



Configuring LocalDirector

This chapter describes how to complete a basic configuration for LocalDirector. Topics include:

- Creating a Basic LocalDirector Configuration, page 3-1
- Configuration Examples, page 3-3

Creating a Basic LocalDirector Configuration

This section describes how to complete a basic configuration for LocalDirector and the server farm.

Basic configuration for LocalDirector includes setting its IP address, testing a connection to the network, and setting a password. Configuration for the server farm includes defining virtual servers and real servers, setting the load-balancing and redirection options, binding virtual servers and real servers, and putting all servers into service. Once basic configuration is complete, see the “Configuration Examples” section for additional procedures that can be used to set up specific LocalDirector features for your site.

To configure LocalDirector system settings (basic configuration), perform the following steps:

-
- Step 1** Make sure LocalDirector is properly installed. Refer to the appropriate LocalDirector hardware installation guide for installation instructions.
 - Step 2** Power on LocalDirector.



Note Because LocalDirector ships with its software loaded in Flash memory, LocalDirector boots without a system disk.

As LocalDirector boots, messages such as the following appear:

```
Finesse Bios V3.
Booting Floppy
Loading from Flash
32MB RAM
Flash=AT29C040A @ 0x300
 i82557 rev 2 Ethernet @ irq11 dev 13 index 0 MAC: 00a0.c965.576f
 i82557 rev 2 Ethernet @ irq15 dev 14 index 1 MAC: 00a0.c965.5b33
 i82557 rev 2 Ethernet @ irq 5 dev 15 index 2 MAC: 00a0.c965.56c1
```

```
LocalDirector 410 Version 3.x Initialization.....done.
Copyright (c) 1998 by Cisco Systems, Inc.
```

Restricted Rights Legend

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c) of the Commercial Computer Software - Restricted Rights clause at FAR sec. 52.227-19 and subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS sec. 252.227-7013.

Cisco Systems, Inc.
170 West Tasman Drive
San Jose, California 95134-1706



Note The messages displayed at bootup differ depending on the LocalDirector platform and the number and type of interfaces installed.

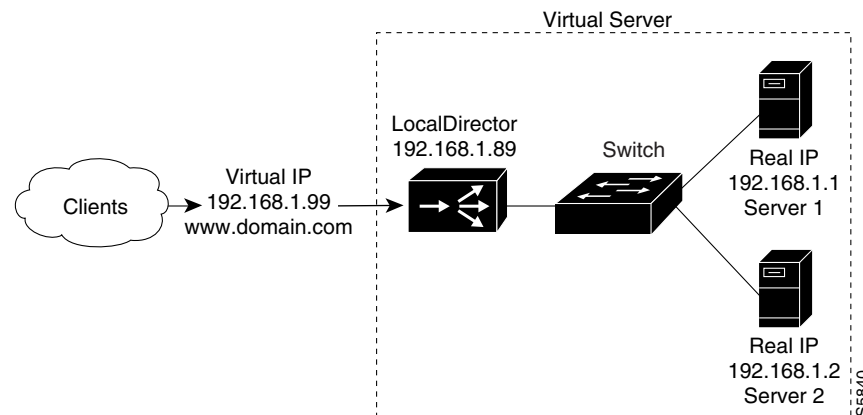
- Step 3** Use the **enable** command to access privileged mode.
- Step 4** Use the **configuration** command to access LocalDirector configuration commands.
- Step 5** Use the **ip address** command to assign the LocalDirector IP address and subnet mask.
- Step 6** Use the **ping** command to determine if LocalDirector is connected or if a host is available on the network. If this test fails, follow these steps:
 - a. Make sure that you have defined an IP address for LocalDirector, and that it is a valid network address.
 - b. Create a virtual address and use the **ping-allow** command to allow you to ping the virtual server. You can also use the **traceroute** command with virtual IP addresses, but you cannot use **traceroute** with the LocalDirector IP address.
 - c. Ping real servers on the LocalDirector network.
 - d. Test each side of the connection by pinging hosts on each network segment. Check all cable connections and ensure they are secure.
- Step 7** Use the **enable password** command to change the privileged mode password.
- Step 8** If preferred, use the **hostname** command to change the host name for the LocalDirector command-line prompt. Note that in a failover configuration, both units have the same host name because the configuration is copied from one unit to the other.

To configure the LocalDirector server farm, perform the following steps:

-
- Step 1** Use the **virtual** command to define virtual servers.
 - Step 2** If preferred, use the **redirection** command to change the default directed mode (Network Address Translation) to dispatched mode for the virtual servers. Additionally, use the **alias ip address** command to specify the virtual IP address on each real server that uses dispatched mode.
 - Step 3** Use the **predictor** command to set the type of load balancing for each virtual server.
 - Step 4** Use the **real** command to define real servers.
 - Step 5** Use the **bind** command to associate each virtual server to a real server.
 - Step 6** Use the **in-service** command to designate real servers and virtual servers as in service. (This can also be done when the virtual servers and real servers are defined.)
 - Step 7** Use the **write terminal**, **show real**, **show virtual**, and **show bind** commands to check the configuration.
 - Step 8** Use the **write memory** command to store the configuration in Flash memory.
 - Step 9** Use the **show config** command to verify the configuration stored in Flash memory.
The basic configuration is complete.
 - Step 10** Enter **Ctrl-Z** to exit the configuration mode. Use the **disable** command to exit privileged mode.
-

Figure 3-1 shows an example of a basic configuration with one virtual server bound to two real servers.

Figure 3-1 Basic LocalDirector Configuration



Configuration Examples

To help determine virtual and real server configurations, diagram the implementation before you begin. Ensure that any virtual IP address you configure is from a valid IP network. If the virtual address is to be accessed from the Internet, the IP address must be part of a Network Information Center-allocated network number.

This section provides server configuration examples in the following sections:

- Default Configuration Settings, page 3-4
- One Virtual Server and Multiple Real Servers, page 3-5
- Multiple Virtual Servers and One Real Server, page 3-8
- Multiple Virtual Servers and Multiple Real Servers, page 3-9
- Buddy Virtual Setup, page 3-13
- Requesting the Same Server for Multiple Connections, page 3-13
- Cookie-Sticky Setup, page 3-16
- HTTP Redirection Setup, page 3-17
- Client-Assigned Load Balancing, page 3-20
- Content Load Balancing, page 3-23
- Secure Services, page 3-24
- Port-Bound Servers, page 3-26
- Maximum Connections and Weighted Configuration, page 3-28
- Backup Configurations, page 3-30
- Configuring with the data Command, page 3-31
- Configuring UDP Traffic, page 3-31
-
- Integrated Probe for DNS Support, page 3-35
- Hypertext Transfer Protocol Probe Support, page 3-35
- Simple Network Time Protocol Support, page 3-36
- LocalDirector as a Boomerang Content Routing Agent, page 3-37

Default Configuration Settings

Table 3-1 shows default configuration settings for LocalDirector.

Table 3-1 LocalDirector Default Configuration Settings

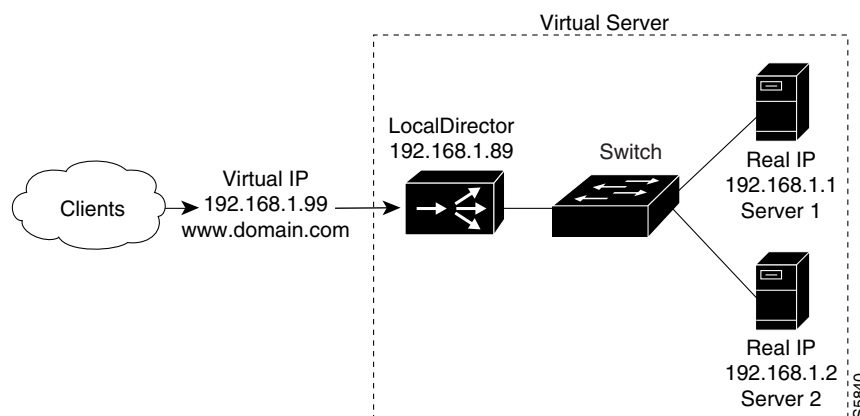
Configuration Setting	Default Value	Minimum Value	Maximum Value
Redirection mode	Directed, local	—	—
IP services	TCP enabled on virtual and real servers	—	—
Syslog console	Not defined	—	—
Routing Information Protocol (RIP)	Disabled	—	—
Timeout value	120 minutes	5 minutes 0 for UDP	65,535 minutes (45.5 days)

Table 3-1 LocalDirector Default Configuration Settings (continued)

Configuration Setting	Default Value	Minimum Value	Maximum Value
Sticky value	0 (disabled)	0 minutes	65,535 minutes (45.5 days)
Reassign value	3	1	4
Threshold value	8	0	65,535
Retry value	1 minute	0	65,535 minutes (45.5 days)
Predictor	Leastconns	—	—
Weight value	1	0	100
Autounfail	On	—	—
Slowstart option	Roundrobin	—	—
Bridging	Allowed	—	—
Maximum connections	0 (unlimited)	—	—
Synguard protection	0 (off)	—	—
Data connections	0 (off)	—	—

One Virtual Server and Multiple Real Servers

In this example, LocalDirector is load balancing all TCP traffic over two servers to provide web services. Figure 3-2 shows the network configuration.

Figure 3-2 One Virtual Server and Multiple Real Servers

All traffic destined for virtual IP address 192.168.1.99 is load balanced across real servers with IP addresses 192.168.1.1 and 192.168.1.2. Only the virtual server appears in the Domain Name System (DNS).

To set up a configuration for one virtual server and multiple real servers, perform the following steps:

- Step 1** Use the **enable** command to enter privileged mode. Type a carriage return at the password prompt if you do not want to assign a privileged password:

```
LocalDirector# enable
Password:<CR>
```

- Step 2** Use the **configuration t** command to enter configuration mode:

```
LocalDirector# configuration t
```

- Step 3** Use the **ip address** command to specify LocalDirector IP address 192.168.1.89, and subnet mask 255.255.255.0:

```
ld(config)# ip address 192.168.1.89 255.255.255.0
```

- Step 4** Use the **interface ethernet** command with the **auto** option (if your interface card supports this option) to automatically set the speed of the Ethernet interface:

```
ld(config)# interface ethernet 0 auto
ld(config)# interface ethernet 1 auto
```

Otherwise, to set the speed manually, use the **interface ethernet** command with the option that is appropriate for your card:

```
ld(config)# interface ethernet 0 {10baset | 100basetx | 100full}
```



Note The four-port RNS adapter cards cannot use the **auto** option. These cards display the following string in the **show interface** command output:

```
rns23x0
```

- Step 5** Use the **shutdown** command to disable unused interface ports:

```
ld(config)# shutdown interface ethernet 2
ld(config)# shutdown interface ethernet 3
```

- Step 6** Use the **name** command to identify IP address 192.168.1.99 as domain, and the **virtual** command to define domain as a virtual server:

```
ld(config)# name 192.168.1.99 domain
ld(config)# virtual domain
```

- Step 7** Use the **name** command to identify IP address 192.168.1.1 as server 1 and 192.168.1.2 as server 2:

```
ld(config)# name 192.168.1.1 server1
ld(config)# name 192.168.1.2 server2
```

- Step 8** Use the **real** command to identify server 1 and server 2 as real servers, and the **is (in-service)** option to enable the real servers to start accepting connections:

```
ld(config)# real server1 is
ld(config)# real server2 is
```

- Step 9** Use the **bind** command to associate domain with server 1 and server 2 and establish the load-balancing relationship between the virtual and the real servers:

```
ld(config)# bind domain server1 server2
```

Step 10 Use the **is** command to bring the virtual server into service:

```
ld(config)# is virtual domain
```

Step 11 Use the **write terminal** command to view the running configuration before it is saved.

Step 12 Use the **write mem** command to save the new settings:

```
ld(config)# write mem
Building configuration...
[OK]
```

Step 13 View the saved configuration with the **show configuration** command:

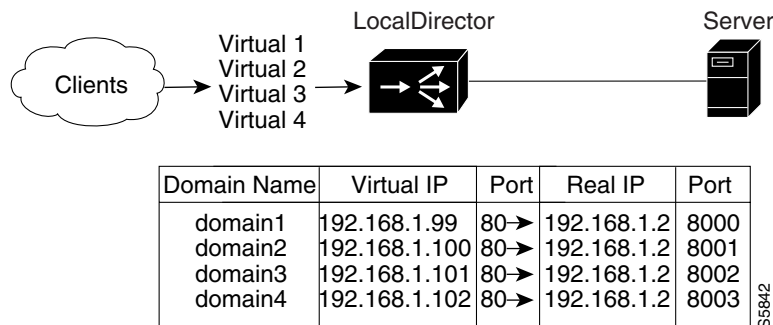
```
ld(config)# show configuration
: Saved
: LocalDirector 420 Version 3.1.0.106
syslog output 20.3
no syslog console
enable password dfeaf10390e560aea745ccba53e044 encrypted
hostname localdirector
no shutdown ethernet 0
no shutdown ethernet 1
shutdown ethernet 2
shutdown ethernet 3
interface ethernet 0 100basetx
interface ethernet 1 100basetx
interface ethernet 2 100basetx
interface ethernet 3 100basetx
mtu 0 1500
mtu 1 1500
mtu 2 1500
mtu 3 1500
multiring all
no secure 0
no secure 1
no secure 2
no secure 3
ping-allow 0
ping-allow 1
ping-allow 2
ping-allow 3
ip address 192.168.1.89 255.255.255.0
route 0.0.0.0 0.0.0.0 172.20.30.1 1
no rip passive
failover ip address 0.0.0.0
no failover
password cisco
no snmp-server contact
no snmp-server location
virtual 192.168.1.99:0:0:tcp is
real 192.168.1.2:0:0:tcp is
real 192.168.1.1:0:0:tcp is
name 192.168.1.1 server1
name 192.168.1.2 server2
name 192.168.1.99 domain
bind 192.168.1.99:0:0:tcp 192.168.1.2:0:0:tcp
bind 192.168.1.99:0:0:tcp 192.168.1.1:0:0:tcp
localdirector(config)#
```

Multiple Virtual Servers and One Real Server

In this example, four virtual addresses are bound to a single web server, as shown in Figure 3-3, allowing you to provide multiple DNS entries for one server. In other words, one real server supports multiple domain names. Virtual IP addresses 192.168.1.99, 192.168.1.100, 192.168.1.101, and 192.168.1.102 are identified as domain 1, domain 2, domain 3, and domain 4, respectively. Port 80 traffic for each virtual IP address is bound to different ports on real server IP address 192.168.1.2.

All web traffic destined for domain 1 accesses information on real server 192.168.1.2 through port 8000. Traffic destined for domain 2 accesses information on real server 192.168.1.2 through port 8001, and so on.

Figure 3-3 Multiple Virtual Servers and One Real Server



Also, by defining a virtual server as an IP address and a port, you can restrict traffic to a specific port. Port 80 is specified for each of the virtual servers, and ports 8000, 8001, 8002, and 8003 are specified for the real server. The virtual server ports and real server ports are bound to each other directly with a unique bind-id on the real server for each port that is bound. In addition, if the application running on port 8000 fails, the entire server is not failed by LocalDirector; the remaining ports continue to accept connections.

To configure multiple virtual servers for one real server, perform the following steps:

Step 1 Use the **name** command to identify the IP addresses of the virtual and real servers:

```
ld(config)# name 192.168.1.99 domain1
ld(config)# name 192.168.1.100 domain2
ld(config)# name 192.168.1.101 domain3
ld(config)# name 192.168.1.102 domain4
ld(config)# name 192.168.1.2 server
```

Step 2 Use the **real** command to identify the IP address named server as the real server that is accepting connections on ports 8000, 8001, 8002, and 8003:

```
ld(config)# real server:8000
ld(config)# real server:8001
ld(config)# real server:8002
ld(config)# real server:8003
```

Step 3 Use the **virtual** command to identify the named IP addresses domain 1, domain 2, domain 3, and domain S4 as virtual servers accepting connections on port 80:

```
ld(config)# virtual domain1:80
ld(config)# virtual domain2:80
ld(config)# virtual domain3:80
ld(config)# virtual domain4:80
```

- Step 4** Use the **bind** command to direct port 80 network traffic from each virtual server to a different port on the real server:

```
ld(config)# bind domain1:80 server:8000
ld(config)# bind domain2:80 server:8001
ld(config)# bind domain3:80 server:8002
ld(config)# bind domain4:80 server:8003
```

- Step 5** Use the **is real** command to set the service state for all real server ports to in service:

```
ld(config)# is real server:8001
ld(config)# is real server:8002
ld(config)# is real server:8003
ld(config)# is real server:8000
```

- Step 6** Use the **is virtual** command to set the service state for all virtual server ports to in service:

```
ld(config)# is virtual domain1:80
ld(config)# is virtual domain2:80
ld(config)# is virtual domain3:80
ld(config)# is virtual domain4:80
```

- Step 7** Use the **show bind** command to display the association between the virtual server ports and real server ports:

```
ld(config)# show bind
```

Virtual Machine(s)	Real Machines
domain2:80:0:tcp(IS)	server:8001:0:tcp(IS)
domain1:80:0:tcp(IS)	server:8000:0:tcp(IS)
domain4:80:0:tcp(IS)	server:8003:0:tcp(IS)
domain3:80:0:tcp(IS)	server:8002:0:tcp(IS)

```
ld(config)#
```

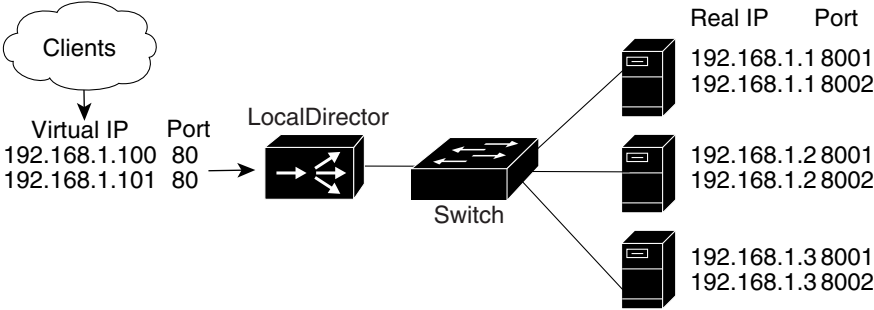
Multiple Virtual Servers and Multiple Real Servers

You can combine multiple virtual servers and multiple real servers so that each virtual server sends network traffic to the same port across real servers, as shown in Figure 3-4. All TCP traffic destined for virtual server 192.168.1.100 is load balanced across the three real servers on port 8001. Traffic destined for virtual server 192.168.1.101 is load balanced across the real servers on port 8002.

A combination of virtual servers and real servers can also be used to load balance traffic across server clusters, as shown in Figure 3-5.

Each virtual server can have a different load-balancing option set with the **predictor** command. For example, 192.168.1.100 can be configured to use the **leastconns** option, and 192.168.1.101 can be configured to use the **weighted** option.

Figure 3-4 Multiple Virtual Servers and Multiple Real Servers



Domain Name	Virtual IP	Real IP	Port
domain1	192.168.1.100 Port 80	192.168.1.1	8001
www.domain1.com		192.168.1.2	8001
		192.168.1.3	8001
domain2	192.168.1.101 Port 80	192.168.1.1	8002
www.domain2.com		192.168.1.2	8002
		192.168.1.3	8002

55841

To configure multiple virtual servers for multiple real servers, perform the following steps:

Step 1 Use the **real** command to identify three real servers, each accepting connections on ports 8001 and 8002. Use the **is** option to put the real servers in service:

```
ld(config)# real 192.168.1.1:8001 is
ld(config)# real 192.168.1.1:8002 is
ld(config)# real 192.168.1.2:8001 is
ld(config)# real 192.168.1.2:8002 is
ld(config)# real 192.168.1.3:8001 is
ld(config)# real 192.168.1.3:8002 is
```

Step 2 Use the **virtual** command to create two virtual servers accepting connections on port 80:

```
ld(config)# virtual 192.168.1.100
ld(config)# virtual 192.168.1.101
```

Step 3 Use the **bind** command to direct network traffic from port 80 on the two virtual servers to ports 8001 and 8002 on the three real servers:

```
ld(config)# bind 192.168.1.100:80 192.168.1.1:8001
ld(config)# bind 192.168.1.100:80 192.168.1.2:8001
ld(config)# bind 192.168.1.100:80 192.168.1.3:8001
ld(config)# bind 192.168.1.101:80 192.168.1.1:8002
ld(config)# bind 192.168.1.101:80 192.168.1.2:8002
ld(config)# bind 192.168.1.101:80 192.168.1.3:8002
```

Step 4 Use the **is virtual** command to set the service state for all virtual server ports to in service:

```
ld(config)# is virtual 192.168.1.100:80
ld(config)# is virtual 192.168.1.101:80
```

Step 5 Use the **show bind** command to display the association between the virtual and real servers:

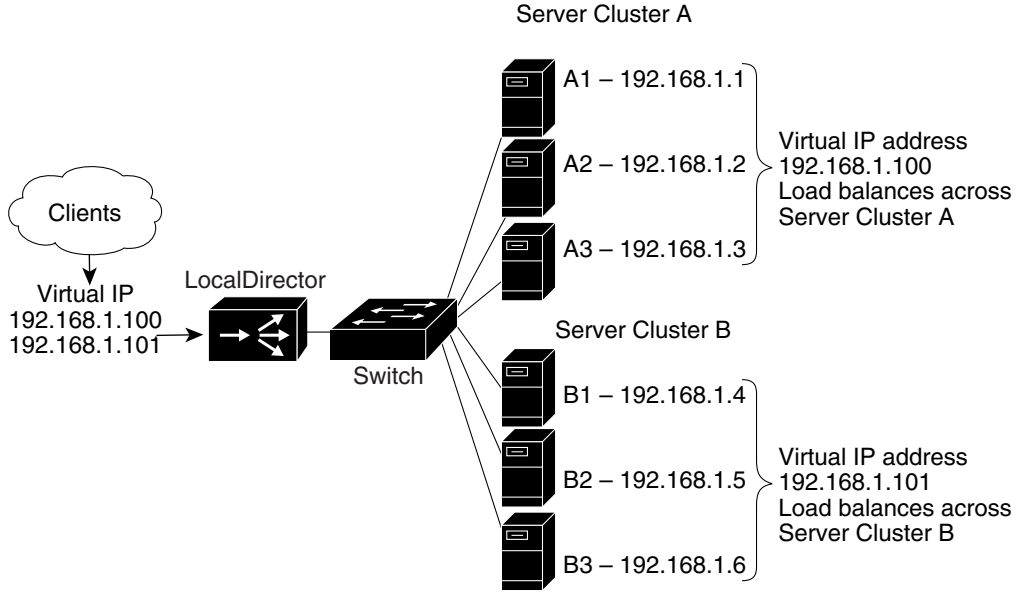
```
ld(config)# show bind

      Virtual Machine(s)          Real Machines
192.168.1.101:80:0:tcp(IS)
                                192.168.1.3:8002:0:tcp(IS)
                                192.168.1.2:8002:0:tcp(IS)
                                192.168.1.1:8002:0:tcp(IS)
192.168.1.100:80:0:tcp(IS)
                                192.168.1.3:8001:0:tcp(IS)
                                192.168.1.2:8001:0:tcp(IS)
                                192.168.1.1:8001:0:tcp(IS)

ld(config)#
```

In Figure 3-5, TCP connections to domain 1 are load balanced across Server Cluster A containing real servers 192.168.1.1, 192.168.1.2, and 192.168.1.3. Connections to domain 2 are load balanced across Server Cluster B containing real servers 192.168.1.4, 192.168.1.5, and 192.168.1.6.

Figure 3-5 Load Balancing Across Server Clusters



bind-ip Addresses

Domain Name	Virtual IP Address	Real IP Address
domain 1	192.168.1.100	192.168.1.1
www.domain1.com		192.168.1.2
		192.168.1.3
domain 2	192.168.1.101	192.168.1.4
www.domain2.com		192.168.1.5
		192.168.1.6

55843

To load balance across server clusters, perform the following steps:

Step 1 Use the **real** command to identify the six real servers, and the **is** option to put the real servers in service:

```
ld(config)# real 192.168.1.1 is
ld(config)# real 192.168.1.2 is
ld(config)# real 192.168.1.3 is
ld(config)# real 192.168.1.4 is
ld(config)# real 192.168.1.5 is
ld(config)# real 192.168.1.6 is
```

Step 2 Use the **virtual** command to identify the two virtual servers:

```
ld(config)# virtual 192.168.1.100
ld(config)# virtual 192.168.1.101
```

Step 3 Use the **bind** command to direct network traffic from virtual server 192.168.1.100 to real servers 192.168.1.1, 192.168.1.2, and 192.168.1.3, and from virtual server 192.168.1.101 to real servers 192.168.1.4, 192.168.1.5, and 192.168.1.6:

```
ld(config)# bind 192.168.1.100 192.168.1.1 192.168.1.2 192.168.1.3
ld(config)# bind 192.168.1.101 192.168.1.4 192.168.1.5 192.168.1.6
```

Step 4 Use the **is** command to put the virtual servers in service:

```
ld(config)# is virtual 192.168.1.100
ld(config)# is virtual 192.168.1.101
```

Step 5 Use the **show bind** command to display the association between the virtual and real servers:

```
ld(config)# show bind
Virtual                               Real
192.168.1.101:0:0:tcp (IS)            192.168.1.6:0:0:tcp (IS)
                                       192.168.1.5:0:0:tcp (IS)
                                       192.168.1.4:0:0:tcp (IS)
192.168.1.100:0:0:tcp (IS)            192.168.1.3:0:0:tcp (IS)
                                       192.168.1.2:0:0:tcp (IS)
                                       192.168.1.1:0:0:tcp (IS)
```

Buddy Virtual Setup

Use the **buddy** command to create a buddy group (in the following procedure, *my_app*) consisting of a list of virtual servers. Certain commands and parameters that affect one virtual server affect all other virtual servers in the buddy group (for example, to share the generic information from the **sticky** command). A virtual server can reside in only one buddy group; if it currently exists in a group, it must be removed from that group before it can be added to a new group. An unlimited number of virtual servers can exist within a buddy group.

Step 1 The following example creates the buddy group *my_app* and adds virtual servers 10.0.0.100 and 10.0.0.200.

```
ld(config)# buddy my_app 10.0.0.100:0:0:tcp
ld(config)# buddy my_app 10.0.0.200:0:0:tcp
ld(config)# sticky 10.0.0.200 10
```

Step 2 Display the contents of the group with the **show buddy** command.

```
ld(config)# show buddy
      Buddy Group      Virtual Machine(s)
      my-app           10.0.0.100:0:0:tcp
                      10.0.0.200:0:0:tcp
```

With this setup, when the **sticky** command is used for virtual server 10.0.0.200 and then a client visits the virtual server 10.0.0.100 after visiting 10.0.0.200, that client will be sent to the same real server as on the 10.0.0.100 connection.

Requesting the Same Server for Multiple Connections

The **sticky** command ensures that the same client gets the same real server for multiple connections. The **sticky** command allows you to get back to the same real server and retain the statefulness of the client/server connection. For example, if a client is completing an online form, the **sticky** command

ensures that multiple connections are sent to the same server to complete the transaction. If this command is not used, each connection attempt to a virtual server is routed according to the predictor option in effect for that virtual server, without regard to prior history of the foreign host.

The **sticky** command does not time the length of a client connection; it times periods of inactivity. For example, if the **sticky** command is set to 5, and the client is active, new requests from the client are not sent to another server through load balancing, even if more than 5 minutes have elapsed. However, if 5 minutes of connection inactivity have elapsed, the requests from the client can be sent to another real server.

For Secure Socket Layer (SSL) connections, use the **ssl** option of the **sticky** command. LocalDirector performs a proxy connection until an SSL session ID is obtained from the HTTP headers. If there is already a sticky association in the LocalDirector cache, the host or server for that association is used. If there is no sticky association, a real server is selected, and a sticky association is created.

**Note**

If you have a proxy server in front of LocalDirector, the **sticky** command cannot work because all clients will appear to have the same source IP address. We recommend that you put LocalDirector in front of the proxy server. Use port-bound virtual servers to create less imbalance.

**Note**

If you have a proxy server in front of LocalDirector, use the **ssl** option of the **sticky** command to use the session ID as the connection identifier instead of the default **generic** option, which uses the source IP address.

To configure a stateful client/server connection using SSL, perform the following steps:

- Step 1** Use the **virtual** command to identify 192.168.1.1 as a virtual server accepting traffic on port 80 (HTTP) and port 443 (SSL):

```
ld(config)# virtual 192.168.1.1:80
ld(config)# virtual 192.168.1.1:443
```

- Step 2** Use the **real** command to identify 192.168.1.220 and 192.168.1.221 as real servers accepting traffic on port 80 (HTTP) and port 443 (SSL):

```
ld(config)# real 192.168.1.220:80 is
ld(config)# real 192.168.1.221:80 is
ld(config)# real 192.168.1.220:443 is
ld(config)# real 192.168.1.221:443 is
```

- Step 3** Use the **bind** command to bind the virtual servers accepting traffic on port 80 to the real servers accepting traffic on port 80:

```
ld(config)# bind 192.168.1.1:80 192.168.1.220:80 192.168.1.221:80
```

- Step 4** Use the **bind** command to bind the virtual servers accepting traffic on port 443 to the real servers accepting traffic on port 443:

```
ld(config)# bind 192.168.1.1:443 192.168.1.220:443 192.168.1.221:443
```

- Step 5** Use the **sticky** command to ensure that requests to virtual server 192.168.1.1:443 are sent to the same real server until 10 minutes of inactivity elapse:

```
ld(config)# sticky 192.168.1.1:443:0:tcp 10 ssl
```

Step 6 Use the **is** command to put the virtual servers in service:

```
ld(config)# is virtual 192.168.1.1:80:0:tcp
ld(config)# is virtual 192.168.1.1:443:0:tcp
```

Step 7 Use the **show bind** command to display server bind associations:

```
ld(config)# show bind
Virtual                               Real
192.168.1.1:443:0:tcp (IS)             192.168.1.221:443:tcp (IS)
                                         192.168.1.220:443:tcp (IS)
192.168.1.1:80:0:tcp (IS)             192.168.1.221:80:tcp (IS)
                                         192.168.1.220:80:tcp (IS)
```

Step 8 Use the **show sticky** command to display the **sticky** command setting:

```
ld(config)# show sticky
Machine                               Sticky
    192.168.1.1:443:0:tcp             10          ssl
```

Step 9 Use the **show configuration** command to view the entire configuration:

```
ld(config)# show configuration
: Saved
: LocalDirector 420 Version 3.1.0.106
syslog output 20.3
no syslog console
enable password dfeaf10390e560aea745ccba53e044 encrypted
hostname localdirector
no shutdown ethernet 0
no shutdown ethernet 1
shutdown ethernet 2
shutdown ethernet 3
interface ethernet 0 100basetx
interface ethernet 1 100basetx
interface ethernet 2 100basetx
interface ethernet 3 100basetx
mtu 0 1500
mtu 1 1500
mtu 2 1500
mtu 3 1500
multiring all
no secure 0
no secure 1
no secure 2
no secure 3
ping-allow 0
ping-allow 1
ping-allow 2
ping-allow 3
no rip passive
failover ip address 0.0.0.0
no failover
password cisco
no snmp-server contact
no snmp-server location
virtual 192.168.1.1:443:0:tcp is
virtual 192.168.1.1:80:0:tcp is
real 192.168.1.221:443:0:tcp is
```

```

real 192.168.1.220:443:0:tcp is
real 192.168.1.221:80:0:tcp is
real 192.168.1.220:80:0:tcp is
bind 192.168.1.1:443:0:tcp 192.168.1.220:443:0:tcp
bind 192.168.1.1:80:0:tcp 192.168.1.221:80:0:tcp
bind 192.168.1.1:80:0:tcp 192.168.1.220:80:0:tcp
sticky 192.168.1.1:443:0:tcp 10 ssl

```

Cookie-Sticky Setup

The **cookie-insert** option is a sticky connection based on a cookie created by LocalDirector. LocalDirector receives (acting as a proxy server) new client connections and scans the HTTP GET request for a cookie.

- If a cookie is not detected, LocalDirector assigns the connection to a real server and creates the cookie.
- If a cookie is detected, LocalDirector extracts it to determine the sticky real server ID and the expiration time. If the time has not expired, LocalDirector forwards the connection to the sticky real server. If the time has expired, LocalDirector assigns the connection to a new sticky real server and creates a new cookie.

The **cookie-passive** option is a sticky connection based on a cookie created by the sticky real server. LocalDirector learns the cookie and stores it in memory. LocalDirector receives (acting as a proxy server) new client connections and scans the HTTP GET request for a cookie that matches one in memory.



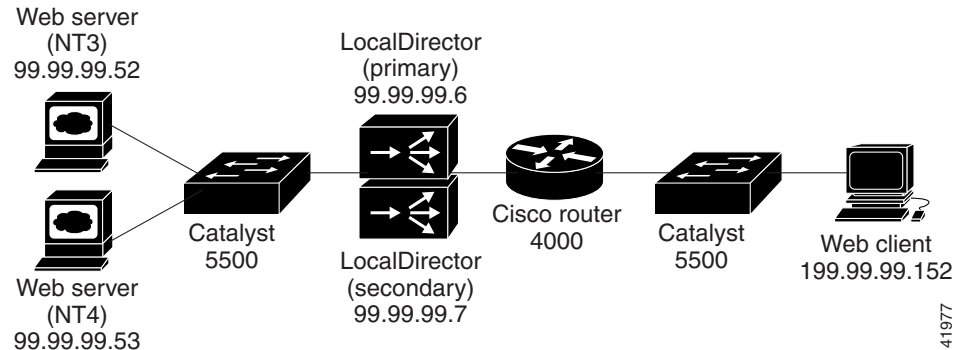
Note

Make sure the LocalDirector clock and the clock of the client and server are set correctly for proper operation.

- If there is a match and the time has not expired, LocalDirector forwards packets to the previously selected sticky real server. LocalDirector then scans packets from the sticky real server until the first HTTP response is detected.
- If there is a match but the time has expired, LocalDirector assigns the connection to a new real server. If there is no set-cookie token in the HTTP response, LocalDirector stops acting as the proxy server for the connection and forwards all packets directly to the sticky real server.

If there is a set-cookie token in the HTTP response, LocalDirector stores the new cookie in memory before it stops acting as a proxy server for the connection.

If the sticky real server does not answer or returns a TCP RST, LocalDirector assigns the connection to a new real server. Refer to the network environment shown in Figure 3-6 for this setup example.

Figure 3-6 Example for the Cookie-Sticky Setup

To configure the cookie-sticky setup, perform the following steps:

-
- Step 1** Change the date and time on the primary LocalDirector. You must use Coordinated Universal Time (UTC).
- ```
ld(config)# clock set 18:01:00 jan 18 2000
```
- Step 2** Create two real server IP addresses, one for web server NT3 and one for web server NT4.
- Step 3** Set a virtual server IP address on LocalDirector and set the virtual server to an in-service state.
- Step 4** Set the load-balancing predictor to the round-robin method.
- ```
ld(config)# predictor virtual_id roundrobin
```
- Step 5** Set the **cookie-insert** mode on LocalDirector (in this example, you must set the minutes value, which appears as 4 minutes).
- ```
ld(config)# sticky virtual_id 04 cookie-insert cookie-name
```
- 

**Note**

With the **cookie-insert** option, LocalDirector generates the cookie to the client. With the **cookie-passive** option, the server generates the cookie to the client. LocalDirector learns and stores the cookie, and passes it to the client. LocalDirector makes an association based on the cookie name.

---

## HTTP Redirection Setup

HTTP redirection is another method that lets you reliably implement persistent or sticky connections. HTTP redirection also allows LocalDirector to perform effective load balancing for Secure Socket Layer (SSL) and non-SSL connections, as well as for connections to an Internet Service Provider (ISP) that pass through a proxy server.

In a typical load-balancing environment, traffic comes from various client networks across the Internet to the virtual address on LocalDirector. LocalDirector then load balances this traffic between real servers.

A client must be connected to the same real server throughout a session for an HTTP server to maintain state on the connection. As previously discussed, the Sticky feature allows LocalDirector to load balance each client connection in a session to the same real server. Once the **sticky** command is applied to a real server for a particular virtual server, all client connections to the virtual server are directed to the same real server.

The current implementation of sticky uses the client IP address, a unique cookie value, or the SSL session ID to identify the client. Each of these implementations may have disadvantages that the HTTP redirection feature can solve.

If you apply the **sticky** command based on the client IP address, it is not effective for load-balancing techniques when connections to an ISP pass through a proxy server. In this scenario, the **sticky** command directs all client connections originating from behind the proxy server to the same real server, and this quickly creates a load imbalance. Also, persistence is broken if the ISP has multiple proxy servers behind a server load balancer (SLB). The SLB changes the client's address to different proxy addresses as attempts are made to load balance client requests. It is common practice for site security to configure the proxy server directly in front of LocalDirector.

The cookie-sticky option provides a mechanism for inserting a unique key for each user of a virtual server. This feature is only used in non-Secure Socket Layer (non-SSL) connections. The cookie-sticky feature provides a means to solve the proxy server problem and to give better load distribution at the server site.

The SSL session ID is effective only when the server is running SSL. Most configurations today only use SSL when necessary because of the heavy processing load required to maintain SSL connections.

Many web sites want to support mixed SSL and non-SSL connections. An example of mixed SSL and non-SSL connections is the web site of an online vendor who provides a *shopping cart* for users to select items for purchase while browsing all the items for sale. The contents of the shopping cart are not considered confidential, and most online vendor administrators establish this area of the web site as a non-SSL connection. However, as soon as a client is ready to purchase items, (often referred to as checkout time), the online vendor wants to be able to switch to a secure SSL connection to collect confidential information (such as a credit card number and expiration date) for the purchase. At this point during the client's session, the online vendor does not want to lose the client's need to mix non-SSL and SSL connections.

## Configuring HTTP Redirection on LocalDirector

The following configuration example uses one LocalDirector and two web servers. LocalDirector has a virtual server with the IP address 10.1.1.1 and the two real servers have addresses 10.1.1.10 and 10.1.1.11. The real servers and the virtual server are registered in DNS as follows:

```
10.1.1.1 www.acme.com
10.1.1.10 coyote.acme.com
10.1.1.11 roadrunner.acme.com
```

To use HTTP redirection to load balance between real machines, perform the following steps:

- 
- Step 1** Use the **virtual** command to create a virtual server to accept connections from the network.
- ```
localdirector(config)# virtual 10.1.1.1:80:0:tcp is
```
- Step 2** Use the **url** command to define a URL. You can also create a short identifier to replace a long URL string.
- ```
localdirector(config-if)# url coyote http://coyote.acme.com/%p
localdirector(config)# url roadrunner http://roadrunner.acme.com/%p
```

**Note**

The %p macro is used here so that LocalDirector includes the incoming path on the redirect it sends out. For instance if the client requests `http://www.acme.com/main.html`, the redirect would be for `http://coyote.acme.com/main.html`. If %p was not used, the redirect would be for `http://coyote.acme.com`.

- Step 3** Use the **direct-ip** command to create the same direct IP address for the virtual server and real servers in LocalDirector.

```
localdirector(config)# direct-ip 10.1.1.10:80:0:tcp is
```

```
localdirector(config)# direct-ip 10.1.1.11:80:0:tcp is
```

The **direct-ip** command creates the same direct IP address for the virtual server and the real servers in LocalDirector and binds them together. The client then actually goes to the virtual server when redirected. If the **direct-ip** command is given only one IP address and port combination, that address is used on both the real server and the virtual server. When the client sends a request to that address, LocalDirector answers on behalf of the server with its MAC address. An optional second IP address and port combination may be given to the **direct-ip** command. If so, the first address is the virtual server and the second address is the real server. This allows server administrators the option of not changing server IP addresses or advertising internal IP addresses to the world.

- Step 4** Use the **link** command to link each IP address.

```
localdirector(config)# link coyote 10.1.1.10:80:0:tcp
```

```
localdirector(config)# link roadrunner 10.1.1.11:80:0:tcp
```

The **link** command is not necessary for HTTP redirection to work; however, it provides the means to dynamically update URLs. When a direct IP address is linked to a URL, the URL inherits the connection counters and status of the direct IP address. The connection counters are necessary if you are using the **leastconns** option of the **predictor** command; the connection counters let the predictor accurately know how many connections have currently been redirected to that particular URL. If the direct IP address is changed to out of service (OOS), the URL that it is linked to will automatically be set to OOS also.

- Step 5** Use the **backup** command to create a backup for both direct IP addresses.

```
localdirector(config)# backup 10.1.1.10:80:0:tcp 10.1.1.1:80:0:tcp
```

```
localdirector(config)# backup 10.1.1.11:80:0:tcp 10.1.1.1:80:0:tcp
```

Backups are used to prevent you from bookmarking a broken link. If you are redirected to `coyote.acme.com`, for instance, you may choose to bookmark that site. If you come back to that bookmark later and `coyote` happens to be failed, the backup will send you back to `www.acme.com`, which allows you to be redirected to a server that is in service.

## Restrictions

The following restrictions apply to HTTP redirection mode:

- URL real servers cannot be bound to virtual servers that already have an SSL sticky timeout, FTP proxy service, or any other sticky or proxy service.
- URLs can only be used as backups for virtual servers, not real servers.
- You will get an error message if you try to use the **direct-ip** command with a virtual server that has been associated with a content *rule\_name* using the **virtual** command.

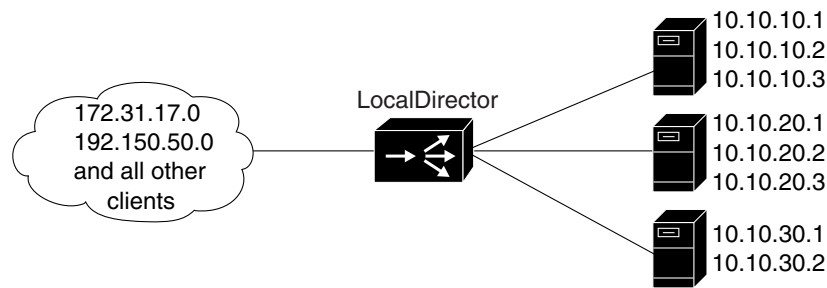
- You can only use the **direct-ip** command with TCP servers.
- Web servers must be able to dynamically create absolute URLs using the HOST field of the content (GET) request. IP addresses needed to create direct IP addresses (DIPs) must be routable or registered with the Network Information Center (NIC). URLs used must be DNS-registered unless routable IP addresses are used.

## Client-Assigned Load Balancing

Use the **assign** command to associate real servers to the bind-ids of virtual servers and direct client requests to a specific instance of a virtual server, as shown in Figure 3-7. Depending on the bindings of virtual servers to real servers, different real machines are used for different clients.

In this example, clients from 172.31.17.0 are directed through the virtual server with a bind-id of 1 to the real servers with IP addresses 10.10.20.1, 10.10.20.2, and 10.10.20.3. Client requests from 192.150.50.0 go through the virtual server with a bind-id of 2 to real servers with IP addresses 10.10.30.1 and 10.10.30.2. Requests from any other source IP address go to the virtual server with the default bind-id of 0, and that virtual server is bound to real servers 10.10.10.1, 10.10.10.2, and 10.10.10.3.

**Figure 3-7 Client-Assigned Load Balancing**



| Client Assignment            | Virtual Servers  | Real Server Bindings                            |
|------------------------------|------------------|-------------------------------------------------|
| All other requests (default) | 192.168.1.1:80:0 | 10.10.10.1:80<br>10.10.10.2:80<br>10.10.10.3:80 |
| 172.31.17.0                  | 192.168.1.1:80:1 | 10.10.20.1:80<br>10.10.20.2:80<br>10.10.20.3:80 |
| 192.150.50.0                 | 192.168.1.1:80:2 | 10.10.30.1:80<br>10.10.30.2:80                  |

11969

To configure client-assigned load balancing, perform the following steps:

- Step 1** Use the **virtual** command to create a virtual server that accepts client traffic not assigned to a specific bind-id. Use the **real** command to create three real servers. Use the **bind** command to bind three real servers to the virtual server:

```
ld(config)# virtual 192.168.1.1:80:tcp
ld(config)# real 10.10.10.1:80:0:tcp
ld(config)# real 10.10.10.2:80:0:tcp
ld(config)# real 10.10.10.3:80:0:tcp
ld(config)# bind 192.168.1.1:80:0:tcp 10.10.10.1:80:0:tcp
ld(config)# bind 192.168.1.1:80:0:tcp 10.10.10.2:80:0:tcp
ld(config)# bind 192.168.1.1:80:0:tcp 10.10.10.3:80:0:tcp
```

- Step 2** Use the **virtual** command to create a virtual server with a bind-id of :1. Use the **real** command to create three real servers. Use the **bind** command to bind three real servers to the virtual server:

```
ld(config)# virtual 192.168.1.1:80:1:tcp
ld(config)# real 10.10.20.1:80:0:tcp
ld(config)# real 10.10.20.2:80:0:tcp
ld(config)# real 10.10.20.3:80:0:tcp
ld(config)# bind 192.168.1.1:80:1:tcp 10.10.20.1:80:0:tcp
ld(config)# bind 192.168.1.1:80:1:tcp 10.10.20.2:80:0:tcp
ld(config)# bind 192.168.1.1:80:1:tcp 10.10.20.3:80:0:tcp
```

- Step 3** Use the **virtual** command to create a virtual server with a bind-id of :2. Use the **real** command to create three real servers. Use the **bind** command to bind three real servers to the virtual server:

```
ld(config)# virtual 192.168.1.1:80:2:tcp
ld(config)# real 10.10.30.1:80:0:tcp
ld(config)# real 10.10.30.2:80:0:tcp
ld(config)# real 10.10.30.3:80:0:tcp
ld(config)# bind 192.168.1.1:80:2:tcp 10.10.30.1:80:0:tcp ld(config)# bind
192.168.1.1:80:2:tcp 10.10.30.2:80:0:tcp
ld(config)# bind 192.168.1.1:80:2:tcp 10.10.30.3:80:0:tcp
```

- Step 4** Use the **assign** command to assign clients from network 172.31.17.0 to the virtual server with a bind-id of :1, and assign clients from 192.168.50.0 to the virtual server with a bind-id of :2. Using a zero in the fourth octet of the subnet mask includes the entire range for the client IP address:

```
ld(config)# assign 192.168.1.1:80:1:tcp 172.31.17.0 255.255.255.0
ld(config)# assign 192.168.1.1:80:2:tcp 192.168.50.0 255.255.255.0
```

- Step 5** Use the **is** command to put the virtual servers and real servers in service.

```
ld(config)# is virtual 192.168.1.1:80:2:tcp
ld(config)# is virtual 192.168.1.1:80:0:tcp
ld(config)# is virtual 192.168.1.1:80:1:tcp
ld(config)# is real 10.10.30.2:80:0:tcp
ld(config)# is real 10.10.30.1:80:0:tcp
ld(config)# is real 10.10.20.3:80:0:tcp
ld(config)# is real 10.10.20.2:80:0:tcp
ld(config)# is real 10.10.10.3:80:0:tcp
ld(config)# is real 10.10.10.2:80:0:tcp
ld(config)# is real 10.10.10.1:80:0:tcp
ld(config)# is real 10.10.20.1:80:0:tcp
```

**Step 6** Use the **show configuration** command to view the entire configuration:

```
ld(config)# show configuration
: Saved
: LocalDirector 420 Version 3.1.0.106
syslog output 20.3
no syslog console
enable password dfeaf10390e560aea745ccba53e044 encrypted
hostname localdirector
no shutdown ethernet 0
no shutdown ethernet 1
shutdown ethernet 2
shutdown ethernet 3
interface ethernet 0 100basetx
interface ethernet 1 100basetx
interface ethernet 2 100basetx
interface ethernet 3 100basetx
mtu 0 1500
mtu 1 1500
mtu 2 1500
mtu 3 1500
multiring all
no secure 0
no secure 1
no secure 2
no secure 3
ping-allow 0
ping-allow 1
ping-allow 2
ping-allow 3
ip address 172.20.30.81 255.255.255.0
no rip passive
failover ip address 0.0.0.0
no failover
password cisco
no snmp-server contact
no snmp-server location
virtual 192.168.1.1:80:2:tcp is
virtual 192.168.1.1:80:0:tcp is
virtual 192.168.1.1:80:1:tcp is
real 10.10.30.2:80:0:tcp is
real 10.10.30.1:80:0:tcp is
real 10.10.20.3:80:0:tcp is
real 10.10.20.2:80:0:tcp is
real 10.10.10.3:80:0:tcp is
real 10.10.10.2:80:0:tcp is
real 10.10.10.1:80:0:tcp is
real 10.10.20.1:80:0:tcp is
bind 192.168.1.1:80:2:tcp 10.10.30.2:80:0:tcp
bind 192.168.1.1:80:2:tcp 10.10.30.1:80:0:tcp
bind 192.168.1.1:80:0:tcp 10.10.10.3:80:0:tcp
bind 192.168.1.1:80:0:tcp 10.10.10.2:80:0:tcp
bind 192.168.1.1:80:0:tcp 10.10.10.1:80:0:tcp
bind 192.168.1.1:80:1:tcp 10.10.20.3:80:0:tcp
bind 192.168.1.1:80:1:tcp 10.10.20.2:80:0:tcp
bind 192.168.1.1:80:1:tcp 10.10.20.1:80:0:tcp
assign 192.168.1.1:80:2:tcp 192.168.50.0 255.255.255.0
assign 192.168.1.1:80:1:tcp 192.31.17.0 255.255.255.0
```

---

## Content Load Balancing

LocalDirector server load-balancing techniques can be maximized with the implementation of content load-balancing services. Content load balancing in LocalDirector is based on Layer 4 through Layer 7 protocol implementations such as Hypertext Transfer Protocol (HTTP) message exchanges between web servers and client browsers. LocalDirector also handles persistence integrity with HTTP 1.1 clients in a content load-balancing environment.

The content load balancing feature in LocalDirector looks for the specified rule matches within the HTTP header that are generated as a result of the URL request from the HTTP client or browser. LocalDirector parses the string containing the URL and HTTP header data and uses content-rule matches to determine which server receives the GET request from the client.

The following is an example of the complete contents of the HTTP header generated by the client and sent to the HTTP server when the URL entered is `http://www.acme.com/home/index.html`.

```
GET /home/index.html HTTP/1.0\r\n
Connection: Keep-Alive\r\n
User-Agent: Mozilla/4.74 [en] (WinNT; U)\r\n
Host: www.acme.com\r\n
Accept: Image/gif, Image/x-xbitmap, Image/jpeg..
Accept-Encoding: gzip\r\n
Accept-Language: en\r\n
Accept-Charset: iso-8859-1, *, utf-8\r\n
\r\n
```

You can configure LocalDirector to search and match content strings in the incoming HTTP header. You define the load-balancing rules that can be used by LocalDirector based upon a successful match of a content string. Also, you can use the **service** command to define LocalDirector actions for services supported through the virtual server. For example, when you use the **clb-close** argument with the **service** command, you ensure that LocalDirector uses load-balancing services for the initial content rule matched. LocalDirector ensures that the destination server receives a `Connection:close` message so that each request is load balanced to the appropriate real server for pipelined or persistent connections.

The content load balancing feature requires the assignment of a specific content rule to a virtual server at the time that virtual server is created. Each configuration of a virtual server can contain a pointer to a content rule. Wildcard matching functions are implemented with the following restrictions:

| Wildcard Character | Description of Function                                                   | Literal Input |
|--------------------|---------------------------------------------------------------------------|---------------|
| ?                  | In the content string, one character is this position can have any value. | \?            |
| *                  | Multiple characters of any value can appear in the content string.        | \*            |

LocalDirector continues to support proxy services for TCP exchanges, as well as secondary proxy services such as client-to-server persistence based on LocalDirector insertion of cookies or server-generated cookies (cookie sticky).

## Configuring a Virtual Server with Content Load Balancing

To configure a virtual server with content load balancing, perform the following steps:

- Step 1** Use the **content-rule** command to create the name and definition of the content rule for content load balancing. In the following example, the name of the content rule is `home1`, the data packet is searched to a depth of 1024 bytes, and the search should contain the URL substring `www.acme.com/home/index.htm`.

```
LocalDirector(config)# content-rule home1 depth 1024 "/home/index.htm*www.acme.com/home/"
```



**Note** Do not create a rule name that contains only the asterisk (\*) wildcard.

- Step 2** Use the **virtual** command to define the virtual server and specify the content rule that is associated with this virtual server.

```
LocalDirector(config)# virtual 10.10.10.1:442:1:tcp:is
```

- Step 3** Optionally, use the **virtual** command to also define a default virtual server. This default virtual server receives all traffic that does not match any of the content rules defined for a particular virtual-id.

```
LocalDirector(config)# virtual 10.10.10.12:0:0:tcp is
```

If you create a default virtual server, the first virtual server in the list displayed with the **show virtual** command is the default virtual server. All packets for this virtual server that do not contain a pattern match to one of the content rules are directed to the default virtual server.

If you do not create a default virtual server and no content match is found, Local Director sends a TCP RESET to the client, and the connection is closed because no rule was matched and a default rule did not exist.

- Step 4** Optionally, use the **show rule** command to display information for all content rules defined for this LocalDirector. Note that the display for all rule names containing more than 48 characters is abbreviated with three dots (...).

```
LocalDirector(config)# show rule
Rule Name Depth Content to Match
gold001 1024 "/files/customer/gold/?????.lst"
spec204 2048 "/public/info/sales/corporate/
filesys/gateway/lev..."
images:152 256 "/files/images/*.gif"
images:153 256 "/files/images/*.jpg"
```

You can also use the **show rule** command to show the entire rule by adding the rulename to the command.

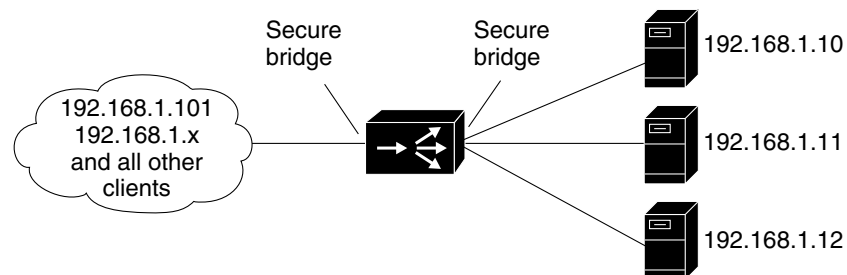
```
LocalDirector(config)# show rule spec204
Rule Name Depth Content to Match
spec204 2048 "/public/info/sales/corporate/filesys/gateway/leveraged/index"
```

## Secure Services

The example shown in Figure 3-8 uses LocalDirector security features of SecureAccess, SecureBind, and SecureBridge to secure a network both internally and externally.

In this example, the company has a Class C network with an IP address of 192.168.1.0. The virtual address in DNS is 192.168.1.8 for www.domain.com. Requests from internal clients are directed to different real servers. The IP address of the system administrator is 192.168.1.101.

**Figure 3-8 Secure Services**



| Client Assignment                              | Virtual Servers                                                | Real Server Bindings                                        |
|------------------------------------------------|----------------------------------------------------------------|-------------------------------------------------------------|
| All other requests (default)                   | 192.168.1.8:80:0                                               | 192.168.1.10:80<br>192.168.1.11:80<br>192.168.1.12:80       |
| Assigned to internal clients 192.168.1.x       | 192.168.1.8:80:1                                               | 192.168.1.10:8000<br>192.168.1.11:8000<br>192.168.1.12:8000 |
| Assigned to system administrator 192.168.1.101 | 192.168.1.8:2310:1<br>192.168.1.8:2311:1<br>192.168.1.8:2312:1 | 192.168.1.10:23<br>192.168.1.11:23<br>192.168.1.12:23       |

12014

To configure secure services, perform the following steps:

**Step 1** Use the **secure** command to turn on the SecureBridge feature:

```
ld(config)# secure 0
ld(config)# secure 1
```

**Step 2** Use the **virtual** command to create a virtual address for Internet web access:

```
ld(config)# virtual 192.168.1.8:80:0:tcp is
```

**Step 3** Use the **virtual** command to create a virtual address for internal web access:

```
ld(config)# virtual 192.168.1.8:80:1:tcp is
```

**Step 4** Use the **virtual** command to create virtual addresses for the system administrator to access the real servers through a Telnet session:

```
ld(config)# virtual 192.168.1.8:2310:1 is
ld(config)# virtual 192.168.1.8:2311:1 is
ld(config)# virtual 192.168.1.8:2312:1 is
```

**Step 5** Use the **real** command to define the three real servers that are serving the Internet web:

```
ld(config)# real 192.168.1.10:80:0:tcp is
ld(config)# real 192.168.1.11:80:0:tcp is
ld(config)# real 192.168.1.12:80:0:tcp is
```

**Step 6** Use the **real** command to define the three real servers that are serving the internal web:

```
ld(config)# real 192.168.1.10:8000:0:tcp is
ld(config)# real 192.168.1.11:8000:0:tcp is
ld(config)# real 192.168.1.12:8000:0:tcp is
```

**Step 7** Use the **real** command to define the three real servers that provide Telnet access for the system administrator:

```
ld(config)# real 192.168.1.10:23 is
ld(config)# real 192.168.1.11:23 is
ld(config)# real 192.168.1.12:23 is
```

**Step 8** Use the **bind** command to bind the system administrator virtual addresses to the real servers:

```
ld(config)# bind 192.168.1.8:2310:1 192.168.1.10:23
ld(config)# bind 192.168.1.8:2311:1 192.168.1.11:23
ld(config)# bind 192.168.1.8:2312:1 192.168.1.12:23
```

**Step 9** Use the **bind** command to bind the Internet web server virtual address to the real servers:

```
ld(config)# bind 192.168.1.8:80:0:tcp 192.168.1.10:80:tcp
ld(config)# bind 192.168.1.8:80:0:tcp 192.168.1.11:80:tcp
ld(config)# bind 192.168.1.8:80:0:tcp 192.168.1.12:80:tcp
```

**Step 10** Use the **bind** command to bind the internal web server virtual address to the real servers:

```
ld(config)# bind 192.168.1.1:80:1:tcp 192.168.1.10:8000:0:tcp
ld(config)# bind 192.168.1.1:80:1:tcp 192.168.1.11:8000:0:tcp
ld(config)# bind 192.168.1.1:80:1:tcp 192.168.1.12:8000:0:tcp
```

**Step 11** Use the **assign** command to give only the system administrator access to the Telnet virtual addresses. The subnet mask of 255.255.255.255 restricts access to the individual IP address.

```
ld(config)# assign 192.168.1.8:2310:1 192.168.1.101 255.255.255.255
ld(config)# assign 192.168.1.8:2311:1 192.168.1.101 255.255.255.255
ld(config)# assign 192.168.1.8:2312:1 192.168.1.101 255.255.255.255
```

**Step 12** Use the **assign** command to give only internal clients access to the virtual addresses for the internal web. The subnet mask of 255.255.255.0 restricts access to the network IP address:

```
ld(config)# assign 192.168.1.8:80:1:tcp 192.168.1.0 255.255.255.0
```

## Port-Bound Servers

Both TCP and UDP services can be directed to specific servers. Figure 3-9 shows how to send HTTP traffic to server A, send Telnet traffic to server B, and direct all other traffic to servers C and D. Three virtual servers have IP address 192.168.1.100; one accepts only HTTP traffic (port 80), one accepts Telnet traffic (port 23), and the other accepts all other connections (default).

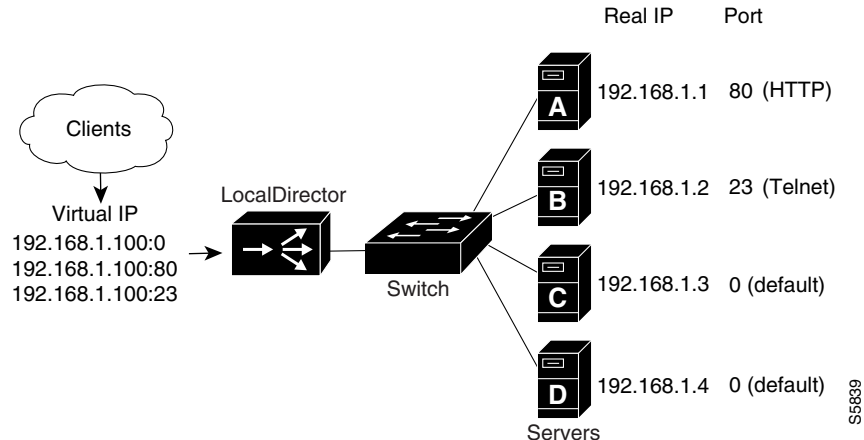


### Note

If you do not specify a port when defining a server, the port is listed as the default. The default port for a server accepts all network connections, except for those directed to a specific port by using the port binding feature.

Names can also be used to refer to the real and virtual servers in this example.

Figure 3-9 Application-Specific Servers



To configure port-bound servers, perform the following steps:

- Step 1** Use the **real** command to identify a real server accepting connections on port 80, a real server accepting connections on port 23, and two real servers accepting default traffic:

```
ld(config)# real 192.168.1.1:80 is
ld(config)# real 192.168.1.2:23 is
ld(config)# real 192.168.1.3 is
ld(config)# real 192.168.1.4 is
```

- Step 2** Use the **virtual** command to identify three virtual servers for IP address 192.168.1.100: one accepting connections on port 80, one accepting connections on port 23, and one accepting default traffic:

```
ld(config)# virtual 192.168.1.100:80 is
ld(config)# virtual 192.168.1.100:23 is
ld(config)# virtual 192.168.1.100 is
```

- Step 3** Use the **bind** command to direct HTTP traffic for virtual server 192.168.1.100, port 80 to real server 192.168.1.1, port 80:

```
ld(config)# bind 192.168.1.100:80 192.168.1.1:80
```

- Step 4** Use the **bind** command to direct Telnet traffic for virtual server 192.168.1.100, port 23 to real server 192.168.1.2, port 23:

```
ld(config)# bind 192.168.1.100:23 192.168.1.2:23
```

**Step 5** Use the **bind** command to direct all other connections for virtual server 192.168.1.100 to real servers 192.168.1.3 and 192.168.1.4:

```
ld(config)# bind 192.168.1.100 192.168.1.3 192.168.1.4
```

**Step 6** Use the **show bind** command to display the association between the virtual and real servers:

```
ld(config)# show bind
Virtual Real
192.168.1.100:0:0:TCP(IS) 192.168.1.4:0:0:TCP(IS)
 192.168.1.3:0:0:TCP(IS)
192.168.1.100:23:0:TCP(IS) 192.168.1.2:23:0:TCP(IS)
192.168.1.100:80:0:TCP(IS) 192.168.1.1:80:0:TCP(IS)
```

---

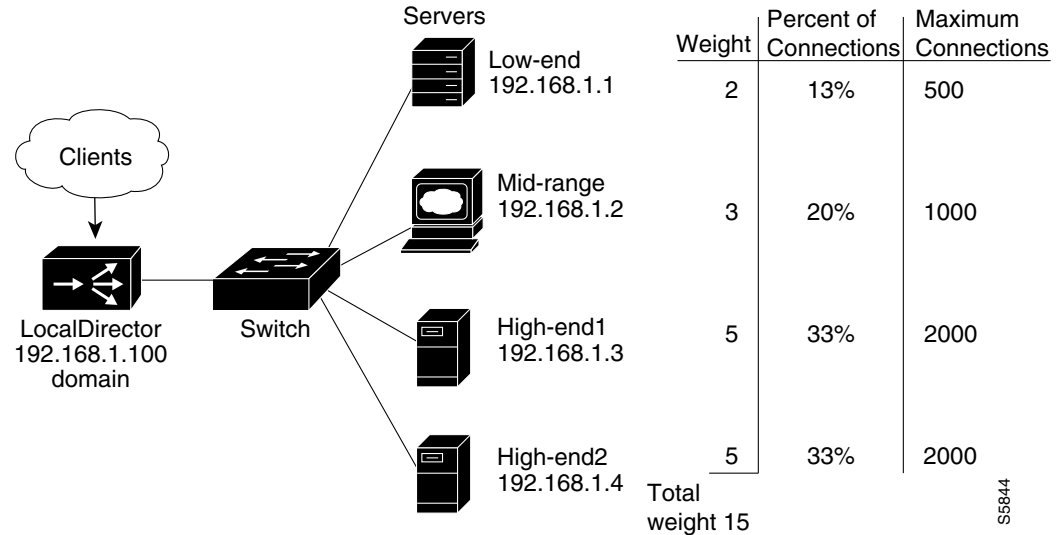
## Maximum Connections and Weighted Configuration

The **maxconns** command allows you to specify the maximum number of connections that each real server can have at one time. A server administrator can set the maximum connections to a level that avoids exceeding the capacity threshold of the server. Often, server administrators have a good idea of the load that a server can bear, and the **maxconns** command can be used to prevent a server from failing because of capacity overload. Clients requesting connections to a server farm with no available connections receive a timeout message.

A higher percentage of connections can be directed to servers that offer increased performance by selecting the **weighted** option of the **predictor** command and setting values with the **weight** command.

Figure 3-10 shows four servers with varying performance indices, maximum connections settings, and weight values. In this example, a weight of 2 is assigned to the low-end server, sending 13 percent of the connections to that server. This particular server cannot accept more than 500 simultaneous connections, so **maxconns** is set to 500. The same reasoning applies to the midrange server and the two high-end servers.

Figure 3-10 Maximum Connections and Weighted Performance



To configure maximum connections and weighted performance, perform the following steps:

- Step 1** Use the **virtual** command to identify 192.168.1.100 as a virtual server. Use the **is** option to put the server in service:

```
ld(config)# virtual 192.168.1.100 is
```

- Step 2** Use the **name** command to associate a name with the virtual server:

```
ld(config)# name 192.168.1.100 domain
```

- Step 3** Use the **real** command to identify four real servers. Use the **is** option to put the servers in service:

```
ld(config)# real 192.168.1.1 is
ld(config)# real 192.168.1.2 is
ld(config)# real 192.168.1.3 is
ld(config)# real 192.168.1.4 is
```

- Step 4** Use the **name** command to associate names with the real servers:

```
ld(config)# name 192.168.1.1 low-end
ld(config)# name 192.168.1.2 mid-range
ld(config)# name 192.168.1.3 high-end1
ld(config)# name 192.168.1.4 high-end2
```

- Step 5** Use the **bind** command to direct traffic for virtual server domain to real servers low-end, mid-range, high-end1, and high-end2:

```
ld(config)# bind domain low-end mid-range high-end1 high-end2
```

- Step 6** Use the **predictor** command to set load balancing with the **weighted** option:

```
ld(config)# predictor domain weighted
```

- Step 7** Use the **weight** command to assign weight values to each of the real servers:

```
ld(config)# weight low-end 2
ld(config)# weight mid-range 3
ld(config)# weight high-end1 5
ld(config)# weight high-end2 5
```

**Step 8** Use the **maxconns** command to limit the number of connections that each real server can accept:

```
ld(config)# maxconns low-end 500
ld(config)# maxconns mid-range 1000
ld(config)# maxconns high-end1 2000
ld(config)# maxconns high-end2 2000
```

**Step 9** Use the **show bind** command to display the association between the virtual and real servers:

```
ld(config)# show bind
Virtual Machines Real Machines
 192.168.1.100:tcp (IS)
 192.168.1.1:tcp (IS)
 192.168.1.2:tcp (IS)
 192.168.1.3:tcp (IS)
 192.168.1.4:tcp (IS)
```

**Step 10** Use the **show weight** command to display the weight values assigned to the real servers:

```
ld(config)# show weight
Real Machine(s) Weight Timeout
192.168.1.1:tcp 2 never
192.168.1.2:tcp 3 never
192.168.1.3:tcp 5 never
192.168.1.4:tcp 5 never
```

## Backup Configurations

To ensure that TCP services continue to run if a server has failed or is out of service, you can identify an alternative destination for server traffic by specifying a backup. The term “backup” is used to define a hot-standby for a real or virtual server defined on LocalDirector. The backup can be a virtual or real server; thus, it is possible to use the **backup** command in any combination.

For real servers, a backup is used if the real server has failed or is out of service. For a virtual server, a backup is used if all real servers (and their backups) bound to the virtual server have failed or are out of service. If the virtual server itself is out of service, a TCP RST is sent to the client requesting the connection.



### Note

A server cannot be used as a backup for itself. For example, a real server cannot serve as a backup for a virtual server to which it is bound. If this configuration is attempted, an error message is generated.

When the server being backed up returns to service, connections are no longer directed to the backup server and they are sent according to the LocalDirector configuration.

You can back up real servers with virtual addresses, and you can back up virtual servers with a real server. You can use a backup server when the real or virtual server is not in service (for example, it has failed or is out of service).



### Note

If the backup server itself is not available, LocalDirector does not check the backup of the backup server. The backup feature checks only one level.

The following example shows a virtual server backing up another virtual server. If real servers 10.1.1.1 and 10.1.1.2 fail (thus failing virtual server 10.10.10.10), the real servers bound to virtual server 10.10.10.20 are used instead.

```
ld(config)# virtual 10.10.10.10
ld(config)# real 10.1.1.1
ld(config)# real 10.1.1.2
ld(config)# bind 10.10.10.10 10.1.1.1 10.1.1.2
ld(config)# virtual 10.10.10.20
ld(config)# real 10.1.1.3
ld(config)# real 10.1.1.4
ld(config)# real 10.1.1.5
ld(config)# bind 10.10.10.20 10.1.1.3 10.1.1.4 10.1.1.5
ld(config)# backup 10.10.10.10 10.10.10.20
```

In the following example, real server 10.1.1.3 is backing up the two real servers bound to virtual server 10.10.10.10:

```
ld(config)# virtual 10.10.10.10
ld(config)# real 10.1.1.1
ld(config)# real 10.1.1.2
ld(config)# bind 10.10.10.10 10.1.1.1 10.1.1.2
ld(config)# real 10.1.1.3
ld(config)# backup 10.10.10.10 10.1.1.3
```

The same result can be achieved with the following configuration, in which real server 10.1.1.3 backs up the two real servers directly, instead of backing up the virtual server:

```
ld(config)# virtual 10.10.10.10
ld(config)# real 10.1.1.1
ld(config)# real 10.1.1.2
ld(config)# bind 10.10.10.10 10.1.1.1 10.1.1.2
ld(config)# real 10.1.1.3
ld(config)# backup 10.1.1.1 10.1.1.3
ld(config)# backup 10.1.1.2 10.1.1.3
```

## Configuring with the data Command

Some web servers (such as Microsoft Windows NT 4.0) continue to establish connections to a real server even though the application daemon is not functioning. Use the **data** command to limit the number of connections sent to a server that is not sending data.

LocalDirector tracks connections and can detect when clients are requesting data and the server is not responding with data. If the DataIn Conns value of a server exceeds the defined threshold and all other real servers bound to the virtual server are at less than 80 percent of their DataIn capacity, the server is failed. Display the DataIn Conns value with the **show real** command.

## Configuring UDP Traffic

This section describes how to set up UDP traffic on the virtual servers and real servers, and a mix of TCP and UDP traffic on the virtual servers and real servers.

Configuration for UDP is similar to TCP, except the **udp** protocol option is used instead of the **tcp** option in the *virtual\_id* and *real\_id* server parameters. Additionally, because UDP is a connectionless protocol, the connection is kept alive through use of a timer.

The connection duration timer starts on the first packet of the flow that is received on the virtual server. Set the timer with the **timeout** command, as shown in the following example. Connections remain valid until the timer times out because of inactivity on the connection flow.

To set up a UDP configuration for a virtual server and a real server, perform the following steps:

- 
- Step 1** Use the **virtual** command to identify 192.10.10.101 as a virtual server running UDP through port 300. Use the **is** option to put the server in service.
- ```
ld(config)# virtual 192.10.10.101:300:0:udp is
```
- Step 2** Use the **timeout** command to set the connection duration period on the virtual server to 11 minutes.
- ```
ld(config)# timeout 192.10.10.101:300:0:udp 11
```
- Step 3** Use the **real** command to identify 192.10.10.1:300:0 as a real server running UDP through port 300. Use the **is** option to put the server in service.
- ```
ld(config)# real 192.10.10.1:300:0:udp is
```
- Step 4** Use the **bind** command to associate the virtual server to the real server:
- ```
ld(config)# bind 192.10.10.101:300:0:udp 192.10.10.1:300:0:udp
```
- 

When an application can use a known connection port, the **buddy** command can be used to group the connections, ensuring that activity on one stream updates timers on all streams.

To set up a buddy group that contains TCP and UDP virtual servers representing TCP and UDP real servers, perform the following steps:

- 
- Step 1** Use the **virtual** command to identify 10.0.0.100 as a virtual server running TCP and UDP. Use the **is** option to put the server in service.
- ```
ld(config)# virtual 10.0.0.100:0:0:tcp is
ld(config)# virtual 10.0.0.100:0:0:udp is
```
- Step 2** Use the **buddy** command to create the buddy group tcp-udp and add the TCP and UDP virtual servers to it:
- ```
ld(config)# buddy tcp-udp 10.0.0.100:0:0:tcp
ld(config)# buddy tcp-udp 10.0.0.100:0:0:udp
```

- Step 3** Use the **real** command to identify three real servers running both TCP and UDP. Use the **is** option to put the servers in service.

```
ld(config)# real 10.0.0.1:0:0:tcp is
ld(config)# real 10.0.0.1:0:0:udp is
ld(config)# real 10.0.0.2:0:0:tcp is
ld(config)# real 10.0.0.2:0:0:udp is
ld(config)# real 10.0.0.3:0:0:tcp is
ld(config)# real 10.0.0.3:0:0:udp is
```

- Step 4** Use the **bind** command to associate the virtual servers to the real servers. Notice the TCP virtual servers are bound to the TCP real servers.

```
ld(config)# bind 10.0.0.100:0:0:tcp 10.0.0.3:0:0:tcp
ld(config)# bind 10.0.0.100:0:0:tcp 10.0.0.2:0:0:tcp
ld(config)# bind 10.0.0.100:0:0:tcp 10.0.0.1:0:0:tcp
ld(config)# bind 10.0.0.100:0:0:udp 10.0.0.3:0:0:udp
ld(config)# bind 10.0.0.100:0:0:udp 10.0.0.2:0:0:udp
ld(config)# bind 10.0.0.100:0:0:udp 10.0.0.1:0:0:udp
```

## Configuring Accelerated Server Load Balancing

Accelerated server load balancing (ASLB) requires two Ethernet links between LocalDirector and the Catalyst 6000 family switch. One link must be connected to the VLAN that the router is on (VLAN 10 in Figure 3-11). The other link must be connected to the VLAN that the real servers are on (VLAN 20 in Figure 3-11).

To set up ASLB according to the information in Figure 3-11, perform the following steps:

- Step 1** Use the **virtual** command to identify 192.168.201.55:80 as a virtual server:

```
ld(config)# virtual 192.168.201.55:80
```

- Step 2** Use the **real** command to identify the three servers:

```
ld(config)# real 192.168.201.1:80
ld(config)# real 192.168.201.2:80
ld(config)# real 192.168.201.3:80
```

- Step 3** Use the **bind** command to associate the virtual server to the real servers:

```
ld(config)# bind 192.168.201.55:80 192.168.201.1:80
ld(config)# bind 192.168.201.55:80 192.168.201.2:80
ld(config)# bind 192.168.201.55:80 192.168.201.3:80
```

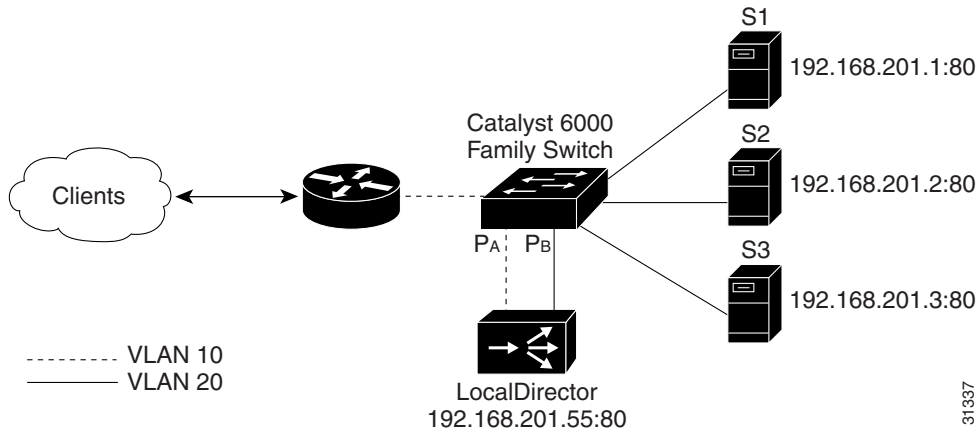
- Step 4** Use the **predictor** command to set load balancing with the **roundrobin** option:

```
ld(config)# predictor 192.168.201.55:80 roundrobin
```

- Step 5** Use the **redirection** command to enable ASLB for the virtual server with the **dispatch assisted** option:

```
ld(config)# redirection 192.168.201.55:80 dispatch assisted
```

Figure 3-11 Sample ASLB Configuration



### ASLB Packet Flow

When an inbound connection synchronization (TCP SYN) packet arrives with a destination MAC address of the LocalDirector virtual server in the Layer 3 header, the switch forwards the packet to LocalDirector at port P<sub>A</sub>. (See Figure 3-11.) LocalDirector makes the load balancing decision, changes the destination MAC address to that of the real server, and forwards the packet to the switch at port P<sub>B</sub>.



**Note**

LocalDirector inserts its own MAC address in the SYN packet before forwarding the packet to the switch. In standalone mode, LocalDirector always preserves original MAC addresses when forwarding packets to the switch.

The switch creates an inbound ASLB Multi-Layer Switching (MLS) entry in the Layer 3 forwarding tables as it forwards the packet to the real server. All subsequent packets that match the inbound ASLB MLS entry are Layer 3-switched (accelerated) to the real server, unless the packet is fragmented or contains a connection finish (TCP FIN) or connection reset (TCP RST).

When the outbound SYN packet from the real server arrives, the switch forwards the packet to LocalDirector at port P<sub>B</sub>. LocalDirector changes the source MAC address to that of the LocalDirector virtual server and forwards the packet to the switch at port P<sub>A</sub>. The switch creates an outbound ASLB MLS entry in the Layer 3 forwarding table as it forwards the packet to the router. All subsequent packets that match the outbound ASLB MLS entry are switched directly to the router, unless the packet is fragmented or contains a FIN or an RST.

Packets containing a FIN or an RST travel the same path as a SYN packet. The switch purges the inbound ASLB MLS entry when it forwards the FIN or RST packet to LocalDirector. The switch purges the outbound ASLB MLS entry as it forwards the FIN or RST packet to LocalDirector.



**Note**

LocalDirector must be in dispatch assisted mode for ASLB to work. Use the **redirection** command to set dispatch assisted mode.

## Integrated Probe for DNS Support

The integrated probe for DNS feature allows LocalDirector to automatically fail or recover real servers that are running DNS servers, based on probe results. Probes are constantly sent to the DNS servers to determine their status. If a DNS server fails to respond to a certain number of probes, it is marked as FAILED. As soon as the DNS server starts responding to DNS probes again, it is returned to the in-service state.

### Configuring the Integrated Probe for DNS

In the following configuration example, LocalDirector is configured with two virtual servers 10.10.10.10 and 10.10.10.20; and three real servers 10.10.10.50, 10.10.10.60, and 10.10.10.70. Real servers 10.10.10.50 and 10.10.10.60 are bound to virtual server 10.10.10.10; and real servers 10.10.10.60 and 10.10.10.70 are bound to virtual server 10.10.10.20.

To activate the DNS probe, perform the following steps:

- 
- Step 1** Use the **probe virtual** command to enable LocalDirector DNS probe requests to be sent every 10 seconds.

```
LocalDirector(config)# probe virtual 10.10.10.10 dns 1
```

You can only configure a DNS probe on a virtual or real server that uses the User Datagram Protocol (UDP). You need to specify a real server with the **probe real** command to override the interval time set with the **probe virtual** command.

- Step 2** Use the **probedns nodename** command to configure the name of the node to be used in the DNS query requests sent to the DNS servers.

```
LocalDirector(config)# probedns nodename server1.com
```

- Step 3** Use the **probeconfig** command to configure the threshold value that limits the number of DNS requests sent without a response. When this threshold is exceeded, LocalDirector marks the DNS server as failed.

```
LocalDirector(config)# probeconfig dns 5
```

---

## Hypertext Transfer Protocol Probe Support

LocalDirector can now validate the activity of web servers, also known as HTTP services, running within a LocalDirector server farm. Although a machine may still be enabled and running, an HTTP service on a particular server or port may be disabled for some reason. LocalDirector can determine the activity of web servers (HTTP services) by inspecting specific information returned by the server to the client that is initiating a request for a web page or service. By looking or probing for specific information, LocalDirector can determine whether the application is still running, and take appropriate action if it is not.

LocalDirector supports two types of HTTP request methods: GET and HEAD. A GET request, which is the most common type of HTTP request, receives header information followed by the contents of the page being displayed. A HEAD request receives only the header information. Header information

includes items such as the return code and possibly other header information. Because the vital information that probes are looking for is in the header, the default request method that the probes use is the HEAD method.

## Configuring LocalDirector for HTTP Probe Support

To configure LocalDirector for HTTP probing, perform the following steps:

- 
- Step 1** Make sure that the machine (web server) that you want to probe has already been configured within the LocalDirector server farm.  
  
The web server that you want to probe must be defined as a real server within LocalDirector and must be associated with (bound to) a virtual server. This allows the defined probe to probe that virtual server, which in turn sends probes to all other real servers that are bound to the virtual server.
  - Step 2** Use the **probe** command to create a specific probe type for the machine you want to probe, and to set the probing interval.  
  
Optionally, use the **show probe** command to display information about both HTTP and DNS probes. Use the **no probe** command to remove a probe type from a machine.
  - Step 3** Use the **probehttp** command to change any of the default HTTP probe settings.
  - Step 4** Optionally, use the **show probehttp** command to display information specific to HTTP probes, such as the file being probed, the expected return code, and the request type.
  - Step 5** Use the **probeconfig** command to enable probing and set the threshold for HTTP or DNS probes.
  - Step 6** Optionally, use the **show probeconfig** command to display the threshold values of HTTP and DNS probes.
  - Step 7** Use the **no probeconfig** command to stop or disable either HTTP or DNS probes.
- 

## Simple Network Time Protocol Support

Some networks may require synchronized timekeeping between a set of distributed clients and servers. The Simple Network Time Protocol (SNTP) provides this type of synchronized timekeeping. Support for SNTP allows LocalDirector to receive periodic updates to its internal time from time servers, thus letting LocalDirector send, receive, and process SNTP packets as an SNTP client. You can run SNTP in client mode or client broadcast mode.

### Client Mode

When you run SNTP in client mode on LocalDirector, a time request is created that is passed to the network. This request is then sent to the first server that you previously defined using the **sntp server** command. When the server receives the time request, the raw 64-bit SNTP timestamp is converted to the appropriate time format. In this mode, the interval between time requests is defined by using the **sntp poll** command.

### Broadcast Client Mode

When you run SNTP in broadcast client mode on LocalDirector, LocalDirector acts as an SNTP client and listens for timestamp updates on a defined multicast address (224.0.1.1) from a server. The timestamp is converted to the appropriate time format. No messages are sent by the client.

Client mode and broadcast client mode are mutually exclusive. Changing from client mode to broadcast client mode erases any servers you previously defined using the **sntp server** command.

## Configuring LocalDirector for SNTP Client Mode

To configure LocalDirector for SNTP client mode, perform the following steps:

- Step 1** Use the **sntp server** command to set the IP address of the servers to which the SNTP time requests are being sent.

```
localdirector(config)# sntp server 172.16.4.4
```

- Step 2** Use the **sntp poll** command to set the interval in seconds between the time requests that are sent to the server.

```
localdirector(config)# sntp poll 16
```

- Step 3** Use the **show sntp** command to confirm SNTP status and settings.

```
localdirector (config)# show sntp
Current time: Tues Mar 6 2001, 20:39:57 UTC
Last SNTP update: Mon Mar 5 2001, 22:45:47 UTC
from server:172.16.4.4
Poll interval:16
Broadcast client mode: disabled
```

## Configuring LocalDirector for SNTP Broadcast Client Mode

To configure LocalDirector for SNTP broadcast client mode, perform the following steps:

- Step 1** Use the **sntp broadcast client** command to configure LocalDirector as an SNTP broadcast client.

```
localdirector(config)# sntp broadcast client
```

- Step 2** Use the **show sntp** command to confirm SNTP status and settings.

```
Current time: Tues Mar 6 2001, 20:39:57 UTC
Last SNTP update: Mon Mar 5 2001, 22:45:47 UTC
from server:213.188.64.14
Poll interval:16
Broadcast client mode: enabled
```

## LocalDirector as a Boomerang Content Routing Agent

Content routing routes user requests to the replicated-content site (typically a mirror site) that can serve them most quickly and efficiently. A Content Router, such as the Cisco Content Router 4400, routes a content routing agent (also known as client) to the “closest” (best) replicated-content site, based on network delay using a software process called boomerang. In LocalDirector Version 4.2.1, you can set up LocalDirector to be a content routing agent using boomerang software.

The Content Router can be deployed on a network in two different ways. It can be set up in direct mode, which uses Domain Name System (DNS). Alternatively, LocalDirector can be set up in Web Cache Communication Protocol (WCCP) mode. Both deployments involve setting up a content routing agent at each content site within each domain you want the content router to support. Content routing agents are machines (such as LocalDirector) that have been properly configured with boomerang software. The Content Router must also be configured with boomerang commands.

For more information about setting up the boomerang Content Router 4400, refer to the *Cisco Content Router 4400 User Guide*.

## Configuring the LocalDirector Content Routing Agent

To set up LocalDirector as a content routing agent, perform the following steps.



### Note

To complete Step 1 and Step 2, refer to the chapter titled “Configuring the System Software” in the *Cisco Content Router 4400 User Guide*. This document is also available at the following URL on Cisco.com:  
<http://www.cisco.com/univercd/cc/td/doc/product/webscale/cr/cr4400/cr4400ug/index.htm>

- 
- Step 1** Make sure that a Content Router, which is configured with boomerang software and resides elsewhere on the network, has been initialized and configured for direct mode or WCCP mode.
  - Step 2** Make sure that a shared keyword has been created using the **key** command. This command is used to encrypt data that is sent between the Content Router and the LocalDirector content routing agent.
  - Step 3** Use the **dns-boomerang enable** command to enable the boomerang software on LocalDirector.



### Note

You can create routing agent domains using the **dns-boomerang client** command without first enabling boomerang on LocalDirector. However, the agent will not respond to incoming boomerang messages from the Content Router until the agent is enabled.

- 
- Step 4** Optionally, use the **show dns-boomerang enable** command to show whether boomerang is enabled on LocalDirector.
  - Step 5** Use the **dns-boomerang client** command to configure a domain on LocalDirector and set up LocalDirector as a content routing agent.
  - Step 6** Optionally, use the **show dns-boomerang client** command to display configuration information about all content routing agents that are configured for a Content Router.
  - Step 7** Optionally, use the **show dns-boomerang counters** command to show the current counters maintained for each LocalDirector agent domain.
  - Step 8** Optionally, use the **clear dns-boomerang counters** command to clear the counters maintained for each LocalDirector agent domain.
-