTimestamp.format(new Date(lTimeStamp))+","+
                           sNFA+","+
                           sClusterIPAddress+","+
                           iPortNumber+","+
                           sServerIPAddress+","+
                           iWeight+","+
                           iTotalConnections+","+
                           iActiveConnections+","+
                           iPortLoad+","+
                           bHasPortLoad+","+
                           iSystemLoad+","+
                           bHasSystemLoad);
      }
      else {
        // Got some other kind of LOG_Record (there are none yet)
      }
      logRecord = logReader.readRecord();
    }
  }
}
```

Figure 327. (Part 5 of 5). LOG_SampleReader.java

Appendix D. Special Notices

This publication is intended to help you to plan for, install, configure, use, tune and troubleshoot IBM SecureWay Network Dispatcher, the Load Balancing component of IBM WebSphere Performance Pack. The information in this publication is not intended as the specification of any programming interfaces that are provided by IBM WebSphere Performance Pack. See the PUBLICATIONS section of the IBM Programming Announcement for IBM WebSphere Performance Pack 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 information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. 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.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

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:

Advanced Peer-to-Peer Networking	AIX
APPN	AS/400
Deep Blue	eNetwork
IBM	Netfinity
NetView	Operating System/2
OS/2	OS/390
OS/400	RS/6000
S/390	SecureWay
SP	System/390
TXSeries	VisualAge
WebSphere	

The following terms are trademarks of other companies:

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

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

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

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

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

UNIX is a registered trademark in the United States and/or 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 E. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

E.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 417.

- *IBM WebSphere Performance Pack: Web Content Management with IBM AFS Enterprise File System*, SG24-5857
- *IBM WebSphere Performance Pack: Caching and Filtering with IBM Web Traffic Express*, SG24-5859
- *Internet Security in the Network Computing Framework*, SG24-5220
- *IBM WebSphere Performance Pack Usage and Administration*, SG24-5233
- *Network Computing Framework Component Guide,* SG24-2119
- *TCP/IP Tutorial and Technical Overview*, GG24-3376

E.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
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

E.3 Other Publications

The following publication is also relevant as a further information source:

- *SecureWay Network Dispatcher User’s Guide Version 2.1 for Solaris, Windows NT and AIX*, GC31-8496

E.4 Referenced Web Sites

- `http://www.redbooks.ibm.com`
- `http://www.javasoft.com`
- `http://www.software.ibm.com/webservers/appserv`

- http://www.ibm.com
- http://www.ibm.com/software
- http://www.ibm.com/java/jdk/download/index.html
- http://info.internet.isi.edu/in-notes/rfc/files/rfc1700.txt
- http://w3.itso.ibm.com
- http://w3.ibm.com
- http://www.elink.ibmlink.ibm.com/pbl/pbl

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-ROM redbooks from the redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this redbooks site.

  Redpieces are redbooks in progress; not all redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders by e-mail including information from the redbooks fax order form to:

	e-mail address
In United States	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

ACL	access control list
AFS	Enterprise File System
AIX	advanced interactive executive
ATM	Asynchronous Transfer Mode
API	application programming interface
APPN	Advanced Peer-to-Peer Network
ARP	Address Protocol Request
CB	Component Broker
CBR	Content Based Routing
CORBA	Common Object Request Broker Architecture
DMZ	demilitarized zone
DNS	Domain Name System
EJB	Enterprise JavaBeans
EJS	Enterprise Java Services
FDDI	Fiber Distributed Data Interface
FTP	File Transfer Protocol
GC	garbage collector
GEM	Global Enterprise Manager
GUI	graphical user interface
HACMP	High Availability Cluster Multiprocessing
HP-UX	Hewlett-Packard UNIX
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
IP	Internet Protocol
ISP	Internet Service Provider
ISS	Interactive Session Support
ITSO	International Technical Support Organization
JDK	Java Development Kit
JRE	Java Runtime Environment
JSP	JavaServer Pages
JVM	Java Virtual Machine
LAN	local area network
LDAP	Lightweight Directory Access Protocol
MAC	Media Access Control
Mbps	megabits per second
MBps	megabytes per second
MIB	management information base
MVS	multiple virtual storage
NAP	network access points
NCSA	National Center for Supercomputing Applications
ND	Network Dispatcher
NNTP	NetNews transfer protocol
OS/2	Operating System/2
PICS	Platform for Internet Content Selection
POP	points of presence
POP3	Post Office Protocol 3
RCA	Remote Cache Access
RMI	Remote Method Invocation
RPC	remote procedure call
SDA	Server Directed Affinity
SSL	Secure Sockets Layer
SMTP	Simple Mail Transfer Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TEC	Tivoli Enterprise Console
UDP	User Datagram Protocol
URL	Universal Resource Locator, Uniform Resource Locator
WAN	wide area network
WAND	Wide Area Network Dispatcher
WTE	Web Traffic Express
WLM	workload manager
WWW	World Wide Web
XML	Extensible Markup Language

Index

ITSO Redbook Evaluation

IBM WebSphere Performance Pack: Load Balancing with IBM SecureWay Network Dispatcher
SG24-5858-00

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!)

SG24-5858-00

Printed in the U.S.A.

IBM WebSphere Performance Pack: Load Balancing with IBM SecureWay Network Dispatcher

SG24-5858-00