Cisco Content Services Switch Basic Configuration Guide

Software Version 7.10
December, 2002

Corporate Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 526-4100

Customer Order Number: DOC-7813886=
Text Part Number: 78-13886-03

Cisco Content Services Switch Basic Configuration Guide

CHAPTER 3

INDEX

FIGURES

About This Guide

This guide provides instructions for the basic configuration of the Cisco 11500 or 11000 series content services switch (hereinafter referred to as the CSS). Information in this guide applies to all CSS models except where noted. For information on CSS administration, refer to the *Cisco Content Services Switch Administration Guide*. For configuration information on advanced features, refer to the *Cisco Content Services Switch Advanced Configuration Guide.*

The CSS software is available in a Standard or Enhanced feature set. The Enhanced feature set contains all of the Standard feature set and also includes Network Address Translation (NAT) Peering, Domain Name Service (DNS), Demand-Based Content Replication (Dynamic Hot Content Overflow), Content Staging and Replication, and Network Proximity DNS. Proximity Database and SSH are optional features.

Note You must enter a software license key when you boot the CSS for the first time. After you boot the CSS, you can activate a CSS software option (for example, SSH) that you purchased using the **license** command. For more information, refer to the *Cisco Content Services Switch Hardware Installation Guide*, Chapter 3, Booting and Configuring the CSS.

Audience

This guide is intended for the following trained and qualified service personnel who are responsible for configuring the CSS:

- Web master
- System administrator
- System operator

How to Use This Guide

This section describes the chapters and contents in this guide.

Chapter	Description
Chapter 1, Configuring Services	Create and configure services. This chapter also contains an overview on the association between services, owners, and content rules.
Chapter 2, Configuring Owners	Create and configure owners.
Chapter 3, Configuring Content Rules	Create and configure content rules.
Chapter 4, Configuring Sticky Parameters for Content Rules	Configure sticky parameters for content rules.
Chapter 5, Configuring Source Groups, ACLs, EQLs, URQLs, NQLs, and DQLs	Configure source groups, Access Control Lists, Extension Qualifier Lists, and Uniform Resource Locator Qualifier Lists, Network Qualifier Lists, and Domain Qualifier Lists.
Chapter 6, Configuring HTTP Header Load Balancing	Configure HTTP header load balancing.
Chapter 7, Configuring Caching	Configure content caching.

Related Documentation

In addition to this document, the Content Services Switch documentation set includes the following:

Document Title	Description
Release Note for the Cisco 11500 Series Content Services Switch	This release note provides information on operating considerations, caveats, and CLI commands for the Cisco 11500 series CSS.
Release Note for the Cisco 11000 Series Content Services Switch	This release note provides information on operating considerations, caveats, and CLI commands for the Cisco 11000 series CSS.
Cisco 11500 Series Content Services Switch Hardware Installation Guide	This guide provides information for installing, cabling, and booting the 11500 series CSS. In addition, this guide provides information about CSS specifications, cable pinouts, and troubleshooting.
Cisco 11000 Series Content Services Switch Getting Started Guide	This guide provides information for installing, cabling, and booting the 11000 series CSS. In addition, this guide provides information about CSS specifications, cable pinouts, and troubleshooting.

Document Title	Description
Cisco Content Services Switch Administration Guide	This guide describes how to perform administration tasks on the CSS including logging into the CSS, upgrading your CSS software, and configuring the following: • Management ports, interfaces, and circuits • DNS, ARP, RIP, IP, and bridging features • OSPF • Logging, including displaying log messages and interpreting sys.log messages • User profile and CSS parameters • SNMP • RMON • Offline Diagnostic Monitor (Offline DM) menu
Cisco Content Services Switch Advanced Configuration Guide	This guide describes how to perform advanced CSS configuration tasks, including: • Domain Name Service (DNS) • DNS Sticky • Content Routing Agent • Client Side Accelerator • Network proximity • VIP and virtual IP interface redundancy • Box-to-box redundancy • Demand-based content replication and content staging and replication • Secure Socket Layer (SSL) termination with the SSL Acceleration Module • Firewall load balancing • CSS scripting language

Document Title	Description
Cisco Content Services Switch Command Reference	Provides an alphabetical list of all CSS Command Line Interface commands including syntax, options, and related commands.
Cisco Content Services Switch Device Management User's Guide	Provides an overview on using the Device Management user interface, an HTML-based Web application that you use to configure and manage a CSS.

Symbols and Conventions

This guide uses the following symbols and conventions to identify different types of information.

Caution A caution means that a specific action you take could cause a loss of data or adversely impact use of the equipment.

Warning **A warning describes an action that could cause you physical harm or damage the equipment.**

Note A note provides important related information, reminders, and recommendations.

Bold text indicates a command in a paragraph.

`Courier text` indicates text that appears on a command line, including the CLI prompt.

`Courier bold text` indicates commands and text you enter in a command line.

Italics text indicates the first occurrence of a new term, book title, and emphasized text.

1. A numbered list indicates that the order of the list items is important.
    a. An alphabetical list indicates that the order of the secondary list items is important.

- A bulleted list indicates that the order of the list topics is unimportant.
  - An indented list indicates that the order of the list subtopics is unimportant.

Obtaining Documentation

These sections explain how to obtain documentation from Cisco Systems.

World Wide Web

You can access the most current Cisco documentation on the World Wide Web at this URL:

http://www.cisco.com

Translated documentation is available at this URL:

http://www.cisco.com/public/countries_languages.shtml

Documentation CD-ROM

Cisco documentation and additional literature are available in a Cisco Documentation CD-ROM package, which is shipped with your product. The Documentation CD-ROM is updated monthly and may be more current than printed documentation. The CD-ROM package is available as a single unit or through an annual subscription.

Ordering Documentation

You can order Cisco documentation in these ways:

- Registered Cisco.com users (Cisco direct customers) can order Cisco product documentation from the Networking Products MarketPlace:

  http://www.cisco.com/cgi-bin/order/order_root.pl

- Registered Cisco.com users can order the Documentation CD-ROM through the online Subscription Store:

  http://www.cisco.com/go/subscription

- Nonregistered Cisco.com users can order documentation through a local account representative by calling Cisco Systems Corporate Headquarters (California, U.S.A.) at 408 526-7208 or, elsewhere in North America, by calling 800 553-NETS (6387).

Documentation Feedback

You can submit comments electronically on Cisco.com. In the Cisco Documentation home page, click the **Fax** or **Email** option in the "Leave Feedback" section at the bottom of the page.

You can e-mail your comments to bug-doc@cisco.com.

You can submit your comments by mail by using the response card behind the front cover of your document or by writing to the following address:

Cisco Systems
Attn: Document Resource Connection
170 West Tasman Drive
San Jose, CA 95134-9883

We appreciate your comments.

Obtaining Technical Assistance

Cisco provides Cisco.com as a starting point for all technical assistance. Customers and partners can obtain online documentation, troubleshooting tips, and sample configurations from online tools by using the Cisco Technical Assistance Center (TAC) Web Site. Cisco.com registered users have complete access to the technical support resources on the Cisco TAC Web Site.

Cisco.com

Cisco.com is the foundation of a suite of interactive, networked services that provides immediate, open access to Cisco information, networking solutions, services, programs, and resources at any time, from anywhere in the world.

Cisco.com is a highly integrated Internet application and a powerful, easy-to-use tool that provides a broad range of features and services to help you with these tasks:

- Streamline business processes and improve productivity
- Resolve technical issues with online support
- Download and test software packages
- Order Cisco learning materials and merchandise
- Register for online skill assessment, training, and certification programs

If you want to obtain customized information and service, you can self-register on Cisco.com. To access Cisco.com, go to this URL:

http://www.cisco.com

Technical Assistance Center

The Cisco Technical Assistance Center (TAC) is available to all customers who need technical assistance with a Cisco product, technology, or solution. Two levels of support are available: the Cisco TAC Web Site and the Cisco TAC Escalation Center.

Cisco TAC inquiries are categorized according to the urgency of the issue:

- Priority level 4 (P4)—You need information or assistance concerning Cisco product capabilities, product installation, or basic product configuration.
- Priority level 3 (P3)—Your network performance is degraded. Network functionality is noticeably impaired, but most business operations continue.

- Priority level 2 (P2)—Your production network is severely degraded, affecting significant aspects of business operations. No workaround is available.
- Priority level 1 (P1)—Your production network is down, and a critical impact to business operations will occur if service is not restored quickly. No workaround is available.

The Cisco TAC resource that you choose is based on the priority of the problem and the conditions of service contracts, when applicable.

Cisco TAC Web Site

You can use the Cisco TAC Web Site to resolve P3 and P4 issues yourself, saving both cost and time. The site provides around-the-clock access to online tools, knowledge bases, and software. To access the Cisco TAC Web Site, go to this URL:

http://www.cisco.com/tac

All customers, partners, and resellers who have a valid Cisco service contract have complete access to the technical support resources on the Cisco TAC Web Site. The Cisco TAC Web Site requires a Cisco.com login ID and password. If you have a valid service contract but do not have a login ID or password, go to this URL to register:

http://www.cisco.com/register/

If you are a Cisco.com registered user, and you cannot resolve your technical issues by using the Cisco TAC Web Site, you can open a case online by using the TAC Case Open tool at this URL:

http://www.cisco.com/tac/caseopen

If you have Internet access, we recommend that you open P3 and P4 cases through the Cisco TAC Web Site.

Cisco TAC Escalation Center

The Cisco TAC Escalation Center addresses priority level 1 or priority level 2 issues. These classifications are assigned when severe network degradation significantly impacts business operations. When you contact the TAC Escalation Center with a P1 or P2 problem, a Cisco TAC engineer automatically opens a case.

To obtain a directory of toll-free Cisco TAC telephone numbers for your country, go to this URL:

http://www.cisco.com/warp/public/687/Directory/DirTAC.shtml

Before calling, please check with your network operations center to determine the level of Cisco support services to which your company is entitled: for example, SMARTnet, SMARTnet Onsite, or Network Supported Accounts (NSA). When you call the center, please have available your service agreement number and your product serial number.

Configuring Services

This chapter describes how to configure services, configure load for services, configure global keepalives, and use script keepalives with services. This chapter also contains an overview of the association between services, owners, and content rules. Information in this chapter applies to all CSS models except where noted.

This chapter contains the following sections:

Service, Owner, and Content Rule Overview

The CSS enables you to configure services, owners, and content rules to direct requests for content to a specific destination service (for example, a server or a port on a server). By configuring services, owners, and content rules, you optimize and control how the CSS handles each request for specific content.

- A **service** is a destination location where a piece of content resides physically (a local or remote server and port). You add services to content rules. Adding a service to a content rule includes it in the resource pool that the CSS uses for load-balancing requests for content. A service may belong to multiple content rules.
- An **owner** is generally the person or company who contracts the Web hosting service to host their Web content and allocate bandwidth as required. Owners can have multiple content rules.
- A **content rule** is a hierarchical rule set containing individual rules that describe which content (for example, .html files) is accessible by visitors to the Web site, how the content is mirrored, on which server the content resides, and how the CSS should process requests for the content. Each rule set must have an owner.

  The CSS uses content rules to determine:

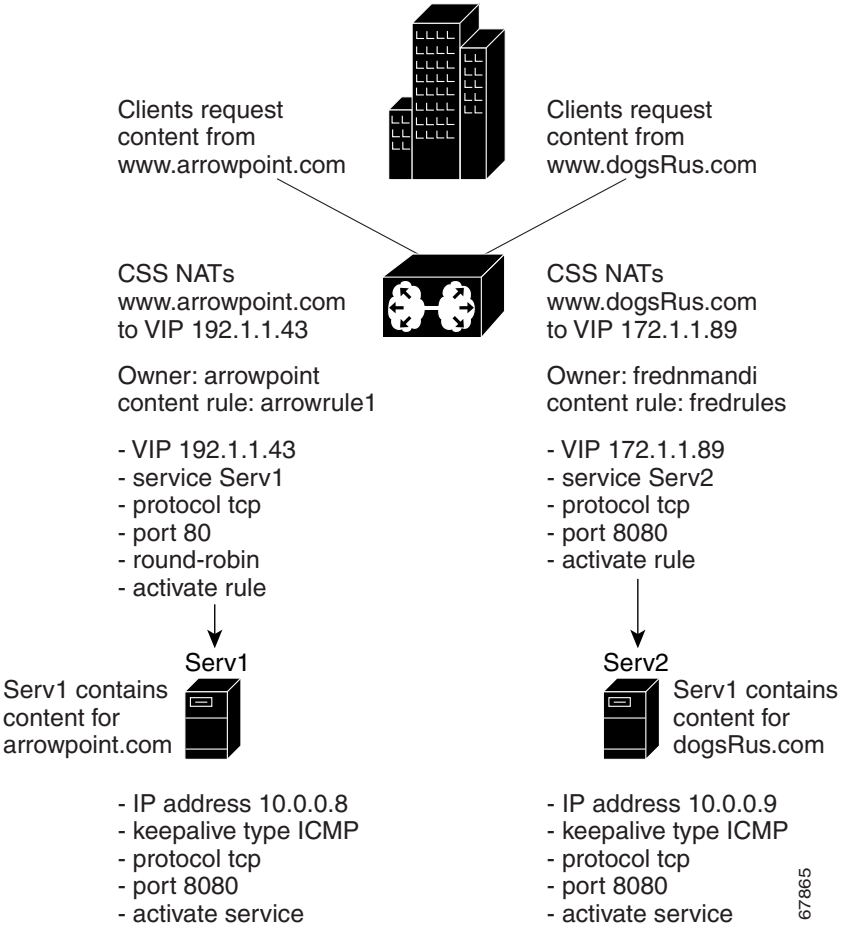
  - Where the content physically resides, whether local or remote
  - Where to direct the request for content (which service or services)
  - Which load balancing method to use

When a request for content is made, the CSS:

1. Uses the owner content rule to translate the owner Virtual IP address (VIP) or domain name using Network Address Translation (NAT) to the corresponding service IP address and port.
2. Checks for available services that match the content request.
3. Uses content rules to choose which service can best process the request for content.
4. Applies all content rules to service the request for content (for example, load-balancing method, redirects, failover, stickiness).

Figure 1-1 illustrates the CSS service, owner, and content rule concepts.

Figure 1-1 Services, Owners, and Content Rules Concepts



Service Configuration Quick Start

Table 1-1 provides a quick overview of the basic steps required to configure a service. Each step includes the command line interface (CLI) command required to complete the task. For a complete description of each feature and all the options associated with the CLI commands, see the sections following Table 1-1.

Table 1-1 Service Configuration Quick Start

Task and Command Example

1. Enter config mode by typing **config**.
 	
`# config`
`(config)#`

2. Create services. When you create a service, the CLI enters that service mode, as shown in the command response below. To create additional services, reenter the **service** command.
 	
`(config)# service serv1`
`(config-service[serv1])#`
`(config-service[serv1])# service serv2`
`(config-service[serv2])#`

3. Assign an IP address to each service. The IP address is the actual IP address of the server.
 	
`(config-service[serv2])#`
`(config-service[serv2])# ip address 10.3.6.2`
`(config-service[serv2])# service serv1`
`(config-service[serv1])# ip address 10.3.6.1`

4. Activate each service.
 	
`(config-service[serv1])# active`
`(config-service[serv1])# service serv2`
`(config-service[serv2])# active`
`(config-service[serv2])# exit`

5. Display all service configurations (optional). `(config-service[serv2])# show service summary`

Configuring Services

The following sections describe how to create and configure content services.

- Creating a Service
- Assigning an IP Address to the Service
- Specifying a Port
- Specifying a Protocol
- Specifying a Domain Name
- Configuring an Advanced Load Balancing String
- Configuring a Service HTTP Cookie
- Prefixing “http://” to a Redirect String or a Domain
- Configuring Weight
- Specifying a Service Type
- Configuring Service Access
- Configuring Service Cache Bypass
- Configuring Network Address Translation for Transparent Caches
- Configuring a Service to Bypass a Cache Farm
- Configuring Keepalives for a Service
- Activating a Service
- Suspending a Service
- Removing a Service

Note The CSS supports Adaptive Session Redundancy (ASR) on 11500 series CSS peers in an active-backup VIP redundancy and virtual IP interface redundancy environment to provide stateful failover of existing flows. For details on ASR, refer to the *Cisco Content Services Switch Advanced Configuration Guide*, Chapter 6, Configuring VIP and Virtual IP Interface Redundancy.

Creating a Service

A service can be a destination location or entity that contains and provides Internet content (for example, a server, an application on a server such as FTP, or streaming audio). A service has a name that is associated with an IP address, and optionally, a protocol and a port number.

By creating a service, you identify the service and enable the CSS to recognize it. You can then apply content rules to services that allow the CSS to:

- Direct requests for content to the service
- Deny requests for content from the service

Enter the service name from 1 to 31 characters. For example, to create service *serv1*, enter:

```
(config)# service serv1
```

The CSS transitions into the newly created service mode.

```
(config-service[serv1])#
```

Assigning an IP Address to the Service

To enable the CSS to direct requests for content to the appropriate service, you must assign an IP address or range of IP addresses to a service. Assigning an IP address to a service identifies the service to the CSS. When the CSS receives a request for content, it translates the VIP (and potentially, the port) to the service IP address (or addresses) and port.

For example, to assign an IP address to *serv1*, enter:

```
(config-service[serv1])# ip address 172.16.1.1
```

The **ip address range** command allows you to specify a range of IP addresses starting with the IP address you specified using the **ip address** command. Enter a number from 1 to 65535. The default range is 1. For example, if you enter an IP address of 172.16.1.1 with a range of 10, the IP addresses range from 172.16.1.1 through 172.16.1.10.

When using the **ip address range** command, use IP addresses that are within the subnet you are using. The CSS does not arp for IP addresses that are not on the circuit subnet. For example, if you configure the circuit for 10.10.10.1/24 and

configure the VIP range as 10.10.10.2 range 400, the CSS will not arp for any IP addresses beyond 10.10.10.254. Using the same example only with a VIP range of 200, the CSS will arp for all IP addresses in the range.

For example, enter:

```
(config-service[serv1])# ip address 172.16.1.1 range 10
```

To restore a service IP address to the default of 0.0.0.0, enter:

```
(config-service[serv1])# no ip address
```

Note The CSS sends keepalives only to the first address in a service range. If you configure a scripted keepalive, it should contain the first address in a service range as one of its arguments.

For the CSS to forward requests to a service on any of the addresses in a range, the CSS must successfully arp for the first address in the range. This behavior is independent of keepalives.

Specifying a Port

Use the **port** command to specify a service TCP/UDP port number or range of port numbers. The TCP or UDP destination port number is associated with a service. Enter the port number as an integer from 0 to 65535. The default is 0 (any).

For example, enter:

```
(config-service[serv1])# port 80
```

To specify a port to be used for keepalives, use the service mode **keepalive port** command.

Use the **range** option to specify a range of port numbers *starting* with the port number you specified using the **port** command. Enter a range number from 1 to 65535. The default range is 1. For example, if you enter a port number of 80 with a range of 10, the port numbers will range from 80 through 89. You can use the **port range** command only on local (default) services.

For example, enter:

```
(config-service[serv1])# port 80 10
```

To set the port to the default of 0 (any), enter:

```
(config-service[serv1])# no port
```

Specifying a Protocol

To specify a service IP protocol, use the **protocol** command. The default setting for this command is **any**, for any IP protocol. The options for this command are:

- **protocol tcp** - The service uses the TCP protocol suite
- **protocol udp** - The service uses the UDP protocol suite

For example, enter:

```
(config-service[serv1])# protocol tcp
```

To set the protocol to the default of **any**, enter:

```
(config-service[serv1])# no protocol
```

Specifying a Domain Name

Use the **domain** command to specify the domain name to prepend to a requested piece of content when an HTTP redirect service generates an "object moved" message for the service. The CSS uses the configured domain name in the redirect message as the new location for the requested content. The CSS prepends the domain name to the requested URL. If the domain name is not configured, the CSS uses the domain in the host-tag field from the original request. If no host tag is found, the CSS uses the service IP address to generate the redirect.

Note You can only use a service redirect domain on a service type redirect. You must specify the **domain** command for a redirect service to obtain an applicable HTTP redirect.

Note You cannot configure the **domain** and **(config-service) redirect-string** commands simultaneously on the same service.

Note The **redirect-string** and **(config-service) domain** commands are similar. The CSS returns the **redirect-string** command string as configured. With the **(config-service) domain** command, the CSS prepends the domain to the original requested URL.

Enter the service domain name as an unquoted text string with no spaces and a maximum length of 64 characters.

Note The CSS automatically prepends the domain name with http://.

For example, enter:

```
(config-service[serv1])# domain www.arrowpoint.com
```

or

```
(config-service[serv1])# domain 172.16.3.6
```

To clear the redirect domain for this service, enter:

```
(config-service[serv1])# no domain www.arrowpoint.com
```

or

```
(config-service[serv1])# no domain 172.16.3.6
```

Configuring an Advanced Load Balancing String

To specify an advanced load-balancing string for a service, use the **string** command. Use this command in conjunction with the advanced load-balancing methods **url**, **cookie**, or **cookieurl**. For information on advanced load-balancing methods, refer to Chapter 4, Configuring Sticky Parameters for Content Rules.

Enter a string from 1 to 15 characters. For example, enter:

```
(config-service[serv1])# string 172.16.3.6
```

To remove a string from a service, enter:

```
(config-service[serv1])# no string
```

Configuring a Service HTTP Cookie

Use the **string** command to specify the HTTP cookie for the service. The syntax for this service mode command is:

string *cookie_name*

Enter the *cookie_name* as an unquoted text string with no spaces and a maximum of 15 characters.

For example, enter:

```
(config-service[serv1])# string userid3217
```

To remove the cookie for a service, enter:

```
(config-service[serv1])# no string
```

Prefixing "http://" to a Redirect String or a Domain

Use the **prepend-http** command to prepend "http://" to a redirect string or domain configured for a service. The default is to prepend "http://" to a redirect string or domain.

For example, enter:

```
(config-service[serv1])# prepend-http
```

To disable prepending "http://" to a redirect string or domain configured on a service, enter:

```
(config-service[serv1])# no prepend-http
```

Configuring Weight

To specify the relative weight of the service, use the **weight** command in service mode. The CSS uses this weight when you configure ACA or weighted roundrobin load balancing on a content rule. By default, all services have a weight of 1. A higher weight will bias flows towards the specified service. To set the weight for a service, enter a number from 1 to 10. The default is 1.

Note For background information on ACA load-balancing decisions based on server weight, see "Using ArrowPoint Content Awareness Based on Server Load and Weight" later in this chapter.

For example, enter:

```
(config-service[serv1])# weight 2
```

To restore the weight to the default of 1, enter:

```
(config-service[serv1])# no weight
```

Note When you add a service to content rules, the service weight as configured in service mode is applied to each rule as a server-specific attribute. To define a content rule-specific server weight, use the **add service weight** command. This command overrides the server-specific weight and applies only to the content rule to which you add the service. For information on the **add service weight** command, refer to Chapter 3, Configuring Content Rules.

Specifying a Service Type

Use the **type** command to specify the type for a service. If you do not define a type for a service, the default service type is local. The syntax and options for this service mode command are:

- **type nci-direct-return** - Specify the service is NAT Channel indication for direct return.

  Note Use the **type nci-direct-return** command to configure NAT Peering. For information on NAT Peering, refer to Chapter 7, Configuring Caching.

- **type nci-info-only** - Specify the service is NAT Channel indication for information only.

- **type proxy-cache** - Define the service as a proxy cache. This is a cache-specific option. This option bypasses content rules for requests coming *from* the cache server. Bypassing content rules in this case prevents a loop between the cache and the CSS. For a description of a proxy cache, refer to Chapter 7, Configuring Caching.
- **type redirect** - Define the service as a remote service to enable the CSS to redirect content requests to the remote service when a local service is not available (for example, the local service has exceeded its configured load threshold). To configure a load threshold for a content rule, use the **load-threshold** command in owner-content mode (refer to Chapter 3, Configuring Content Rules, "Specifying a Load Threshold"). If you have multiple remote services defined as **type redirect**, the CSS uses the roundrobin load-balancing method to load balance requests between them.

  When you add a type redirect service to a content rule, you must also configure a URL to match on the content. For example, "/*" or "/vacations.html".
- **type redundancy-up** - Specify the router service in a redundant uplink.
- **type rep-cache-redir** - Specify the service is a replication cache with redirect.
- **type rep-store** - Specify the service is a replication store.
- **type rep-store-redir** - Specify the service is a replication store with redirect. No content rules are applied to requests from this service type.
- **ssl-accel** - Specify that this is an SSL acceleration service for the SSL Acceleration Module (Cisco 11500 series CSS only). This allows you to:
  - Configure the service as an SSL acceleration service.
  - Add the SSL proxy list to an SSL service through the **(config-service) add ssl-proxy-list** command.

  For more information on configuring SSL termination, refer to the *Cisco Content Services Switch Advanced Configuration Guide*.
- **type transparent-cache** - Specify the service as a transparent cache. This is a cache-specific option. No content rules are applied to requests from this service type. Bypassing content rules in this case prevents a loop between the cache and the CSS. For a description of a transparent cache, refer to Chapter 7, Configuring Caching.

For example, to enable the CSS to redirect content requests for serv1, specify **redirect** in the serv1 content rule:

```
(config-service[serv1])# type redirect
```

To restore the service type to the default setting of **local**, enter:

```
(config-service[serv1])# no type
```

How the CSS Accesses Server Types

When you configure a Layer 3 or 4 content rule, the rule hits the local services. If:

- The local services are not active or configured, the rule hits the primary sorry server.
- The primary sorry server fails, the rule hits the secondary sorry server.

Redirect services and redirect content strings cannot be used with Layer 3 or 4 rules because they use the HTTP protocol.

When you configure a Layer 5 content rule, the CSS directs content requests to local services. If:

- The local services are not active or configured, the rule sends the HTTP redirects with the location of the redirect services to the clients.
- The local and redirect services are not active or configured, the rule forwards the HTTP requests to the primary sorry server.
- All services are down except the secondary sorry server, the rule forwards the HTTP requests to the secondary sorry server.

For information on adding a service to a content rule or adding primary and secondary sorry servers, refer to Chapter 3, Configuring Content Rules.

Configuring Service Access

Use the **access** command to associate an access mechanism with a service for use during publishing, subscribing, and demand-based replication activities. You must use this command for each service that offers publishing services. This command is optional for subscriber services; the subscriber service inherits the access mechanism from the publisher.

When you use this command to associate an FTP access mechanism with a service, the base directory of an existing FTP record becomes the tree root. To maintain coherent mapping between WWW daemons and FTP daemons, make the FTP access base directory equivalent to the WWW daemon root directory as seen by clients. For information on creating an FTP record, refer to the **(config) ftp-record** command in the *Cisco Content Services Switch Adminstration Guide*, Chapter 1, Logging in and Getting Started.

Enter the access FTP record as the name of the existing FTP record. Enter an unquoted text string with no spaces.

For example, enter:

```
(config-service[serv1])# access ftp arrowrecord
```

To remove a service access mechanism, enter:

```
(config-service[serv1])# no access ftp
```

Configuring Service Cache Bypass

Use the **cache-bypass** command to prevent the CSS from applying content rules to requests originating from a proxy or transparent-cache type service when it processes the requests. By default, no content rules are applied to requests from a proxy or transparent-cache type service.

Note For a description of proxy and transparent caching, refer to Chapter 7, Configuring Caching.

For example, enter:

```
(config-service[serv1])# cache-bypass
```

To allow the CSS to apply content rules to requests from a proxy or transparent-cache type service, enter:

```
(config-service[serv1])# no cache-bypass
```

Configuring Network Address Translation for Transparent Caches

Use the **transparent-hosttag** command to enable destination Network Address Translation (NAT) for the transparent cache service type.

Note Currently, you can use the **transparent-hosttag** command only with a CSS operating in a Client Side Accelerator (CSA) environment. For details on CSA, refer to the *Content Service Switch Advanced Configuration Guide*, Chapter 4, Configuring a Client Side Accelerator.

Note For a description of a transparent cache, refer to Chapter 7, Configuring Caching.

For example, enter:

```
(config-service[serv1])# transparent-hosttag
```

To disable destination NATing for the transparent cache service type, enter:

```
(config-service[serv1])# no transparent-hosttag
```

Configuring a Service to Bypass a Cache Farm

Use the **bypass-hosttag** command to allow the Client Side Accelerator (CSA) on the CSS to bypass a cache farm and establish a connection with the origin server to retrieve non-cacheable content. The domain name from the host tag field is used to look up the origin IP address on the CSA.

Note Currently, you can use the **bypass-hosttag** command only with a CSS operating in a CSA environment. For details on CSA, refer to the *Cisco Content Services Switch Advanced Configuration Guide*, Chapter 4, Configuring a Client Side Accelerator.

For example, enter:

```
(config-service[serv1])# bypass-hosttag
```

To disable bypassing cache for non-cacheable content, enter:

```
(config-service[serv1])# no bypass-hosttag
```

Configuring Maximum TCP Connections

To define the maximum number of TCP connections on a service, use the **max connections** command. Enter the maximum number of connections from 6 to 65534. The default is 65534, which indicates that there is no limit on the number of connections.

```
(config-service[serv1])# max connections 7
```

To set the maximum TCP connections to the default of 65534, enter:

```
(config-service[serv1])# no max connections
```

Note Do not use service max connections on UDP content rules. The service connection counters do not increment and remain at 0 because UDP is a connectionless protocol.

Configuring Keepalives for a Service

With keepalive messages, you can determine whether or not a service is still functioning. The CSS supports a total of 2048 keepalives. These keepalives include:

- ICMP, HTTP-GET, HTTP-HEAD, TCP, FTP, SSL, and script keepalives configured and assigned to a service through the **(config-service) keepalive type** command. Each time you assign one of these keepalives to a service through this command, the CSS counts it as one keepalive.
- Global keepalives configured in keepalive configuration mode. You can apply multiple services to a global keepalive reducing the amount of configuration required for each service. The CSS counts a global keepalive as one keepalive regardless of the number of services assigned to it.

  Global keepalives supersede the individual keepalive parameters configured in service mode. For information on configuring global keepalives, see the "Configuring Keepalives in Global Keepalive Mode" section later in this chapter.

The CSS divides the keepalive types into two categories, Class A and Class B keepalives. The CSS supports a maximum of 2048 Class A keepalives. The CSS supports a maximum of 512 Class B keepalives. Table 1-2 lists the keepalive types in each class, the maximum number of each type, and the maximum number of each keepalive type that can execute concurrently.

Table 1-2 Keepalive Class, Types, and Limitations

Class	Type	CSS Maximum	Concurrent Maximum
A (The CSS limits 2048 keepalives per Class A.)	ICMP	2048	2048
	HTTP-HEAD non-persistent	2048	2048
	SSL (Hello)	2048	2048
	TCP	2048	2048
B (The CSS limits 512 keepalives per Class B.)	FTP	256	32
	HTTP-GET persistent and non-persistent	256	32
	HTTP-HEAD persistent	256	32
	Script	256	16

Caution Do not configure more than 2048 total keepalives, including a total of 512 Class B keepalives. Any services assigned to keepalives over the supported total number will not be eligible for content rule selection.

When you configure a keepalive for a service (or associate a service with a global keepalive), the CSS periodically sends a message to the service based on the keepalive frequency to determine the state of the service. See the "Configuring Keepalive Frequency" section. The CSS considers the service to be alive when a service responds to the keepalive message.

The CSS transitions the service to the dying state when the service fails to respond to a keepalive message. The CSS tests whether the failed service is functional by sending a keepalive message at time intervals based on the retry period. See the "Configuring Keepalive Type" section.

The CSS transitions the service to the dead state if the service fails to respond a maximum number of retries to the keepalive message. See the "Configuring Keepalive Retryperiod" section. Then the CSS removes the service from the load-balancing algorithm. The CSS continues to test whether the service is functional at time intervals based on the retry period.

Thus, using the default values of a 5-second keepalive frequency interval, a 5-second retry period interval, and maximum of three failures, a service can transition from the alive state to the dead state in 15 seconds; a 5-second interval between a keepalive response and the initial keepalive failure based on the keepalive frequency, and two failures, each occurring at 5-second intervals based on the retry period.

However, if the keepalives are Class B type keepalives, the time for a service to transition from an alive state to the dead state may take longer. This transition delay occurs because the CSS executes smaller numbers of Class B keepalives at the same time. For example, if you configure 256 HTTP-GET keepalives using the default values for frequency, retry period, and maximum failure, and all services fail, the time for all of the services to transition from the alive state to the dead state is 120 seconds; 8 groups of 32 services, each group transitioning in 15 seconds.

To configure keepalive message parameters for a service, use the **keepalive** command. The following sections describe the attributes you can configure for keepalives:

- Configuring Keepalive Frequency
- Configuring Keepalive Retryperiod
- Configuring Keepalive Maxfailure
- Configuring Keepalive Type
- Configuring HTTP Keepalive Method
- Configuring Keepalive Port
- Configuring Keepalive HTTP Response Code
- Configuring Keepalive URI
- Configuring a Keepalive Hash Value
- Showing Keepalive Information for a Service

For details on using script keepalives, see the "Using Script Keepalives With Services" section in this chapter.

Configuring Keepalive Frequency

Use the **keepalive frequency** command to specify the time in seconds between sending keepalives messages to a service. Specify a frequency from 2 to 255 seconds. The default is 5 seconds.

For example, enter:

```
(config-service[serv1])# keepalive frequency 15
```

To reset the frequency to its default value of 5, enter:

```
(config-service[serv1])# no keepalive frequency
```

Note When configuring the CSS for FTP keepalives, do not configure the keepalive frequency or the keepalive retryperiod to a value less than 15 seconds. Note that the CSS does not prevent you from configuring smaller values. Also, the default value for the keepalive frequency or the keepalive retryperiod is five seconds. You must use the **keepalive frequency** and **keepalive retryperiod** commands to override the defaults.

Note The timeout value for a keepalive is related to the configured keepalive frequency. For versions 7.10.3.05 and greater, the timeout is 2 seconds less than the keepalive frequency with a minimum of 1 second. From version 5.20 up to version 7.10.3.05, the timeout is one second less than the keepalive frequency.

Caution In WebNS 5.1 and earlier versions, if you configure more than 16 script keepalives, the CSS automatically adjusts the keepalive frequency time to a value that best fits the resource usage. Note that this adjustment also affects the keepalive retry period value (see "Configuring Keepalive Type" later in this chapter) by adjusting that value to a number that is one-half the adjusted frequency time. If this occurs, you may observe in the output of the **show service** command that your previously set keepalive frequency and retry period times change to a different value, as determined by the CSS.

Configuring Keepalive Retryperiod

Use the **keepalive retryperiod** command to specify the keepalive retry period for a service. When a service has failed to respond to a given keepalive message (the service has transitioned to the dying state), the retry period specifies how frequently the CSS tests the service to see if it is functional. Enter the retry period as an integer from 2 to 255 seconds. The default is 5 seconds.

Note When configuring the CSS for FTP keepalives, do not configure the keepalive frequency or the keepalive retryperiod to a value less than 15 seconds. Note that the CSS does not prevent you from configuring smaller values. Also, the default value for the keepalive frequency or the keepalive retryperiod is five seconds. You must use the **keepalive frequency** and **keepalive retryperiod** commands to override the defaults.

For example, to configure a retry period of 60 seconds, enter:

```
(config-service[serv1])# keepalive retryperiod 60
```

To reset the retry period to its default value of 5, enter:

```
(config-service[serv1])# no keepalive retryperiod
```

Configuring Keepalive Maxfailure

Use the **keepalive maxfailure** command to specify the number of times a service can fail to respond to a keepalive message before being considered down. Specify a maximum failure number from 1 to 10. The default is 3.

For example, enter:

```
(config-service[serv1])# keepalive maxfailure 5
```

To reset the maximum failure number to its default value of 3, enter:

```
(config-service[serv1])# no keepalive maxfailure
```

Configuring Keepalive Type

Use the **keepalive type** command to specify the type of keepalive message, if any, appropriate for a service or to associate a service with a global keepalive.Each time you assign one of these keepalives to a service through this command, the CSS counts it as one keepalive.

Caution Do not configure more than 2048 total keepalives, including a total of 512 Class B keepalives. Any services assigned to keepalives over the supported total number will not be eligible for content rule selection.

The syntax and options for this service mode command are:

- **keepalive type ftp** *ftp_record* - Keepalive method in which the CSS logs in to an FTP server as defined in the FTP record file. Enter the name of the existing FTP record for an FTP server as an unquoted text string with no spaces. To create an FTP record, use the **(config) ftp-record** command.

  The FTP keepalive type is a Class B type. The CSS supports a maximum of 256 FTP keepalives and concurrently executes a maximum of 32 keepalives of this type at a time.

  When configuring the CSS for FTP keepalives, do not configure the keepalive frequency or the keepalive retryperiod to a value less than 15 seconds. Note that the CSS does not prevent you from configuring smaller values. Also, the default value for the keepalive frequency or the keepalive retryperiod is five seconds. You must use the **keepalive frequency** and **keepalive retryperiod** commands to override the defaults.

- **keepalive type http** - A persistent HTTP index page request. By default, HTTP keepalives attempt to use persistent connections.

  For configuring the method for the HTTP keepalive type, see the "Configuring HTTP Keepalive Method" section. The HTTP-HEAD persistent, and HTTP-GET persistent keepalive types are a Class B types. Of each of these types, the CSS supports a maximum of 256 keepalives and concurrently executes a maximum of 32 keepalives at a time.

  If an HTTP persistent keepalive fails to make a persistent connection, then it attempts to make a non-persistent connection. If the non-persistent connection succeeds, then the keepalive succeeds. At the next interval, the keepalive attempts a persistent connection.

- **keepalive type http non-persistent** - A non-persistent HTTP index page request. This command disables the default persistent behavior.

  For configuring the method for the HTTP keepalive type, see the "Configuring HTTP Keepalive Method" section. The HTTP-GET non-persistent keepalive type is a Class B type. Of this type, the CSS supports a maximum of 256 keepalives and concurrently executes a maximum of 32 keepalives at a time.

  The HTTP-HEAD non-persistent keepalive type is a Class A type. The CSS supports a maximum of 2048 HTTP-HEAD non-persistent keepalives and concurrently executes a maximum of 2048 keepalives of this type at a time.

- **keepalive type icmp** - An ICMP echo message (ping). This is the default keepalive type.

  The ICMP keepalive type is a Class A type. The CSS supports a maximum of 2048 ICMP keepalives and concurrently executes a maximum of 2048 keepalives of this type at a time.

- **keepalive type named** *name* - Specify the name of a previously created global keepalive to associate the server with a global keepalive. Before using this command, ensure that the global keepalive is activated through the **(config-keepalive) active** command. Assigning a service to a global keepalive overrides any keepalive properties you assigned in service mode. For details about creating global (named) keepalives in keepalive configuration mode, see "Configuring Keepalives in Global Keepalive Mode" later in this chapter.

- **keepalive type none** - Do not send keepalive messages to a service.

- **keepalive type script** *script_name* {**"***arguments***"**} {**use-output**} - Script keepalive to be used by the service. The script is played each time the keepalive is issued. Enter the name of an existing script keepalive.

  The optional *arguments* variable passes arguments into the keepalive script. Enter a quoted text string with a maximum of 128 characters including spaces.

  The **use-output** option allows the script to parse the output for each executed command. This optional keyword allows the use **grep** and file direction within a script.By default, the script does not parse the output. For details on using script keepalives, see "Using Script Keepalives With Services" later in this chapter.

Note To preserve CSS system resources, use script keepalives only when needed. If an ICMP or HTTP keepalive message is sufficient to validate the service, then use that type of message instead of a script keepalive.

- **keepalive type ssl** - SSL HELLO keepalives for this service. Use this keepalive for all backend services supporting SSL. The CSS sends a client HELLO to connect the SSL server. After the CSS receives a HELLO from the server, the CSS closes the connection with a TCP RST.

  The SSL keepalive type is a Class A type. The CSS supports a maximum of 2048 SSL keepalives and concurrently executes a maximum of 2048 keepalives of this type at a time.

  When the 11500 series CSS is using an SSL module, use the keepalive type of **none**. The SSL module is an integrated device in the CSS and does not require the use of keepalive messages for the service.

- **keepalive type tcp** - A TCP session that determines service viability (3-way handshake and reset (RST)). By default and in compliance with RFC 1122, the CSS sends a RST to close the socket on a server port for TCP keepalives. A RST is faster than a FIN, because a RST requires only one packet, while a FIN can take up to four packets. If your servers require a graceful closing of a socket using a FIN, you can use a script keepalive. For an example TCP script keepalive that sends a FIN to close a socket, refer to the *Cisco Content Services Switch Advanced Configuration Guide*, Chapter 11, Using the CSS Scripting Language, in the "Script Keepalive Examples" section.

  The TCP keepalive type is a Class A type. The CSS supports a maximum of 2048 TCP keepalives and concurrently executes a maximum of 2048 keepalives of this type at a time.

For example, to set serv1 keepalive type to **ftp**, enter:

```
(config-service[serv1])# keepalive type ftp
```

Configuring HTTP Keepalive Method

Use the **keepalive method** command to specify the HTTP keepalive method for a service. The syntax and options for this service mode command are:

- **method get** - The CSS issues an HTTP GET method to the service, computes a hash value on the page, and stores the hash value as a reference hash. Subsequent GETs require a 200 OK status (HTTP command completed OK response) and the hash value to equal the reference hash value. If the 200 OK status is not returned, or if the 200 OK status is returned but the hash value is different from the reference hash value, the CSS considers the service down.

  When you specify the content information of an HTTP Uniform Resource Identifier (URI) for an HTTP keepalive, the CSS calculates a hash value for the content. If the content information changes, the hash value no longer matches the original hash value and the CSS assumes that the service is down. To prevent the CSS from assuming that a service is down due to a hash value mismatch, specify the **keepalive method** as **head**.
- **method head** (default) - The CSS issues an HTTP HEAD method to the service and a 200 OK status is required. The CSS does not compute a reference hash value for this type of keepalive. If the 200 OK status is not returned, the CSS considers the service down.

For example, enter:

```
(config-service[serv1])# keepalive method get
```

If you change the keepalive method on an active service, make sure that you suspend and reactivate the service for the change to take effect.

Note By default, HTTP keepalives attempt to use persistent connections. If an HTTP persistent keepalive fails to make a persistent connection, then it attempts to make a non-persistent connection. If the non-persistent connection succeeds, then the keepalive succeeds. At the next interval, the keepalive attempts a persistent connection.

Configuring Keepalive Port

Use the **keepalive port** command to specify the port number used for keepalives. Enter the number as an integer from 0 to 65535. The default setting is based on the configured service port number. Otherwise, the default setting is based on the keepalive type. If the keepalive type is:

- HTTP or TCP - The default port number is 80
- FTP - The port number is 21 and is not configurable.

Note If you do not configure a keepalive port, the TCP keepalive uses the service port configured with the **(config-service) port** command. If you do not configure either port, the TCP keepalive uses port 80.

For example, to specify port 8080 as the keepalive port for service *serv1*, enter:

```
(config-service[serv1])# keepalive port 8080
```

To reset the TCP keepalive port to its default of 0, enter:

```
(config-service[serv1])# no keepalive port
```

Configuring Keepalive HTTP Response Code

For a Cisco 11500 series CSS, use the **keepalive http-rspcode** command to specify the response code expected from the HTTP daemon when the CSS issues a HEAD request. This command could be helpful for checking a redirect by specifying 302 response code, or triggering another non-200 HTTP response code. Enter the response code as an integer from 100 to 999. The default is 200.

For example, to specify a response code of 302, enter:

```
(config-service[serv1])# keepalive http-rspcode 302
```

To reset the response code to its default value of 200, enter:

```
(config-service[serv1])# no keepalive http-rspcode
```

Configuring Keepalive URI

Use the **keepalive uri** command to specify the HTTP keepalive content information for a service. Enter the content information of the URI as a quoted text string with a maximum of 64 characters. Do not include the host information in the string. The CSS derives the host information from the service IP address and the keepalive port number.

For example, enter:

```
(config-service[serv1])# keepalive uri "/index.html"
```

To clear the content information for the keepalive, enter:

```
(config-service[serv1])# no keepalive uri
```

When you specify the content information of a URI for an HTTP keepalive, the CSS calculates a hash value for the content. If the content information changes, the hash value no longer matches the original hash value and the CSS assumes that the service is down. To prevent the CSS from assuming that a service is down due to a hash value mismatch, define **keepalive method** as **head**. The CSS does not compute a hash value for this type of keepalive.

If you specify a Web page with changeable content and do not specify the head keepalive method, you must suspend and reactivate the service each time the content changes.

Configuring a Keepalive Hash Value

Use the **hash** command to override the default MD5 hash for a keepalive. The CSS compares the hash value against the computed hash value of all HTTP GET responses. A successful comparison results in the keepalive maintaining an ALIVE state.

To configure the hash value:

1. Configure the keepalive. The example below creates a keepalive GET to a test page.

```
(config)# service serv1
(config-service[serv1])# ip address 10.0.3.21
(config-service[serv1])# keepalive type http
(config-service[serv1])# keepalive method get
(config-service[serv1])# keepalive uri "/testpage.html"
(config-service[serv1])# keepalive hash
"1024b91e516637aaf9ffca21b4b05b8c"
```

```
(config-service[serv1])# active
```

2. Display the hash value using the **show keepalive** command. For example, enter:

```
(config-service[serv1])# show keepalive
Keepalives:

Name: serv1
    Index: 0          State: ALIVE
    Description: Auto generated for service serv1
    Address: 10.0.3.21  Port: 80
    Type:             HTTP:GET:/testpage.html
    Hash:             1024b91e516637aaf9ffca21b4b05b8c
    Frequency:        5
    Max Failures:     3
    Retry Frequency:  5
    Dependent Services:
```

3. Use the hash value from the keepalive display to configure the keepalive hash. Enter the MD5 hash as a quoted hexadecimal string with a maximum of 32 characters. For example, enter:

```
(config-service[serv1])# keepalive hash
"1024b91e516637aaf9ffca21b4b05b8c"
```

An excerpt of the service configuration from the running-config is as follows:

```
service serv1
    ip address 10.0.3.21
    keepalive type http
    keepalive method get
    keepalive uri "/testpage.html"
    keepalive hash "1024b91e516637aaf9ffca21b4b05b8c"
    active
```

To clear a hash value and return to the default hash value, enter:

```
(config-service[serv1])# no keepalive hash
```

Showing Keepalive Information for a Service

To display keepalive information for a service, use the **show service** command. For more information on this command and what it displays, see the "Showing Service Configurations" section later in this chapter.

Activating a Service

Once you configure a service, you must activate it to enable the CSS to access it for content requests. Activating a service puts it into the resource pool for load-balancing content requests and starts the keepalive function.

Note Once a service is activated the following commands cannot be changed for the active service: **ip address**, **port**, **protocol**, **type**, **transparent-hosttag**, and **bypass-hosttag**. If you need to make modifications to an active service, you must first suspend it.

The following command activates service *serv1*:

```
(config-service[serv1])# active
```

Note For the Cisco 11500 series CSS, the CSS supports one active SSL service for each SSL Acceleration Module in the chassis (one SSL service per slot). You can configure more than one SSL service for a slot but only a single SSL service can be active at a time. Before you can activate the service, you must add an SSL proxy list to an **ssl-accel** type service and then activate the SSL proxy list.

Suspending a Service

Suspending a service removes it from the pool for future load-balancing content requests. Suspending a service does not affect existing content flows, but it prevents additional connections from accessing the service for its content. You may want to suspend a service prior to performing maintenance on the service. The following command suspends service *serv1*:

```
(config-service[serv1])# suspend
```

Note When you suspend a service, the CSS rebalances the remaining services using the failover setting.

Removing a Service

When you remove a service, the CSS:

- Removes the service from all content rules to which the service has been added.
- Rebalances the remaining services. The CSS does not apply the failover setting.

Note You cannot retrieve service information once you issue the **remove service** command.

Removing a Service From a Content Rule

To remove a service from a content rule, use the **remove service** command in the specific owner-content mode. To display a list of services added to a content rule, enter:

```
(config-owner-content[arrowpoint-rule1])# remove service ?
    server1
    server3
```

To remove service *server1* from owner *arrowpoint* content rule *rule1*, enter:

```
(config-owner-content[arrowpoint-rule1])# remove service server1
```

Removing a Service From a Source Group

To remove a service from a source group, use the **remove service** command in the specific group mode. To display a list of services added to a source group, enter:

```
(config-group[ftpgroup])# remove service ?
    server7
    serviceftp
```

To remove service *serviceftp* from source group *ftpgroup*, enter:

```
(config-group[ftpgroup])# remove service serviceftp
```

Showing Service Configurations

Before activating a service, you may want to display the service configuration to ensure that all the parameters are correct. The **show service** command enables you to display information for a specific service or all services currently configured in the CSS, depending on the location from where you issue the command.

You can issue the following **show service** commands from any mode:

- **show service** - Display configurations for each service
- **show service** *service_name* - Display service information for a specific service
- **show service summary** - Display a summary of each service

From a specific service mode, the **show service** command displays configuration information only for that service. When you issue this command from any other mode, it displays configuration information for all services.

For example, enter:

```
(config)# show service
Name: s1            Index:  10
 Type: Local        State:  Alive
 Rule: (192.168.101.15  ANY  ANY )
 Session Redundancy: Disabled
 Redirect Domain:
 Redirect String:
 Keepalive: (ICMP   5   3   5 )
 Last Clearing of Stats Counters  03/15/2002  13:45:01
 Mtu:               1500       State Transitions:   0
 Total Local Connections: 0    Total Backup Connections: 0
 Total Connections:   0   Max Connections:     0
 Total Reused Conns:  0
 Weight:            1          Load:               2
 DFP:               Disable
```

The **show service summary** command displays a summary of all service currently configured. For example, enter:

```
(config)# show service summary
Service Name     State Conn Weight Avg    State
                                   Load   Transitions
serv17            DOWN   0    1    254         1
serv18            ALIVE  0    0    254         5
NS6               ALIVE  0    0    254         3
SL3@192.16.10.25  ALIVE  0    1    212         1
```

To display configuration information for all services, enter:

```
# show service
```

To display information for a specific service, use the **show service** command with the service name. For example, enter:

```
# show service serv86
```

If you are in service mode, to display the configuration information for the current service, enter:

```
(config-service[serv86])# show service
```

Note The connection counters displayed with the **show service** command do not increment and remain at 0 for UDP flows. UDP is a connectionless protocol.

Table 1-3 describes the fields in the **show service** output.

Table 1-3 Field Descriptions for the show service Command

Field	Description
Name	The name of the service.
Index	The CSS assigned unique numeric index.
Type	The type for the service. If you do not define a type for the service, the default service type is local. The possible types are: • **nci-direct-return** - A NAT Channel Indication (NCI) service for NAT peering. • **nci-info-only** - The service is NAT Channel indication for information only. • **proxy-cache** - The service is a proxy cache. This type bypasses content rules for requests from the cache. • **redirect** - The service is not directly accessible and requires redirection. • **redundancy-up** - The service is a redundant uplink. • **rep-cache-redir** - The service is a replication cache with redirect. • **rep-store** - The service is a replication store server for hot content. • **rep-store-redir** - The service is a replication store to which content requests are redirected. • **ssl-accel** - Specify that this is an SSL acceleration service for an SSL Acceleration Module (Cisco 11500 series CSS only). • **transparent-cache** - The service is a transparent cache. No content rules are applied to requests from the cache.

Table 1-3 Field Descriptions for the show service Command (continued)

Field	Description
State	The state of the service. The State field displays the service as either Alive, Dying, Down, or Suspended. The Dying state reports that a service is failing according to the parameters configured in the following service mode commands: **keepalive retryperiod**, **keepalive frequency**, and **keepalive maxfailure**. When a service enters the Down state, the CSS does not forward any new connections to it (the service is removed from the load balancing rotation for the content rule). However, the CSS keeps all existing connections to the service (connections to that service are not "torn down").
Rule	The address, protocol, and port information for the service.
Redirect Domain	The domain name to be used when an HTTP redirect service generates an "object moved" message for the service.
Session Redundancy	Indicates whether Adaptive Session Redundancy (ASR) is enabled or disabled for the service. For details on ASR, refer to the *Cisco Content Services Switch Advanced Configuration Guide*.
SSL-Accel Slot	The slot in the CSS chassis where the SSL module is located. An SSL service requires the SSL module slot number to correlate the SSL proxy list to a specific SSL module. For details on SSL, refer to the *Cisco Content Services Switch Advanced Configuration Guide*.
Session Cache Size	The size of the SSL session ID cache for the service. The cache size is the maximum number of SSL session IDs that can be stored in a dedicated session cache on an SSL module.
Redundancy Global Index	The unique global index value for Adaptive Session Redundancy assigned to the service using the **redundant-index** command in service configuration mode.
Redirect String	The HTTP redirect string to be used when an HTTP redirect service generates an "object moved" message for the service.

Table 1-3 Field Descriptions for the show service Command (continued)

Field	Description
Keepalive	The keepalive type, frequency, maxfailure, and retryperiod. The possible keepalive types are: • **ftp** - The keepalive method that accesses an FTP server by logging into an FTP server as defined in an FTP record file. • **http** - An HTTP index page request. By default, HTTP keepalives attempt to use persistent connections. For an HTTP Head keepalive, the response code is also displayed. • **icmp** - An ICMP echo message (default) • **named** - Global keepalive defined in keepalive configuration mode. • **none** - Do not send keepalive messages to the service. • **script** - Script keepalive to be used by the service. The script is played each time the keepalive is issued. • **ssl** - SSL HELLO keepalives for this service. Use this keepalive for all backend services supporting SSL. When the 11500 series CSS is using an SSL module, use the keepalive type of **none**. • **tcp** - TCP connection handshake request. The keepalive frequency value is the time in seconds between sending keepalive messages to the service. The default is 5. The range is from 2 to 255. The keepalive maxfailure value is the number of times the service can fail to respond to a keepalive message before being considered down. The default is 3. The range is from 1 to 10. The keepalive retryperiod value is the time in seconds between sending retry messages to the service. The default is 5. The range is from 2 to 255.
Last Clearing of Stats Counters	The date and time when the State Transitions, Total Connections, or Total Reused Conns. counters were last cleared (reset to 0). The date and time stamp initially shown reflects when the service was activated or 01/01/00 00:00:00 if the service is down.

Table 1-3 Field Descriptions for the show service Command (continued)

Field	Description
Mtu	The size of the largest datagram that can be sent or received on the service.
State Transitions	The total number of state transitions on the service. If the State Transitions field is 0, this can be due to a resetting of the counter through either the global configuration mode **zero service state-transitions** command or the content mode **zero state-transitions** command. The counter can also be 0 if the service is down, or if the service is alive but no traffic is running.
Total Local Connections	Total number of TCP connections mastered by the CSS in an Adaptive Session Redundancy configuration.
Current Local Connections	Number of current active TCP connections on the CSS in an Adaptive Session Redundancy configuration.
Total Backup Connections	Total number of TCP connections backed up by the CSS for the master CSS in an Adaptive Session Redundancy configuration.
Current Backup Connections	Number of curent TCP connections that the CSS is backing up in an Adaptive Session Redundancy configuration.
Total Connections	The total number of connections that have been mapped to the service. In an Adaptive Session Redundancy configuration, Total Connections equals the sum of the Total Local Connections and the Total Backup Connections. If the Total Connections field is 0, this can due to a resetting of the counter through either the global configuration mode **zero service total-connections** command or the content mode **zero total-connections** command. The counter can also be 0 if the service is down, or if the service is alive but no traffic is running.
Max Connections	The configured maximum number of TCP connections on the service. The range is from 6 to 65534. The default is 65534.

Table 1-3 Field Descriptions for the show service Command (continued)

Field	Description
Total Reused Conns.	The total number of connections that were reused for multiple content requests during persistent connections. If the Total Reused Conns field is 0, this can be due to a resetting of the counter through either the global configuration mode **zero service total-reused-connections** command or the content mode **zero total-reused connections** command. The counter can also be 0 if the service is down, or if the service is alive but no traffic is running.
Weight	The service weight used with load metrics to make load allocation decisions. The weight is used in ArrowPoint Content Awareness (ACA) and weighted roundrobin load balancing decisions. The range is from 1 to 10. The default is 1.
Load/Average Load	The current and average load for the service.
DFP	State of the dynamic feedback protocol (DFP). Possible states are Enable or Disable. The DFP state is Disable if either DFP is not configured or DFP is configured and you have configured a weight on a service using the **add service weight** command in owner-content configuration mode. For details on DFP, see "Configuring Dynamic Feedback Protocol for Server Load-Balancing" later in this chapter.

Clearing Service Statistics Counters

Use the **zero service** command to clear a specific service statistics counter for all existing CSS services and to set that counter to zero. The reset statistics appear as 0 in the **show service** display. The **zero service** command is available in all modes.

Use the following **zero service** commands from any mode:

- **zero service total-connections** - Set the Total Connections counter to zero for all services
- **zero service total-reused-connections** - Set the Total Reused Conns. counter to zero for all services
- **zero service state-transitions -** Set the State Transitions counter to zero for all services

For example, to clear the Total Connections counter for all services, enter:

```
(config)# zero service total-connections
```

Note If you use the **zero** command in content mode, this command clears the service statistics for all services that have been added to a specified content rule, not for all content rules.

When you are in content mode, you can also use the **zero** command to clear the statistics counter for a specified service associated with the content rule. For details on clearing service statistics associated with a content rule, refer to Chapter 3, Configuring Content Rules.

Configuring Load for Services

This section covers:

- Service Load Overview
- Configuring Load for Services
- Showing Global Service Loads

Service Load Overview

Server load is a mechanism to express the current load experienced by a server. The CSS calculates load by using the variances in normalized response times from client to server to determine a server's load *number*. A server with a heavier processing load would be biased toward a more significant, larger load number.

To configure global load parameters for the eligibility and ineligibility of CSS services, use the **load report**, **load teardown timer**, and **load ageout timer** commands (discussed later in this section).

You can adjust load calculations by changing the load *step* size, which is the difference in milliseconds between load numbers. The CSS can determine the load step dynamically, or you can configure the initial load step using the global **load step** command.

The load on a service has a range of 2 to 255, with an eligible load of 2 to 254. An eligible service is an active service that can receive flows. A service with a load of 255 is offline.

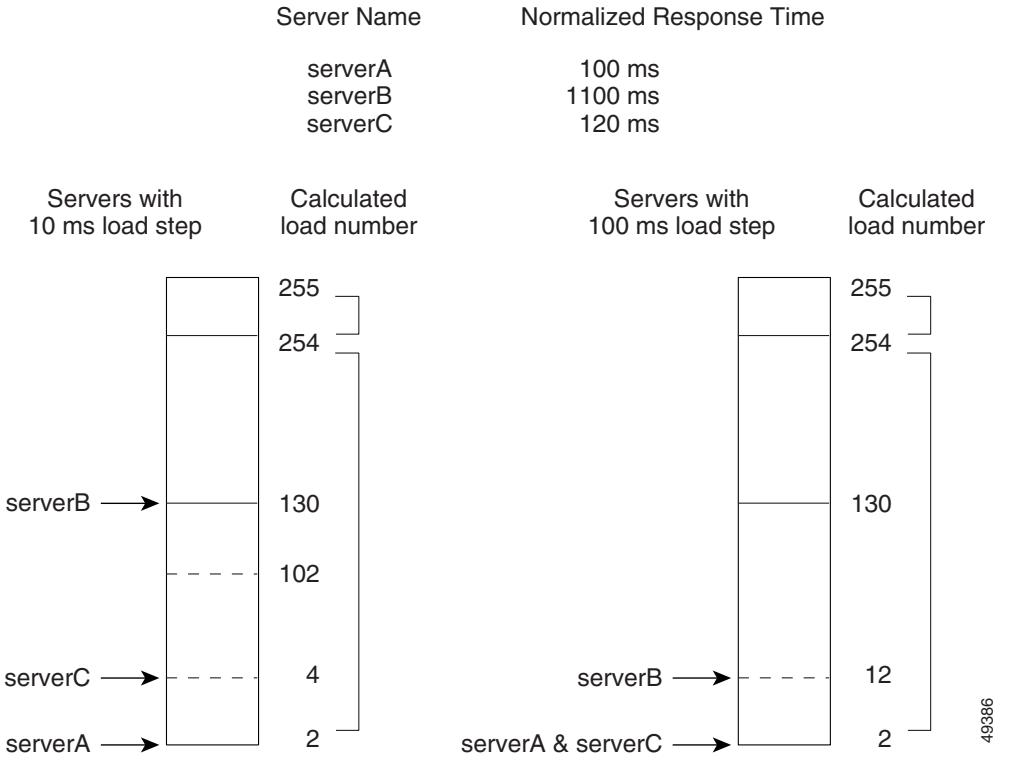
A service becomes ineligible to receive flows when its load number exceeds the configured load threshold. The CSS uses the configured ageout timer value to return the service to the eligible state.

For the CSS to consider the server loads as different, response times of the servers must differ by the configured load step or greater. If the response times differ by less than the configured load step, the CSS considers the servers to have the same load.

Note Redirect services have load numbers associated with them, but the load numbers are either 2 (available) or 255 (unavailable).

Figure 1-2 shows servers A, B, and C with response times of 100 ms, 1100 ms, and 120 ms, respectively. One group of servers has load step configured to 10 ms. The second group of servers has load step configured to 100 ms.

Figure 1-2 Load Calculation Example with Three Servers



For the servers set to the 10 ms load step, the difference in response time between:

- ServerA and serverB is 1000 ms. Because this value is greater than the configured load step of 10 ms, the CSS considers the server loads different.
- ServerA and serverC is 20 ms. Because this value is greater than the configured load step of 10 ms, the CSS considers the server loads different.

For the servers set to 100 ms load step, the difference in response time between:

- ServerA and serverB is 1000 ms. Because this value is greater than the configured load step of 100 ms, the CSS considers the server loads different.
- ServerA and serverC is 20 ms. Because this value is less than the configured load step of 100 ms, the CSS considers servers A and C to have the same load.

Increasing the load step causes the load for servers to be closer to each other. Decreasing the load step causes the load for servers to be further from each other.

To enable you to configure an accurate load threshold for a server, you can calculate a load number for a server. To calculate a server load number:

1. Take the difference between the server with the lowest response time and the server for which you want to determine a load number.
2. Divide the difference by the configured load step.
3. Add this number to the calculated load step of the server with the lowest response time, which is always 2.

For example, to calculate the load number for serverC with the 10 ms load step:

1. Take the difference in server response time between serverA and serverC (20 ms).
2. Divide it by the configured load step (10 ms). The result equals 2.
3. Add 2 to serverA's (server with lowest response time) calculated load (2) to determine serverC's calculated load of 4.

Using ArrowPoint Content Awareness Based on Server Load and Weight

ArrowPoint Content Awareness (ACA) load-balancing algorithm balances traffic between a group of servers. You can configure the CSS to make ACA load-balancing decisions based on:

- Server load
- Server weight and load

Using ACA Based on Server Load

ACA determines the best service for each content request based on server load and size of the content being requested. ACA estimates the file size based on previous requests for the same content. A service with a lower load receives more flows than a service with a higher load.

Using ACA Based on Server Weight and Load

Server weight is a mechanism to express the processing capabilities of a server. Weights allow you to configure the CSS to prefer one group of servers over another. When you configure weights, the number of hits per server is relative to the weight configured on that server. A higher weight will bias flows toward the specified server. For example, in Figure 1-2, ServerA with a weight of two is hit twice as much as ServerB that has a weight of one. ServerC has a weight of 10 and is hit 10 times as much as ServerB. All servers with the same weight are hit equally in a roundrobin manner.

The CSS can use a server's weight in tandem with server load to determine server availability. When you configure ACA on a content rule to use both weight and load, the CSS calculates the number of requests per weight level based on the number of servers with that weight. The CSS then balances the requests among the servers based on their individual loads. The number of requests per weight level is equal to weight level * number of servers *10. The CSS then increments the weight level, and uses the same mechanism to balance requests among the servers in the next weight level.

For information on configuring weight for a service, see "Configuring Weight" described earlier in this chapter. Also refer to Chapter 3, Configuring Content Rules, "Specifying a Service Weight".

Configuring Load for Services

The following sections cover:

- Configuring Global Load Step
- Configuring Global Load Threshold
- Configuring Global Load Reporting
- Configuring Load Tear Down Timer
- Configuring Load Ageout Timer

Configuring Global Load Step

Use the **load step** command to set the global load step, which is the difference in milliseconds between load numbers. Load numbers have a range from 2 to 254. By default, the CSS starts at a load step of 10 ms and then dynamically calculates the load step as it accumulates minimum and maximum response times for the services.

When you configure the load step to reduce the flows to a slower service, consider the differences in response times between services. For example:

- Increasing the load step causes the load for services to be closer to each other, thus increasing the number of flows to a slower service.
- Decreasing the load step causes the load for services to be further from each other, decreasing the flows to a slower service.

The options and syntax for this global configuration mode command are:

- **load step** *msec* **dynamic** (default) - Set the initial load step. The CSS uses the default of 10 ms as the initial load step, modifying it after the CSS collects sufficient response time information.
- **load step** *msec* **static** - Set a constant load step. The CSS uses this load step value instead of making dynamic calculations.

Enter the load step in milliseconds from 10 to 1000000000. The default is 10 ms. For example, to set the load step to 100 ms, enter:

```
(config)# load step 100
```

To set the load step to the default of 10 ms, enter:

```
(config)# no load step
```

Configuring Global Load Threshold

Use the **load threshold** command to define the global load number which the CSS uses to determine if a service is eligible to receive flows. If the service load exceeds the threshold, the service becomes ineligible to receive flows until the CSS ages the service into the eligible state.

Enter the threshold as a number from 2 to 254. The default is 254, which is the maximum threshold services can reach before becoming unavailable. To view the global load on services, use the **show load** command (see Table 1-4 for details).

For example, to set the load threshold to 25, enter:

```
(config)# load threshold 25
```

Note If you do not configure a load threshold for the content rule with the **(config-owner-content) load-threshold** command, the rule inherits this global load threshold.

To set the load threshold to the default of 254, enter:

```
(config)# no load threshold
```

Configuring Global Load Reporting

Use the **load reporting** command to enable the CSS to generate teardown reports and derive load numbers. A teardown report is a summary of response times for services when flows are being torn down. The CSS uses the teardown report to derive the load number for a service. The default is load reporting enable.

If you are not concerned about load reporting, disable it and it may increase performance (depending on flows and load reporting already occurring). To disable load reporting, enter:

```
(config)# no load reporting
```

To reenable load reporting, enter:

```
(config)# load reporting
```

Configuring Load Tear Down Timer

Use the **load teardown-timer** command to set the maximum time between teardown reports. A teardown report is a summary of response times for services when flows are being torn down. The CSS uses the teardown report to derive the load number for a service.

When the CSS has sufficient teardown activity for a service, it generates a teardown report and the teardown timer is reset. If a teardown report is not triggered at the end of the teardown timer interval due to insufficient activity, the CSS generates a teardown report based on its current activity. If there is no activity, no report is generated and the timer resets.

Note The teardown timer is overridden when a service is reset. After 10 teardown reports are recorded, the timer is reset to its configured value.

Enter the teardown timer as the number of seconds between teardown reports. enter an integer from 0 to 1000000000. The default is 20. The value of 0 disables the timer. For example, to set the teardown timer to 120 seconds, enter:

```
(config)# load teardown-timer 120
```

To reset the teardown time interval to its default of 20 seconds, enter:

```
(config)# no load teardown-timer
```

Configuring Load Ageout Timer

Use the **load ageout-timer** command to set the time interval in seconds in which the CSS ages out stale load information for a service. When the ageout timer interval expires, the CSS erases the information and resets the service load to 2. Load information is stale when the teardown report number recorded on a service has not incremented during the ageout time interval because no flows (long or short) are being torn down on the service.

At the beginning of the time interval, the ageout timer saves the number of the current teardown report. When the CSS generates a a new teardown report, the report number in the CSS increments and any services in the report saves this number. At the end of the ageout time interval, the CSS compares the initial teardown number, saved at the beginning of the time interval, with the current teardown number saved by each service. If the number of a service is less than or equal to the timer number, the load information is stale. The CSS erases it and the service load is reset to 2.

Enter the ageout timer as the number of seconds to age out load information for a service. Enter an integer from 0 to 1000000000. The default is 60. A value of 0 disables the timer.

For example, enter:

```
(config)# load ageout-timer 180
```

To set the ageout time to the default of 60, enter:

```
(config)# no load ageout-timer
```

Showing Global Service Loads

Use the **show load** command to display the global load configuration and service load information. For example, enter:

```
(config)# show load
```

Table 1-4 describes the fields in the **show load** output.

Table 1-4 Field Descriptions for the show load Command

Field	Description
Global load information	The configured state of load reporting (enabled or disabled). Reporting is disabled by default.
Step Size	The configured method in which the load step size is calculated: • Dynamic indicates that the CSS calculates the step size. • Static indicates that the configured step size is used.
Configured	The configured load step. The value is the difference in milliseconds between load numbers. If the step size method is dynamic, this is the initial load step. The CSS modifies the value after it collects sufficient response time information from the services.
Actual	The actual load step. The value is the difference in milliseconds between load numbers. If the step size method is configured, the actual value will be the same as the Configured field.
Threshold	The configured global load number that the CSS uses to determine if a service is eligible to receive flows. The range is from 2 to 254. The default is 254.

Table 1-4 Field Descriptions for the show load Command (continued)

Field	Description
Ageout-Timer	The configured time interval in seconds in which stale load information for a service is aged out. When the ageout timer interval expires, the CSS erases the information and resets the service load to 2. The range is an integer from 0 to 1000000000. The default is 60. A value of 0 disables the timer.
Teardown-timer	The maximum time between teardown reports. The range is from 0 to 1000000000. The default is 20. A value of 0 disables the timer.
Configured	The configured maximum time between teardown reports. The range is from 0 to 1000000000. The default is 20. A value of 0 disables the timer.
Actual	The actual time between teardown reports.
Service Name	The name of the service.
Average Load Number	The average load number for the service.

Configuring Keepalives in Global Keepalive Mode

Global keepalive configuration mode allows you to create a global keepalive and configure its properties. Once you create and configure a keepalive, you can apply it to any service. Applying a keepalive to multiple services reduces the amount of configuration required for each service.

Global keepalives are independent of service mode. In service mode, you can also configure individual keepalive properties for a service (see "Configuring Keepalives for a Service" earlier in this chapter). Global keepalives supersede the individual keepalive parameters configured in service mode.

The CSS supports a total of 2048 keepalives. These keepalives include:

- ICMP, HTTP-GET, HTTP-HEAD, TCP, FTP, SSL, and script keepalives configured and assigned to a service through the **(config-service) keepalive type** command. Each time you assign one of these keepalives to a service

through this command, the CSS counts it as one keepalive. For information on configuring keepalive attributes for a service, see "Configuring Keepalives for a Service" earlier in this chapter.

- Global keepalives configured in keepalive configuration mode. You can apply multiple services to a global keepalive reducing the amount of configuration required for each service. The CSS counts a global keepalive as one keepalive regardless of the number of services assigned to it.

The CSS divides the keepalive types into two categories, Class A and Class B keepalives. The CSS supports a maximum of 2048 Class A keepalives. The CSS supports a maximum of 512 Class B keepalives. Table 1-2 lists the keepalive types in each class, the maximum number of each type, and the maximum number of each keepalive type that can execute concurrently.

Table 1-5 Keepalive Class, Types, and Limitations

Class	Type	CSS Maximum	Concurrent Maximum
A (The CSS limits 2048 keepalives per Class A.)	ICMP	2048	2048
	HTTP-HEAD non-persistent	2048	2048
	SSL (Hello)	2048	2048
	TCP	2048	2048
B (The CSS limits 512 keepalives per Class B.)	FTP	256	32
	HTTP-GET persistent and non-persistent	256	32
	HTTP-HEAD persistent	256	32
	Script	256	16

Caution Do not configure more than 2048 total keepalives, including a total of 512 Class B keepalives. Any services assigned to keepalives over the supported total number will not be eligible for content rule selection.

To access keepalive configuration mode, use the **keepalive** command from circuit, global, interface, and IP configuration modes. The prompt changes to (config-keepalive [name]). You can also use this command from keepalive mode to access another keepalive.

The following sections describe how to configure global keepalives:

- Naming a Global Keepalive
- Configuring a Global Keepalive Description
- Configuring a Global Keepalive IP Address
- Configuring a Global Keepalive Frequency
- Configuring a Global Keepalive Retryperiod
- Configuring a Global Keepalive Maxfailure
- Configuring a Global Keepalive Type
- Configuring a Global Keepalive Method
- Configuring a Global Keepalive Port
- Configuring a Global Keepalive HTTP Response Code
- Configuring a Global Keepalive URI
- Configuring a Global Keepalive Hash Value
- Activating the Global Keepalive
- Suspending a Global Keepalive
- Associating a Service with a Global Keepalive
- Showing Keepalive Configurations

For details on using script keepalives, see the "Using Script Keepalives With Services" section in this chapter.

Naming a Global Keepalive

Use the **keepalive** command to access the keepalive configuration mode and configure global keepalive properties, which you can apply to any service. Enter the name of the new keepalive you want to create or the name of an existing keepalive. Enter an unquoted text string with no spaces and a length of 1 to 31 characters. To see a list of existing keepalive names, use the **keepalive ?** command.

For example, to create the global keepalive *keepimages*, enter:

```
(config)# keepalive keepimages
```

When you access this mode, the prompt changes to `(config-keepalive [keepimages])`.

```
(config-keepalive[keepimages])#
```

To remove an existing keepalive, enter:

```
(config)# no keepalive keepimages
```

Configuring a Global Keepalive Description

Use the **description** command to specify the description for a keepalive. Enter the description as a quoted text string with a maximum of 64 characters, including spaces.

For example, to enter a description for the global keepalive *keepimages*, enter:

```
(config-keepalive[keepimages])# description "This keepalive is for
the image servers"
```

To delete a description, enter:

```
(config-keepalive[keepimages])# no description
```

Configuring a Global Keepalive IP Address

Use the **ip address** command to specify the IP address where the keepalive messages are sent. Enter the IP address in dotted-decimal notation.

For example, to enter an IP address for keepalive *keepimages*, enter:

```
(config-keepalive[keepimages])# ip address 192.168.7.6
```

Configuring a Global Keepalive Frequency

Use the **frequency** command to specify the time between sending keepalive messages to the IP address. Enter the frequency time in seconds as an integer from 2 to 255. The default is 5.

Note When configuring the CSS for FTP keepalives, do not configure the keepalive frequency or the keepalive retryperiod to a value less than 15 seconds. Note that the CSS does not prevent you from configuring smaller values. Also, the default value for the keepalive frequency or the keepalive retryperiod is five seconds. You must use the **frequency** and **retryperiod** commands to override the defaults.

Note The timeout value for a keepalive is related to the configured keepalive frequency. For versions 7.10.3.05 and greater, the timeout is 2 seconds less than the keepalive frequency with a minimum of 1 second. From version 5.20 up to version 7.10.3.05, the timeout is one second less than the keepalive frequency.

Caution In WebNS 5.1 and earlier versions, if you configure more than 16 script keepalives the CSS automatically adjusts the keepalive frequency time to a value that best fits the resource usage. Note that this adjustment also affects the keepalive retry period value (see "Configuring a Global Keepalive Retryperiod" later in this chapter) by adjusting that value to a number that is one-half the adjusted frequency time. If this occurs, you may observe in the output of the **show service** command that your previously set keepalive frequency and retry period times change to a different value, as determined by the CSS.

For example, to set the frequency time to 10 seconds, enter:

```
(config-keepalive[keepimages])# frequency 10
```

To reset the frequency to its default value of 5, enter:

```
(config-keepalive[keepimages])# no frequency
```

Configuring a Global Keepalive Retryperiod

Use the **retryperiod** command to specify the retry period to send messages to the keepalive IP address. Enter the retry period as an integer from 2 to 255 seconds. The default is 5 seconds.

Note When configuring the CSS for FTP keepalives, do not configure the keepalive frequency or the keepalive retryperiod to a value less than 15 seconds. Note that the CSS does not prevent you from configuring smaller values. Also, the default value for the keepalive frequency or the keepalive retryperiod is five seconds. You must use the **frequency** and **retryperiod** commands to override the defaults.

For example, to configure a retry period of 60 seconds, enter:

```
(config-keepalive[keepimages])# retryperiod 60
```

To reset the retry period to its default value of 5, enter:

```
(config-keepalive[keepimages])# no retryperiod
```

Configuring a Global Keepalive Maxfailure

Use the **maxfailure** command to specify the number of times the IP address can fail to respond to a keepalive message before the CSS considers it down. Enter the maximum failure as an integer from 1 to 10. The default is 3.

For example, to set the global keepalive maxfailure number to 7, enter:

```
(config-keepalive[keepimages])# maxfailure 7
```

To reset the maximum failure number to its default value of 3, enter:

```
(config-keepalive[keepimages])# no maxfailure
```

Configuring a Global Keepalive Type

Use the **type** command to specify the type of keepalive message assigned to a keepalive. The syntax and options for this keepalive mode command are:

- **type ftp** *ftp_record* - Keepalive method by which the CSS logs in to the FTP server as defined in the FTP record file. Enter the name of the existing FTP record for an FTP server as an unquoted text string with no spaces. To create an FTP record, use the **(config) ftp-record** command.

  The FTP keepalive type is a Class B type. The CSS supports a maximum of 256 FTP keepalives and concurrently executes a maximum of 32 keepalives of this type at a time.

  When configuring the CSS for FTP keepalives, do not configure the keepalive frequency or the keepalive retryperiod to a value less than 15 seconds. Note that the CSS does not prevent you from configuring smaller values. Also, the default value for the keepalive frequency or the keepalive retryperiod is five seconds. You must use the **frequency** and **retryperiod** commands to override the defaults.

- **keepalive type http** - A persistent HTTP index page request. By default, HTTP keepalives attempt to use persistent connections.

  For configuring the method for the HTTP keepalive type, see the "Configuring a Global Keepalive Method" section. The HTTP-HEAD persistent, and HTTP-GET persistent keepalive types are a Class B types. Of each of these types, the CSS supports a maximum of 256 keepalives and concurrently executes a maximum of 32 keepalives at a time.

  If an HTTP persistent keepalive fails to make a persistent connection, then it attempts to make a non-persistent connection. If the non-persistent connection succeeds, then the keepalive succeeds. At the next interval, the keepalive attempts a persistent connection.

- **keepalive type http non-persistent** - A non-persistent HTTP index page request. This command disables the default persistent behavior.

  For configuring the method for the HTTP keepalive type, see the "Configuring a Global Keepalive Method" section. The HTTP-GET non-persistent keepalive type is a Class B type. Of this type, the CSS supports a maximum of 256 keepalives and concurrently executes a maximum of 32 keepalives at a time.

The HTTP-HEAD non-persistent keepalive type is a Class A type. The CSS supports a maximum of 2048 HTTP-HEAD non-persistent keepalives and concurrently executes a maximum of 2048 keepalives of this type at a time.

- **keepalive type icmp** - An ICMP echo message (ping). This is the default keepalive type. The ICMP keepalive type is a Class A type. The CSS supports a maximum of 2048 ICMP keepalives and concurrently executes a maximum of 2048 keepalives of this type at a time.
- **type script** *script_name* {**"***arguments***"**} {**use-output**} - Script keepalive to be used by the service. The script is played each time the keepalive is issued. Enter the name of an existing script keepalive.

  The optional *arguments* variable passes arguments into the keepalive script. Enter a quoted text string with a maximum of 128 characters including spaces.

  The **use-output** option allows the script to parse the output for each executed command. This optional keyword allows the use **grep** and file direction within a script. By default, the script does not parse the output. For details on script keepalives, see "Using Script Keepalives With Services" later in this chapter.

  The script keepalive type is a Class B type. The CSS supports a maximum of 256 script keepalives and concurrently executes a maximum of 16 keepalives of this type at a time.

  Note To preserve CSS system resources, use script keepalives only when needed. If an ICMP or HTTP keepalive message is sufficient to validate the service, then use that type of message instead of a script keepalive.

- **type ssl** - SSL HELLO keepalives for this service. Use this keepalive for all backend services supporting SSL. The CSS sends a client HELLO to connect the SSL server. After the CSS receives a HELLO from the server, the CSS closes the connection with a TCP RST. The SSL keepalive type is a Class A type. The CSS supports a maximum of 2048 SSL keepalives and concurrently executes a maximum of 2048 keepalives of this type at a time.

  When the 11500 series CSS is using an SSL module, use the keepalive type of **none**. The SSL module is an integrated device in the CSS and does not require the use of keepalive messages for the service.

- **type tcp** - A TCP session that determines service viability (3-way handshake and a reset (RST)). By default and in compliance with RFC 1122, the CSS sends a RST to close the socket on a server port for TCP keepalives. A RST is faster than a FIN, because a RST requires only one packet, while a FIN can take up to four packets. If your servers require a graceful closing of a socket using a FIN, you can use a script keepalive. For an example TCP script keepalive that sends a FIN to close a socket, refer to the *Cisco Content Services Switch Advanced Configuration Guide*, Chapter 11, Using the CSS Scripting Language, in the "Script Keepalive Examples" section.

  The TCP keepalive type is a Class A type. The CSS supports a maximum of 2048 TCP keepalives and concurrently executes a maximum of 2048 keepalives of this type at a time.

For example, to set the global keepalive *keepimages* to **type tcp**, enter:

```
(config-keepalive[keepimages])# type tcp
```

Configuring a Global Keepalive Method

Use the **method** command to specify the HTTP keepalive method assigned to the global keepalive. The syntax and options for the keepalive mode command are:

- **method get** - The CSS issues an HTTP GET method to the service, computes a hash value on the page, and stores the hash value as a reference hash. Subsequent GETs require a 200 OK status (HTTP command completed OK response) and the hash value to equal the reference hash value. If the 200 OK status is not returned, or if the 200 OK status is returned but the hash value is different from the reference hash value, the CSS considers the service down.

  When you specify the content information of an HTTP Uniform Resource Identifier (URI) for an HTTP keepalive, the CSS calculates a hash value for the content. If the content information changes, the hash value no longer matches the original hash value and the CSS assumes that the service is down. To prevent the CSS from assuming that a service is down due to a hash value mismatch, specify the **method** as **head**.
- **method head** (default) - The CSS issues an HTTP HEAD method to the service and a 200 OK status is required. The CSS does not compute a reference hash value for this type of keepalive. If the 200 OK status is not returned, the CSS considers the service down.

For example, to specify the HTTP get keepalive method, enter:

```
(config-keepalive[keepimages])# method get
```

If you change the keepalive method on an active service, make sure that you suspend and reactivate the service for the change to take effect.

Note By default, HTTP keepalives attempt to use persistent connections. If an HTTP persistent keepalive fails to make a persistent connection, then it attempts to make a non-persistent connection. If the non-persistent connection succeeds, then the keepalive succeeds. At the next interval, the keepalive attempts a persistent connection.

Configuring a Global Keepalive Port

Use the **port** command to specify the port number used for global keepalives. Enter the number as an integer from 0 to 65535. The default setting is based on the TCP keepalive port number. Otherwise, the default setting is based on the keepalive type. If the keepalive type is:

- HTTP or TCP - The default port number is 80
- FTP - The port number is 21 and is not configurable.

For example, to specify port 8080 as the global keepalive port, enter:

```
(config-keepalive[keepimages])# port 8080
```

To reset the keepalive port to its default of 0, enter:

```
(config-keepalive[keepimages])# no port
```

Configuring a Global Keepalive HTTP Response Code

Use the **http-rspcode** command to specify the response code expected from the HTTP daemon when the CSS issues a HEAD request. This could be helpful to check a redirect by specifying 302, or triggering another non-200 HTTP response code. Enter the response code as an integer from 100 to 999. The default is 200.

For example, to specify a response code of 302, enter:

```
(config-keepalive[keepimages])# http-rspcode 302
```

To reset the response code to its default value of 200, enter:

```
(config-keepalive[keepimages])# no http-rspcode
```

Configuring a Global Keepalive URI

Use the **uri** command to specify the content information for an HTTP global keepalive. Enter the content information for a URI as a quoted text string with a maximum length of 64 characters. Do not include the host information in the string. The CSS derives the host information from the service IP address and the keepalive port number.

When you specify the content information for an HTTP keepalive, the CSS calculates a hash value for the content. If the content information changes, the hash value no longer matches the original hash value and the CSS assumes that the service is down. To prevent the CSS from assuming that a service is down due to a hash value mismatch, specify the **keepalive method** as **head**. If you specify a Web page with changeable content and do not specify the keepalive method as **head**, you must suspend and reactivate the service each time the content information changes.

For example, to specify the content information for the global keepalive, enter:

```
(config-keepalive[keepimages])# uri "/index.html"
```

To clear the content information assigned to this keepalive, enter:

```
(config-keepalive[keepimages])# no uri
```

Configuring a Global Keepalive Hash Value

Use the **hash** command to override the default MD5 hash for a keepalive. The CSS compares the hash value against the computed hash value of all HTTP GET responses. A successful comparison results in the keepalive maintaining an ALIVE state.

To configure the hash value:

1. Configure the global keepalive. For example, enter:

```
(config-keepalive[keepimages])# method get
(config-keepalive[keepimages])# uri "/testpage.html"
(config-keepalive[keepimages])# hash
"1024b91e516637aaf9ffca21b4b05b8c"
```

2. Configure the service. For example, enter:

```
(config)# service imageserver1
(config-service[imageserver1])# ip address 10.0.3.21
(config-service[imageserver1])# keepalive type named keepimages
(config-service[imageserver1])# active
```

3. Display the hash value using the **show keepalive** command. For example, enter:

```
(config-keepalive[keepimages])# show keepalive

Keepalives:

Name: imageserver1
    Index:           0           State:      ALIVE
    Description:     Auto generated for service serv1
    Address:         10.0.3.21    Port:80
    Type:            HTTP GET:/testpage.html
    Hash:            1024b91e516637aaf9ffca21b4b05b8c
    Frequency:       5
    Max Failures:    3
    Retry Frequency: 5
    Dependent Services:
```

4. Use the hash value from the keepalive display to configure the keepalive hash. Enter the MD5 hash value as a quoted hexadecimal string with a maximum of 32 characters. For example, enter:

```
(config-keepalive[keepimages])# hash
"1024b91e516637aaf9ffca21b4b05b8c"
```

An excerpt of the service configuration from the running-config is as follows:

```
service imageserver1
    ip address 10.0.3.21
    keepalive type http
    keepalive method get
    keepalive uri "/testpage.html"
    keepalive hash "1024b91e516637aaf9ffca21b4b05b8c"
    active
```

To clear a hash value and return to the default hash value, enter:

```
(config-keepalive[keepimages])# no hash
```

Activating the Global Keepalive

Use the **active** command to activate the global keepalive. Activating a keepalive enables the CSS to start sending keepalive messages to the IP address.

For example, to activate the global keepalive *keepimages*, enter:

```
(config-keepalive[keepimages])# active
```

Suspending a Global Keepalive

Use the **suspend** command to deactivate the keepalive.

For example, enter:

```
(config-keepalive[keepimages])# suspend
```

Associating a Service with a Global Keepalive

Use the **keepalive type named** command to associate a service with a global keepalive. The service maintains the global keepalive attributes when you add the service to content rules.

For example, to associate *imageserver1* with global keepalive *keepimages*, enter:

```
(config-service[imageserver1])# keepalive type named keepimages
```

Showing Keepalive Configurations

To display global keepalive configurations, use the **show keepalive** command. To display a list of existing keepalives, use the **show keepalive ?** command.

This command provides the following options:

- **show keepalive** - Display information for all keepalives
- **show keepalive** *keepalive_name* - Display information for a specific keepalive
- **show keepalive-summary** - Display summary information for all keepalives

For example, enter:

```
(config)# show keepalive

Keepalives:

Name:          keepimages  Index: 1   State: ALIVE ( ICP Check )
Description:   This keepalive is for image servers
Address:       172.16.1.7 Port: 80
Type: HTTP:HEAD-302:/index.html
Frequency: 5
Max Failures: 3
Retry Frequency: 5
Dependent Services: imageserver1

Name: rualive   Index: 2           State: ALIVE
Description: Auto generated for service serv2
Address:        172.16.1.8         Port: 80
Type: HTTP:HEAD:/index.html
Frequency: 5
Max Failures: 3
Retry Frequency: 5
Dependent Services: serv2

(config)# show keepalive-summary

Keepalives:
Alive1         DOWN     192.25.1.7
Alive2         ALIVE    192.25.1.8
```

Table 1-6 describes the fields in the **show keepalive** output.

Table 1-6 Field Descriptions for the show keepalive Command

Field	Description
Name	The name of the keepalive.
Index	The CSS assigned unique index value for each keepalive.
State	The state of the keepalive. The possible states are down, alive, dying, suspended, and no services.
Description	The description for the keepalive.
Address	The IP address where the keepalive messages are sent.
Port	The port number for the keepalive.
Type	The type of keepalive message assigned to the keepalive. The possible types are FTP, HTTP, ICMP, script, SSL, TCP, or named. For an HTTP Head keepalive, the response code is also displayed.
Frequency	The time in seconds between sending keepalive messages to the IP address. The range is from 2 to 255. The default is 5.
Max Failures	The configured number of times the IP address can fail to respond to a keepalive message before being considered down. The range is from 1 to 10. The default is 3.
Retry Frequency	The retry period in seconds to send messages to the keepalive IP address. The range is from 2 to 255. The default is 5.
Dependent Services	Services currently configured to use the keepalive. This is mainly used for named keepalive types.

Using Script Keepalives With Services

Script keepalives are scripts that you can create to provide custom keepalives for your specific service requirements. To create the scripts, use the rich CSS Scripting Language that is included in your CSS software. For details on using the CSS Scripting Language, including using **socket** commands and examples of keepalive scripts, refer to the *Cisco Content Services Switch Advanced Configuration Guide*.

Currently, a CSS provides keepalives for FTP, HTTP, ICMP, SSL, and TCP. For information on global keepalives, see "Configuring Keepalives in Global Keepalive Mode" earlier in this chapter. For information on configuring keepalive messages, see "Configuring Keepalives for a Service" earlier in this chapter.

Using script keepalives allow you to extend the CSS keepalive functionality beyond the default keepalives. For example, you can develop a script specifically to connect a CSS to a Post Office Protocol 3 (POP3) mail server.

Once you create a script offline, you can upload it to the CSS and configure the script keepalive option on a service.

The CSS supports a maximum of 256 script keepalives. If you specify a script to parse the output for each executed command, you can configure only 16 keepalives that use script output.

Note You can also configure a script keepalive without having the corresponding script present on the CSS. In this case, a constant Down state remains on the service until you upload the appropriate script to the CSS. This allows you to develop and implement a configuration before uploading all the scripts to the CSS.

Script Keepalive Considerations

When you configure a script keepalive, follow the same general guidelines as those for global keepalive types, with the exceptions noted in these sections. For details on global keepalives, see "Configuring Keepalives in Global Keepalive Mode" earlier in this chapter.

The CSS Scripting Language allows you to pass 128 characters in a quoted argument. Assuming an average of seven characters per argument (plus a space delimiter), you can potentially use a maximum of 16 arguments in one script.

The CSS executes each line in a script keepalive. If your application requires numerous script keepalives (for example, greater than 60), keep each script as short and concise as possible. A smaller script yields much faster script execution results than a larger size script. To maximize CSS system performance, avoid complex protocols or extensive scripts (for example, no database queries, not performing a full login with validation), which can take the CSS longer to execute.

Use the script naming convention of *ap-kal-type,* so that when you press tab or "?", you can easily see the keepalive scripts available for use. For example, an SMTP script would be named ap-kal-smtp. The script name can have a maximum of 32 characters. The arguments must be in a quoted text string with a maximum of 128 characters.

For the configured script keepalive to find the corresponding script, the script must reside in the /<*current running version*>/script directory. When you configure a script keepalive, use only script names. (A CSS does not accept path names.) If the script is present elsewhere on the CSS, the script keepalive assumes it does not exist.

Note Because many scripts have a multistep process such as connecting, sending a request, and waiting for a specific type of response, configure a higher **frequency** time value for script keepalives than for standard keepalives. A time interval of 10 seconds or higher ensures that the script keepalive has enough time to finish. Otherwise, state transitions may occur more often than is usual.

Because a CSS reads an entire script into memory, there is a maximum script keepalive size of 200 KB (approximately 6,000 lines). If a script exceeds this limit, it will not load. This should be more than adequate for all applications. For example, the script keepalives included with your CSS software are approximately 1 KB. To further conserve CSS memory, services can share a common script keepalive so that only one instance of the script needs to reside in memory. However, you must configure the script keepalive for each service where you want the script to run.

To see a complete list of all scripts available in the /<current running version>/script directory, press the Tab key or "?". Optionally, you can type a script name not found in the list, then you can upload the script later. You can manipulate scripts using the **archive**, **clear**, and **copy** commands. You can also upload a script from a local hard drive to the /script directory on the CSS, or download a script from the /script directory on the CSS to a local hard drive.

Configuring Script Keepalives

Note For a large number of services that use script keepalives, use a smaller subset of global keepalives to handle the work for them. For information on global keepalives, see "Configuring Keepalives in Global Keepalive Mode" earlier in this chapter.

Use the **keepalive type script** command to configure script keepalives. The syntax for this service configuration mode command is:

keepalive type script *script_name* {**"***arguments***"**} {**use-output**}

The optional **use-output** keyword allows the script to parse the output for each executed command. This optional keyword allows the use of **grep** and file direction within a script. You can configure a maximum of 16 script keepalives (out of a maximum of 256 script keepalives) to use script output. By default, the script does not parse the output.

For example, to configure an httplist keepalive, enter:

```
(config-service[serv1)# keepalive type script ap-kal-httplist
"10.10.102.105 /default.htm"
```

In the previous command example, the **keepalive** command configures the serv1 service keepalive to be of type script with the script name ap-kal-httplist and the arguments "10.10.102.105 /default.htm". The output is not parsed by the script.

To disable a script keepalive on a service, enter:

```
(config-service[serv1])# keepalive type none
```

Viewing a Script Keepalive in a Service

When you add a script keepalive to a service, the CSS recognizes that the script is the keepalive for the service in the **show service** screen. The script name appears in the Keepalive field, and any potential arguments appear directly below in the Script Arguments field. If there are no script arguments, then the Script Arguments field does not appear.

For example, enter:

```
(config-service[serv1])# show service

Name: serv1                     Index: 1
    Type: Local                   State: Alive
    Rule (10.10.102.105 ANY ANY)
   Session Redundancy: Disabled
   Redirect Domain:
   Redirect String:
    Keepalive: (SCRIPT ap-kal-httplist 10   3   5)
    Script Arguments: "10.10.102.105 /default.htm"
    Script Error: None
    Script Run Time: 1 second
    Script Using Output Parsing:  No
   Last Clearing of Stats Counters  03/15/2002  13:45:01
    Mtu:               1500       State Transitions:   0
    Connections:       0          Max Connections:     0
    Total Connections: 0          Total Reused Conns:  0
    Weight:            1          Load:                2
```

Note If a script keepalive terminates with an error, you can use the Script Error and Script Run Time fields to help troubleshoot the problem.

You can also use the **show running-config** command to display the script keepalive and its arguments.

For example, enter:

```
(config-service[serv1])# show running-config

service serv1
    ip address 10.10.102.105
    keepalive frequency 10
    keepalive type script ap-kal-httplist "10.10.102.105
    /default.htm"
    active
```

The example above shows the script keepalive and arguments that have been configured on a service. If no arguments are specified in the script, then the quoted text following the script name will not appear.

Script Keepalive Status Codes

A script can return a status code of zero or non-zero. On a return of non-zero, the CSS flags the service state as Dying or Down; on a return of zero, the CSS flags the service state as Alive. For example, enter:

```
! Connect to the remote host
socket connect host einstein port 25 tcp
! Purposely fail
exit script 1
```

Because the above script fails when it executes the **exit** command, the script returns a non-zero value. By default, the script will fail with a syntax error if the **connect** command fails. Be sure to check the logic of your scripts to ensure that the CSS returns the correct value.

Script Keepalives and Upgrading WebNS Software

When you upgrade the WebNS software in your CSS, the upgrade process creates a new /<*current running version*>/script directory. You must copy your custom scripts (including custom script keepalives) to the new /<*current running version*>/script directory so that the CSS can find them.

Use the following procedure to ensure that your custom script keepalives operate properly after upgrading the software.

1. Upgrade the WebNS software in your CSS. Refer to the *Cisco Content Services Switch Administration Guide*.
2. Copy the scripts from the old /<*current running version*>/script directory to the new /<*current running version*>/script directory.
3. Reboot the CSS.

Configuring Dynamic Feedback Protocol for Server Load-Balancing

The Dynamic Feedback Protocol (DFP) is a mechanism that allows load-balanced servers to dynamically report changes in their status and their ability to provide services to a CSS. A status report sent to a CSS from a server contains a relative weight/number of connections to define the load and availability of each server. A CSS incorporates server feedback into the load-balancing decision process in order to:

- Obtain server availability information
- Identify load imbalances over multiple sites
- Distribute traffic more evenly

This section includes the following topics:

DFP Overview

The DFP manager (running on the CSS as a task and part of the load manager) is responsible for establishing TCP connections with the DFP agents that reside on each server. A DFP manager can communicate simultaneously with a maximum of 127 DFP agents. DFP agents can be software running on the actual server itself or may be separate hardware devices that collect and consolidate information from one or more servers for load-balancing purposes. DFP agents are available from a number of third-party sources.

DFP agents collect relative weights from the load-balanced servers and periodically send new or adjusted weights to the DFP manager in the form of load vectors. The CSS load manager distributes the incoming connections or services to the servers in the order of weight assigned to the load-balanced servers. The load manager uses the reported weights to choose the best available server, resulting in optimal performance of servers and less response time.

Note If you configure a weight on a service using the **add service weight** command in owner-content configuration mode, the configured weight takes precedence over the service weight reported by the DFP agent for that content rule. In turn, the DFP-reported weight take precedence over the weight configured on a service in service configuration mode.

The CSS uses load-balancing algorithms such as roundrobin, weighted roundrobin, Arrowpoint Content Aware (ACA), least connections, and so on to distribute the incoming connections or service requests. Weighted roundrobin can take advantage of the server weights reported by the DFP agents.

The weighted roundrobin load-balancing method uses weight to specify how many consecutive connections to give to the highest-weighted server before moving on to the next highest-weighted server. As a server's load changes, the DFP agent recalculates the weight for each server and reports the updated weights to the DFP manager, thereby influencing how the load manager distributes the service requests. For more information on CSS server load-balancing, refer to Chapter 3, Configuring Content Rules.

Functions of a DFP Agent

Besides reporting server weight/connection information to the DFP manager, the ability for multiple DFP agents to exist on a server platform provides several benefits to the load-balancing process. A DFP agent can inform the CSS that the server:

- Is congested
- Is under utilized
- Should not be used for load balancing for a period of time

Types of DFP Messages

The following messages are defined for communication between the DFP agent and the DFP manager in the CSS:

- The preference information message reports the status or weight of an IP server and is sent from the DFP agent to the DFP manager.
- The server state message, sent from the DFP manager to the agent, informs the agent that the load manager has decided to take the server in or out of service.
- The DFP parameters send configuration information from the DFP manager to the agent. Currently the only configuration parameter passed is the keepalive interval.

DFP messages consist of a DFP header called a signal header followed by message vectors. Vectors are optional commands that exist in the defined messages. Each message vector contains a vector header, which is the first part of each vector in the DFP message, followed by data specific to the defined vector. The vector header allows the DFP manager or the DFP agent to discard any vectors or commands that it does not understand.

Defined vectors for DFP include:

- **Security Vector** - Allows each DFP message to be verified.
- **Load Vector** - Contains the actual load information being reported for the real servers and represents the servers' preferred capability.
- **Keepalive Vector -** Part of the DFP connection configuration. The keepalive vector allows the load manager to inform the DFP agent of the minimum time interval by which the agent must send information over the DFP connection to the CSS.

If a CSS receives a message that contains a vector type that it does not understand, The CSS discards the unknown vector.

DFP System Flow

When you configure a DFP agent on a CSS, the DFP manager initiates a single TCP connection with the DFP agent (regardless of the number of servers the agent supports) with the parameters specified in the DFP agent configuration. The DFP manager sends a keepalive vector in a DFP message to change the default keepalive time if required.

After the connection is established, the DFP agent periodically sends update information in the form of a load-vector. If an agent has no information to send, it still must send an empty DFP packet to prevent the connection from being torn-down.

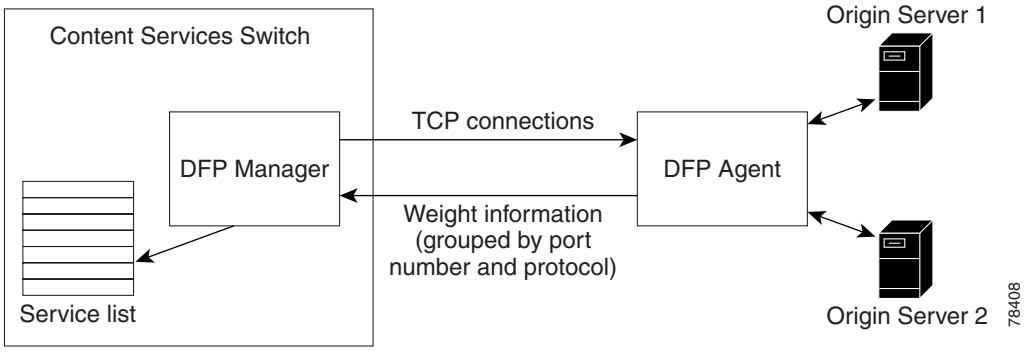
If a DFP agent is responsible for collecting information from multiple servers, the servers are grouped by their port number and protocol-type, and a separate load vector is required for each grouping. A DFP agent can report weights for a maximum of 128 servers in a single weight report. Upon receiving information about an adjusted weight, DFP manager updates the weights of the server reported in the list of load-balanced servers.

If DFP is disabled, a CSS uses the weight configured on a service in owner-content configuration mode using the **add service weight** command (for that content rule only) or the weight configured on the service in service configuration mode, in that order. If no weight is configured on the service, the CSS uses a default weight of 1 to load balance the service. If a connection between a DFP agent and the DFP manager closes because of a timeout, a CSS uses the default weight for load balancing until the DFP manager reestablishes the connection with the DFP agent and obtaining a new weight report.

If the configured DFP agent supports MD5 (Message Digest Algorithm Version 5) security, you can specify a shared key text string in the DFP manager. MD5 encryption is a one-way hash function that provides strong encryption protection. The CSS provides an MD5 secure connection between the DFP manager and the DFP agent on the server. In this secure environment, the CSS discards DFP messages from the server unless the messages contain the MD5 code.

Figure 1-3 summarizes the relationship between the DFP manager (in the CSS) and a DFP agent.

Figure 1-3 DFP Manager to DFP Agents System Flow Example



Configuring a DFP Agent

To configure a DFP agent listening for DFP connections on a particular IP address and TCP port combination on a server and to enable the DFP manager on the CSS, use the **dfp** command. You can configure a maximum of 127 DFP agents for the DFP manager in the CSS. Use the **no dfp** command to disable the DFP agent connection to a particular IP address.

The syntax for the **dfp** command is:

dfp *ip_or_host* {*port*} {**key "***secret***"**|[**des-encrypted** *encrypted_key*|**"***encrypt_key***"**]} {**timeout** *seconds*} {**retry** *count*} {**delay** *time*} {**max-agent-wt** *weight*}

- *ip_or_host* - The IP address or host name of the configured DFP agent. Enter an IP address in dotted-decimal notation (for example, 192.168.11.1) or a mnemonic host name (for example, myhost.mydomain.com).
- *port* **-** Optional. The server TCP port that the configured DFP agent uses to listen for connections from the CSS DFP manager. Valid entries are 0 to 65535. The default is 14001.

Note Do not configure a service TCP keepalive to connect to the same port that the DFP agent uses to listen for connections from the DFP manager. This type of configuration causes the built-in DFP keepalive to fail.

- **key "***md5secret***"** - Optional. An MD5 (Message Digest Algorithm Version 5) security key used for encryption to provide a secure data exchange between the CSS DFP manager and the DFP agents. MD5 encryption is a one-way hash function that provides strong encryption protection. Enter the secret as a case-sensitive quoted text string (maximum of 64 characters). It can include any printable ASCII character except tabs.

  For DFP to function properly, ensure that you configure the same key on each DFP agent that you configured on the DFP manager. If the key on a agent does not match the key on the DFP manager, no connection will be established and the DFP agent will not be able to send a weight report to the CSS. In this case, when the DFP manager fails to establish a connection with an agent for a given key, the CSS logs the following informational message in `SYSLOG`: `Secret key might not be same as DFP agent's key. Check secret key.`
- **des-encrypted** - The optional keyword that specifies that a Data Encryption Standard (DES) key follows.
- *encrypted_key* - The DES key that the CSS previously encrypted. The CSS does not re-encrypt this key and saves it in the running-config as you entered it. Enter an unquoted case-sensitive text string with no spaces and a maximum of 128 characters.
- **"***encrypt_key***"** - The DES encryption key that you want the CSS to encrypt. The CSS saves the encrypted key in the running-config as you entered it. Enter a quoted case-sensitive text string with no spaces and a maximum of 64 characters.
- **timeout** *seconds* - Optional. The maximum inactivity time period (the keepalive time) for the connection between the CSS DFP manager and the server DFP agent. If the inactivity time period exceeds the timeout value, the DFP manager closes the connection. The DFP manager attempts to reopen the connection as often as specified by the value of the **retry** option. The range is from 1 to 10000 seconds. The default is 3600 seconds (1 hour).
- **retry** *count* **-** Optional. The number of times the CSS DFP manager tries to reopen a connection with the server DFP agent. The range is from 0 (for continuous retries) to 65535. The default is 3 retry attempts.
- **delay** *time* **-** Optional. The delay time, in seconds, between each connection reestablishment attempt. Valid entries are 1 (immediately) to 65535 seconds (18 hours). The default value is 5 seconds.

- **max-agent-wt** *value* - Optional. Maximum value of the weight reported by a DFP agent. A CSS uses this option to scale the reported weight when the weight range of a DFP agent does not match the weight range of the DFP manager. For example, the DFP manager weight range is 0 to 255. If a DFP agent reports weight in the range 0 to 16, the CSS scales up the agent-reported weight to match the weight range of the DFP manager. If an agent reports weight in the range 0 to 65535, the CSS scales down the agent-reported weight to match the weight range of the DFP manager.

  If a DFP agent reports a weight greater than the maximum configured weight, then the CSS rejects the weight report and does not use the weight in load balancing decisions. In this case, the CSS also logs an error in SYSLOG. Enter an integer from 1 to 65535. The default is 255.

For example, the following command configures the DFP manager to communicate with the DFP agent at the specified address running with the following options and variables:

- DFP agent IP address - 192.168.1.2
- Port - 14001 (default)
- MD5 security key - “hello”
- Connection timeout - 6000 seconds
- Number of connection retries - 3
- Delay between connection retries - 60 seconds

```
(config)# dfp 192.168.1.2 14001 key “hello” timeout 6000 retry 3
delay 60
```

To disable the DFP agent, enter:

```
(config)# no dfp 192.168.1.2
```

Maintaining a Consistent Weight Range Among Services

The CSS has a weight range of 1 through 10; the DFP manager has a weight range of 0 through 255. Because of this difference in weight ranges, you may need to manually adjust the weights configured on the DFP agent for different services to maintain the same service weight range that exists outside of DFP.

For example, suppose that you configure on the same content rule three services (serv1, serv2, and serv3) with weights of 1, 2, and 5, respectively. If the DFP agent reports a weight of 20 for serv1, serv1 will now receive 20 connections for every two connections on serv2 and five connections on serv3. This places a disproportionate load on serv1, especially if serv2 and serv3 represent fast servers with plenty of unused resources.

To solve this problem and to maintain the same weight range for all three services, you can do either of the following:

- Force the DFP agent to report a weight in the range of 1 to 10 for serv1
- Have the DFP agent report weights for all three services to maintain the same weight range

Displaying Configured DFP Agents

For reporting purposes, you can view the configured DFP agents on a CSS using the **show dfp** command. This command displays a list of all DFP agents or the DFP agents at the specified IP address or host name arranged by their IP addresses, the port number on which the agent is connected to the DFP manager, the current state of the DFP agent, the keepalive time for the DFP TCP connection, and the DES-encrypted key of the agent, if any.

The syntax for this command is:

show dfp *ip_or_host*

The *ip_or_host* variable allows you to specify the DFP agent or agents running at a particular IP address or host name.

For example, to display configuration information for all DFP agents, enter:

```
# show dfp
```

Table 1-7 describes the fields in the **show dfp** output.

Table 1-7 Field Descriptions for the show dfp Command

Field	Description
IP Address	The IP address of the configured DFP agent.
Port	The port number of the configured DFP agent. The default is 14001.
State	The state of the DFP agent. Possible states are Active, Dead, or Connecting.
KAL	The configured maximum inactivity time in seconds for the TCP connection between the DFP manager and the DFP agent. When this time elapses, the CSS tears down the connection.
MD5 Key	The DES-encrypted key of the DFP agent if configured.

Displaying Services Supported by Configured DFP Agents

To view the individual weights of load-balanced services reported by a configured DFP agent, use the **show dfp-reports** command. This command groups the weights by the port number of reported services, the type of protocol, and the IP address of servers.

The syntax for this command is:

show dfp-reports {*ip_or_host* {**port** *number* {**protocol** *text* {**ip** *ip_or_host*}}}}

The options and variables for this command are:

- *ip_or_host* - The IP address or host name of the configured DFP agent. Enter an IP address in dotted-decimal notation (for example, 192.168.11.1) or a mnemonic host name (for example, myhost.mydomain.com).
- **port** *number* - Optional. The port number for the load-balanced server or service. Valid entries are 0 to 65535. The default is 14001.
- **protocol** *text* - Optional. The type of protocol for the load-balanced server or service. Possible values are TCP, UDP, HTTP, or FTP.

- **ip** *ip_or_host* - Optional. The IP address or host name of the load-balanced server or service. Enter an IP address in dotted-decimal notation (for example, 192.168.11.1) or a mnemonic host name (for example, myhost.mydomain.com).

The following example shows the weight reported by a DFP agent configured at 192.168.1.2, for server 192.168.1.3. Weights are first grouped by port number of reported servers, and then by protocol.

```
# show dfp-reports 192.168.1.2 port 80 protocol tcp ip 192.168.1.3
```

Table 1-8 describes the fields in the **show dfp-reports** output.

Table 1-8 Field Descriptions for the show dfp-reports Command

Field	Description
Service	The name of the configured service for which the DFP agent is reporting
Weight	The last weight reported by the DFP agent for the service
Time-Stamp	The month, day, and time of the last-received report
# of Reports	The total number of reports

Displaying DFP Information

To display DFP information, see the following sections:

- Using the show service Command
- Using the show rule services Command

Using the show service Command

Use the **show service** command to display service-specific information. The **show service** command output includes a DFP field that indicates the state of DFP. Possible states are Enable or Disable. The state is Enable when DFP is configured and there is no weight configured on the service in owner-content configuration mode. The state is Disable if DFP is not enabled or if DFP is enabled and you have configured a service weight in owner-content configuration mode using the **add service weight** command.

For details on the **show service** command, see "Showing Service Configurations" earlier in this chapter.

Using the show rule services Command

Use the **show rule services** command in owner-content mode to display weights configured for services in service mode, owner-content mode, and DFP, as well as other service-related information. The output of the command includes the weight assigned to each service preceded by a code letter. The code letters have the following meanings:

- D, the weight reported by a DFP agent
- R, the weight configured for a service using the **add service weight** command in owner-content mode
- S, the weight configured for a service using the **weight** command in service mode

For details on the **show rule services** command, refer to Chapter 3, Configuring Content Rules.

Where to Go Next

For information on creating and configuring owners, refer to Chapter 2, Configuring Owners.

Configuring Owners

This chapter describes how to create and configure owners. Information in this chapter applies to all CSS models except where noted.

This chapter contains the following sections:

Owner Configuration Quick Start

Table 2-1 provides a quick overview of the steps required to configure owners. Each step includes the CLI command required to complete the task. For a complete description of each feature and all the options associated with the CLI command, see the sections following Table 2-1.

Table 2-1 Owner Configuration Quick Start

Task and Command Example

1. Enter config mode by typing **config**.
`# config`
`(config)#`

2. Create an owner.
`(config)# owner arrowpoint`
`(config-owner[arrowpoint])#`

3. Specify the owner email address. `(config-owner[arrowpoint])# email-address bobo@arrowpoint.com`
4. Specify the owner mailing address. `(config-owner[arrowpoint])# address "373 grand ave usa"`
5. Specify the owner billing information. `(config-owner[arrowpoint])# billing-info "finance"`
6. Display owner information (optional). `(config-owner[arrowpoint])# show owner`

Creating an Owner

An **owner** is generally the person or company who contracts the web hosting service to host their web content and allocate bandwidth as required. Use the **owner** command to create an owner for a content rule. When you create an owner, you enable the CSS to identify the entity (for example, person, company name, or other meaningful title) that owns content rules. The CSS can contain many owners and maintain a configurable profile for each owner.

When creating an owner, you may want to use the owner's DNS name. Enter the owner name as an unquoted text string from 1 to 31 characters in length. The following example creates the owner *arrowpoint*:

```
(config)# owner arrowpoint
```

Once you create an owner, the CLI enters into owner mode.

```
(config-owner[arrowpoint])#
```

To remove an owner, use the **no owner** command. When you remove an owner, you also remove all content rules created for the owner. For example, enter:

```
(config-owner[arrowpoint])# no owner arrowpoint
```

Configuring an Owner DNS Balance Type

Use the **dnsbalance** command to determine where to resolve a request for a domain name into an IP address. By default, the content rule will use the DNS load balancing method assigned to the owner. The DNS load-balancing method configured for the owner applies to all of the owner's content rules. To set a different method to a specific content rule, use the **(config-owner-content) dnsbalance command**.

The syntax and options for this owner mode command are:

- **dnsbalance leastloaded** - Resolve the request to the least-loaded of all local or remote domain sites. The CSS first compares load numbers. If the load number between domain sites is within 50, then the CSS compares their response times. The site with the faster response time is considered the least-loaded site.
- **dnsbalance preferlocal** - Resolve the request to a local VIP address. If all local systems exceed their load threshold, the CSS chooses the least-loaded remote CSS VIP address as the resolved address for the domain name.
- **dnsbalance roundrobin** (default) - Resolve the request by evenly distributing the load to resolve domain names among content domain sites, local and remote. The CSS does not include sites that exceed their local load threshold.

For example, enter:

```
(config-owner[arrowpoint])# dnsbalance leastloaded
```

To reset the DNS load balancing method to its default setting of **roundrobin**, enter:

```
(config-owner[arrowpoint])# no dnsbalance
```

Specifying Owner Address

To enter an address for an owner, use the **address** command in owner mode. Enter a quoted text string with a maximum of 128 characters.

For example, enter:

```
(config-owner[arrowpoint])# address "373 granite ave usa"
```

To delete an owner address, enter:

```
(config-owner[arrowpoint])# no address
```

Specifying Owner Billing Information

To enter billing information for an owner, use the **billing-info** command in owner mode. Enter the billing information assigned to an owner as a quoted text string with a maximum length of 128 characters. For example, enter:

```
(config-owner[arrowpoint])# billing-info "finance"
```

To delete an owner billing address, enter:

```
(config-owner[arrowpoint])# no billing-info
```

Specifying Case

To define whether or not the CSS employs case-sensitivity when matching content requests to an owner's content rule, use the **case** command. The default is **case insensitive**.

For example, a client requests content from *arrowpoint/index.html*. If owner *arrowpoint* is configured for:

- **case sensitive**, the request must match content *index.html* exactly
- **case insensitive**, the request can be any combination of uppercase and lowercase letters (for example, *Index.html*, *INDEX.HTML*)

To configure owner *arrowpoint* content rules to be case-sensitive, enter:

```
(config-owner[arrowpoint])# case sensitive
```

To return to the default, enter:

```
(config-owner[arrowpoint])# case insensitive
```

Specifying Owner DNS Type

To set the peer name exchange policy for a specific owner, use the **dns** command. The default is none, which does not set a peer name exchange policy. For information on configuring DNS, refer to the *Cisco Content Services Switch Advanced Configuration Guide*.

The syntax and options for this owner mode command are:

- **dns accept** - Accept all content rules proposed by the CSS peer
- **dns push** - Push (send) all content rules onto the CSS peer
- **dns both** - Accept all content rules proposed by the CSS peer and push all rules onto the CSS peer

For example, enter:

```
(config-owner[arrowpoint])# dns push
```

To remove an owner's peer name exchange policy, enter:

```
(config-owner[arrowpoint])# no dns
```

Specifying Owner Email Address

To enter an email address for an owner, use the **email-address** command in owner mode. For example, enter:

```
(config-owner[arrowpoint])# email-address bobo@arrowpoint.com
```

To remove an owner email address, enter:

```
(config-owner[arrowpoint])# no email-address
```

Removing an Owner

To remove an owner, use the **no owner** command from config mode. To remove an owner, you must first exit from the owner mode. You cannot be in the owner mode that you wish to remove.

For example, to remove an owner, enter:

```
(config)# no owner arrowpoint
```

Caution Removing an owner also deletes the content rules associated with it.

Showing Owner Information

The **show owner** command enables you to display owner information for an owner. An owner is an entity that owns Web content and is using the CSS to manage access to that content.

You can issue the following **show owner** commands from the specified command modes to display configuration information and statistics for an owner:

- **show owner** {*owner_name* {**statistics**}} - Display configuration information and statistics for an owner. This command is used in ACL, Circuit, Global, Group, Interface, Service, SuperUser, and User modes. The **show owner** command displays configuration information for all owners. The **show owner** *owner_name* command displays configuration information for a specified owner.

- **show owner** {**statistics**} - Display configuration information and statistics for the current owner, or for the owner of the current content rule. This command is used in Owner and Content mode. The **show owner** command with no options displays configuration information only.

For example, to display configuration information for a specific owner from the ACL, Circuit, Global, Group, Interface, Service, SuperUser, or User modes, enter:

```
# show owner test.com
```

To display configuration information for the owner in Owner or Content mode, enter:

```
(config-owner[test.com])# show owner
```

Table 2-2 describes the fields in the **show owner** *name* output.

Table 2-2 Field Descriptions for the show owner name Command

Field	Description
Name	The name of the owner.
Billing Info	The billing information about the owner.
Address	The postal address for the owner of the Web hosting service.
Email Address	The email address for the owner.
DNS Policy	The peer DNS exchange policy for the owner. The possible policies are: • **accept** - Accept all content rules proposed by the CSS peer. • **push** - Advertise the owner and push all content rules onto the CSS peer. • **both** - Advertise the owner and push all content rules onto the CSS peer, and accept all content rules proposed by the CSS peer. • **none** - The default DNS exchange policy for the owner. The owner is hidden from the CSS peer.
Case Matching	Whether the matching of content requests to the owner's rules is case sensitive or insensitive.

To display statistics for an owner from the ACL, Circuit, Global, Group, Interface, Service, SuperUser, or User modes, enter:

```
# show owner test.com statistics
```

To display statistics for the owner from either Owner or Content mode, enter:

```
(config-owner[test.com])# show owner statistics
```

Table 2-3 describes the fields in the **show owner** *name* **statistics** output.

Table 2-3 Field Descriptions for the show owner name statistics Command

Field	Description
DNS Policy	The peer DNS exchange policy for the owner. The possible policies are: • **accept** - Accept all content rules proposed by the CSS peer. • **push** - Advertise the owner and push all content rules onto the CSS peer. • **both** - Advertise the owner and push all content rules onto the CSS peer, and accept all content rules proposed by the CSS peer. • **none** - The default DNS exchange policy for the owner. The owner is hidden from the CSS peer.
Hits	Number of connections processed under the rules of the owner.
Bytes	Total number of bytes transferred that matched the rules of the owner.
Frames	Total frames transferred that matched the rules of the owner.
Redirects	Total number of flows that have been redirected due to persistent connections or stickiness.

Table 2-3 Field Descriptions for the show owner name statistics Command (continued)

Field	Description
Spoofs	Number of times that client connections have been replied to by the CSS while it simultaneously negotiates a connection with the back-end service.
Case Matching	Whether or not the matching of content requests to the rules of the owner is case sensitive or insensitive.
Reject Overload	Not used.
Reject No Services	Number of times that connections were rejected due to no available services.
Drops	Not used.
NAT Translations	Not used.

Showing Owner Summary

The **show summary** command enables you to display a summary of the following owner information for all owners or a specific owner:

- Owners
- Content rules
- Services
- Service hits

You can issue the following **show summary** commands from any mode:

- **show summary** - Display a summary of all owner information
- **show summary** *owner_name* - Display a summary of owner information for a specific owner

For example, enter:

```
(config)# show summary
```

Table 2-4 describes the fields in the **show summary** output.

Table 2-4 Field Descriptions for the show summary Command

Field	Description
Global Bypass Counters	
No Rule Bypass Count	The number of times that a flow passes through even though it did not match one of the existing content rules.
ACL Bypass Count	The number of times that the ACL immediately sends traffic to its destination, bypassing the content rule.
URL Params Bypass Count	The number of times that content requests match on content rules that have param-bypass set to enable. The CSS forwards the content requests to the origin server.
Cache Miss Bypass Count	The number of times that TCP connections from the cache servers bypassed content rules so the cache server could access the origin server for the requested content.
Garbage Bypass Count	The number of times that the CSS examined content requests and deemed them unrecognizable or corrupt. As a result, the CSS forwards the content request to the origin server rather than the cache server.
Owner	The owner name.
Content Rules	The rule associated with the owner.
State	The state of the rule (active or suspended).
Services	The services associated with the rule.
Service Hits	The number of hits on the service.

Where to Go Next

Once you create and configure an owner, refer to Chapter 3, Configuring Content Rules, for information on configuring content rules. Content rules instruct the CSS on how to handle requests for the owner's content. You create and configure a content rule within a specific owner mode. This method ensures that the configured content rule applies only to a specific owner.

Configuring Content Rules

This chapter describes how to create and configure content rules. Information in this chapter applies to all CSS models except where noted.

This chapter contains the following sections:

Service, Owner, and Content Rule Overview

The CSS enables you to configure services, owners, and content rules to direct requests for content to a specific destination service (for example, a server or a port on a server). By configuring services, owners, and content rules, you optimize and control how the CSS handles each request for specific content.

- A **service** is a destination location where a piece of content physically resides (a local or remote server and port). You add services to content rules. Adding a service to a content rule includes it in the resource pool that the CSS uses for load balancing requests for content. A service may belong to multiple content rules. To configure services, refer to Chapter 1, Configuring Services.
- An **owner** is generally the person or company who contracts the web hosting service to host their web content and allocate bandwidth as required. To configure owners, refer to Chapter 2, Configuring Owners.
- A **content rule** is a hierarchical rule set containing individual rules that describe which content (for example, .html files) is accessible by visitors to the web site, how the content is mirrored, on which server the content resides, and how the CSS should process requests for the content. Each rule set must have an owner.

When a request for content is made, the CSS:

1. Uses the owner content rule to translate the owner Virtual IP address (VIP) or domain name using Network Address Translation (NAT) to the corresponding service IP address and port.
2. Checks for available services that match the content request.
3. Uses content rules to choose which service can best process the request for content.
4. Applies all content rules to service the request for content (for example, load-balancing method, redirects, failover, stickiness).

The CSS uses content rules to determine:

- Where the content physically resides, whether local or remote.
- Where to direct the request for content (which service or services).
- Which load-balancing method to use.

The type of rule also implies the Layer at which the rule functions.

- A Layer 3 content rule implies a destination IP address of the host or network.
- A Layer 4 content rule implies a combination of destination IP address, protocol, and port.
- A Layer 5 content rule implies a combination of destination IP address, protocol, port, and URL that may or may not contain an HTTP cookie or a domain name.

Note A Layer 5 content rule supports the HTTP CONNECT, GET, HEAD, POST, PUSH, and PUT methods. The CSS recognizes and forwards the following HTTP methods directly to the destination server in a transparent caching environment. Note that the CSS does not load balance these HTTP methods. RFC-2068: OPTIONS, TRACE; RFC-2518: PROPFIND, PROPPATCH, MKCOL, MOVE, LOCK, UNLOCK, COPY, DELETE.

Content rules are hierarchical. That is, if a request for content matches more than one rule, the characteristics of the most specific rule apply to the flow. The CSS uses this order of precedence to process requests for the content, with 1 being the highest match and 9 being the lowest match. The hierarchy for content rules is as follows:

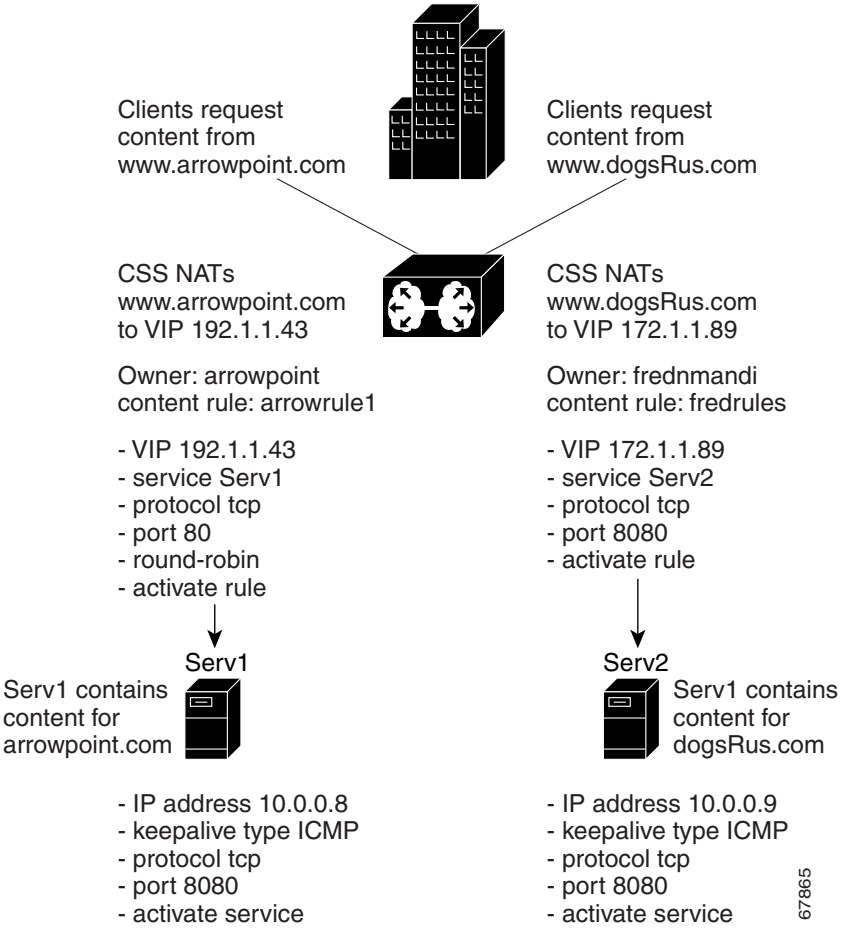
1. Domain name, IP address, protocol, port, URL
2. Domain name, protocol, port, URL
3. IP address, protocol, port, URL
4. IP address, protocol, port
5. IP address, protocol
6. IP address
7. Protocol, port, URL
8. Protocol, port
9. Protocol

Note The CSS evaluates the content rule hierachary before it evaluates the Layer 5 rule URL, cookie strings, or HTTP header information.

Note In some environments, URL, cookie strings, or HTTP header information can span over multiple packets. In these environments, the CSS can parse multiple packets for Layer 5 information before making load-balancing decisions. Through the global configuration mode **spanning-packets** command, the CSS can parse up to 20 packets with a default of 6. The CSS makes the load-balancing decision as soon as it finds a match and does not require parsing of all of the configured number of spanned packets. Because parsing multiple packets does impose a longer delay in connection, performance can be impacted by longer strings that span mulitple packets. For information on using the **spanning-packets** command, see "Specifying the Number of Spanned Packets" later in this chapter.

Figure 3-1 illustrates the CSS service, owner, and content rule concepts.

Figure 3-1 Services, Owners, and Content Rules Concepts



Content Rule Configuration Quick Start

Table 3-1 provides a quick overview of the steps required to create and configure a Layer 3 content rule. Each step includes the CLI command required to complete the task. For a complete description of each feature and all the content rule configuration options, see the sections following Table 3-1.

Ensure that you have already created and configured a service and owner for the content rules. The command examples in Table 3-1 create a Layer 3 content rule for owner *arrowpoint*.

Table 3-1 Content Rule Configuration Quick Start

Task and Command Example

1. Enter into config mode by typing **config**.
`# config`
`(config)#`

2. Enter into the owner mode for which you wish to create content rules. `(config)# owner arrowpoint`

3. Create the content rule for the owner.
`(config-owner[arrowpoint])# content rule1`
The CSS enters into the owner-content rule mode.
`(config-owner-content[arrowpoint-rule1]#`

4. Configure a Virtual IP address (VIP) or domain name for the owner content. This example configures a VIP, which implies a Layer 3 content rule.
`(config-owner-content[arrowpoint-rule1]# vip address 192.168.3.6`
If you require a Layer 4 content rule, specify a protocol in the content rule and a specific TCP/UDP port number (in addition to the VIP address or domain name).
`(config-owner-content[arrowpoint-rule1]# protocol tcp`
`(config-owner-content[arrowpoint-rule1]# port 80`
If you require a Layer 5 content rule, specify a URL in the content rule (in addition to the protocol and port number).
`(config-owner-content[arrowpoint-rule1]# url "//www.arrowpoint.com/*"`

Table 3-1 Content Rule Configuration Quick Start (continued)

Task and Command Example
5. Specify a load balancing type. `(config-owner-content[arrowpoint-rule1]# balance aca`

6. Add previously configured services to the content rule.
`(config-owner-content[arrowpoint-rule1]# add service serv1`
`(config-owner-content[arrowpoint-rule1]# add service serv2`

7. Activate the content rule. `(config-owner-content[arrowpoint-rule1]# active`
8. Display the content rules (optional). `(config-owner-content[arrowpoint-rule1]# show rule`

Naming and Assigning a Content Rule to an Owner

To name a content rule and assign it to an owner, use the **content** command. By assigning content rules to an owner, you can manage access to the content. Assign content rules to an owner by creating the content rule in the mode for that owner. The CSS identifies content rules by the names you assign. Enter a content rule name from 1 to 31 characters.

The following example assigns:

- The name *rule1* to the content rule
- Content rule *rule1* to owner *arrowpoint*

```
(config-owner[arrowpoint])# content rule1
```

Once you assign a content rule to an owner, the CLI prompt changes to reflect the specific owner and content rule mode.

```
(config-owner-content[arrowpoint-rule1])#
```

Within owner and content mode, you can configure how the CSS will handle requests for the content. To remove an existing content rule from an owner, use the **no content** command from owner mode. For example, enter:

```
(config-owner[arrowpoint])# no content rule1
```

Configuring a Virtual IP Address

Note The CSS supports Adaptive Session Redundancy (ASR) on 11500 series CSS peers in an active-backup VIP redundancy and virtual IP interface redundancy environment to provide stateful failover of existing flows. For details on ASR, refer to the *Cisco Content Services Switch Advanced Configuration Guide*, Chapter 6, Configuring VIP and Virtual IP Interface Redundancy.

A Virtual IP address (VIP) is an address that an Internet Domain Name System (DNS) provides when asked to resolve a domain name. For example, a DNS server may translate *www.arrowpoint.com* to the VIP 192.217.4.15. Internet Service Providers (ISPs) generally assign VIPs. ISPs request VIPs from the Internet Assigned Name Authority (IANA).

Assigning a VIP to owner content enables the CSS to translate (using Network Address Translation (NAT)) the VIP to the IP address of the service where the content resides.

Note The CSS allows you to configure a domain name instead of a VIP. See the next section for information on configuring a domain name. You may configure either a VIP, a domain name, or both in a content rule.

To enable the CSS to translate an owner's Internet IP address to the IP address of the service where the content resides, configure a VIP to the owner content. By translating a VIP to the service IP address, the CSS enhances network security because it prevents users from accessing your private network IP addresses.

Caution Ensure that all VIPs are unique IP addresses. Do not configure a VIP to the same address as an existing IP address on your network or a static ARP entry.

Note When you configure a rule without a VIP (wildcard VIP rule), the rule matches on any VIP that matches the other configured rule attributes (for example, port and protocol). When you configure a rule without a VIP and without a port (double-wildcard caching rule), the rule matches on any VIP or port that matches the other configured rule attributes (for example, protocol). If you have a

configuration that requires either type of rule, be aware that the client request will match on this rule when the client request attempts to connect directly to a server IP address. For more information on double-wildcard caching rules, refer to Chapter 7, Configuring Caching.

The variables and options for the **vip address** command include:

- *ip_address* or *host* - The IP address or name for the content rule. Enter the address in either dotted-decimal IP notation (for example, 192.168.11.1) or mnemonic host-name format (for example, myhost.mydomain.com).
- **range** *number* - The range option and variable allows you to specify a range of IP addresses starting with the VIP address. Enter a number from 1 to 65535. The default range is 1. The *ip_or_host* variable is the first address in the range. For example, if you enter a VIP of 172.16.3.6 with a range of 10, the VIP addresses will range from 172.16.3.6 to 172.16.3.15.

Note When you use an FTP content rule with a configured VIP address range, be sure to configure the corresponding source group with the same VIP address range (refer to Chapter 5, Configuring Source Groups, ACLs, EQLs, URQLs, NQLs, and DQLs).

To configure a Virtual IP address (VIP), issue the **vip address** command and specify either an IP address or a host name. For example, enter:

```
(config-owner-content[arrowpoint-rule1])# vip address 192.168.3.6
```

Note When you ping a VIP, the CSS responds only if there is at least one live service, live sorry server, or redirect string configured for the VIP, or if the service is associated with a source group. If the services or sorry servers are down and you have not defined a redirect string for the VIP, the CSS does not respond to the ping.

To configure a Virtual IP address (VIP) with a range of 10, use the **vip address** command with the **range** option. For example, enter:

```
(config-owner-content[arrowpoint-rule1])# vip address 192.168.3.6
range 10
```

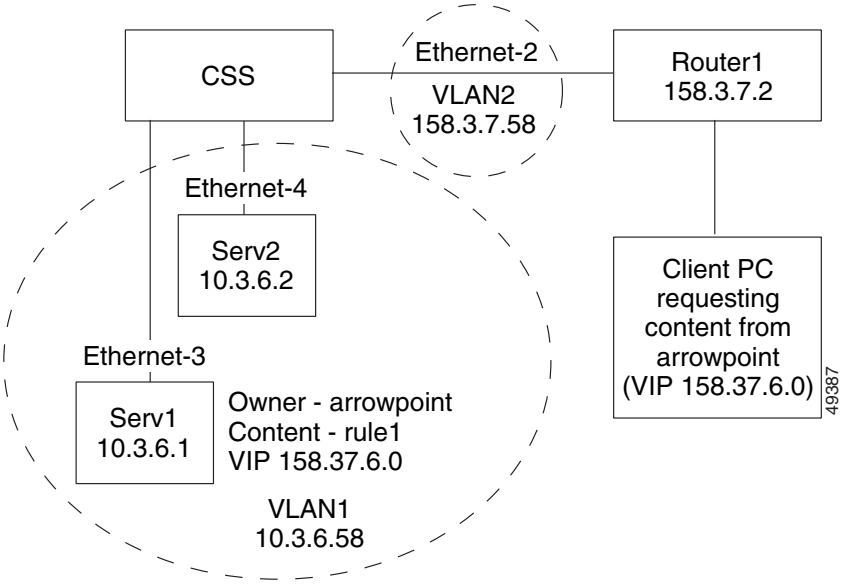
When using the **vip address range** command, use IP addresses that are within the subnet you are using. The CSS does not ARP for IP addresses that are not on the circuit subnet. For example, if you configure the circuit for 10.10.10.1/24 and configure the VIP range as 10.10.10.2 range 400, the CSS will not ARP for any IP addresses beyond 10.10.10.254. Using the same example with a VIP range of 200, the CSS will ARP for all IP addresses in the range.

To remove a VIP from a content rule, enter:

```
(config-owner-content[arrowpoint-rule1])# no vip address
```

Figure 3-2 shows an example of configuring a VIP. In this example, a user requests content from *arrowpoint*. The content physically resides on the server with IP address 10.3.6.1. By configuring VIP 158.37.6.0 to the content, the CSS translates the VIP to the server IP address where the content actually resides without exposing internal IP addresses.

Figure 3-2 Example of Configuring a Virtual IP Address



Configuring a Domain Name Content Rule

The CSS allows you to use a domain name in place of, or in conjunction with, a VIP in a content rule. Using a domain name in a content rule enables you to:

- Enable service provisioning to be independent of IP-to-domain name mappings
- Provision cache bandwidth as needed based on domain names

Note Domain names in content rules are case insensitive, regardless of the **case** command setting.

To configure a domain name in a content rule, use the **url** command and place two slash characters (//) at the front of the quoted *url_name* or *url_path*.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# url
"//www.arrowpoint.com/*"
```

Normally, port 80 traffic does not use a port number in the domain name. To specify a port other than port 80, enter the domain name with the port number exactly. Separate the domain name and the port number with a colon. For example, enter:

```
(config-owner-content[arrowpoint-rule1])# url
"//www.arrowpoint.com:8080/*"
```

Use domain name rules rather than VIP rules when you have several transparent caches and you want certain domains to use the most powerful cache server. You want all other domains load balanced among the remaining cache servers. For this configuration, set up a domain name rule for the specific domains you want directed to the powerful cache server. Then configure a wildcard VIP rule (specify port 80 and no VIP) to balance all other HTTP traffic among the remaining caches.

You may use a single VIP in front of a server that is hosting many domain names. Over time, some of the domain names may receive more traffic and could benefit from having their content on a separate server. To segregate the traffic, configure the domain names you want directed to specific services. You do not need to configure additional VIPs for the domain names because the CSS will use the domain names as the matching criteria in the content rules.

Matching Content Rules on Multiple Domain Names

When you have a requirement for a content rule to match on multiple domain names, you can associate a Domain Qualifier List (DQL) to the rule. A DQL is a list of domain names that you configure. You can use a DQL on a rule to specify that content requests for each domain in the list will match on the rule.

You can determine the order that the domain names are listed in the DQL. You can arrange the names in a DQL by assigning an index number as you add the name to the list.

DQLs exist independently of any range mapping. You can use them as matching criteria to balance across servers that do not have IP addresses or port ranges. If you want to use range mapping when using a service range, you need to consider the index of any domain name in the DQL. If you are not using service ranges with DQLs, you do not need to configure any index and the default index is 1.

For example, you could configure a DQL named Woodworker.

```
(config)# dql Woodworker
```

The domain names you could add as part of the DQL include *www.wood.com*, *www.woodworker.com*, *www.maple.com*, *www.oak.com*. You could configure *www.wood.com* and *www.woodworker.com* to have the same mapping index. You can enter indexes from 1 to 1000 and provide an optional quoted description for each index.

For example, enter:

```
(config-dql[Woodworker]# domain www.wood.com index 1 "This is the
same as the woodworker domain"

(config-dql[Woodworker]# domain www.woodworker.com index 1

(config-dql[Woodworker]# domain www.maple.com index 2

(config-dql[Woodworker]# domain www.oak.com index 3
```

If you specify a DQL as a matching criteria for content rule WoodSites, and there are two services, S1 and S2, associated with the rule, the CSS checks the services at mapping time for ranges. To add a DQL to a content rule, use the **url** command as shown:

```
(config-owner-content[WoodSites])# url "/*" dql Woodworker
```

For example, if the CSS receives a request for *www.oak.com* along with other criteria, a match on the WoodSites rule occurs on DQL index 3. If the rule has the roundrobin load balancing method, the CSS examines a service (S2 for this example) to determine the backend connection mapping parameters. If you configured S2 with a VIP address of 10.0.0.1 with a range of 5, the addresses include 10.0.0.1 through 10.0.0.5. Because this service has a range of addresses and 0 (any) as its port, the DQL index of 3 matches the service VIP range index of 3, which is address 10.0.0.3.

To delete a DQL, use the **no dql** command. For example, enter:

```
(config)# no dql Woodworker
```

Note You cannot delete a DQL currently in use by a content rule.

For a complete description of DQLs, refer to Chapter 5, Configuring Source Groups, ACLs, EQLs, URQLs, NQLs, and DQLs.

Configuring a Content Rule using a Domain Name and a Virtual IP Address

Use a domain name and a virtual IP address (VIP) in a content rule when you want the CSS to match content requests going to a specific domain at a specific VIP. If the CSS is serving more than one VIP at the domain name, configure two domain name content rules and specify the different VIPs.

This configuration is shown in the sample running-config below. Note that because the IP addresses in the example below are contiguous, you could use the **vip address range** command to specify a VIP range of 2.

```
content domainRule1
    vip address 192.168.1.1
    protocol tcp
    port 80
    url "//www.domain.com/*"
    add service Serv1
    activate

content domainRule2
    vip address 192.168.1.2
    protocol tcp
```

```
    port 80
    url "//www.domain.com/*"
    add service Serv1
    activate
```

If your network topology does not require that the CSS ARP-reply for VIPs, you do not need to configure separate content rules for the domain name and VIP. In this situation, a domain name content rule without a VIP is sufficient because it will match on all content requests going to the domain regardless of the VIP.

An example of a topology where ARP-replying is not required is when an upstream router has the CSS statically configured as the next hop router for the VIPs. A domain name content rule is as follows:

```
content domainRule3
    protocol tcp
    port 80
    url "//www.domain.com/*"
    add service Serv1
    active
```

Using Wildcards in Domain Name Content Rules

You can use wildcards in domain names as part of the matching criteria for a content rule. Domain name wildcards work within the content rule hierarchy. That is, if a request for content matches more than one rule (including a wildcard domain name), the characteristics of the most specific rule determine how the CSS sets up the flow.

Note You cannot use wildcards with either a Domain Qualifier List (DQL) or a Uniform Resource Locator Qualifier List (URQL).

For example, the following content rule criteria have the highest precedence because, as a set, they provide the greatest specificity in matching content:

Domain name, IP address, protocol, port, URL

If you want to create a content rule using all these criteria, such as the configuration shown below, then the content rule matches only on the JPEG files that are found in the domain whose name starts with "*arr*", as well as the other criteria, including VIP address, protocol, and port number.

```
(config-owner-content[arrowpoint-rule1])# vip address 192.168.3.6
```

```
(config-owner-content[arrowpoint-rule1])# protocol tcp
(config-owner-content[arrowpoint-rule1])# port 80
(config-owner-content[arrowpoint-rule1])# url "//arr*.com/*.jpg"
```

When the CSS encounters a content rule with a wildcard domain name and matches according to the content rule hierarchy, it stops the search at that point. This behavior is consistent with the way that the CSS manages content rules in general.

For example, if the content request matches on the rule with VIP address 192.168.3.6 and URL /*, the CSS does not continue the search to match on a second rule with a wildcard VIP address (no address specified) and a URL of */*.jpg*. The specific address match makes the first rule more specific than the second rule.

To further clarify, if the match occurs on a rule with *//arrowpoint*.com/**, the search stops at that point and does not continue to match on a rule with *//arr*.com/*.gif*, because the first rule is a more specific match. Also note that a fully-specified domain name rule (arrowpoint.com) is more specific than a wildcard domain name rule (arr*.com).

For example, to have the content rule match on all instances of the text string "*arr*" in the domain name portion of the content rule, enter:

```
(config-owner-content[arrowpoint-rule1])# url "//www.arr*.com/*"
```

General Guidelines for Domain Name Wildcards in Content Rules

A domain name is made up of text strings called "words" and word separators called "dots" (.). The CSS parses the domain name from right word to left word. The CSS allows wildcards to be used as part of the domain name in one word or more than one word, but the wildcard cannot start the word.

For example, the CSS supports the following domain names:

- www.arr*.com
- arr*.com
- *.arr*.com
- arr*.home.com

Notice that the wildcard character either appears by itself as a domain word, or appears to the *right* of any characters that start a domain word. However, a wildcard character cannot start a domain name word.

For example, point.com:

- *point.com
- *.*point.com
- *point.home.com

Note You cannot use wildcards on the *rightmost* portion (for example, .com, .org, .gov) of the domain name. For this reason, the wildcard domain name syntax f* is not supported. You can use wildcards in any other words that make up the domain name.

Adding Services to a Content Rule

To add an existing service to a content rule, use the **add** command. Adding a service to a content rule includes it in the resource pool that the CSS uses for load balancing requests for content. The maximum number of services that you can add into a single content rule is 64. Note that a service may belong to multiple content rules. To see a list of services you can add to a content rule, use **add service ?** command.

Note You can only add local services to a content rule that contains either a Domain Qualifier List (DQL) or a service port range.

The **add service** command enables you to add the following types of services to a content rule:

- Service
- Primary Sorry Server
- Secondary Sorry Server

When you configure a Layer 3 or 4 content rule, the rule hits the local services. If:

- The local services are not active or configured, the rule hits the primary sorry server.
- The primary sorry server fails, the rule hits the secondary sorry server.

Redirect services and redirect content strings cannot be used with Layer 3 or 4 rules because they use the HTTP protocol.

When you configure a Layer 5 content rule, the CSS directs content requests to local services. If:

- The local services are not active or configured, the rule sends the HTTP redirects with the location of the redirect services to the clients.
- The local and redirect services are not active or configured, the rule forwards the HTTP requests to the primary sorry server.
- All services are down except the secondary sorry server, the rule forwards the HTTP requests to the secondary sorry server.

For information on configuring service types, refer to Chapter 1, Configuring Services, "Specifying a Service Type".

Note In some environments, URL, cookie strings, or HTTP header information can span over multiple packets. In these environments, the CSS can parse multiple packets for Layer 5 information before making load-balancing decisions. Through the global configuration mode **spanning-packets** command, the CSS can parse up to 20 packets with a default of 6. The CSS makes the load-balancing decision as soon as it finds a match and does not require parsing of all of the configured number of spanned packets. Because parsing multiple packets does impose a longer delay in connection, performance can be impacted by longer strings that span mulitple packets. For information on using the **spanning-packets** command, see "Specifying the Number of Spanned Packets" later in this chapter.

Adding a Service to a Content Rule

Use the **add service** command to add a service to a content rule. The maximum number of services that you can add into a single content rule is 64.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# add service serv2
```

Specifying a Service Weight

When you add a service to a content rule, you can assign a weight for the service using the **add service weight** option. The CSS uses this weight when you configure ACA or weighted roundrobin load balancing on the content rule. When you assign a higher weight to the service, the CSS redirects more requests to the service.

To set the weight for a service, enter a number from 1 to 10. The default is the weight configured for this service through the **(config-service) weight** command. By default, all services have a weight of 1.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# add service serv2 weight 3
```

Note When you add a service to content rules, the service weight as configured in service mode is applied to each rule as a server-specific attribute. Use the **add service weight** command to define a content rule-specific server weight. This command overrides the server-specific weight and applies only to the content rule to which you add the service. For information on the setting a weight on a service, refer to Chapter 1, Configuring Services, "Configuring Weight".

Adding a Primary Sorry Server to a Content Rule

Use the **primarySorryServer** command to configure the primary sorry service for a content rule. The CSS directs content requests to the primary sorry server when all other services are unavailable. You can configure this service to contain content, or to provide a drop or redirect message. This service is not used in load balancing.

Note If you configure the **persistence reset remap** command in the global configuration and **no persistent** command on the content rule, when a local service becomes available again, the CSS remaps any new or in progress persistent connections to the local server from the sorry server. Otherwise, new connections go to the available local services but in progress persistent connections stay on the sorry server.

Enter the server name as a case-sensitive unquoted text string with no spaces.

Note You can only add a primary sorry server to a rule if its range for the IP address or port is equal to the range for the IP address or port of each service on the rule. For example, if the rule has two services each with a range of three addresses, the primary sorry server must have a range of three addresses.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# primarySorryServer
slowserver
```

To remove a primary sorry service, enter:

```
(config-owner-content[arrowpoint-rule1])# no primarySorryServer
```

Adding a Secondary Sorry Server to a Content Rule

Use the **secondarySorryServer** command to configure the secondary sorry service for a content rule. A secondary sorry service is a backup service the CSS uses when the primary sorry service is unavailable. You can configure this service to contain content, or to provide a drop or redirect message. This service is not used in load balancing.

Enter the server name as a case-sensitive unquoted text string with no spaces.

Note You can only add a secondary sorry server to a rule if its range for the IP address or port is equal to the range for the IP address or port of each service on the rule. For example, if the rule has two services each with a range of three addresses, the secondary sorry server must have a range of three addresses.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# secondarySorryServer
slowestserver
```

To remove a secondary sorry service, enter:

```
(config-owner-content[arrowpoint-rule1])# no secondarySorryServer
```

Adding a Domain Name System to a Content Rule

To specify a DNS name that maps to a content rule, use the **add dns** command. The options for this command are:

- **add dns** *dns_name* - The DNS name mapped to the content rule. Enter the name as a case-sensitive unquoted text string with no spaces and a length of 1 to 31 characters.
- **add dns** *dns_name ttl_value* - The DNS name mapped to the content rule with the optional Time to Live (TTL) value in seconds. This value sets how long the DNS client remembers the IP address response to the query. Enter a value from 0 to 255. The default is 0.

Note When using the content **add dns** command, you must add DNS names in lowercase only. If you enter DNS names with a combination of uppercase and lowercase characters, a startup error appears and you must reenter the names in lowercase characters.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# add dns arrowpoint 120
```

To remove a DNS name mapped to the content rule, enter:

```
(config-owner-content[arrowpoint-rule1])# remove dns arrowpoint
```

Note To configure DNS server functionality on the CSS, use the **(config) dns-server** command.

Disabling a Domain Name System in a Content Rule

To disable DNS in a content rule, use the **dns-disable-local** command. The CSS informs other CSSs through an Application Peering Protocol (APP) session that the services related to the content rule are not available for DNS activities. However, the services remain active for other functions.

For example, to disable DNS for a specific content rule, enter:

```
(config-owner-content[arrowpoint-rule1])# dns-disable-local
```

To enable DNS in the content rule, use the **no dns-disable-local** command. For example, enter:

```
(config-owner-content[arrowpoint-rule1])# no dns-disable-local
```

Activating a Content Rule

Activating content enables the CSS to provide access to the content. To activate content, use the **active** command in the content mode to activate specific content.

Note Once a content rule is activated the following commands cannot be changed for the active content rule: **port**, **protocol**, **balance**, **dnsbalance**, **header-field-rule**, and **url**. In addition, you cannot remove the last remaining service from the content rule. If you need to make modifications to an active content rule, you must first suspend it.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# active
```

Suspending a Content Rule

Suspending a content rule deactivates it. Suspending a content rule:

- Prevents the CSS from providing access to the content
- Does not affect existing flows to the content

To suspend a content rule, use the **suspend** command in content mode. For example, enter:

```
(config-owner-content[arrowpoint-rule1])# suspend
```

Removing a Content Rule

To remove an existing content rule, use the **no content** command from owner mode. For example, enter:

```
(config-owner[arrowpoint])# no content rule1
```

Removing a Service from a Content Rule

To remove an existing service from a content rule, use the **remove** command from owner-content mode. Removing a service removes it from the resource pool that the CSS uses for balancing the load of requests for content governed by a rule. When you remove a service, the remaining services are rebalanced.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# remove service serv1
```

Configuring a Protocol

Specifying a protocol in a content rule enables the CSS to direct requests for content associated with the content rule to use a specific protocol. You may specify the following protocols for content:

- **any** (default, meaning the rule will match on a TCP or UDP port)
- **tcp**
- **udp**

To configure the TCP protocol for content, enter:

```
(config-owner-content[arrowpoint-rule1])# protocol tcp
```

To reset the protocol to the default of **any**, enter:

```
(config-owner-content[arrowpoint-rule1])# no protocol
```

Configuring a Port

Specifying a port enables the CSS to associate a content rule with a specific TCP/UDP port number. Specify a port number ranging from 0 to 65535. The default is 0, which indicates any port.

To configure a port for content, enter:

```
(config-owner-content[arrowpoint-rule1])# port 80
```

To reset the port number to the default of 0 value, enter:

```
(config-owner-content[arrowpoint-rule1])# no port
```

Configuring Load Balancing

To specify the load-balancing algorithm for a content rule, use the **balance** command available in content configuration mode. The options are:

- **balance aca** - ArrowPoint Content Awareness load-balancing algorithm (refer to Chapter 1, Configuring Services, "Using ArrowPoint Content Awareness Based on Server Load and Weight"). ACA balances the traffic over the services based on load or on server weight and load.
- **balance destip** - Destination IP address division algorithm. The CSS directs all client requests with the same destination IP address to the same service. This option is typically used in a caching environment.
- **balance domain** - Domain name division algorithm. The CSS divides the alphabet evenly across the number of caches. It parses the host tag for the first four letters following the first dot and then uses these characters of the domain name to determine to which server it should forward the request. This option is typically used in a caching environment.
- **balance domainhash** - Internal CSS hash algorithm based on the domain string. The CSS parses the host tag and does an exclusive XOR hash across the entire host name. It then uses the XOR hash value to determine to which server to forward the request. This method guarantees that all requests with the same host tag will be sent to the same server in order to increase the probability of a cache hit. This option is typically used in a caching environment.

Note If you are using the domainhash load-balancing method with proxy cache services, you may see duplicate sites across caches because the CSS balances on the first GET request in a persistent connection unless the subsequent GET request does not match a rule with the same proxy service specified. If you are concerned with duplicate hits across caches, reset persistence to remap and disable persistence on the rule. Issue the **(config) persistence reset remap** command globally and the **(config-owner-content) no persistent** command on the content rule.

- **balance leastconn** - Least connection algorithm. This balance method chooses a running service that has the least number of connections.

  We do not recommend that you use UDP content rules with the leastconn load-balancing algorithm. The service connection counters do not increment and remain at 0 because UDP is a connectionless protocol. Because the counters remain at 0, the CSS will give inconsistent results.

- **balance roundrobin** - Roundrobin algorithm (default). The CSS resolves the request by evenly distributing the load to resolve domain names among local and remote content domain sites.
- **balance srcip** - Source IP address division algorithm. The CSS directs all client requests coming from the same source IP address to the same service. This option is generally used in a caching configuration.
- **balance url** - URL division algorithm. The CSS divides the alphabet evenly across the number of caches. It then parses the URL for the first four characters located after the portion of the URL matched on by the rule. For example, if the URL in a content rule is configured for "*/news/**", the CSS will balance on the first four characters following "*/news/*". This option is typically used in a caching environment.
- **balance weightedrr** - Weighted roundrobin algorithm. The CSS uses roundrobin but weighs some services more heavily than others depending on the server's configured weight. All servers have a default weight of 1. To set a server weight, use the **add service weight** command in owner-content mode.
- **balance urlhash** - Internal CSS hash algorithm based on the URL string. The CSS parses the URL and performs an XOR hash across the URL. It then uses the XOR hash value to determine to which server to forward the request. This

method guarantees that all requests for the same URL will be sent to the same server in order to increase the probability of a cache hit. This option is typically used in a caching environment.

Note A Layer 5 content rule supports the HTTP CONNECT, GET, HEAD, POST, PUSH, and PUT methods. The CSS recognizes and forwards the following HTTP methods directly to the destination server in a transparent caching environment. Note that the CSS does not load balance these HTTP methods. RFC-2068: OPTIONS, TRACE; RFC-2518: PROPFIND, PROPPATCH, MKCOL, MOVE, LOCK, UNLOCK, COPY, DELETE.

In a transparent caching environment (for example, no VIP address on a Layer 5 content rule), the CSS bypasses these HTTP methods, and they are forwarded to the destination server.

For example, to specify **weightedrr** load balancing, enter:

```
(config-owner-content[arrowpoint-rule1])# balance weightedrr
```

To revert the balance type to the default of roundrobin, enter:

```
(config-owner-content[arrowpoint-rule1])# no balance
```

Configuring a DNS Balance Type

Use the **dnsbalance** command to determine where to resolve a request for a domain name into an IP address. The syntax and options for this content mode command are:

- **dnsbalance preferlocal** - Resolve the request to a local VIP address. If all local systems exceed their load threshold, the CSS chooses the least-loaded remote system VIP address as the resolved address for the domain name.
- **dnsbalance roundrobin** - Resolve the request by evenly distributing the load to resolve domain names among local and remote content domain sites. The CSS does not include sites that exceed their local load threshold.

- **dnsbalance leastloaded** - Resolve the request to the least-loaded of all local or remote domain sites. The CSS first compares load numbers. If the load number between domain sites is within 50, then the CSS compares their response times. The site with the fastest response time is considered the least-loaded site.
- **dnsbalance useownerdnsbalance** - Resolve the request by using the DNS load balancing method assigned to the owner. This is the default method for the content rule. If you do not configure an owner method, the CSS uses the default owner DNS load-balancing method of roundrobin. To configure a DNS balancing method for an owner, refer to Chapter 2, Configuring Owners, "Configuring an Owner DNS Balance Type".

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# dnsbalance roundrobin
```

To restore the DNS balance type to the default setting of using the owner's method, enter:

```
(config-owner-content[arrowpoint-rule1])# no dnsbalance
```

Configuring Hotlists

Use the **hotlist** command to define a hotlist that lists the content most requested (hot content) during a user-defined period of time. The CSS enables you to configure hotlist attributes for content rules. Defining hotlist attributes for a content rule enables you to determine which content is heavily accessed. With this information, you can accurately determine which content should be replicated.

Note You must configure and enable a hotlist for replication-store and replication-cache to work.

You can configure the following attributes for hotlists for specific content from config-owner-content mode:

- **hotlist** - Enable the hotlist. To enable a hotlist for a specific content rule, use the **hotlist** command from the corresponding owner-content mode. For example, enter:

```
(config-owner-content[arrowpoint-rule1])# hotlist
```

To disable a hotlist, enter:

```
(config-owner-content[arrowpoint-rule1])# no hotlist
```

- **hotlist interval** - Set the hotlist refresh interval. Enter the interval time in minutes from 1 to 60. The default is 1. For example, enter:

  ```
  (config-owner-content[arrowpoint-rule1])# hotlist interval 10
  ```

  To restore the hotlist interval to the default of 1, enter:

  ```
  (config-owner-content[arrowpoint-rule1])# no hotlist interval
  ```

- **hotlist size** - Set the size of the hotlist. Enter the total number of entries maintained for this rule from 1 to 100. The default is 10. For example, enter:

  ```
  (config-owner-content[arrowpoint-rule1])# hotlist size 10
  ```

  To restore the hotlist size to the default of 10, enter:

  ```
  (config-owner-content[arrowpoint-rule1])# no hotlist size
  ```

- **hotlist threshold** - Set the hotlist threshold. Enter an integer from 0 to 65535 to specify the threshold above which a piece of content is considered hot. The default is 0. For example, enter:

  ```
  (config-owner-content[arrowpoint-rule1])# hotlist threshold 9
  ```

  To restore the hotlist threshold default of 0, enter:

  ```
  (config-owner-content[arrowpoint-rule1])# no hotlist threshold
  ```

- **hotlist hitcount -** Set the hotlist type to hit count, how may times the content was accessed. For example, enter:

  ```
  (config-owner-content[arrowpoint-rule1])# hotlist type hitcount
  ```

  To restore the hotlist type to the default setting **hitCount**, enter:

  ```
  (config-owner-content[arrowpoint-rule1])# no hotlist type
  ```

To display hotlist information, use the **show domain hotlist** command. Table 3-2 describes the fields in the **show domain hotlist** output.

Table 3-2 Field Descriptions for the show domain hotlist Command

Field	Description
Hotlist Enabled/Disabled	Enable the domain hotlist. The domain hotlist is disabled by default.
Size	The configured maximum number of domain entries contained in the hotlist. The range is from 1 to 100. The default is 10.
Interval	The configured interval, in minutes, to refresh the domain hotlist and start a new list. The interval range is from 1 to 60. The default is 1.
Threshold	The configured number of domain hits per interval, which must be exceeded for a domain to be considered hot and added to the list. The threshold range is from 0 to 65535. The default is 0, which indicates that the threshold is disabled.
# Hot Domains	The total number of hot domains.
Hits	The number of hits for a hot domain.
Domain	The name of the hot domain associated with the Hits field.

Configuring a Domain Hotlist

Use the **domain** command to enable the domain hotlist and configure domain hotlist parameters. A domain hotlist lists the most accessed domains on a CSS during a user-defined period of time. The syntax and options are:

- **domain hotlist** - Enable the domain hotlist. The domain hotlist is disabled by default.
- **domain hotlist interval** *minutes* - Configure the interval to refresh the domain hotlist and start a new list. Enter the interval from 1 to 60 minutes. The default is 1 minute.
- **domain hotlist size** *max_entries* - Configure the maximum number of domain entries contained in the hotlist. Enter the maximum number of entries from 1 to 100. The default is 10 entries.

- **domain hotlist threshold** *number* - Configure the threshold, which is the number of domain hits per interval that must be exceeded for a domain to be considered hot and added to the list. Enter the threshold from 0 to 65535. The default is 0, which disables the threshold.

To enable a domain hotlist, enter:

```
(config)# domain hotlist
```

To disable the domain hotlist, enter:

```
(config)# no domain hotlist
```

To display the domain hotlist and its configuration, use the **show domain hotlist** command (see Table 3-2).

Specifying a Uniform Resource Locator

Use the **url** command to specify the Uniform Resource Locator (URL) for content and enable the CSS to access a remote service when a request for content matches the rule. Enter the URL as a quoted text string with a maximum length of 252 characters. Before you can change the URL for the content rule, you must remove the current URL first.

Note Do not include the ? or # parameter character in the URL string. The CSS terminates the URL at these parameter characters.

The syntax and options for this content mode command are:

- **url "/***url_name***"** - Specify the URL for the content as a quoted text string with a maximum length of 252 characters.
- **url "/**{*url_path*}**/*" eql** *eql_name* - Specify the URL for any content file that has its file extension defined in the specified Extension Qualifier List (EQL).
- **url "/**{*url_path*}**/*" dql** *dql_name* {*eql_name*} - Specify the URL for any content file that has its domain name defined in the specified Domain Qualifier List (DQL). You cannot use a DQL in conjunction with a domain name in a URL. You may optionally include an EQL after the DQL name to specify specific file extensions as part of the DQL matching criteria.

- **url urql** *urql_name* - Specify a URQL consisting of a group of URLs to this content rule. Note that you cannot specify both **url urql** and **application ssl** for the same content rule. You cannot configure a URQL with subscriber services.

The variables are:

- *url_name* - The URL for the content. Enter a quoted text string with a maximum length of 252 characters. You must place a slash character (/) at the beginning of the URL (for example, "/announcements/prize.html").

  To specify a domain name, place two slashes (//) at the beginning of the URl. For example, "//www.arrowpoint.com/*" allows the rule to match on HTTP traffic that contains the www.arrowpoint.com domain name in the HTTP host tag.

  Normally, port 80 traffic does not use a port number in the domain name. To specify a port other than port 80, enter the domain name with the port number exactly. Separate the domain name and the port number with a colon. For example, enter:

  ```
  (config-owner-content[arrowpoint-rule1])# url
  "//www.arrowpoint.com:8080/*"
  ```

  To use stickiness based on Secure Socket Layer (SSL) session ID, set the URL to /*. Also, set the port to 443 with the **(config-owner-content) port** command and enable stickiness with the **(config-owner-content) advanced-balance ssl** command. Then specify an SSL application type.

  You can specify certain wildcard operations for wildcard matching. Use a "*" character to specify a wildcard match. You can specify a maximum of eight directories. Each directory name can be a maximum of 32 characters with a total maximum of 252 characters in the URL. You can specify only one wildcard per URL.

  Examples of supported wildcards are:

  - **/*.html** - Matches all requests with the .html extension
  - **/announcements/*** - Matches all requests for files in the *announcements* directory
  - **/announcements/*.html** - Matches requests for files in the *announcements* directory having .html extensions
  - **/announcements/new/*.jpg** - Matches requests for all files in the *announcements/new* directory that contain the .jpg extension

- *url_path* - An optional path to any content file that has its file extension defined in the EQL. Enter a quoted text string. You must place:
  - A slash character (/) at the beginning of the quoted path
  - /* characters at the end of the quoted path

  For example, "/announcements/new/*".
- *eql_name* - The name of the EQL. To see a list of EQLs, use the **eql ?** command.
- *urql_name* - The name of the URQL. You can only assign one URQL per rule. To see a list of URQLs, use the **urql ?** command.

Note For caching environments, you can configure a domain content rule by placing two slash characters (//) at the front of the *url_name* or *url_path*. The rule matches HTTP traffic that contains the domain name in the HTTP host tag.

For example, to specify a URL that matches all requests for content in the *announcements* directory with .html extensions, enter:

```
(config-owner-content[arrowpoint-products.html])# url
"/announcements/*.html"
```

To remove a URL, enter:

```
(config-owner-content[arrowpoint-products.html])# no url
```

To remove a URQL from a URL, enter:

```
(config-owner-content[arrowpoint-products.html])# no url urql
```

To display a URL for a content rule, use the **show rule** command for the content rule.

Specifying an Extension Qualifier List in a Uniform Resource Locator

Server selections are based on the Uniform Resource Locator (URL) specified in the owner content rule. To enable the CSS to access a service when a request for content matches the extensions contained in a previously defined EQL, specify

the URL and EQL name for the content. For information on creating an EQL, refer to Chapter 5, Configuring Source Groups, ACLs, EQLs, URQLs, NQLs, and DQLs.

Specify a URL as a quoted text string with a maximum of 252 characters followed by **eql** and the EQL name.

Note Do not specify a file extension in the URL when you use an EQL in the URL or the CSS will return an error message. For example, the CSS will return an error message for the **url "/*.txt" eql Cacheable** command. The following command is valid; **url "/*" eql Cacheable**.

For example, enter:

```
(config-owner-content[arrowpoint-products.html])# url "/*" eql
graphics
```

The following example enables the CSS to direct all requests to the correct service for content that matches:

- Pathnames (*/customers/products*)
- Extensions listed in the EQL (*graphics*)

```
(config-owner-content[arrowpoint-products.html])# url
"/customers/products/*" eql graphics
```

To display a content rule EQL, use the **show rule** command.

Specifying the Number of Spanned Packets

Use the **spanning-packets** command to configure the number of packets spanned for the search of the HTTP header termination string. In some environments, URL, cookie strings, or HTTP header information can span over multiple packets. In these environments, the CSS can parse up to 20 packets for Layer 5 information before making load balancing decision. By default, the CSS parses 6 packets.

The CSS makes the load-balancing decision as soon as it finds a match and does not require parsing of all of the configured number of spanned packets. Because parsing multiple packets does impose a longer delay in connection, performance can be impacted by longer strings that span mulitple packets.

To change the number of packets, enter a number from 1 to 20. The default value is 6. For example, to configure the number of packets spanned to 10, enter:

```
(config)# spanning-packets 10
```

To reset the number of packets spanned to the default value of 6, enter:

```
(config)# no spanning-packets
```

Specifying a Load Threshold

Use the **load-threshold** command to set the normalized load threshold for the availability of each local service on a content rule. When the service load metric exceeds this threshold, the local service becomes unavailable and is redirected to remote services. To define a remote service, use the service mode **type redirect** command (refer to Chapter 1, Configuring Services, "Specifying a Service Type").

Enter the load threshold as an integer from 2 to 254. The default is 254, which is the maximum threshold a service can reach before becoming unavailable. To view the load on services, use **show service**. For example, enter:

```
(config-owner-content[arrowpoint-rule1])# load-threshold 100
```

To reset the load threshold to its default value of 254, enter:

```
(config-owner-content[arrowpoint-rule1])# no load-threshold
```

Redirecting Requests for Content

Use the **redirect** command to set HTTP status code 302 for a content rule and specify the alternate location of the content governed by a rule. Use this command to:

- Make the content unavailable to subsequent requests at its current address.
- Provide a URL to send back to the requestor. You must add a URL to the content rule for redirect to force the HTTP request. For example, url "/*". Enter the URL as a quoted text string with a maximum of 64 characters.

Note If you also set status code 404 (drop message) for content, code 302 takes priority.

Do not configure a service for a redirect-only content rule.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# redirect
"//www.arrowpoint.com/newlocation.html"
```

To delete the redirect URL, enter:

```
(config-owner-content[arrowpoint-rule1])# no redirect
```

Enabling TCP Flow Reset Reject

Use the **flow-reset-reject** command to enable the CSS flow manager subsystem to send a TCP RST (reset) frame when a flow for requested content is mapped to a destination IP address that is no longer reachable. The **flow-reset-reject** command prevents a CSS client from hanging up and retransmitting when the request can never be serviced. In addition, for UDP flows the command allows the CSS to purge the flow cache of the UDP flow so that another request gets remapped to a different IP address, if necessary, without attempting to use the previously mapped IP address. The **flow-reset-reject** command is applied on a per content rule basis.

Use the **no** form of this command to disable the sending of the TCP RST frame to the client.

To enable the CSS to send a TCP RST frame, enter:

```
(config-owner-content[rule1])# flow-reset-reject
```

To reset the CSS back to the default state of not sending a TCP RST frame, enter:

```
(config-owner-content[rule1])# no flow-reset-reject
```

Configuring Persistence, Remapping, and Redirection

During the life of a persistent connection, a CSS must determine if it needs to move a client connection to a new service based on content rules, load balancing, and service availability. In some situations, moving the client connection is not necessary; in other situations, it is mandatory. This section describes how to configure the CSS to make these decisions using:

- Content rule persistence
- Bypass persistence
- HTTP Redirection
- Service Remapping

Content Rule Persistence

When a CSS receives a request for content from a client, the software checks if the request matches on a content rule to determine the best service to handle the request. If the request matches on a content rule, the CSS establishes a client connection to the best service specified by the content rule. By default, the CSS keeps the client on the same connection for an entire flow session as long as a new content request:

- Matches on the same content rule that specified the current service
- Matches on a new content rule that contains the current service, even if a different best service is specified by the content rule

This CSS behavior is known as *content rule persistence*. If you are using transparent caches (which prefetch content) or mirrored-content servers, this scheme works well because the same content is available on each service.

Use the **persistent** command in content configuration mode to maintain a persistent connection with a server as long as the above criteria are met. By default, persistence is enabled. Disabling persistence allows the CSS to move a connection to a better service on the same rule or to use cache bypass functionality (EQLs or failover bypass).

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# persistent
```

Use the **no persistent** command on a content rule with:

- A balance method of domain or domain hash when using proxy caches
- A balance method of url or urlhash when using transparent caches
- A failover method of bypass when using transparent caches
- An EQL bypass with a transparent cache
- Adding a sorry server to a content rule

To disable persistence:

```
(config-owner-content[arrowpoint-rule1])# no persistent
```

Note If a request for content on a persistent connection matches on a new content rule that does not contain the current service, or persistence is disabled and there is a better service configured in the content rule, the CSS redirects or remaps the current connection to a new best service based on the setting of the **persistence reset** command, if configured. If you do not configure **persistence reset**, the CSS performs an HTTP redirect by default. For details on HTTP redirection, see "Configuring HTTP Redirection and Service Remapping" later in this chapter.

Configuring Bypass Persistence

If a CSS bypasses a service (for example, a transparent cache is down and **failover bypass** is configured) and the next content request on the same TCP connection matches on a content rule that contains the transparent cache that was down, the CSS will continue to bypass the cache, by default, even after the bypassed cache is back online. In this case, the CSS typically sends the content request to the origin server. This behavior is called *bypass persistence*.

You can configure the CSS to redirect or remap a bypassed connection using the **bypass persistence** global config command in conjunction with the **persistence reset** command.

Use the **bypass persistence** command to determine if the CSS performs either a remapping or redirection operation to reset a bypassed service when a content request matches on a content rule, but a previous request caused the bypass. This global command affects all flows. By default, bypass persistence is enabled.

For example, enter:

```
(config)# bypass persistence disable
```

The CSS uses remapping or redirection to reset the connection according to the setting of the **persistence reset** method.

```
(config)# bypass persistence enable
```

The CSS does not use remapping or redirection to reset the connection and continues to bypass a service.

Configuring HTTP Redirection and Service Remapping

If you need to place different content on different servers (for example, to conserve server disk space, for load-balancing considerations, or when using proxy caches), content rule persistence is not useful. In this case, you can disable persistence by using the **no persistent** command, described in "Content Rule Persistence" earlier in this section.

When the CSS receives a request for content that is not available on the current service, it must reset the current connection to the service and establish a new connection to another service (for example, a different proxy cache or the origin server) that contains the requested content. You can accomplish this in either of the following ways:

- **Redirection** - An HTTP technique that resets both the client-to-CSS (front-end) connection and the CSS-to-service (back-end) connection, then establishes a new flow to the best service that contains the requested content.
- **Service Remapping** - A technique that resets only the back-end connection to the current service and then creates a new back-end connection to the best service that contains the requested content. This technique is faster and more efficient than redirection because the CSS does not need to reset and then reestablish the front-end connection. With service remapping, the CSS strictly manages portmapping to prevent the occurrence of duplicate port numbers.

Note Service remapping is incompatible with stateless redundancy failover (the **redundancy-l4-stateless** command). Service remapping enables CSS portmapping, which source-port NATs all flows. Stateless redundancy failover requires that the CSS not NAT source ports. For more information on stateless redundancy failover, refer to the *Cisco Content Services Switch Advanced Configuration Guide*, Chapter 7, Configuring Redundant Content Services Switches.

Use the **persistence reset** command with the content rule **no persistent** command to cause an HTTP redirection or perform a back-end remapping operation when resetting a connection to a new back-end service. The global **persistence reset** command affects all flow setups that require redirection or remapping.

For example, to enable redirection:

```
(config)# persistence reset redirect
```

For example, to enable service remapping:

```
(config)# persistence reset remap
```

Note The CSS does not use remapping when selecting redirect type services. Refer to Chapter 1, Configuring Services, "Specifying a Service Type".

Specifying an HTTP Redirect String

Use the **redirect-string** command to specify an HTTP redirect string to be used when an HTTP redirect service generates an "object moved" message for the service. The CSS uses the entire configured redirect string as the new location for the requested content. If no string is configured, the CSS prepends the domain configured with the **(config-service) domain** command to the original request. If neither the redirect string nor the domain name are configured, the CSS uses the domain in the host-tag field from the original request combined with the requested HTTP content. If no host tag is found, the CSS uses the IP address of the service to generate the redirect.

Note You can use a redirect string only on a service type redirect.

Note The **redirect-string** and **(config-service) domain** commands are similar. The CSS returns the **redirect-string** command string verbatim as configured. However, the CSS prepends the domain configured with the **(config-service) domain** command to the original requested URL.

Note You cannot configure the **redirect-string** and **(config-service) domain** commands simultaneously on the same service.

The syntax for this service mode command is:

redirect-string *string*

Enter the HTTP redirect string as a quoted or an unquoted text string with no spaces and a maximum length of 64 characters. Use quotation marks when you want to include a question mark in the string.

For example, enter:

```
(config-service[serv1])# redirect-string www.arrowpoint.com
```

To remove the redirect string from the service, enter:

```
(config-service[serv1])# no redirect-string www.arrowpoint.com
```

Using Show Remap

Use the **show remap** command to display the configured **persistence reset** and **bypass persistence** settings. This command is available in all modes except RMON, URQL, and VLAN configuration modes.

Table 3-3 describes the fields in the **show remap** output.

Table 3-3 Field Descriptions for the show remap Command

Field	Description
Group SFP Port Map Info	This field is currently not used.
Persistence Reset Method	The configured persistence reset method when resetting a connection to a new back-end service. The possible methods are: • **redirect** - Causes an HTTP redirection when resetting a connection to a new back-end service. An HTTP redirection resets both sides of the connection. • **remap** - Uses a back-end remapping operation when resetting a connection to a new back-end service.
Bypass Persistence	The configured bypass persistence setting. The possible settings are: • **disable** - The CSS performs either a service remapping or HTTP redirection operation to reset a bypassed service when a content request matches on a content rule, but a previous request caused the bypass. • **enable** - The CSS does not perform remapping or redirection to reset the connection and continues to bypass a service. By default, bypass persistence is enabled.

Defining Failover

Note The CSS supports Adaptive Session Redundancy (ASR) on 11500 series CSS peers in an active-backup VIP redundancy and virtual IP interface redundancy environment to provide stateful failover of existing flows. For details on ASR, refer to the *Cisco Content Services Switch Advanced Configuration Guide*, Chapter 6, Configuring VIP and Virtual IP Interface Redundancy.

To define how the CSS handles content requests when a service fails or is suspended, use the **failover** command. For the CSS to use this setting, ensure that you configure a keepalive for each service; that is, do not set the keepalive type to **none** (the keepalive default is ICMP). The CSS uses the keepalive settings to monitor the services to determine server health and availability.

The failover command applies to the following caching load balancing types:

- **balance domain**
- **balance url**
- **balance srcip**
- **balance destip**
- **balance domainhash**
- **balance urlhash**

Note If you remove a service (using the **remove service** command), the CSS rebalances the remaining services. The CSS does not use the failover setting.

This command supports the following options:

- **failover bypass** - Bypass all failed services and send the content request directly to the origin server. This option is used in a proxy or transparent cache environment when you want to bypass the failed cache and send the content request directly to the server that contains the content.
- **failover linear** (default) - Distribute the content request evenly between the remaining services.
- **failover next** - Send the content requests to the cache service next to the failed service. The CSS selects the service to redirect content requests to by referring to the order in which you configured the services.

For example, enter:

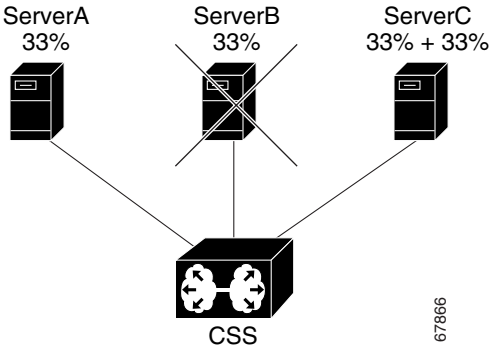
```
(config-owner-content[arrowpoint-rule1])# failover bypass
```

To restore the default setting of **failover linear**, enter:

```
(config-owner-content[arrowpoint-rule1])# no failover
```

Figure 3-3 shows three cache services configured for failover **next**. If ServerB fails, the CSS sends ServerB content requests to ServerC, which was configured after ServerB in the content rule.

Figure 3-3 ServerB Configured for Failover Next



As shown in Figure 3-4, if ServerC fails, the CSS sends ServerC content requests to ServerA because no other services were configured after ServerC.

Figure 3-4 ServerC Configured for Failover Next

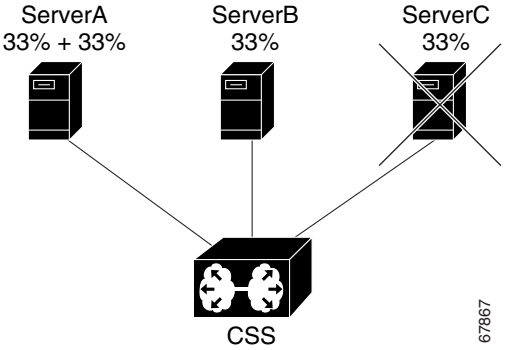


Figure 3-5 shows three cache services configured for failover **linear**. If you suspend ServerB or if it fails, the CSS does not rebalance the services. It evenly distribute ServerB cache workload between servers A and C.

Note that Figure 3-5 and Figure 3-6 use the alphabet to illustrate division balance.

Figure 3-5 Suspended or Failed Service Configured for Failover Linear

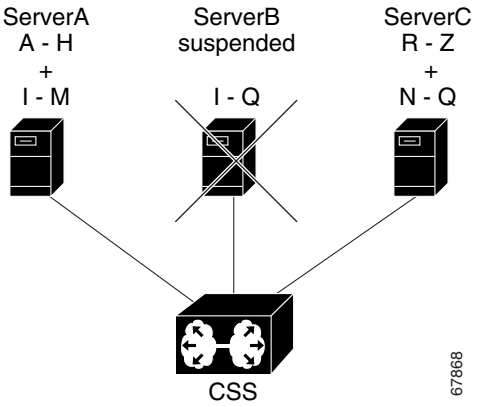
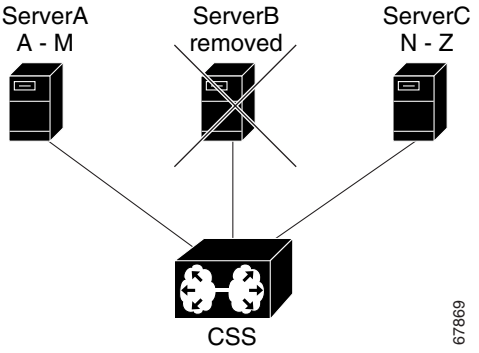


Figure 3-6 also shows three cache services configured for failover **linear**, but in this example, you *remove* ServerB using the **remove service** command from owner-content mode. Because the CSS does not apply the failover setting when you remove a service, it rebalances the remaining services.

Figure 3-6 Removing a Service Configured for Failover Linear



Specifying an Application Type

To specify the application type associated with a content rule, use the **application** command. The application type enables the CSS to correctly interpret the data stream to match and parse the content rule. Otherwise, the data stream packets are rejected. Define an application type for non-standard ports.

When configuring Layer 5 content rules for an application other than HTTP, use the appropriate **application** type to enable the Layer 5 rule to function.

Note A Layer 5 content rule supports the HTTP CONNECT, GET, HEAD, POST, PUSH, and PUT methods. The CSS recognizes and forwards the following HTTP methods directly to the destination server in a transparent caching environment. Note that the CSS does not load balance these HTTP methods. RFC-2068: OPTIONS, TRACE; RFC-2518: PROPFIND, PROPPATCH, MKCOL, MOVE, LOCK, UNLOCK, COPY, DELETE

The **application** command enables you to specify the following application types:

- **bypass** - Bypass the matching of a content rule and sends the request directly to the origin server
- **ftp-control** - Process FTP data streams
- **http** (default) - Process HTTP data streams
- **realaudio-control -** Process RealAudio Control data streams
- **ssl -** Process Secure Socket Layer (SSL) protocol data streams

Note You cannot specify both **url urql** and **application ssl** for the same content rule.

Note Cisco recommends that the **application ssl** command always be configured in conjunction with the **advanced-balance ssl** command (refer to Chapter 4, Configuring Sticky Parameters for Content Rules, "Specifying an Advanced Load-Balancing Method for Sticky Content"). The **application ssl** command causes the CSS to spoof a connection so that you see the response come back from the server. The **advanced-balance ssl** command causes the CSS to look for the SSL session ID coming from the server and stick the client to the server based on that session ID. Once a flow is setup, the **application ssl** command causes the CSS to treat the flow as a Layer 4 flow and does not inspect the flow for Layer 5 data in order to prevent the CSS from mis-interpreting encrypted data.

For example, in a content rule that specifies port 21, you may want to configure the application type as **ftp-control**. Configuring the content rule to application type **ftp-control** instructs the CSS to process only FTP requests coming into port 21.

```
(config-owner-content[arrowpoint-rule1])# application ftp-control
```

For example, the following owner portion of a startup-config shows a content rule configured for **application ftp-control**.

```
!************************* OWNER ************************
owner arrowpoint
content ftprule
vip address 192.3.6.58
protocol tcp
port 21
application ftp-control
```

```
add serv1
add serv3
active
```

To remove an application type, enter:

```
(config-owner-content[arrowpoint-rule1])# no application
```

Enabling Content Requests to Bypass Transparent Caches

Use the **param-bypass** command to enable content requests to bypass transparent caches when the CSS detects special terminators in the requests. These terminators include "#" and "?" which indicate that the content is dependent on the arguments that follow the terminators. Because the content returned by the server is dependent on the content request itself, the returned content is deemed as not cacheable, and the content request is directed to the origin server.

This command contains the following options:

- **param-bypass disable** (default) - Content requests with special terminators do not bypass transparent caches.
- **param-bypass enable** - Content requests with special terminators bypass transparent caches and are forwarded to the origin server.

For example, to enable the **param-bypass** command, enter:

```
(config-owner-content[arrowpoint-rule1])# param-bypass enable
```

Showing Content

The **show content** command enables you to display content entries in the Content Service Database (CSD) of the 11500 series CSS. This command is available in all modes.

To display content from a specific module, and content entry location, in either the CSS 11503 or CSS 11506, specify the **show content** command as follows:

show content slot *slot_number* {**start-index** *index_number*}

The variables and options are:

- **slot** *slot_number* - Display content from the module located in a specific slot in the CSS 1150 or CSS 1150 chassis. For the CSS 11503, the available choices are 1 through 3. For the CSS 11506, the available choices are 1 through 6. If you do not specify a slot number, the CSS displays the content entries from the SCM in slot 1 of the CSS.
- **start-index** *index_number* - Display content entries starting at the specified *index_number* parameter. This variable defines where you want to start browsing CSS content. Starting from the specified index number, you receive up to a maximum of 64k of information. To see additional information, issue the **show content** command again, starting from the last index number displayed. To specify an index number, enter a number from 0 to 4095. If you do not specify a start-index the CSS displays the content entries starting from 0.

Use the **show content** command with no options or variables to show all content entries in the Content Service Database for a CSS 11501, 11503, or 11506.

For example, to look at the content from the module in CSS 11503 chassis slot 2, starting at index 150, enter:

```
(config)# show content slot 2 start-index 150
```

Table 3-4 describes the fields in the **show content** output.

Note URQL entries are flagged with an asterisk (*) in the **show content** output.

Table 3-4 Field Descriptions for the show content Command

Field	Description
Pieces of Content for Slot	The chassis slot number in which the module resides.
Subslot	The module slot number in which the Session Processor resides.
Total Content	The total number of content entries.
Index	Unique index for known content in the CSD.
<address>	The IP address of the content.

Table 3-4 Field Descriptions for the show content Command (continued)

Field	Description
Protocol	The IP Protocol of the content.
Port	Protocol port of the content.
Best Effort	The QoS class of the content. This field is not used by the CSS at this time.
Streamed	Identifies if the piece of content is streaming media (video or audio). This field is not used by the CSS at this time.
URL	The Universal Resource Locator of the content.
Domain	The domain name of the content.

Showing Content Rules

The **show rule** command displays content rule information for specific content rules or all content rules currently configured in the CSS. Use the following **show rule** commands from any mode:

- **show rule** - Display all owners and content rules currently configured in the CSS
- **show rule-summary** - Display a summary of owner content information
- **show rule** *owner_name* - Display information identical to the show rule command, but only for the specified owner's content
- **show rule** *owner_name content_rule_name* - Display information identical to the show rule command, but only for a specific owner and content
- **show rule** *owner_name content_rule_name* **acl** - Display the ACL attributes for the specified content rule
- **show rule** *owner_name content_rule_name* **all** - Displays all attributes for the specified content rule
- **show rule** *owner_name content_rule_name* **dns** - Display the DNS attributes for the specified content rule
- **show rule** *owner_name content_rule_name* **header-field** - Display the header-field attributes for the specified content rule

- **show rule** *owner_name content_rule_name* **hot-list** - Display the hotlist attributes for the specified content rule
- **show rule** *owner_name content_rule_name* **services** - Display the services for the specified content rule
- **show rule** *owner_name content_rule_name* **statistics** - Display the statistics for the specified content rule
- **show rule** *owner_name content_rule_name* **sticky** - Display the sticky attributes for the specified content rule

To display all content rule information, enter:

```
# show rule
```

To display the summary for all content rules, enter:

```
# show rule-summary
```

To display all rule attributes for an owner, enter:

```
# show rule owner content_rule all
```

Note The CntRuleName and OwnerName fields display the first 16 characters of the configured data. The URL field displays the first 10 characters of configured data.

Table 3-5 describes the fields in the **show rule** output.

Table 3-5 Field Descriptions for the show rule Command

Field	Description
Name	The name of the content rule.
Owner	The owner of the rule.
Author	The author (Local CSS or remote CSS peer) of the rule.
Index	A CSS assigned unique index for the rule. The number is based in the order that the rule was created.
State	The state of the rule (active or suspend).

Table 3-5 Field Descriptions for the show rule Command (continued)

Field	Description
Type	The application type associated with the rule. The possible values are: • **bypass** - Bypass the matching of the content rule and send the request directly to the origin server • **http** - Process HTTP data streams (default) • **ftp-control** - Process FTP data streams • **realaudio-control** - Process RealAudio Control data streams • **ssl** - Process Secure Socket Layer (SSL) protocol data streams
L3	Destination IP address.
L4	Destination protocol and port.
URL	The URL for the content.
URQL	The name of the associated URL Qualifier list.
EQL	The name of the associated EQL.
DQL	The name of the associated DQL.
Header Field Group	The name of the associated header-field group.
Total Bytes	Total bytes to the content rule.
Total Frames	Total frames to the content rule.
Total Redirects	Total redirects by the content rule (when the **redirect** command is configured for a content rule). This field increments whenever a request for content is redirected to an alternate location.
Total Rejects	Total rejects by the content rule. This field increments when all services for a content rule are unavailable.
Overload Rejects	Total rejects on the content rule due to overload on the rule's available services.

Table 3-5 Field Descriptions for the show rule Command (continued)

Field	Description
Balance	The load-balancing algorithm for the content rule. The possible values are: • **ACA** - ArrowPoint Content Awareness algorithm. The CSS correlates content request frequency with the server's cache sizes to improve cache hit rates for that server. • **destip** - Destination IP address division. The CSS directs all client requests with the same destination IP address to the same service. • **domain** - Domain name division. The CSS uses the domain name in the request URI to direct the client request to the appropriate service. • **domainhash** - Internal CSS hash algorithm based on the domain string. The CSS uses the algorithm to hash the entire domain string. Then, the CSS uses the hash result to choose the server.

Table 3-5 Field Descriptions for the show rule Command (continued)

Field	Description
Balance (continued)	• **leastconn** - Least connections. The CSS chooses a running service that has the least number of connections. • **roundrobin** - Roundrobin algorithm (default). • **srcip** - Source IP address division. The CSS directs all client requests with the same source IP address to the same service. • **url** - URL division. The CSS uses the URL (omitting the leading slash) in the redirect URL to direct the client requests to the appropriate service. • **urlhash** - Internal CSS hash algorithm based on the URL string. The CSS uses the algorithm to hash the entire URL string. Then, the CSS uses the hash result to choose the server. • **weightedrr** - Weighted roundrobin algorithm. The CSS uses the roundrobin algorithm but weighs some services more heavily than others. You can configure the weight of a service when you add it to the rule.

Table 3-5 Field Descriptions for the show rule Command (continued)

Field	Description
Advanced Balance	The advanced load-balancing method for the content rule, including stickiness (as described in Chapter 4, Configuring Sticky Parameters for Content Rules). The possible values are: • **arrowpoint-cookie** - Enables the content rule to stick the client to the server based on the unique service identifier information of the selected server in the ArrowPoint-generated cookie. • **cookies** - Enables the content rule to stick the client to the server based on the configured string found in the HTTP cookie header. You must specify a port in the content rule to use this option. The CSS will then spoof the connection. • **cookieurl** - This is the same as advanced-balance cookies, but if the CSS cannot find the cookie header in the HTTP packet, this type of failover looks up the URL extensions (that is, the portion after the "?" in the URL) based on the same string criteria. You can use this option with any Layer 5 HTTP content rule.

Table 3-5 Field Descriptions for the show rule Command (continued)

Field	Description
Advanced Balance (continued)	• **none** - Disables the advanced-balancing method for the rule. This is the default setting. • **sticky-srcip** - Enables the content rule to stick a client to a server based on the client IP address, also known as Layer 3 stickiness. You can use this option with Layer 3, 4, or 5 content rules. • **sticky-srcip-dstport** - Enables the content rule to stick a client to a server based on both the client IP address and the server destination port number, also known as Layer 4 stickiness. You can use this option with Layer 4 or 5 content rules. • **ssl** - Enables the content rule to stick the client to the server based on the Secure Socket Layer (SSL) version 3 session ID assigned by the server. The application type must be SSL for the content rule. You must specify a port in the content rule to use this option. The CSS will then spoof the connection. • **url** - Enables the content rule to stick a client to a server based on a configured string found in the URL of the HTTP request. You must specify a port in the content rule to use this option. The CSS will then spoof the connection.
Sticky Mask	The subnet mask used for stickiness. The default is 255.255.255.255.
Sticky Inactivity timeout	The inactivity timeout period on a sticky connection for a content rule before the CSS removes the sticky entry from the sticky table. The range is from 0 to 65535 minutes. The default value is 0, which means this feature is disabled.

Table 3-5 Field Descriptions for the show rule Command (continued)

Field	Description
Sticky No Cookie Found Action	The action the CSS should take for a sticky cookie content rule when it cannot locate the cookie header or the specified cookie string in the client request. The possible values are: • **loadbalance** - The CSS uses the configured balanced method when no cookie is found in the client request. This is the default setting. • **redirect "URL"** - The CSS redirects the client request to a specified URL string when no cookie found in the client request. When using this option, you must also specify a redirect URL. Enter the redirect URL as a quoted text string from 0 to 64 characters. • **reject** - The CSS rejects the client request when no cookie is found in the request. • **service name** - The CSS sends the no cookie client request to the specified service when no cookie is found in the request.

Table 3-5 Field Descriptions for the show rule Command (continued)

Field	Description
Sticky Server Down Failover	The action that the CSS should take when a sticky string is found but the associated service has failed or is suspended. The possible values are: • **Balance** - The failover method uses a service based on the configured load balancing method (default). • **Redirect** - The failover method uses a service based on the currently configured redirect string. If a redirect string is not configured, the load balancing method is used. • **Reject** - The failover method rejects the content request. • **Sticky-srcip** - The failover method uses a service based on the client IP address. This is dependent on the sticky configuration. • **Sticky-srcip-dstport** - The failover method uses a service based on the client IP address and the server destination port. This is dependent on the sticky configuration.
ArrowPoint Cookie Path	The pathname where you want to send the ArrowPoint cookie. The default path of the cookie is "/".
ArrowPoint Cookie Expiration	The expiration time that the CSS compares with the time associated with the ArrowPoint cookie. If you do not set an expiration time, the cookie expires when the client exits the browser.

Table 3-5 Field Descriptions for the show rule Command (continued)

Field	Description
String Match Criteria	The string criteria to derive string results and the method to choose a destination server for the result. The string result is a sticky string in the cookie header, URL, or URL extension based on a sticky type being configured. See the following fields.
String Range	The starting and ending byte positions within a cookie, URL, or URL extension from a client. By specifying the range of bytes, the CSS processes the information located only within the range. • The range is from 1 to 1999. The default starting byte position is 1. • The range is from 2 to 2000. The default ending byte position is 100.
String Prefix	The string prefix located in the sticky range. If you do not configure the string prefix, the string functions start from the beginning of the cookie, URL, or URL extension, depending on the sticky type. If the string prefix is configured but is not found in the specified sticky range, load balancing defaults to the roundrobin method. The default has no prefix (“”).
String Eos-Char	The ASCII characters as the delimiters for the sticky string.
String Ascii-Conversion	Whether to enable or disable the ASCII conversion of escaped special characters within the specified sticky range before applying any processing to the string. By default, ACSII conversion is enabled.
String Skip-Len	The number of bytes to skip after the end of the prefix to find the string result. The default is 0. The range is from 0 to 64.

Table 3-5 Field Descriptions for the show rule Command (continued)

Field	Description
String Process-Len	The number of bytes, after the end of the prefix designated by the **string prefix** command and skipping the bytes designated by the **string skip-length** command, that the string operation will use. The range is from 0 to 64. The default is 0.
String Operation	The method to choose a destination server for a string result; derived from the settings of the string criteria commands. The possible values are: • **match-service-cookie** - Choose a server by matching a service cookie in the sticky string. This is the default setting. When a match is not found, the server is chosen by using the configured balance method (for example, roundrobin). This is the default method. • **hash-a** - Apply a basic hash algorithm on the hash string to generate the hash key. • **hash-crc32** - Apply the CRC32 algorithm on the hash string to generate a hash key. • **hash-xor** - Exclusive OR (XOR) each byte of the hash string to derive the final hash key.
Redirect	Text used to build an HTTP 302 redirect message that is sent to the client when the rule is matched.
Persistence	Whether or not a persistent connection with a server is maintained. By default, persistence is enabled.
Param-Bypass	Whether or not content requests bypass transparent caches when the CSS detects special terminators in the requests. These terminators include “#” and “?” which indicate that the content is dependent on the arguments that follow the terminators. Bypass is disabled by default.

Table 3-5 Field Descriptions for the show rule Command (continued)

Field	Description
Session Redundancy	Indicates whether Adaptive Session Redundancy (ASR) is enabled or disabled on the rule. For details on ASR, refer to the *Cisco Content Services Switch Advanced Configuration Guide*.
Redund Glb Index	The unique global index value for Adaptive Session Redundancy assigned to the content rule using the **redundant-index** command in owner-content configuration mode.
IP Redundancy	The state of IP redundancy if configured on the rule. Possible values are: Master, Backup, or Down. If IP redundancy is not configured, the state is Not Redundant.
Flow Timeout Multiplier	Number of seconds that a flow remains idle before the CSS reclaims the flow resources, as configured with the **flow-timeout-multiplier** command. For details on the **flow-timeout-multiplier** command, refer to the *Cisco Content Services Switch Administration Guide*.
Rule Services	Content rule services to configuration and statistic information, as follows.
Local Load Threshold	The normalized load threshold for the availability of each local service on the content rule. When the service load metric exceeds this threshold, the local service becomes unavailable and is redirected to the remote services. The range is from 2 through 254. The default is 254, which is the maximum load. A load of 255 indicates that the service is down
PrimarySorryServer	The primary service to be used when all other services for the content rule are unavailable.
SecondSorryServer	The secondary service to be used when all other services for the content rule are unavailable.
Name	The names of the services.
Hits	The number of content hits on the service.

Table 3-5 Field Descriptions for the show rule Command (continued)

Field	Description
Wgt	The weight for the service used when you configure ACA, weighted roundrobin, and DFP load balancing on the content rule. With a higher weight, the CSS redirects more requests to the service. The letters preceding the weight numbers have the following meanings: • D = Weight reported by DFP • R = Weight configured for a service using the **add service weight** command in owner-content mode • S = Weight configured for a service using the **weight** command in service mode
State	The state of the service.
Ld	The service load. The range is from 2 to 255. 255 indicates that the service is unavailable.
KAlive	The service keepalive type.
Conn	The number of connections currently mapped to the service.
DNS	The number of times that the CSS DNS resolver chose the service as the answer to a DNS client query.
DNS Names	Domain Name System names.
DNS TTL	The time to Live value in seconds, which determines how long the DNS client remembers the IP address response to the query.

Table 3-5 Field Descriptions for the show rule Command (continued)

Field	Description
DNS Balance	Where the CSS resolves a request for a domain name into an IP address. The possible values are: • **leastloaded** - Resolves the request to the least-loaded local or remote domain site. The CSS first compares load numbers. If the load number between domain sites is within 50, then the CSS compares their response times. The site with the fastest response time is considered the least-loaded site. • **Preferlocal** - Resolves the request to a local VIP address. If all local systems exceed their load threshold, the CSS chooses the least-loaded remote system VIP address as the resolved address for the domain name. • **roundrobin** - Resolves the request by evenly distributing the load to resolve domain names amongst content domain sites, local and remote. The CSS does not include sites that exceed their local load threshold. • **useownerdnsbalance** - Resolves the request by using the DNS load-balancing method assigned to the owner. This is the default method for the content rule. If you do not implicitly set an owner method, the CSS uses the default owner DNS load-balancing method of roundrobin.

Table 3-5 Field Descriptions for the show rule Command (continued)

Field	Description
Hotlist	Whether or not hotlist is enabled.
Size	The total number of hotlist entries that is maintained for the rule. The range is from 1 to 100. The default is 10.
Type	The hotlist type. Currently, the CSS supports only the hit count hotlist type, which is the default setting. Hit count is the number of times that the content is accessed.
Threshold	The hit count per interval threshold below which content is not considered hot. The range is from 0 to 65535. The default is 0.
Interval	The interval, in minutes, for refreshing the hotlist. The range is from 1 to 60. The default is 1.
Associated ACLs	The ACLs associated with a content rule.
TCP RST Client If Service Unreachable	Whether or not the **flow-reset-reject** command is enabled to allow the CSS's flow manager subsystem to send a TCP RST (reset) frame when a flow is mapped to a service that is no longer reachable. By default, the **flow-reset-reject** command is disabled.

Clearing Counters in a Content Rule

The CSS allows you to clear counters:

- Associated with all content rules or only the current content rule
- Associated with a single service or for all services in a content rule

Use the **zero** command and its options to clear the counters for content rules or services associated with content rules, and set the counters to zero.

This section covers:

- Clearing Counters for Content Rules
- Clearing Service Statistics Counters in a Content Rule

Clearing Counters for Content Rules

To reset the counters for all content rules to zero, use the **zero all** command. The reset counter statistics appear as zero in the **show summary** display.

Note If you issue the **zero** command without an option, only the counters for the current content rule are set to zero.

For example, enter:

```
(config-owner-content[rule1])# zero all
```

Clearing Service Statistics Counters in a Content Rule

To clear a service statistics counter for all CSS services associated with a content rule, use the **zero** command. To clear a service statistics counter for a specific service in the content rule, use the **zero** command and identify the name of the service. In this case, only the counter for the specified service is set to zero.

The reset statistics appear as 0 in the **show service** display.

You can issue the following **zero** commands from content mode:

- **zero total-connections** - Set the Total Connections counter to zero for all services associated with the specified content rule
- **zero total-reused-connections** - Set the Total Reused Conns. counter to zero for all services associated with the specified content rule
- **zero state-transitions -** Set the State Transitions counter to zero for all services associated with the specified content rule

You can issue the following **zero** commands from content mode:

- **zero total-connections service** *service_name* - Set the Total Connections counter to zero for only the specified service associated with the content rule
- **zero total-reused-connections service** *service_name* - Set the Total Reused Conns. counter to zero for only the specified service associated with the content rule
- **zero state-transitions service** *service_name* **-** Set the State Transitions counter to zero for only the specified service associated with the content rule

For example, to clear a counter for all services associated with the specified content rule, enter:

```
(config-owner-content[rule1])# zero total-connections
```

For example, to clear a counter for a specific service in a content rule, enter:

```
(config-owner-content[rule1])# zero total-connections service
serv1
```

Where to Go Next

Once you create content rules you can configure sticky parameters for the content rules. For information on configuring sticky parameters, refer to Chapter 4, Configuring Sticky Parameters for Content Rules.

Configuring Sticky Parameters for Content Rules

This chapter describes how to configure sticky parameters for content rules. The information in this chapter applies to all CSS models, except where noted. This chapter contains the following sections:

Sticky Overview

During a session, the CSS maintains an association between a client and a server. This association is referred to as *stickiness*. Stickiness enables transactions over the Web when the a client must remain on the same server for the entire session. Depending on the content rule, the CSS "sticks" a client to an appropriate server after the CSS has determined which load balancing method to use.

If the CSS determines that a client is already stuck to a particular service, then the CSS places the client request on that service, regardless of the load balancing criteria specified by the matched content rule. If the CSS determines that the client is not stuck to a particular service, it applies normal load balancing to the content request.

Client *cookies* uniquely identify clients to the services providing content. A cookie is a small data structure used by a server to deliver data to a Web client and request that the client store the information. In certain applications, the client returns the information to the server to maintain the state between the client and the server.

When the CSS examines a request for content and determines through content rule matching that the content is sticky, it examines any cookie or URL present in the content request. The CSS uses this information to place the content request on the appropriate server.

The CSS 11501 supports a 128K sticky table. The CSS 11503 or CSS 11506 supports either a 128K or 32K sticky table (depending whether the SCM has 288 MB or 144 MB of memory). When the CSS has 288 MB of memory, it supports a 128K sticky table. When the CSS has 144 MB of memory, it supports a 32K sticky table.The size of the sticky table means that once 128K (or 32K) simultaneous users are on the site, the table wraps and the first users become "unstuck"

This section describes the following topics:

Why Use Stickiness?

When customers visit an e-commerce site, they usually start out by browsing around the site, the Internet equivalent of window shopping. Depending on the application, the site may require that the customer become “stuck” to one server once the connection is established, or the application may not require this until the customer starts to build a shopping cart.

In either case, once the customer adds items to the shopping cart, it is important that all of the customer’s requests get directed to the same server so that all the items are contained in one shopping cart on one server. An instance of a customer's shopping cart is typically local to a particular Web server and is not duplicated across multiple servers.

E-commerce applications are not the only types of applications that require stickiness. Any Web application that maintains client information may require stickiness, such as banking applications or online trading.

Because the application must distinguish each user or group of users, the CSS needs to determine how a particular user is stuck to a specific Web server. The CSS can use a variety of methods, including:

- Source IP address
- Source IP address and destination port
- String found in a cookie or a URL
- SSL session ID

The e-commerce application itself dictates which of these methods is appropriate for a particular e-commerce vendor.

Using Layer 3 Sticky

If an application requires that a user be stuck for the entire session, use Layer 3 sticky, which sticks a user to a server based on the user's IP address. If the volume of your site is such that you will have more than 128K (or 32K) users at a time, or if a large percentage of your customers come to you through a mega-proxy, then consider using either a different sticky method (for example, the advanced-balance method **cookies**, **cookieurl**, or **url**), or increasing your sticky mask.

Note If you use the **sticky-inact-timeout** command to specify the inactivity timeout period on a sticky connection, when the sticky table becomes full and none of the entries have expired from the sticky table, the CSS rejects subsequent needed sticky requests.

The default sticky mask is 255.255.255.255, which means that each entry in the sticky table is an individual IP address. Some mega-proxies allow one user to use several different IP addresses in a range of addresses over the life of one session. This causes some of the TCP connections to get stuck to one server, and other TCP connections to a different server for the same transaction. This would result in possibly losing some items from the shopping cart. To avoid this problem, use one of the more advanced methods of sticking. If you cannot, Cisco Systems recommends using a sticky mask of 255.255.240.0.

Using Layer 4 Sticky

Layer 4 sticky functions identically to Layer 3 sticky, except that it sticks based on a combination of source IP address, protocol, and destination port. Layer 4 sticky also uses either a sticky table and has the same limitations as Layer 3 sticky.

If the CSS sees the same IP address with two different destination ports, it will use two entries. You can also apply sticky mask to Layer 4 sticky.

If you are concerned about whether your site can handle all of the simultaneous sessions, then consider using the Layer 5 advanced-balanced methods of **arrowpoint-cookie**, **cookie**, **cookieurl**, or **url**.

Using Layer 5 Sticky

Layer 5 sticky uses a combination of destination IP address, protocol, port, and URL that may or may not contain an HTTP cookie or a domain name. Layer 5 sticky can function based on a sticky string in a cookie or URL, or based on SSL version 3 session ID. The advanced-balanced methods such as **arrowpoint-cookie**, **cookie**, **cookieurl**, and **url** do not use a sticky table to keep track of IDs. The **advanced-balance ssl** method for SSL sticky does use a sticky table.

Note If you use the **sticky-inact-timeout** command to specify the inactivity timeout period on a sticky connection, when the sticky table becomes full and none of the entries have expired from the sticky table, the CSS rejects subsequent new sticky requests. If the **sticky-inact-timeout** command is specified for a Layer 5 content rule using SSL sticky, the SSL sessions continue even if the sticky table is full but the CSS does not maintain stickiness on the new sessions.

Configuring Sticky on the CSS

Configuring sticky on the CSS requires you to:

- Determine the sticky method you want to use according to the requirements of the site (for example, Layer 3, Layer 4, or one of the string methods).
- Configure a failover method.

If you use advanced-balance methods **cookies**, **url**, or **cookieurl**, you must also:

- Determine whether you want to use an exact string match or a hash, and then configure that function.
- Determine how you want to delimit (configure) the string.

To configure sticky on the CSS:

1. Configure the sticky method using the **advanced-balance** command and its options. The **advanced-balance** command options are described in "Specifying an Advanced Load-Balancing Method for Sticky Content" later in this chapter.
   - To configure Layer 3 sticky, use **advanced-balance sticky-srcip** in the content rule. If necessary, change the sticky mask from the default of 255.255.255.255.
   - To configure Layer 4 sticky, use **advanced-balance sticky-srcip-dstport** in the content rule. If necessary, change the sticky mask from the default of 255.255.255.255.
   - To configure sticky cookies, use **advanced-balance cookies** in the content rule.
   - To configure sticky URL, use **advanced-balance url** in the content rule.
   - To configure sticky cookies with URLs, use **advanced-balance cookieurl** in the content rule.
2. Configure a failover method. Use the **sticky-serverdown-failover** command to define what will happen if a sticky string is found, but the associated service has failed or is suspended. The sticky failover default is for the CSS to use the configured load balancing method. The **sticky-serverdown-failover** options are described in "Configuring Sticky Serverdown Failover" later in this chapter.

   If you configured an advanced-balance method of **sticky-srcip** or **sticky-srcip-dstport**, no further steps are required.

   If you configured the advanced-balance methods **cookies**, **url**, or **cookieurl**, complete Steps 3 and 4.
3. If you are using **advanced-balance cookies**, **url**, or **cookieurl**, determine whether you want to use an exact string match or a hash.

   To use an exact string match:

   a. Enter the **string operation match-service-cookie** command (this is the default for the **string operation** command).

b. For each service configuration, use the service mode **string** command to configure the unique string that you want to use for matching each server.

For example, you have three servers and you want the string matching to be *serverid111* for service1, *serverid112* for service2, and *serverid113* for service3. Configure the Web server applications to use these strings when they set cookies or pass parameters.

For information on the **string operation match-service-cookie** command, see "Specifying a String Operation" later in this chapter.

To use the hash algorithm:

a. Enter the **string operation** command in the content rule.

b. Select an option (**hash-a**, **hash-crc32**, or **hash-xor**) depending on the hash method you wish to use. Hashing requires that each server can accept cookies set by all other servers.

Technical Support recommends using either **hash-xor** or **hash-crc32,** depending on your string possibilities. If the strings are completely dissimilar, use **hash-xor**. If the strings are similar, use **hash-crc32**. For example, if your string values are *abc1*, *abc2*, and *abc3*, the **hash-xor** method cannot provide you with enough variance in the hash values (that is, *abc1* and *abc2* may end up on the same server because they may hash to the same value).

For information on the string operation hash options, see "Specifying a String Operation" later in this chapter.

4. If you are using **advanced-balance cookies**, **url**, or **cookieurl**, determine how you want to delimit (configure) the string. Use the following owner-content **string** commands to delimit the string:

- **string range** - Defining the string range enables you to limit the size of the search. By default the CSS searches the first 100 bytes of the cookie, URL, or parameters in the URL depending on the method. If you know where in the cookie or URL the string is likely to appear, define the string range accordingly. The range is from 1 to 2000. The default is 1 to 100. The string range options are described in "Configuring String Range" later in this chapter.

- **string eos-char** - A maximum of 3 ASCII characters that delimit the end of the string within the string range. Use this option when the string length varies. Note that **string process-length** overrides **string eos-char**. If you do not configure either option, the CSS uses a maximum of 100 bytes for the delimiter.
- **string prefix** - The CSS uses the string prefix (maximum of 30 characters) to locate the string within the string range of the cookie or URL. If the string prefix is specified, but not found, the CSS uses the normal balance method.
- **string process-length** - Specifies the number of bytes within the string range after the end of the prefix plus the skip-length that is used to determine the string. Use this option when the string length is fixed.
- **string skip-length** - Specifies the number of bytes to skip after the end of the prefix within the string range. The range is 0 to 64.

For example, if you are using ipaddr=192.168.3.6&, then use the **string prefix** "ipaddr=" and the **string eos-char** "&" because the IP addresses vary in length.

For example, if you are using server ID=*server111*, then use the **string prefix** "server ID=" and a **string process-length** of 8 because the string length does not vary in length.

Table 4-1 describes sticky rules and how they apply to content rules.

Table 4-1 Applying Sticky Rules to Content Rules

Rule Type	Sticky Configuration	Stickiness Based on...
Layer 3 content rule	**advanced-balance sticky-srcip**	Source IP address using a sticky mask.
Layer 4 content rule	**advanced-balance sticky-srcip-dstport**	Source IP address and destination port using a sticky mask.
Layer 5 content rule not using a sticky string	**advanced-balance sticky-srcip-dstport**	Source IP address and destination port using a sticky mask.

Table 4-1 Applying Sticky Rules to Content Rules (continued)

Rule Type	Sticky Configuration	Stickiness Based on...
Layer 5 content rule using a sticky string	**advanced-balance cookies** or **advanced-balance cookieurl**	Searching for a sticky string in the cookie or URL. If the CSS does not find the sticky string in the cookie or URL, the CSS load-balances each request among the available servers.
Layer 5 content rule with SSL	**advanced-balance ssl**	SSL v3 session ID. If no session ID is present, the CSS uses the source IP address and destination port to maintain stickiness.

Note In some environments, URL, cookie strings, or HTTP header information can span over multiple packets. In these environments, the CSS can parse multiple packets for Layer 5 information before making load-balancing decisions. Through the global configuration mode **spanning-packets** command, the CSS can parse up to 20 packets with a default of 6. The CSS makes the load-balancing decision as soon as it finds a match and does not require parsing of all of the configured number of spanned packets. Because parsing multiple packets does impose a longer delay in connection, performance can be impacted by longer strings that span mulitple packets. For information on using the **spanning-packets** command, refer to Chapter 3, Configuring Content Rules.

Specifying an Advanced Load-Balancing Method for Sticky Content

Use the **advanced-balance** command to specify an advanced load-balancing method for a content rule that includes stickiness. A content rule is "sticky" when additional sessions from the same user or client are sent to the same service as the first connection, overriding normal load balancing. By default, the advanced balancing method is disabled.

The advanced-balance command options **cookies**, **cookieurl**, and **url** use strings for sticking clients to servers. These options are beneficial when the sticky table limit is too small for your application requirements because the string methods do not use the sticky table.

The syntax and options for this content mode command are:

- **advanced-balance arrowpoint-cookie** - Enables the content rule to stick the client to the server based on the unique service identifier information of the selected server in the ArrowPoint-generated cookie. Configure the service identifier by using the **(config-service) string** command. For information on configuring the ArrowPoint-generated cookie, see "Configuring an ArrowPoint Cookie" later in this chapter. You can use this option with any Layer 5 content rule.

  Note If you are using the **arrowpoint-cookie** option of the **advanced-balance** command (see "Configuring Sticky Parameters for E-Commerce Applications" later in this chapter), do not configure string match criteria, the **sticky-no-cookie-found-action** command, or the **sticky-serverdown-failover** command.

- **advanced-balance cookies** - Enables the content rule to stick the client to the server based on the configured string found in the HTTP cookie header. You must specify a port in the content rule to use this option. The CSS will then spoof the connection. A content rule with a sticky configuration set to **advanced-balance cookies** requires all clients to enable cookies on their browser.

When a client makes an initial request, they do not have a cookie. But once they go to a server that is capable of setting cookies, they receive the cookie from the server. Each subsequent request contains the cookie until the cookie expires. A string in a cookie can be used to stick a client to a server. The service mode **string** command enables you to specify where the CSS should locate the string within the cookie.

The CSS processes the cookie using:

- An exact match that you set up when you configure the services.
- Data for a hash algorithm. For more information, see "Comparing Hash Method With Match Method" later in this chapter.

- **advanced-balance cookieurl** - Same as the **advanced-balance cookies** command, but if the CSS cannot find the cookie header in the HTTP packet, this type fails over to look up the URL extensions (that is, the portion after the "?" in the URL) based on the same string criteria. You must specify a port in the content rule to use this option. The CSS will then spoof the connection.

  This option is useful if a Microsoft© IIS web server is used with Cookie Munger, which dynamically places the session state information in the cookie header or URL extension, depending on whether or not the client can accept cookies.

  Some client applications do not accept cookies. When a site depends upon the information in the cookie, administrators sometimes modify the server application so that it appends the cookie data to the parameters section of the URL. The parameters typically follow a "?" at the end of the main data section of the URL.

  The **advanced-balance cookieurl** command sticks a client to a server based on locating the configured string:

  - In the cookie if a cookie exists
  - In the parameters section of the URL if no cookie exists

  The string can either be an exact match or be hashed.

- **advanced-balance none** - Disables the advanced-balancing method for a content rule (default).
- **advanced-balance sticky-srcip** - Enables the content rule to stick a client to a server based on the client IP address, also known as Layer 3 stickiness. You can use this option with Layer 3, 4, or 5 content rules.

- **advanced-balance sticky-srcip-dstport** - Enables the content rule to stick a client to a server based on both the client IP address and the server destination port number, also known as Layer 4 stickiness. You can use this option with Layer 4 or Layer 5 content rules.
- **advanced-balance ssl** - Enables the content rule to stick the client to the server based on the Secure Socket Layer (SSL) version 3 session ID assigned by the server. The application type must be SSL for the content rule. You must specify a port in the content rule to use this option. The CSS will then spoof the connection.

  Sites where encryption is required for security purposes often use SSL. SSL contains session IDs and the CSS can use these session IDs to stick the client to a server. In order for the CSS to successfully provide SSL stickiness, the application must be using SSL version 3 session IDs. Sticky SSL uses the sticky table. If you are concerned about the number of concurrent sessions, and not concerned about security, you should consider using the **cookies**, **cookieurl**, or **url** option.

  Note In addition, you may need to issue the **ssl-l4-fallback disable** command when you want to disable the CSS from inserting the Layer 4 hash value, based on the source IP address and destination address pair, into the sticky table. This may be necessary in a lab environment when testing SSL with a small number of clients and servers, where some retransmissions might occur. In this case, you would not want to use the Layer 4 hash value because it will skew the test results. See "Configuring SSL-Layer 4 Fallback" later in this chapter for details.

  Do not issue the **ssl-l4-fallback disable** command if SSL version 2 is in use on the network.

- **advanced-balance url** - Enables the content rule to stick a client to a server based on a configured string found in the URL of the HTTP request. You must specify a port in the content rule to use this option. The CSS will then spoof the connection.

  Similar to **advanced-balance cookies**, **advanced-balance url** may use either an exact match method or a hash algorithm. The string can exist anywhere in the URL.

- **advanced-balance wap-msisdn** - Enables a Layer 5 content rule to stick a client to a server based on the MSISDN header field in an HTTP request. MSISDN is the header field for wireless clients using the Wireless Application Protocol (WAP). The MSISDN field value can contain the client's telephone number or user ID, which uniquely identifies a client. This command is especially useful for clients using e-commerce applications, for example, making a purchase on a web site.

  If the MSISDN header is present in an HTTP request, the CSS generates a key based on the value in the MSISDN header field. The CSS uses the key to look up an entry in the sticky table. If an entry exists in the sticky table, the CSS sends the client to the sticky server indicated by the table entry. If an entry does not exist in the sticky table, the CSS:

  - Generates a new entry in the sticky table (similar to L3, L4, and SSL sticky)
  - Load balances the request to a server
  - Stores the selected server and the key (hashed value of the MSISDN header) in the sticky entry

  For subsequent requests from the same client, the CSS looks up the same table entry and sends the client to the same server.

  If the MSISDN header field is not present in an HTTP request, the CSS load balances the client request based on the configured balance method. The default load-balancing method is roundrobin.

  In the following example, TCP port 80 traffic destined for 192.168.128.151 is stuck to either server1 or server2 based on the contents of the MSISDN HTTP header field.

```
owner arrowpoint
   content ruleWapSticky
     vip address 192.168.128.151
     protocol tcp
     port 80
     url "/*"
     add service server1
     add service server2
     advanced-balance wap-msisdn
     active
```

For example, to specify **advanced-balance wap-msisdn** for content rule *rule1*, enter:

```
(config-owner-content[arrowpoint-rule1])# advanced-balance
wap-msisdn
```

Note You can use the **advanced-balance wap-msisdn** command alone or with the MSISDN header field type. For a configuration example using both, see "Configuring Wireless Users for E-Commerce Applications" later in this chapter.

To disable the advanced load-balancing method, enter:

```
(config-owner-content[arrowpoint-rule1])# advanced-balance none
```

Configuring SSL-Layer 4 Fallback

Use the **ssl-l4-fallback disable** command when you want to prevent the CSS from inserting the Layer 4 hash value, based on the source IP address and destination address pair, into the sticky table (the default CSS operation). Insertion of the Layer 4 hash value into the sticky table occurs when more than three frames are transmitted in either direction (client-to-server, server-to-client) or if SSL version 2 is in use on the network. If either condition occurs, the CSS inserts the Layer 4 hash value into the sticky table, overriding the further use of the SSL version 3 session ID.

The **ssl-l4-fallback** command is applicable only when the **advanced-balance ssl** method is specified for a content rule, which forces the content rule to stick to a server based on SSL version 3 session ID. The use of the **ssl-l4-fallback** command may be necessary in a lab environment when testing SSL with a small number of clients and servers, where some retransmissions might occur. In this case, you would not want to use the Layer 4 hash value because it will skew the test results.

Note The **ssl-l4-fallback** command is a global configuration mode command and affects all contents rules using the **advanced-balance ssl** method.

The options for this global configuration mode command include:

- **ssl-l4-fallback enable** - The CSS inserts the Layer 4 hash value into the sticky table (default setting).
- **ssl-l4-fallback disable** - The CSS does not insert the Layer 4 hash value into the sticky table and continues to look for SSL version 3 session IDs.

Note Do not issue the **ssl-l4-fallback disable** command if SSL version 2 is in use on the network.

For example, to disable the CSS from inserting the Layer 4 hash value into the sticky table, enter:

```
(config)# ssl-l4-fallback disable
```

To reset the CSS back the default action of inserting a Layer 4 hash value into the sticky table, enter:

```
(config)# ssl-l4-fallback enable
```

Configuring Sticky Serverdown Failover

Use the **sticky-serverdown-failover** command to define what will happen if a sticky string is found, but the associated service has failed or is suspended. The sticky failover default method is for the CSS to use the configured load-balancing method.

Note If you are using the **arrowpoint-cookie** option of the **advanced-balance** command (see "Configuring Sticky Parameters for E-Commerce Applications" later in this chapter), do not configure string match criteria, the **sticky-no-cookie-found-action** command, or the **sticky-serverdown-failover** command.

The syntax and options for this content mode command are:

- **sticky-serverdown-failover balance** - Sets the failover method to use a service based on the configured load-balancing method.
- **sticky-serverdown-failover redirect** - Sets the failover method to use the redirect string configured on a content rule. If you do not configure a redirect string on a content rule, the load-balancing method is used.
- **sticky-serverdown-failover reject** - Rejects the content request.
- **sticky-serverdown-failover sticky-srcip** - Sets the failover method to use a service based on the client source IP address.
- **sticky-serverdown-failover sticky-srcip-dstport** - Sets the failover method to use a service based on the client source IP address and the server destination port.

For example, to set the sticky failover method to **sticky-srcip**, enter:

```
(config-owner-content[arrowpoint-rule1])sticky-serverdown-failover
sticky-srcip
```

To set the sticky failover method to its default setting of using the configured load-balancing method, enter:

```
(config-owner-content[arrowpoint-rule1])# no
sticky-serverdown-failover
```

Configuring Sticky Mask

A client IP address uniquely identifies the client to the CSS. During normal client-server sessions, the IP address is maintained throughout the connection. However, if the connection is lost (for example, due to a dense proxy failover) and the client reconnects with a different IP address, the CSS needs to reconnect the client to the same server that is preserving the client information (for example, information from a shopping cart or financial session).

Use the **sticky-mask** command to mask a group of client IP addresses in order to preserve the client connection state when the client's source IP address changes. The sticky mask specifies which portion of the client IP address the CSS will mask. The default sticky subnet mask is 255.255.255.255.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# sticky-mask
255.255.255.0
```

To restore the sticky subnet mask to the default of 255.255.255.255, enter:

```
(config-owner-content[arrowpoint-rule1])# no sticky-mask
```

Configuring Sticky Inactive Timeout

Use the **sticky-inact-timeout** command to specify the inactivity timeout period on a sticky connection for a content rule before the CSS removes the sticky entry from the sticky table. When you configure this period, the CSS keeps the sticky entry in the sticky table for the specified amount of time. The CSS does not reuse this entry until the time expires. If the sticky table is full and none of the entries have expired, the CSS rejects the new sticky request. If the **sticky-inact-timeout** command is specified for a Layer 5 content rule using SSL sticky, the SSL sessions continue even if the sticky table is full; however the CSS does not maintain stickiness on the new sessions.

When the sticky connection expires, the CSS uses the configured load-balancing method to choose an available server for the request.

When this feature is disabled, the new sticky connection uses the oldest used sticky entry. A sticky association could exist for a time depending on the sticky traffic load on the CSS.

The syntax for this command is:

sticky-inact-timeout *minutes*

Enter the number of minutes of inactivity from 0 to 65535. The default value is 0, which means this feature is disabled. For example, enter:

```
(config-owner-content[arrowpoint-rule1])# sticky-inact-timeout 9
```

To disable the sticky connection inactivity timeout feature, enter:

```
(config-owner-content[arrowpoint-rule1])# no sticky-inact-timeout
```

Configuring Sticky Content for SSL

To use stickiness based on SSL version 3 session ID, configure a specific SSL Layer 5 rule for a service. To configure an SSL Layer 5 rule for a service:

- Set the port to 443 using the **(config-owner-content) port** command.
- Enable the content rule to be sticky based on SSL using the **(config-owner-content) advanced-balance ssl** command.
- Specify the SSL application type using the **(config-owner-content) application ssl** command.

Note Cisco recommends that the **application ssl** command always be configured in conjunction with the **advanced-balance ssl** command. The **application ssl** command causes the CSS to spoof a connection so that you see the response come back from the server. The **advanced-balance ssl** command causes the CSS to look for the SSL session ID coming from the server and stick the client to the server based on that session ID. Once a flow is setup, the **application ssl** command then causes the CSS to treat the flow as a Layer 4 flow and does not inspect the flow for Layer 5 data in order to prevent the CSS from mis-interpreting encrypted data.

For example, the following owner portion of a startup-config shows a content rule configured for SSL. Note that **url "/*"** command in this example is optional. The combination of the **application ssl** and **advanced-balance ssl** commands promotes the rule to Layer 5.

```
!************************** OWNER ***************************!

    owner arrowpoint
    content L5sslsticky
    vip address 192.3.6.58
    add service server87
    add service server88
    balance aca
    protocol tcp
    port 443
    url "/*"
    advanced-balance ssl
    application ssl
    active
```

Configuring String Range

Use the **string-range** command to specify the starting and ending byte positions within a cookie, URL, or URL extension the CSS uses to search for the specified string. By specifying this range of bytes, the CSS processes the information located only within this range. This limits the amount of information that the CSS has to process when examining each cookie, URL, or URL extension, enhancing its performance. By default, the string range is the first 100 bytes of the cookie, URL, or parameters in the URL.

Note If the starting position is beyond the cookie, URL, or URL extension, the CSS does not perform the string function. When the ending position is beyond the cookie, URL, or URL extension, the string processing stops at the end of the corresponding header.

Enter the *start_byte* as the starting byte position of the cookie, URL, or URL extension after the header. Enter an integer from 1 to 1999. The default is 1. Ensure that the starting byte position is less than the end byte.

Enter the *end_byte* as the ending byte position of the cookie, URL, or URL extension. Enter an integer from 2 to 2000. The default is 100. Ensure that the ending byte position is more than the start byte position.

If you are using **advanced-balance**:

- **cookies** - The CSS starts counting after “Cookie: ”(that is, cookie, colon, space).
- **url** - The CSS starts counting after the “/”.
- **cookieurl** - The CSS starts counting after the "Cookie: " string. If the CSS does not find "Cookie: " in the HTTP request, it starts counting after the "?" in the URL of the same request.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# string-range 35
to 55
```

To restore the string range to the default of 1 to 100, enter:

```
(config-owner-content[arrowpoint-rule1])# no string-range
```

Specifying a String Operation

Use the **string operation** command to determine the method to choose a destination server for a string result. The CSS derives the string result from the settings of the **string** criteria commands within the string range. You can choose a server by using the configured balance method or by using the hash key generated by the specified sticky hash type. If the Web servers:

- Are only capable of accepting the cookies that they set, then you must use the exact match method.
- Can accept any cookies that are set by either a cookie server or other servers, then you may use the hash method.

Note If you are using the **arrowpoint-cookie** option of the **advanced-balance** command (see "Configuring Sticky Parameters for E-Commerce Applications" later in this chapter), do not configure string match criteria, the **sticky-no-cookie-found-action** command, or the **sticky-serverdown-failover** command.

Comparing Hash Method With Match Method

When an application uses the exact match method, once a client makes a request to a particular server, the server is responsible for providing the client with a string unique to the server to use for future requests. Typically, if a server receives a string in a request that was set by another server, that string causes an error. In an exact match, the CSS looks for the unique string. If it finds an exact match, then the server is used. If no match is found, the CSS uses the configured load-balancing method to select a server for the client.

When an application uses one of the hash algorithms, all of the servers are capable of accepting any strings set by other servers. The model was designed so you could set up a site where the initial login would send a client to a Web server that assigns cookies to clients. When the CSS receives the first request from a client with the cookie string, it performs the hash operation on the string and chooses a server accordingly. The hash algorithm ensures that a particular string is always sent to a specific server, but it does not have to be a predefined server, as with an exact match.

Using the string operation hash algorithms may allow the Web server application to be used without being modified. When you use the **string operation match-service-cookie** method, you must modify the Web server application so that each server generates a unique string. The hash algorithms may be able to take advantage of strings already generated by the servers.

The syntax and options for this content mode command are:

- **string operation match-service-cookie** - Chooses a server by matching a service cookie in the sticky string. This is the default setting. When a match is not found, the CSS chooses the server by using the configured balance method (for example, roundrobin).
- **string operation** [**hash-a**|**hash-crc32**|**hash-xor**] - Chooses a server by using the hash key generated by the designated hash method. When using **advanced balance cookies** with a hash algorithm, all servers in the same domain must accept cookies regardless of which server created the cookie. This enables all servers configured on the Layer 5 rule to process cookies passed in an HTTP request.
  - **hash-a** - Apply a basic hash algorithm on the hash string to generate the hash key
  - **hash-crc32** - Apply the CRC32 algorithm on the hash string to generate a hash key
  - **hash-xor** - Exclusive OR (XOR) each byte of the hash string to derive the final hash key

  If the selected server is out of service, the CSS performs a rehash to choose another server.

  Technical Support recommends using either **hash-xor** or **hash-crc32** depending on your string possibilities. If the strings are completely dissimilar, use **hash-xor**. If the strings are similar, use **hash-crc32**. For example, if your string values are *abc1*, *abc2*, and *abc3*, the **hash-xor** method cannot provide you with enough variance in the hash values (that is, *abc1* and *abc2* may end up on the same server because they may hash to the same value).

For example, to set the string operation to choose a server by using the string operation **hash-crc32** algorithm, enter:

```
(config-owner-content[arrowpoint-rule1])# string operation
hash-crc32
```

To reset the string operation to its default setting of choosing a server by matching a service cookie in the sticky string, enter:

```
(config-owner-content[arrowpoint-rule1])# no string operation
```

The CSS derives a string result from the following string criteria commands:

- **string ascii-conversion**
- **string eos-char**
- **string prefix**
- **string process-length**
- **string skip-length**

Enabling or Disabling String ASCII Conversion

Use the **string ascii-conversion** command to enable or disable the ASCII conversion of escaped special characters within the specified sticky string range before applying any processing to the string. By default, ASCII conversion is enabled.

For example, to disable ASCII conversion of escaped special characters, enter:

```
(config-owner-content[arrowpoint-rule1])# string ascii-conversion
disable
```

To reenable the ASCII conversion of escaped special characters to its default setting, use the **no** form of the command or the **enable** option. For example, enter:

```
(config-owner-content[arrowpoint-rule1])# no string
ascii-conversion

(config-owner-content[arrowpoint-rule1])# string ascii-conversion
enable
```

Specifying End of String Characters

Use the **string eos-char** command to specify up to three ASCII characters as the delimiters for the sticky string within the string range. For example, in a cookie header, a semicolon (;) character is usually used as a delimiter; in a URL extension, an ampersand (&) character is often used as a delimiter.

The CSS uses the **string eos-char** value if the **(config-owner-content) string process-length** command is not configured. The **(config-owner-content) string process-length** command has higher precedence. If neither command is configured, the CSS uses the maximum of 100 bytes for the final string length. Enter the sticky string end of string characters as a quoted text string with a maximum of three characters.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# string eos-char ";"
```

To clear the end of string characters, enter:

```
(config-owner-content[arrowpoint-rule1])# no string eos-char
```

Specifying a String Prefix

Use the **string prefix** command to specify the string prefix located in the string range. If you do not configure the string prefix, the string functions start from the beginning of the string range for the cookie, URL, or URL extension, depending on the sticky type. By default, the string range is the first 100 bytes of the cookie, URL, or parameters in the URL. If the string prefix is configured but is not found in the string range, the CSS uses the load-balancing method you defined in the **sticky-serverdown-failover** command.

Enter the string prefix as a quoted text string with a maximum of 30 characters. The default is no prefix (““).

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# string prefix "UID="
```

To clear the string prefix, enter:

```
(config-owner-content[arrowpoint-rule1])# no string prefix
```

Specifying a String Process Length

Use the **string process-length** command to specify how many bytes after the end of prefix within the string range designated by the **string prefix** command and skipping the bytes designated by the **string skip-length** command, the string action will use. This command has higher precedence than the **string eos-char** command. If neither command is configured, the CSS uses the maximum of 100 bytes for the final string action. Enter the number of bytes from 0 to 64. The default is 0.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# string process-length 16
```

To set the number of bytes to its default setting of 0, enter:

```
(config-owner-content[arrowpoint-rule1])# no string process-length
```

Specifying a String Skip Length

Use the **string skip-length** command to specify how many bytes to skip after the end of prefix within the string range to find the string result. Enter the number of bytes from 0 to 64. The default is 0. For example, enter:

```
(config-owner-content[arrowpoint-rule1])# string skip-length 3
```

To set the number of bytes to its default setting of 0, enter:

```
(config-owner-content[arrowpoint-rule1])# no string skip-length
```

Configuring Sticky-No-Cookie-Found-Action

Note If you intend to use the **advanced-balance arrowpoint-cookie** command (see "Configuring Sticky Parameters for E-Commerce Applications" later in this chapter), do not configure the **sticky-no-cookie-found-action** command.

Use the **sticky-no-cookie-found-action** command to specify the action the CSS should take for a sticky cookie content rule when it cannot locate the cookie header or the specified cookie string in the **sticky-no-cookie-found-action** command.

The options for the **sticky-no-cookie-found-action** command are:

- **loadbalance** (default) - The CSS uses the configured balance method when no cookie is found in the client request.
- **redirect** *"URL"* - Redirects the client request to a specified URL string when no cookie found in the client request. When using this option, you must also specify a redirect URL. Specify the redirect URL as a quoted text string from 0 to 64 characters.
- **reject** - Rejects the client request when no cookie is found in the request.
- **service** *name* - Sends the no cookie client request to the specified service when no cookie is found in the request.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])#
sticky-no-cookie-found-action redirect
"http://www.lml.com/nocookie.html"
```

To reset **sticky-no-cookie-found-action** to the default of **loadbalance**, enter:

```
(config-owner-content[arrowpoint-rule1])# no
sticky-no-cookie-found-action
```

Showing Sticky Configurations

To display sticky configurations for content, use the **show rule** command. For example, enter:

```
(config-owner-content[arrowpoint-rule1])# show rule
```

For details on the **show rule** command, refer to Chapter 3, Configuring Content Rules.

Configuring Sticky Parameters for E-Commerce Applications

By configuring sticky parameters for e-commerce applications, you can instruct the CSS how to process client requests that do not contain cookies when the requests are destined to a content rule that is sticking based on a string within a cookie. You can also instruct the CSS how to process wireless users by integrating HTTP header load balancing with the **advanced-balance wap-msisdn** command. For applications that use the CSS sticky table, you can remove a sticky table entry after a defined period of activity.

Configuring sticky parameters for e-commerce applications includes:

- Configuring an Advanced Balance ArrowPoint Cookie
- Configuring an ArrowPoint Cookie
- Configuring an Arrowpoint-Cookie Expiration Time
- Configuring an Arrowpoint Cookie Path
- Configuring Sticky-No-Cookie-Found-Action
- Configuring Wireless Users for E-Commerce Applications

Configuring an Advanced Balance ArrowPoint Cookie

Note If you are using the **arrowpoint-cookie** option of the **advanced-balance** command, do not configure string match criteria, the **sticky-no-cookie-found-action** command, or the **sticky-serverdown-failover** command.

Use the **advanced-balance arrowpoint-cookie** command to enable the content rule to stick the client to the server based on the unique service identifier of the selected server in the ArrowPoint-generated cookie. Configure the service identifier by using the **(config-service) string** command. You do not need to configure string match criteria. For information on configuring the ArrowPoint-generated cookie, see "Configuring an ArrowPoint Cookie". You can use this option with any Layer 5 content rule.

For example, to specify **advanced-balance arrowpoint-cookie** for content *rule1*, enter:

```
(config-owner-content[arrowpoint-rule1])# advanced-balance
arrowpoint-cookie
```

To disable the advanced load-balancing method, enter:

```
(config-owner-content[arrowpoint-rule1])# no advanced-balance
```

Configuring an ArrowPoint Cookie

Use the **arrowpoint-cookie** command to configure the ArrowPoint cookie path and expiration time. The CSS generates the ArrowPoint cookie transparently for a client, the client stores it and returns it in subsequent requests, and the CSS later uses it to maintain the client-server stickiness. This cookie contains the sticky information itself and does not refer to a sticky table.

If you configure the **arrowpoint-cookie** method in a content rule, the CSS always checks for the existence of the ArrowPoint cookie when it receives a client request. If this cookie does not exist, the CSS performs server load balancing and generates an ArrowPoint cookie.

When the CSS finds the cookie in the client request, it unscrambles the cookie data and then validates it. Then, the CSS checks the cookie expiration time. If the cookie has expired, the CSS sends a new cookie containing the information about the server where the client was stuck. This appears as an uninterrupted connection.

If the cookie format is valid, the CSS ensures the consistency between the cookie and the CSS configuration. When all the validations are passed, the CSS forwards the client request to the server indicated by the server identifier. Otherwise, the CSS treats this request as an initial request.

The options for this content mode command are:

- **arrowpoint-cookie expiration** - Set an expiration time, which the CSS compares with the time associated with the cookie
- **arrowpoint-cookie path** - Set the cookie path to a configured path
- **arrowpoint-cookie browser-expire -** Allow the browser to expire the cookie
- **arrowpoint-cookie expire services -** Expire the service information when the cookie expires

Configuring an Arrowpoint-Cookie Expiration Time

Use the **arrowpoint-cookie expiration** command to set an expiration time, which the CSS compares with the time associated with the ArrowPoint cookie. If the cookie has expired, the CSS sends a new cookie that includes the server where the client was stuck. This allows for the appearance of an uninterrupted connection. If you do not set an expiration time, the cookie expires when the client exits the browser.

The syntax of this owner-content mode configuration command is:

arrowpoint-cookie expiration *dd:hh:mm:ss*

The variables are:

- *dd* - Number of days. Valid numbers are from 00 to 99.
- *hh* - Number of hours. Valid numbers are from 00 to 99.
- *mm* - Number of minutes. Valid numbers are from 00 to 99.
- *ss* - Number of seconds. Valid numbers are from 00 to 99.

Note Do not use all zeros for days, hours, minutes, and seconds. This value is invalid.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# arrowpoint-cookie
expiration 08:04:03:06
```

To reset the expiration time to when the client exits the browser, enter:

```
(config-owner-content[arrowpoint-rule1])# no arrowpoint-cookie
expiration
```

Configuring Arrowpoint-Cookie Browser Expire

Use the **arrowpoint-cookie browser-expire** command to allow the browser to expire the ArrowPoint cookie based on the expiration time. To configure the expiration time, see the previous section. The syntax of this owner-content configuration mode command is:

arrowpoint-cookie browser-expire

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# arrowpoint-cookie
browser-expire
```

To allow the CSS to expire the cookie, enter:

```
(config-owner-content[arrowpoint-rule1])# no arrowpoint-cookie
browser-expire
```

Note When the cookie expires, all sticky information is lost.

Configuring Arrowpoint-Cookie Expire Services

Use the **arrowpoint-cookie expire-services** command to expire service information when the cookie expires before sending a new cookie. By default, when the cookie expires, the CSS sends a new cookie with the server information from the expired cookie. The syntax of this owner-content configuration mode command is:

arrowpoint-cookie expire-services

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# arrowpoint-cookie
expire-services
```

To reset the default behavior, enter:

```
(config-owner-content[arrowpoint-rule1])# no arrowpoint-cookie
expire-services
```

Configuring an Arrowpoint Cookie Path

Use the **arrowpoint-cookie path** command to set the ArrowPoint cookie path to a configured path. Otherwise, the CSS sets the default path attribute of the cookie to "/". The syntax of this owner-content configuration mode command is:

arrowpoint-cookie path "*path_name*"

Enter the *path_name* where you want to send the cookie. Enter a quoted text string with a maximum of 99 characters. The default path of the cookie is "/".

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# arrowpoint-cookie path
"/cgi-bin/"
```

To reset the cookie path to its default of "/", enter:

```
(config-owner-content[arrowpoint-rule1])# no arrowpoint-cookie
path
```

Configuring Wireless Users for E-Commerce Applications

Use the **advanced-balance wap-msisdn** command with the MSISDN header field to configure wireless users for e-commerce applications. For details on the **advanced-balance wap-msisdn** command, see "Specifying an Advanced Load-Balancing Method for Sticky Content" earlier in this chapter. For details on the MSISDN header field, refer to Chapter 6, Configuring HTTP Header Load Balancing, "Configuring a Header Field Entry".

Wireless clients use the Wireless Application Protocol (WAP) to access Internet content. When a wireless client sends a request for content, the WAP protocol gateway (a device that translates requests from the WAP protocol stack to the WWW protocol stack) generates the MSISDN field and adds it to the HTTP header.

In the following example, TCP port 80 traffic destined for VIP 192.168.128.151 that contains the string "012" in the MSISDN HTTP header field will hit content rule rule012. The CSS will stick this traffic to either server1 or server2 based on the entire contents of the MSISDN field.

TCP port 80 traffic destined for 192.168.128.151 that does not contain the string "012" in the MSISDN HTTP header field, but has the field in the header, will hit content rule ruleNo012. The CSS will roundrobin load-balance the traffic across server21 and server22.

TCP port 80 traffic destined for 192.168.128.151 that does not contain the MSISDN HTTP header field will hit content rule ruleNoWap. The CSS will roundrobin load-balance the traffic across server31 and server32.

```
header-field-group wap012
   header-field 1 wap-msisdn contain "012"

header-field-group wapNo012
   header-field 1 wap-msisdn not-contain "012"

owner arrowpoint
   content rule012
     vip address 192.168.128.151
     protocol tcp
     port 80
     url "/*"
     add service server1
     add service server2
     header-field-rule wap012
     advanced-balance wap-msisdn
     active
```

```
content ruleNo012
 vip address 192.168.128.151
 protocol tcp
 port 80
 url "/*"
 add service server21
 add service server22
 header-field-rule wapNo012
 active

content ruleNoWap
 vip address 192.168.128.151
 protocol tcp
 port 80
 url "/*"
 add service server31
 add service server32
 active
```

Where to Go Next

You can configure source groups, Access Control Lists (ACLS), Extension Qualifier Lists (EQLs), Uniform Resource Locator Qualifier Lists (URQLs), Network Qualifier Lists (NQLs), and Domain Qualifier Lists (DQLs). For information, refer to Chapter 5, Configuring Source Groups, ACLs, EQLs, URQLs, NQLs, and DQLs.

Configuring Source Groups, ACLs, EQLs, URQLs, NQLs, and DQLs

This chapter describes how to configure source groups, Access Control Lists (ACLS), Extension Qualifier Lists (EQLs), Uniform Resource Locator Qualifier Lists (URQLs), Network Qualifier Lists (NQLs), and Domain Qualifier Lists (DQLs). Information in this chapter applies to all CSS models, except where noted.

This chapter contains the following sections:

Configuring Source Groups

Group configuration mode allows you to configure a maximum of 255 source groups on a CSS. A source group is a collection of local servers that initiate flows from within the local web farm. The CSS enables you to treat a source group as a virtual server with its own source IP address.

For example, if you configure several streaming audio transmitters as a group, the CSS will process flows from the group members and give them all the same source IP address.

This section covers:

Source Group Configuration Quick Start

Use the procedure in Table 5-1 to configure a source group for TCP/UDP traffic. To configure a source group for FTP traffic, see the next section. Note that each source group requires a content rule that contains the same services and VIP as the source group.

Table 5-1 Source Group Configuration Quick Start

Task and Command Example

1. Create the source group. Source group names can be a maximum of 16 characters. The following example creates a source group *ftpgroup*.
`(config)# group ftpgroup`
The CLI transitions into config-group mode where you can activate the source group and configure attributes for it.
`(config-group[ftpgroup])#`

2. Configure the source group VIP address to which all service IP addresses will be translated. You can assign the same VIP address to multiple source groups, but only one of the source groups can be active at a time. For example, enter: `(config-group[ftpgroup])# vip address 172.16.36.58`

3. Add previously defined services to the source group. For example, enter:
`(config-group[ftpgroup])# add service server1`
`(config-group[ftpgroup])# add service server2`

4. Activate the source group. Because a VIP address can belong only to one active source group at a time, the CSS will not allow you to activate a second source group that contains the same VIP address as the one in the active source group.
`(config-group[ftpgroup])# active`
To remove service *server1* from the source group, enter:
`(config-group[ftpgroup])# remove service server1`

Table 5-1 Source Group Configuration Quick Start (continued)

Task and Command Example

5. Create a content rule, add the same services and VIP that are configured in the source group, and activate the content rule. The content rule enables the CSS to match requests for the content rule VIP. When either *server1* or *server2* replies to the request, the CSS NATs the server IP addresses to the source group VIP.
 	
For example, enter:
 	
`(config-owner[arrowpoint.com])# content ftpsource1`
 	
`(config-owner-content[arrowpoint.com-ftpsource1])# add service server1`
 	
`(config-owner-content[arrowpoint.com-ftpsource1])# add service server2`
 	
`(config-owner-content[arrowpoint.com-ftpsource1])# vip address 172.16.36.58`
 	
`(config-owner-content[arrowpoint.com-ftpsource1])# activate`

Creating a Source Group

To access group configuration mode, use the **group** command from any mode except ACL and boot configuration modes. The syntax for this command is:

group *groupname*

Enter an existing or a new source group name from 1 to 31 characters.

For example, enter:

```
(config)# group ftpgroup
(config-group[ftpgroup])#
```

To view a list of existing source groups, enter:

```
(config)# group ?
```

Note You can also use the **group** command from within group mode to access or create another source group.

To remove a source group, enter:

```
(config)# no group ftpgroup
```

Note To make certain modifications to an active source group, you must first suspend the source group using the **suspend** command. Such modifications include: changing the IP address to 0 or using the **no ip address command**, adding or removing a service or destination service, or using the **portmap** command.

Source Group Commands

Use the following commands in group mode:

- **active** - Activates a source group.
- **add destination service** *service_name* - Adds a destination service to a source group. You can configure a maximum of 64 destination services per source group. You cannot use a service with the same name in other source groups or the source service list within the same source group. You can use services with duplicate addresses among destination services because the actual service is chosen through content rule selection. The destination service must be active and must be added to a content rule in order for it to perform destination source address NATing for the source group (refer to Chapter 3, Configuring Content Rules).

  Note Adding a destination service to a source group does not allow the destination service flows to be NATed by the source group, when the service initiates the flows. This is because the destination service applies group membership based on rule and service match. To ensure that service initiated connections are NATed, you must also configure ACL match criteria or additional service names with duplicate addresses, then add those services to a source group. The source group used could be the current source group with the destination service or any other configured source group.

- **add service** - Adds a service to a source group. You can configure a maximum of 64 services per source group. A service may belong to only one group at a time. When the source group is active and the same service is hit through a content rule, ACL preferred service, or sorry service, the source group is used to NAT (Network Address Translation) the source address. The service must be active in order for it to perform source address NATing for the source group (refer to Chapter 1, Configuring Services).

  Be aware that you cannot use a service with:

  – The same name in other source groups or the destination service list within the same source group

  – The same address as a source service on another source group

- **vip address** - Specifies the source Virtual IP address (VIP) for the group. The CSS substitutes this IP address for the source address in flows originating from one of the group's sources. You can assign the same VIP address to multiple source groups, but only one of the source groups can be active at a time.

- **remove service** - Removes a previously configured service from a source group.

- **portmap** - Defines the source port translation of flows from the services configured in a source group. By default, portmapping is enabled for source groups on source ports greater than 1023. The CSS translates such source ports to a range starting at 8192. Use the following portmap options to change the default portmapping behavior of the CSS. The syntax and options for this group mode command are:

  – **portmap base-port** *base_number* - Defines the base port (starting port number) for the CSS. Enter a base number from 2016 to 63456. The default is 2016.

    To reset the starting port number to its default value of 2016, use the **no portmap base-port** command.

  – **portmap number-of-ports** *number* - Defines the number of ports in the portmap range for each Switch Processor (SP) in an 11500 series CSS or a Switch Fabric Processor (SFP) in an 11000 series CSS. Enter a number from 2048 to 63488. The default is 63488.

    To reset the number of ports to the default value, use the **no portmap number-of-ports** command.

– **portmap disable** - Instructs the CSS to perform Network Address Translation (NAT) only on the source IP addresses and not on the source ports of UDP traffic hitting a particular source group. Use this option for Wireless Application Protocol (WAP) or other applications where you need to preserve the registered UDP port number for return traffic.

Note This command does not affect TCP flows.

The CSS maintains but ignores any **base-port** or **number-of ports** (see the previous options) values configured in the source group. If you later reenable portmapping for that source group, any configured **base-port** or **number-of ports** values will take effect. The default behavior for a configured source group is to NAT both the source IP address and the source port for port numbers greater than 1023.

To restore the default CSS behavior of NATing source IP addresses *and* source ports for a configured source group, use the **portmap enable** command.

- **suspend** - Suspends a source group. The group and its attributes remain the same but no longer have an effect on flow creation.

Configuring a Source Group for FTP Connections

To use source groups to support FTP sessions to a VIP that is load balanced across multiple services, configure a content rule for the VIP and then a source group.

Note When you use an FTP content rule with a configured VIP address range, be sure to configure the corresponding source group with the same VIP address range (refer to Chapter 3, Configuring Content Rules).

To configure FTP sessions to a VIP:

1. Configure a content rule as required using the VIP that will be load balanced across multiple servers. The following example shows the portion of a running-config for content rule *ftp_rule*. Ensure that you use the **application ftp-control** command to define the application type.

```
content ftp_rule
    vip address 192.168.3.6
    protocol tcp
    port 21
    application ftp-control
    add service serv1
    add service serv2
    add service serv3
    active
```

2. Configure a source group defining the same VIP and services as configured in the content rule.

Note If you are load-balancing passive FTP servers, you must configure services directly in the associated source groups as shown in the following example. Active FTP does not require that you configure services in source groups.

The following running-config example shows source group *ftp_group*.

```
group ftp_group
    vip address 192.168.3.6
    add service serv1
    add service serv2
    add service serv3
    active
```

Configuring Source Groups to Allow Servers to Internet-Resolve Domain Names

The CSS provides support to enable servers to resolve domain names using the Internet. If you are using private IP addresses for your servers and wish to have the servers resolve domain names using domain name servers that are located on the Internet, you must configure a content rule and source group. The content rule and source group are required to specify a public Internet-routable IP address (VIP address) for the servers to allow them to resolve domain names.

To configure a server to resolve domain names:

1. If you have not already done so, configure the server.

   The following example creates *Server1* and configures it with a private IP address 10.0.3.251 and activates it.

   ```
   (config)# service Server1
   (config-service[Server1])# ip address 10.0.3.251
   (config-service[Server1])# active
   ```

2. Create a content rule to process DNS replies. The content rule to process DNS replies is in addition to the content rules you created to process Web traffic. The content rule example below enables the CSS to NAT inbound DNS replies from the public VIP address (192.200.200.200) to the server's private IP address (10.0.3.251).

   The following example creates content rule *dns1* with a public VIP 192.200.200.200 and adds server *Server1*.

   ```
   (config-owner[arrowpoint.com])# content dns1
   (config-owner-content[arrowpoint.com-dns1])# vip address
   192.200.200.200
   (config-owner-content[arrowpoint.com-dns1])# add service Server1
   (config-owner-content[arrowpoint.com-dns1])# active
   ```

3. Create a source group to process DNS requests. The source group enables the CSS to NAT outbound traffic source IP addresses from the server's private IP address (10.0.3.251) to the public VIP address (192.200.200.200).

   To prevent server source port collisions, the CSS NATs the server's source IP address and port by translating the:

   - Source IP address to the IP address defined in the source group.
   - Port to the port selected by the source group. The source group assigns each server a unique port for a DNS query so that the CSS can match the DNS reply with the assigned port. This port mapping enables the CSS to direct the DNS reply to the correct server.

   The following example creates source group *dns1* with public VIP address 192.200.200.200 and adds the service Server1.

```
(config)# group dns1
(config-group[dns1])# vip address 192.200.200.200
(config-group[dns1])# add service Server1
(config-group[dns1])# active
```

Showing Source Groups

To display source group configuration information, use the **show group** commands in SuperUser, User, Global Configuration, and Group modes. The options are:

- **show group** - Display all source group configurations
- **show group** *group_name* - Display the source group configuration specified by group_name
- **show group** *group_name* **portmap** - Display the starting port number and number of ports configured on each SP in a 11500 series CSS (or SFP in a 11000 series CSS)

For example, enter:

```
(config)# show group
```

Table 5-2 describes the fields in the **show group** output.

Table 5-2 Field Descriptions for the show group Command

Field	Description
Group	The name of the group, whether the group is activated (Active) or suspended (Suspend), and the source IP address for the group.
Session Redundancy	Indicates whether Adaptive Session Redundancy (ASR) is enabled or disabled for the source group. For details on ASR, refer to the *Cisco Content Services Switch Advanced Configuration Guide*.
Redundancy Global Index	The unique global index value for Adaptive Session Redundancy assigned to the source group using the **redundant-index** command in group configuration mode.
Associated ACLs	Any ACLs associated with the group.
Source/Destination Services	The source or destination services of the source group.
Name	The name of the service.
Hits	The number of content hits on the service. This field is incremented for traffic from a group server going out from the source group. Traffic coming into the group does not increment the counter.
State	The state of the service. The possible states are Alive, Dying, or Dead.
DNS Load	The DNS load for the service. A load of 255 indicates that the service is down. An eligible load range is from 2 to 254.

Table 5-2 Field Descriptions for the show group Command (continued)

Field	Description
Trans	The number of times that the state of the service has transitioned.
Keepalive	The keepalive type of the service. The possible types are FTP, HTTP, ICMP, NAMED, SCRIPT, or TCP.
Conn	The number of connections currently on the service.
Flow Timeout Multiplier	Number of seconds that a flow remains idle before the CSS reclaims the flow resources, as configured with the **flow-timeout-multiplier** command. For details on the **flow-timeout-multiplier** command, refer to the *Cisco Content Services Switch Administration Guide*.
Group Cumulative Counters	The counters for the group.
Hits/Frames/Bytes	The number of group hits, frames, and bytes. This field is incremented for traffic from a group server going out from the source group. Traffic coming into the group does not increment the counter.
Connection Total/Current	The total number of connections and the current number of connections for the group.
FTP Control Total/Current	The total number of FTP control channels that were mapped and monitored by the CSS, and the current number of those connections that are mapped.
SP (or SFP) Port Map Info	The port map information for each SP in the 11500 series CSS (or SFP in the 11000 series CSS). Includes the status of the **portmap** command (Enabled or Disabled).
SP (or SFP)	The slot and port number of the SP in the 11500 series CSS (or SFP in the 11000 series CSS).
Base Port	The starting SP port number in the 11500 series CSS (or SFP port number in the 11000 series CSS) in the chassis.

Table 5-2 Field Descriptions for the show group Command (continued)

Field	Description
Configured Base Port	The configured starting port number.
Configured Ports SP (or SFP)	The configured number of ports allowed on each SP in the 11500 series CSS (or SFP in the 11000 series CSS).
Current Mapped Ports	The current number of mapped ports.
Last Mapped Port	The most recently mapped port number for each SP in the 11500 series CSS (or SFP in the 11000 series CSS).
High Water Mark	The highest number of ports that this source group has had concurrently mapped since the last group was activated.
No Portmap Errors	The number of times no port could be allocated by the portmapper.

Clearing Source Group Counters

To set the statistics displayed by the **show group** command to zero, use the **zero all** command. The reset counter statistics appear as zero in the **show group** display.

For example, enter:

```
(config-group[ftpgroup])# zero all
```

Configuring an Access Control List

The following sections describe how to configure an Access Control List (ACL):

Access Control List Overview

The CSS provides traffic filtering capabilities with Access Control Lists (ACLs). ACLs filter network traffic by controlling whether packets are forwarded or blocked at the CSS interfaces. You can configure ACLs for routed network protocols, filtering the protocol packets as the packets pass through the CSS.

An ACL consists of clauses that you define. The CSS uses these clauses to determine how to handle each packet it processes. When the CSS examines each packet, it either forwards or blocks the packet based on whether or not the packet matches a clause in the ACL.

Note ACLs are not supported on the CSS Ethernet Management port.

The total number of ACL hits for each packet received by the CSS can vary depending on the type of flow and whether an ACL match occurred. The CSS performs an ACL check for every packet received until the flow is completely set up.

- For Content Hits, a flow can be defined as a stream of UDP and TCP packets between a client and a server. The CSS must receive a number of packets from the client and the server before it can completely set up the flow. All of these packets, received before the flow is completely set up, are subject to ACL checks and can cause increments to the ACL Content Hits counter.
- For Router Hits, all non-TCP or UDP packets subjected to ACL checks cause increments to the ACL Router Hits counter. All UDP and TCP traffic terminating on the CSS (for example, a Telnet or FTP session) cause increments to the ACL Router Hits counter.

ACLs provide a basic level of security for accessing your network. If you do not configure ACLs on the CSS, all packets passing through the CSS could be allowed onto the entire network. For example, you may want to permit all email traffic, but block Telnet traffic. You can also use ACLs to allow one client to access a part of the network and prevent another client from accessing the same area.

Caution ACLs function as a firewall security feature. When you enable ACLs, all traffic not configured in an ACL permit clause *will be denied*. It is extremely important that you first configure an ACL to permit traffic *before you enable ACLs*. If you do not permit any traffic, you will lose network connectivity. Note that the console port is not affected.

Cisco recommends that you configure either a permit all or a deny all clause depending on your ACL configuration. For example, you could first configure a permit all clause and then configure deny clauses for only the traffic you wish to deny. Or, use the default deny all clause and configure permit clauses only for the traffic you wish to permit.

ACL Configuration Quick Start

Use the procedure in Table 5-3 to configure an ACL. Each step includes the CLI command required to complete the task. For a complete description of each feature, see the sections following this procedure.

Table 5-3 ACL Configuration Quick Start

Task and Command Example

1. Create an ACL and access ACL mode. Define the ACL index number from 1 to 99.
`(config)# acl 7`
`(config-acl[7])#`

2. To control traffic on a circuit, configure clauses in the ACL. Enter a clause number from 1 to 254 and define the clause parameters. The syntax for defining a clause is: **clause** *number* **permit**|**deny**|**bypass** *protocol* [*source_info* {*source_port*}] **dest** [*dest_info* {*dest_port*}] {**log**} {**prefer** *servicename*} {**sourcegroup** *name*} For example, enter: `(config-acl[7])# clause 1 deny udp any eq 3 dest any eq 3 log prefer serv7` If you are load-balancing passive FTP servers and you want to use an ACL to apply a source group, you must configure services directly in the source group. For details on using source groups to support FTP sessions, see "Configuring a Source Group for FTP Connections" earlier in this chapter.
3. Apply the ACL to a specific circuit or add the ACL to DNS queries. For example, to apply acl 7 to circuit VLAN1, enter: `(config-acl[7])# apply circuit-(VLAN1)`
4. Enable all ACLS on the CSS. Enter the global **acl enable** command for all ACLs to take effect. You can enable ACL mode even if no ACLs are configured. When you enable ACLs, all traffic not specifically permitted in an ACL permit clause is **denied** by default. For example, enter: `(config)# acl enable`

Caution When you enter the **acl enable** command, all traffic is denied except for traffic specified in an ACL permit clause.

Creating an ACL

To create an ACL and access ACL mode, use the **acl** *index number* command. The index number defines the ACL and can range from 1 to 99. To display a list of existing ACLs, use the **acl ?** command.

```
(config)# acl 7
```

When you access this mode, the prompt changes to the ACL mode of the index number you created. For example, enter:

```
(config-acl[7])#
```

Deleting an ACL

To delete an ACL, use the **no acl** command followed by the index number you wish to delete. For example, enter:

```
(config)# no acl 2
```

Configuring Clauses

To control traffic on a circuit, the CSS enables you to enter clauses in a specific ACL. When implementing an ACL, the number assigned to each clause is very important. The CSS looks at the ACL starting from clause 1 and sequentially progresses through the rest of the clauses. Assign the lowest clause numbers to clauses with the most specific matches. Then, assign higher clause numbers to clauses with less specific matches.

You do not need to enter the clauses sequentially. The CSS automatically inserts the clause in the appropriate order in the ACL. For example, if you enter clauses 10 and 24, and then clause 15, the CSS inserts the clauses in the correct sequence.

Clause *number* is the number you want to assign to the clause. Enter a number from 1 to 254. To create a clause to permit, deny, or bypass traffic on a circuit, use the **clause** command.

Note Ensure that ACLs associated with a source group specified in the **clause** command are globally enabled for the ACL to properly map to the source group (see "Globally Enabling ACLs" later in this chapter).

The syntax for the **clause** command is:

- **clause** *number* **bypass** - Creates a clause in the ACL to *permit* traffic on a circuit and *bypass* (do not apply) content rules that apply to the traffic. The syntax for **clause bypass** is:

  clause *number* **bypass** *protocol* [*source_info* {*source_port*}]
  dest [*dest_info* {*dest_port*}] {**sourcegroup** *name*} {**prefer** *servicename*}

  Note The **bypass** option bypasses traffic only on a content rule, thus does not cause NATing to occur. Do not use the **bypass** option in an ACL clause with a source group. Since this option does not bypass traffic that does not match a rule, it does not effect NATing on a source group in an ACL clause.

- **clause** *number* **deny** - Creates a clause in the ACL to *deny* traffic on a circuit. The syntax for **clause deny** is:

  clause *number* **deny** *protocol* [*source_info* {*source_port*}]
  dest [*dest_info* {*dest_port*}] {**sourcegroup** *name*} {**prefer** *servicename*}

- **clause** *number* **permit** - Creates a clause in the ACL to *permit* traffic on a circuit. When you configure an ACL permit clause, all traffic not specified in a permit clause is denied by default. The syntax for **clause permit** is:

  clause *number* **permit** *protocol* [*source_info* {*source_port*}]
  dest [*dest_info* {*dest_port*}] {**sourcegroup** *name*} {**prefer** *servicename*}

Note If you specify both a source group and a preferred service in a clause, you must specify the source group before you specify the preferred service within the clause.

Table 5-4 provides variables and options for the **clause** command. Bolded syntax defines keywords that you enter on the command line. Italics define variables where you enter a value such as an IP address or host name.

Note ACLs are not supported on the CSS Ethernet Management port.

Note When a destination in an ACL clause is a Layer 5 content rule, the CSS does not spoof the connection. Therefore, the ACL clause does not function as would be expected. As a workaround, you may configure an additional clause to permit the TCP IP addresses and ports. Be aware that content will be matched on both clauses. For example,
clause 14 permit any any destination content Layer5/L5 eq 80 (original clause)
clause 15 permit tcp any destination 200.200.200.200 eq 80 (This is an additional clause to handle the SYN, where the destination IP address is the IP address configured in the Layer 5 content rule. Note that this clause number must be greater than the destination content clause number.)

Table 5-4 Clause Command Options

Variables and Options	Parameters
number	The number you want to assign to the clause. Enter a number from 1 to 254.
action	The action to apply to the clause. Enter one of the following: **bypass**, **deny**, **permit**.
protocol	The protocol for the traffic type. Enter one of the following: **any**, **icmp**, **igp**, **igmp**, **ospf**, **tcp**, **udp**.

Table 5-4 Clause Command Options (continued)

Variables and Options	Parameters
source_info	The source of the traffic. Enter one of the following: • *ip_address* (optionally include *subnet mask* in IP address format only) for the source IP address and optional mask IP address. • *hostname* for the source host name. Enter a host name in mnemonic host-name format. Configure the CSS DNS client first to enable the CSS to translate the host name. • **any** for any combination of source IP address and host name information. • **nql** *nql_name* for an existing Network Qualifier List (NQL) consisting of a list of IP addresses.
source_port	The source port for the traffic. If you do not designate a source port, this clause allows traffic from any port number. Enter one of the following: • **eq** *port* is equal to the port number. • **lt** *port* is less than the port number. • **gt** *port is* greater than the port number. • **neq** *port* is not equal to the port number. • **range** *low high* for a range of port numbers, inclusive. Enter numbers from a range of 1 to 65535. Separate the *low* and *high* number with a space.

Table 5-4 Clause Command Options (continued)

Variables and Options	Parameters
destination_info	The destination information for the traffic. Enter one of the following: • **destination any** for any combination of destination information. • **destination content** *owner_name*|*rule_name* for an owner content rule. Separate the owner and rule name with a \ character. • **destination** *ip_address* (for the destination IP address and optional subnet mask IP address. Include *subnet mask* as IP address only, no CIDR. • **destination** *hostname* for the destination host name. To use a *hostname*, configure the CSS DNS client first to enable the CSS to translate the host name. • **nql** *nql_name* for an existing NQL consisting of host IP addresses. Enter the name of the NQL.

Table 5-4 Clause Command Options (continued)

Variables and Options	Parameters
destination_port	The destination port. Enter one of the following. You may use a port number or port name with the options. • **eq** *port* is equal to the port number. • **lt** *port* is less than the port number. • **gt** *port* is greater than the port number. • **neq** *port* is not equal to the port number. • **range** *low high* for a range of port numbers, inclusive. Enter numbers from a range of 1 to 65535. Separate the *low* and *high* number with a space. • *port names*: **https** = Port 443 Https, **ldap** = Port 389 Ldap, **bgp** = Port 179 Bgp, **ntp** = Port 123 Ntp, **nntp** = Port 119 Nntp, **pop** = Port 110 Pop, **http** = Port 80 Http, **gopher** = Port 70 Gopher, **domain** = Port 53 Domain, **smtp** = Port 25 Smtp, **telnet** = Port 23 Telnet, **ftp** = Port 21 Ftp, **ftp-data** = Port 20 Ftp-data, **none** = None If you do not define a destination port, this clause allows traffic to any port.

Table 5-4 Clause Command Options (continued)

Variables and Options	Parameters
sourcegroup *name*	Define a source group based on matching this ACL clause. Enter the group name. To see a list of source groups, enter: `show group ?`
prefer *service_name*	Define a preferred service based on matching the ACL clause. Enter the service name. To define more than one preferred service, separate each service with a comma (,). You can define a maximum of two services. You cannot configure services learned through an Application Peering Protocol (APP) session as preferred services. A remote service learned through APP is of the form ap-redirect@207.140.138.118 and can been seen on the **show service summary** screen. When configuring an ACL clause, you cannot use this service as a preferred service. If you save this clause in the startup-config and reboot the CSS, a startup error occurs because this service has not been learned through APP at this point. For example, enter: `clause 10 permit any any destination any prefer ap redirect@207.140.138.118`

Deleting a Clause

To delete a clause, use the **no clause** command. For example, enter:

```
(config-acl[7]) no clause 6
```

Logging ACL Activity

When you configure the CSS to log ACL activity, it logs the event of the packet matching the clause and ACL. The CSS sends log information to the location you specified in the **logging** command. For information on the **logging** command, refer to the *Cisco Content Services Switch Administration Guide*.

Note Before you configure logging for a specific ACL clause, ensure that global ACL logging is enabled. To globally enable ACL logging, use the **(config)# logging subsystem acl level debug-7** command.

Because the CSS does not save the clause log enable command in the running-config, you must reenable logging if the CSS reboots.

To configure logging for an ACL clause:

1. Enter the ACL mode for which you want to enable logging.

```
(config)# acl 7
(config-acl[7])#
```

2. Remove the ACL from the circuit. You must remove an ACL from a circuit before making any clause changes.

```
(config-acl[7]) remove circuit-(VLAN1)
```

3. Enable logging for the existing clause.

```
(config-acl[7])# clause 1 log enable
```

4. Reapply the ACL to the circuit.

```
(config-acl[7])# apply circuit-(VLAN1)
```

To disable ACL logging for a specific clause, enter:

```
(config-acl[7])# clause 1 log disable
```

To globally disable logging for all ACL clauses, enter:

```
(config)# no logging subsystem acl
```

Applying an ACL to a Circuit or DNS Queries

Once you configure the ACL, use the **apply** command to assign an ACL to all circuits, an individual circuit, or to DNS queries.

Note You cannot apply an empty ACL to a circuit. If you attempt to do so, the error message `Cannot apply ACL for it has no clauses` appears.

To add a new clause to an existing and applied ACL, reapply the ACL to the circuit with the **apply circuit** command.

To apply any changes to an existing clause on an existing and applied ACL, you must remove the ACL from the circuit with the **(config-acl) remove** command, and then reapply the ACL to the circuit.

To remove a clause currently in use, you must remove its applied ACL from the circuit, delete the clause, and then reapply the ACL to the circuit.

The syntax and options for this ACL mode command are:

- **apply all** - Applies the ACL to all existing circuits
- **apply circuit** - (*circuit_name*) - Applies the ACL to an individual circuit
- **apply dns** - Adds the ACL to DNS queries

Note If you configure a CSS with the **dns-server** command, and the CSS receives a DNS query for a domain name that you configured on the CSS using the **host** command, the DNS query *will not* match on an ACL that is configured with the **apply dns** command.

However, if you configure a domain name on a content rule on a CSS using the **add dns** *domain_ name* command, a DNS query for that domain name *will* match on an ACL that is configured with the **apply dns** command.

For example, to apply acl 7 to circuit VLAN1:

```
(config-acl[7])# apply circuit-(VLAN1)
```

To display a list of circuits, use the **apply ?** command.

Note You must enter the global **acl enable** command for ACLs to take effect. For information on the **acl enable** command, see "Globally Enabling ACLs" later in this chapter.

Removing an ACL from a Circuit or DNS Queries

Use the **remove** command to remove an ACL from all circuits, an individual circuit, or from DNS queries.

Note To remove a clause currently in use, you must remove its applied ACL from the circuit, delete the clause, and then reapply the ACL to the circuit.

The syntax and options for this ACL mode command are:

- **remove all** - Removes the ACL from an individual circuit. To display a list of circuits that you can remove, use the **remove ?** command.
- **remove circuit** (*circuit_name*) - Removes the ACL from a circuit. To display a list of circuits that you can remove, use the **remove ?** command.
- **remove dns** - Removes the ACL from DNS queries.

For example, enter:

```
(config-acl[7])# remove circuit-(VLAN1)
(config-acl[7])# remove dns
```

Globally Enabling ACLs

Global ACL commands allow you to enable or disable all ACLs simultaneously. Global commands are advantageous when managing your network.

Caution When you enter the **acl enable** command, all traffic is denied except for traffic specified in an ACL permit clause.

To globally enable all ACLs, enter:

```
(config)# acl enable
```

To globally disable all ACLs on the CSS, enter:

```
(config)# acl disable
```

Showing ACLs

Use the **show acl** commands to display access control lists and clauses. The **show acl** commands are available in all modes.

When you show an ACL clause that is applied to a circuit, the display includes:

- **Content Hits** - A flow can be defined as a stream of UDP and TCP packets between a client and a server. The CSS must receive a number of packets from the client and the server before it can completely setup the flow. All of these packets, received before the flow is completely setup, are subject to ACL checks and can cause increments to the ACL Content Hits counter.
- **Router Hits** - All non-UDP and -TCP packets subjected to ACL checks cause increments to the ACL Router Hits counter. All UDP and TCP traffic terminating on the CSS (for example, a Telnet or FTP session) cause increments to the ACL Router Hits counter.

When you show an ACL clause that is applied to DNS queries, the display includes a DNS hit counter, which counts DNS lookups.

The syntax is:

- **show acl** - Displays all ACLs and their clauses.
- **show acl** *index* - Displays the clauses for the specified ACL index number (valid numbers are 1 to 99).
- **show acl config** - Shows the ACL global configuration. This display also shows you which ACLs are applied to which circuits.

For example, enter:

```
(config)# show acl 2
```

Table 5-5 describes the fields in the **show acl** output.

Note The total number of ACL hits for each packet received by the CSS can vary depending on the type of flow and whether an ACL match occurred. The CSS performs an ACL check for every packet received until the ACL flow is completely setup. Once the ACL flow is setup, remaining packets received by the CSS that are associated with the flow are not subject to an ACL match and the ACL hit counters do not increment.

Table 5-5 Field Descriptions for the show acl Command

Field	Description
Acl	The number assigned to the ACL (a number from 1 to 99).
Clause	The number assigned to the clause (a number from 1 to 254).
Action	The method that incoming traffic is controlled by the clause (permit, deny, or bypass) and the protocol for the type of traffic.
Source	The configured source of the traffic.
Destination	The configured destination for the traffic.
Log	Whether or not ACL logging is enabled or disabled on the specified clause.
Content Hits	Increments for a packet received by the CSS before flow setup.
Router Hits	Increments for a packet directly forwarded to the CSS through a Telnet or FTP session or from non-TCP or UDP packets.
DNS Hits	Increments for a packet that matches an ACL clause for DNS flows.

Setting the Show ACL Counters to Zero

Use the **zero counts** command to set the content and DNS hit counters in the **show acl** command screen to zero for a specific ACL. You must be in an ACL to use this command. The CSS clears counters only for that ACL. The syntax and options for this command are:

```
(config-acl[7])# zero counts
```

ACL Example

The following ACL provides security for a CSS, Server1, and Server2 on one VLAN (VLAN1). The ACL:

- Permits clients from subnet 172.16.107.x to access servers 1 and 2 on VLAN1 using various applications (for example, Telnet, FTP, TFTP)
- Permits clients from subnet 172.16.107.x to launch a browser with the URL 172.16.107.35 (the Virtual IP address)
- Prevents clients on any subnet other than 172.16.107.x from accessing VLAN1 and servers 1 and 2

The individual clauses provide the following security.

- Clause 20 permits any protocol from source subnet 172.16.107.0 to Server1 (IP address 172.16.107.15).
- Clause 30 permits any protocol from source subnet 172.16.107.0 to Server2 (IP address 172.16.107.16).
- Clause 50 permits bidirectional communication to the VLAN for any ICMP traffic, including keepalives. If you are using service keepalives, you must configure a clause to permit keepalive traffic.
- Clause 60 permits UDP to port 520 on the VLAN for RIP updates. This clause is required if your router is on a subnet other than 172.16.107.x.
- Clause 70 denies everything that has not been permitted in the ACL.

```
!************************** ACL **************************
acl 1
clause 20 permit any 172.16.107.0 255.255.255.0 destination
172.16.107.15
clause 30 permit any 172.16.107.0 255.255.255.0 destination
172.16.107.16
```

```
clause 50 permit ICMP any destination any
clause 60 permit udp any eq 520 destination any
clause 70 deny any any destination any
apply circuit-(VLAN1)
```

Configuring Extension Qualifier Lists

An Extension Qualifier List (EQL) is a collection of file extensions that enable you to match a content rule based on extensions. You activate an EQL by associating it as part of a URL in a Layer 5 content rule. Use the **eql** command to access EQL configuration mode and configure an extension qualifier list. Enter a name that identifies the extension list you want to create. Enter an unquoted text string with no spaces and a length of 1 to 31 characters.

For example, enter:

```
(config)# eql graphics
(config-eql[graphics])#
```

To remove an existing EQL, use the **no eql** command from config mode. For example, enter:

```
(config)# no eql graphics
```

Once you create an EQL, you can configure the following attributes for it:

- **description** - Provides a description for the EQL. Enter a quoted text string with a maximum length of 64 characters. For example, enter:

  ```
  (config-eql[graphics])# description "This EQL specifies graphic
  file extensions"
  ```

- **extension** *name* - Specifies the extension *name* for content on which you want the CSS to match. Enter a text string from 1 to 7 characters. When configuring EQLs for services, make sure you enter an extension for static content such as .avi, .gif, or .jpg. Do not enter extensions for dynamic content such as .asp and .html. The order in which you enter extensions is irrelevant.

  For example, enter:

  ```
  (config-eql[graphics])# extension pcx
  ```

Optionally, you may provide a *description* of the extension type. Enter a quoted text string with a maximum length of 64 characters. For example, enter:

```
(config-eql[graphics])# extension gif "This is a graphics file"
```

To remove an extension from an EQL, use the **no extension** command. For example, enter:

```
(config-eql[graphics])# no extension gif
```

Specifying an Extension Qualifier List in a Uniform Resource Locator

Server selections are based on the Uniform Resource Locator (URL) specified in the owner content rule. To enable the CSS to access a service when a request for content matches the extensions contained in a previously defined Extension Qualifier List (EQL), specify the URL and EQL name for the content.

Specify a URL as a quoted text string with a maximum of 256 characters followed by **eql** and the EQL name.

Note Do not specify a file extension in the URL when you use an EQL in the URL or the CSS will return an error message. For example, the CSS will "return" an error message for the command **url "/*.txt" eql** graphics. The following command is valid; **url "/*" eql graphics**.

For example, enter:

```
(config-owner-content[arrowpoint.com-products.html])# url "/*" eql
graphics
```

The following example enables the CSS to direct all requests to the correct service for content that matches:

- Pathnames (*/customers/products*)
- Extensions listed in the EQL (*graphics*)

```
(config-owner-content[arrowpoint.com-products.html])# url
"/customers/products/*" eql graphics
```

To display an EQL name and extensions configured for a content rule, use the **show rule** command.

For details on the **show rule** command and its output, refer to Chapter 3, Configuring Content Rules.

Showing EQL Extensions and Descriptions

To display a list of existing EQLs names, use **eql ?** command.

For example, enter:

```
(config)# eql ?
```

To display the extensions configured for a specific EQL including any descriptions, use the **show eql** command and the EQL name. For example, enter:

```
(config)# show eql graphics
```

Table 5-6 describes the fields in the **show eql** output.

Table 5-6 Field Descriptions for the show eql Command

Field	Description
EQL	The name of the EQL and its description, if configured
Extensions	The extensions of content requests associated with the EQL and their descriptions, if configured

Configuring Uniform Resource Locator Qualifier Lists

URQL configuration mode allows you to configure a Uniform Resource Locator Qualifier List (URQL). A URQL is a group of URLs for content that you associate with one or more content rules. The CSS uses this list to identify which requests to send to a service. For example, you want all streaming video requests to be handled by your powerful servers. Create a URQL that contains the URLs for the content, and then associate the URQL to a content rule. The CSS will direct all requests for the streaming video URLs to the powerful servers specified in the content rule. Creating a URQL to group the URLs saves you from having to create a separate content rule for each URL.

Note You cannot specify both **url urql** and **application ssl** within the same content rule. You cannot configure a URQL with subscriber services.

Creating a URQL

To access URQL configuration mode, use the **urql** command. The prompt changes to (config-urql [name]). You can also use this command from URQL mode to access another URQL.

Enter the URQL name you want to create or enter an existing URQL. Enter the name as an unquoted text string with no spaces and a maximum of 31 characters. When you create a URQL, it remains suspended until you activate it using the **activate** command in urql mode. To display a list of existing URQL names, enter:

```
(config)# urql ?
```

For example, enter:

```
(config)# urql videos
(config-urql[videos)#
```

To remove an existing URQL, enter the following command in global configuration mode:

```
(config) no urql videos
```

Once you create a URQL:

1. Configure the URLs you want to group in the URQL by:
   a. Specifying the URL entry
   b. Defining the URL
   c. Optionally, describing the URL
2. Designate the domain name of the URLs in a URQL.
3. Add the URQL to a content rule using the owner-content **url** command.
4. Optionally, describe the URQL.

The following sections describe how to complete these tasks.

Configuring a URL in a URQL

Use the **url** command to include the URL for content requests you want as part of this URQL, and optionally provide a description. Configuring a URL in a URQL includes:

- Specifying the URL Entry
- Defining the URL
- Describing the URL

Note You must create the URL entry before you can define the URL, describe it, or associate it with a content rule.

Specifying the URL Entry

To specify a URL entry in a URQL, enter a URL number from 1 to 1000. For example, enter:

```
(config-urql[videos])# url 10
```

To remove a URL entry from a URQL, use the **no url** command. For example, enter:

```
(config-urql[videos])# no url 10
```

To specify additional URL entries in the URQL, reenter the **url** command. For example, enter:

```
(config-urql[videos])# url 20
(config-urql[videos])# url 30
(config-urql[videos])# url 40
```

Defining the URL

To define a URL for the entry, use the **url** command. Enter the URL as a quoted text string with a maximum of 251 characters. Wildcards are not allowed in a URQL URL. For example, enter:

```
(config-urql[videos])# url 10 url "/cooking/cookies.avi"
```

To remove a URL from an entry, use the **no url** *number* **url** command. Use this command to remove a previously assigned URL before you redefine the URL for an entry. For example, enter:

```
(config-urql[videos])# no url 10 url
```

To define additional URL for the entries, reenter the **url** *entry* **url** command. For example, enter:

```
(config-urql[videos])# url 20 url "/cooking/fudge.avi"
(config-urql[videos])# url 30 url "/cooking/pie.avi"
(config-urql[videos])# url 40 url "/cooking/cake.avi"
```

Describing the URL

You may optionally enter a description for the URL. Enter a quoted text string with a maximum length of 64 characters. For example, enter:

```
(config-urql[videos])# url 10 description "making cookies"
```

To remove a description about the URL, enter:

```
(config-urql[videos])# no url 10 description
```

Designating the Domain Name of URLs in a URQL

Use the **domain** command to designate the domain name or IP address of the URLs to a URQL. Enter the domain name in mnemonic host-name format (for example, www.arrowpoint.com) from 1 to 63 characters. Enter the IP address as a valid address for the domain name (for example, 192.168.11.1).

Note You must assign a domain before you can activate a URQL. To change the domain address of an existing URQL, suspend the URQL and then change the domain.

For example, enter:

```
(config-urql[videos])# domain "www.arrowpoint.com"
```

or

```
(config-urql[videos])# domain "192.168.11.1"
```

Adding a URQL to a Content Rule

Once you create and configure a URQL, use the **url urql** command to add it to a previously configured content rule. You can only assign one URQL per rule. Also, a content rule may contain either a URL or a URQL. To see a list of URQLs, use the **urql ?** command.

Note You cannot specify both **url urql** and **application ssl** within the same content rule. You cannot configure a URQL with subscriber services.

For example, enter:

```
(config-owner-content[chefsbest-recipes])# url urql videos
```

To remove a URQL from a content rule, enter:

```
(config-owner-content[chefsbest-recipes])# no url urql
```

To display a URL for a content rule, use the **show rule** command for the content rule. For details on the **show rule** command and its output, refer to Chapter 3, Configuring Content Rules.

Describing the URQL

Use the **description** command to provide a description for a URQL. Enter the description an a quoted text string with a maximum of 64 characters.

For example, enter:

```
(config-urql[videos])# description "cooking streaming video"
```

To clear a description for the URQL, enter:

```
(config-urql[videos])# no description
```

Activating a URQL

Use the **active** command to activate a suspended URQL. When you create a URQL, it is suspended until you use the **active** command to activate it.

Note Before you can activate a URQL, you must assign the domain for the URLs. See "Designating the Domain Name of URLs in a URQL" in this chapter.

For example, enter:

```
(config-urql[videos])# active
```

Suspending a URQL

Use the **suspend** command to deactivate a URQL on all currently assigned content rules. For example, enter:

```
(config-urql[videos])# suspend
```

To reactivate the URQL, use the **(config-urql) active** command.

URQL Configuration in a Startup-Config File

The following example shows a URQL configuration in a startup-config file.

```
!*************************** URQL ************************
urql excellence1
    url 10
    url 30
    url 30 url "/arrowpoint.gif"
    domain "192.168.128.109"
    url 10 url "/"
urql excellence2
    url 10
    url 10 url "/poweredby.gif"
    domain "192.168.128.109"
```

Showing URQLs

To display a list of URQLs, enter:

```
(config)# urql ?
```

To display all configured URQLs, enter:

```
(config)# show urql
```

To display a specific URQL, enter:

```
(config)# show urql videos
```

Table 5-7 describes the fields in the **show urql** output.

Table 5-7 Field Descriptions for the show urql Command

Field	Description
Name	The name of the URQL
Description	The configured description for the URQL
Domain	The domain name or address of the URLs associated with the URQL
Create Type	The create type (static or dynamic)

Table 5-7 Field Descriptions for the show urql Command (continued)

Field	Description
State	The state of the URQL (Active or Suspended)
Rules Associated	The number of rules associated with the URQL

Table 5-8 describes the additional fields when you display a specified URQL.

Table 5-8 Field Descriptions for a Specified URQL

Field	Description
URQL Table Domain	The domain name or address of the URLs associated with the URQL
Number of entries configured	The number of URL entries in the URQL
URL	The URL
Description	The description associated with the URL
Create Type	The create type (static or dynamic)
State	The state of the URL (Active or Suspended)
CSD Entries	The number of CSD entries

Configuring Network Qualifier Lists

NQL configuration mode allows you to configure a Network Qualifier List (NQL). An NQL is a list of networks or specific services, identified by IP address and subnet mask, that you assign to an ACL clause as a source or destination. By grouping networks into an NQL and assigning the NQL to an ACL clause, you have to create only one clause instead of a separate clause for each network.

The CSS enables you to configure a maximum of 512:

- Networks or services per NQL
- NQLs per CSS

This functionality is useful, for example, in a caching environment where you have a network you want to bypass and send content requests directly to the origin servers (servers containing the content). You can also use an NQL for users who prefer a service based on a specific network.

To access NQL configuration mode, use the **nql** command. The prompt changes to (config-nql [name]). You can also use this command from NQL mode to access another NQL.

See the following sections to configure an NQL:

- Creating an NQL
- Describing an NQL
- Adding Networks to an NQL
- Adding an NQL to an ACL Clause
- Showing NQL Configurations

Creating an NQL

Enter the name of the new NQL you want to create or an existing NQL. Enter the name as an unquoted text string with no spaces and a maximum of 31 characters. You can create a maximum of 512 NQLs per CSS.

For example, enter:

```
(config)# nql bypass_nql
(config-nql[bypass_nql])#
```

To display a list of existing NQLs, use the **nql ?** command. If no NQLs currently exist, the CSS prompts you to enter a new name.

To remove an existing NQL, use the **no nql** command. For example, enter:

```
(config)# no nql bypass_nql
```

Describing an NQL

Use the **description** command in NQL mode to provide a description for an NQL. Enter the NQL description as a quoted text string with a maximum length of 63 characters.

For example, enter:

```
(config-nql[bypass_nql])# description "Bypass services"
```

Adding Networks to an NQL

Use the **ip address** command to add a maximum of 512 networks or services to an NQL. Enter an IP address with either a subnet prefix or a subnet mask. You may also add an optional description for the IP address and turn on logging.

The syntax and options are:

ip address *ip_address*[*/subnet_prefix*| *subnet_mask*] {**"***description***"**}{**log**}

The variables and options are:

- *ip_address* - The destination network address. Enter the IP address in dotted-decimal notation (for example, 192.168.0.0).
- *subnet_mask* - The IP subnet mask prefix length in CIDR bitcount notation (for example, /16). The valid prefix length range is 8 to 32. Do not enter a space to separate the IP address from the prefix length.
- *subnet_address* - The IP subnet mask in dotted-decimal notation (for example, 255.255.0.0).
- **"***description***"** - A description of the IP address. Enter a quoted text string with a maximum of 63 characters.
- **log** - Logs an event involving an NQL. If you do not enter this option, events are not logged. To log an NQL event, you must enable global NQL logging. To enable global NQL logging, use the **(config) logging subsystem nql level debug-7** command. For logging information, refer to the *Cisco Content Services Switch Administration Guide*.

For example, to add two networks to the NQL *bypass_nql*, enter:

```
(config-nql[bypass_nql])# ip address 192.168.0.0/16 "Network of
dynamic mail content" log
(config-nql[bypass_nql])# ip address 123.123.123.0/24
```

To log events occurring on a network, you must also enable global NQL logging. For example, enter:

```
(config)# logging subsystem nql level debug-7
```

Note If you do not include a description or turn on logging when you create the entry and later wish to add a description or turn on logging, you must first remove the entry and then add it again with the desired options.

To remove an IP address from an NQL, use the **no ip address** command. For example, enter:

```
(config-nql[bypass_nql])# no ip address 192.168.0.0/16
```

Adding an NQL to an ACL Clause

To add an NQL to an ACL clause:

1. Create the ACL. For example, enter:

   ```
   (config)# acl 10
   ```

2. Define the clause, including the NQL as either a source or destination.

   This clause example bypasses content rules for any traffic from any source going to the destination networks defined in NQL *bypass_nql* on port 80.

   ```
   (config-acl[10])# clause 1 bypass any any destination nql
   bypass_nql eq 80
   ```

Showing NQL Configurations

Use the **show nql** command to display NQL configuration information. The syntax for this command is:

- **show nql** - Displays information for all NQLs. If you enter this command in NQL mode, the CSS displays the addresses only for the current NQL.
- **show nql** *nql_name* - Displays information for the specified NQL. Enter the NQL name as a case-sensitive unquoted text string with no spaces. To see a list of existing NQL names, use the **show nql ?** command.

For example, enter:

```
(config-nql[bypass_nql])# show nql
```

Table 5-9 describes the fields in the **show nql** output.

Table 5-9 Field Descriptions for the show nql Command

Field	Description
Name	The name of the NQL.
Description	The description associated with the NQL.
IP Addresses	The IP addresses and subnet mask supported by the NQL. If configured, a description appears after the address.

Configuring Domain Qualifier Lists

When you have a requirement for a content rule to match on multiple domain names, you can associate a Domain Qualifier List (DQL) to the rule. A DQL is a list of domain names that you configure and assign to a content rule, instead of creating a content rule for each domain. Assigning multiple domain names to a DQL enables you to have many domain names match on one content rule.

You can use a DQL on a rule to specify that content requests for each domain in the list will match on the rule. You can determine the order that the domain names are listed in the DQL. You can arrange the names in a DQL by assigning an index number as you add the name to the list.

Note The CSS supports a maximum of 512 DQLs, with a maximum of 2,500 DQL domain name entries. This means that a single DQL can have up to 2500 entries, or five DQLs can have up to 500 entries for each DQL.

DQLs exist independently of any range mapping. You can use them as a matching criteria to balance across servers that do not have VIP or port ranges. If you want to use range mapping when using range services, you need to consider the index of any domain name in the DQL. If you are not using service ranges with DQLs, you do not need to configure any index and the default index is 1.

For example, you could configure a DQL named Woodworker.

```
(config)# dql Woodworker
```

The domain names you could add as part of the DQL include *www.wood.com*, *www.woodworker.com*, *www.maple.com*, *www.oak.com*. You could configure *www.wood.com* and *www.woodworker.com* to have the same mapping index. You can enter indexes from 1 to 1000 and provide an optional quoted description for each index.

For example, enter:

```
(config-dql[Woodworker]# domain www.wood.com index 1 "This is the
same as the woodworker domain"
(config-dql[Woodworker]# domain www.woodworker.com index 1
(config-dql[Woodworker]# domain www.maple.com index 2
(config-dql[Woodworker]# domain www.oak.com index 3
```

If you specify a DQL as a matching criteria for content rule WoodSites, and there are two services, S1 and S2, associated with the rule, the CSS checks the services at mapping time for ranges. To add a DQL to a content rule, use the **url** command as shown:

```
(config-owner-content[WoodSites])# url "/*" dql Woodworker
```

For example, if the CSS receives a request for *www.oak.com* along with other criteria, a match on the WoodSites rule occurs on DQL index 3. If the rule has the roundrobin balance method configured, the CSS examines a service (S2 for this example) to determine the backend connection mapping parameters. If you configured S2 with a VIP address of 10.0.0.1 with a range of 5, the addresses include 10.0.0.1 through 10.0.0.5. Because this service has a range of address and **any** as its port, the DQL index of 3 matches the service VIP range index of 3, which is address 10.0.0.3.

To access DQL configuration mode, use the **dql** command from any configuration mode except boot, group, RMON alarm, RMON event, and RMON history configuration modes. The prompt changes to (config-dql [name]). You can also use this command from DQL mode to access an existing DQL.

See the following sections to configure a DQL:

- Creating a DQL
- Describing a DQL
- Adding a Domain to a DQL
- Adding a DQL to a Content Rule
- Removing a DQL from a Content Rule
- Showing DQL Configurations

Creating a DQL

To create a new DQL, enter the name of the DQL you want to create as an unquoted text string with no spaces and a maximum of 31 characters. To access an existing DQL, enter the DQL name. To display a list of existing DQL names, use the **dql ?** command.

For example, to configure a DQL:

```
(config)# dql pet_domains
(config-dql[pet_domains])#
```

Describing a DQL

Use the **description** command to provide a description for DQL. Enter the description as a quoted text string with a maximum of 63 characters, including spaces.

For example, enter:

```
(config-dql[pet_domains])# description "pet supplies"
```

Adding a Domain to a DQL

Use the **domain** command to add a domain to the list of domains supported by a DQL. The syntax is:

domain *name* **index** *number* {**"***description***"**}

The variables and option are:

- *name* - The name of the domain. Enter an unquoted text string with a maximum of 63 characters (for example, www.arrowpoint.com). The CSS matches the domain name exactly.
- *number* - The index number for the domain. Enter a number from 1 to 10000. If a domain has more than one domain name, you can assign the same index number to its different names.
- **"***description***"** - A description of the domain name. Enter a quoted text string with a maximum of 63 characters.

Note The CSS supports a maximum of 512 DQLs, with a maximum of 2,500 DQL domain name entries. This means that a single DQL can have up to 2500 entries, or five DQLs can have up to 500 entries for each DQL.

For example, enter:

```
(config-dql[pet_domains])# domain www.birds.com index 1
"idaho-based"
(config-dql[pet_domains])# domain www.cats.com index 2 "worldwide"
(config-dql[pet_domains])# domain www.horses.com index 3
"florida-based"
```

Normally, port 80 traffic does not use a port number in the domain name. To specify a port other than port 80, enter the domain name with the port number exactly. Separate the domain name and the port number with a colon. For example, enter:

```
(config-dql[pet_domains])# domain www.dogs.com:8080 index 4
```

To add or delete a domain name from a DQL that is assigned to a content rule, you must first suspend the content rule using the **suspend** command. You cannot make changes to a DQL currently in use by a content rule.

For example, to remove a domain from the example DQL, enter:

```
(config-dql[pet_domains])# no domain www.birds.com
```

Adding a DQL to a Content Rule

Once you have configured a DQL, use the **url** command to add it to a content rule. You cannot use wildcards in DQL entries.

For example, enter:

```
(config-owner-content[pets.com-rule1])# url "/*" dql pet_domains
```

Removing a DQL from a Content Rule

To remove a DQL that is assigned to a content rule, you must first suspend the content rule using the **suspend** command. You cannot remove a DQL currently in use by a content rule. Once the content rule is suspended, use the **no dql** command to remove the DQL from the content rule.

For example, enter:

```
(config) no dql pet_domains
```

Showing DQL Configurations

Use the **show dql** command to display all DQL configurations. To display a specific DQL, include the DQL name in the command line.

For example, enter:

```
(config-dql[pet_domains])# show dql pet_domains
```

Table 5-10 describes the fields in the **show dql** output.

Table 5-10 Field Descriptions for the show dql Command

Field	Description
Name	The name of the DQL
Index	The CSS unique index which identifies the DQL

Table 5-10 Field Descriptions for the show dql Command

Field	Description
Description	The description for the DQL
Index	The DQL unique index number for this domain
Domain	The name of the domain associated with the index number
Description	The description for the domain

Configuring Virtual Web Hosting

Virtual Web hosting enables you to host a large number of Web sites on a small number of servers (typically 2 to 10 servers) that have mirrored content. Each server may contain hundreds or thousands of Web sites. The servers determine which Web site is being requested based on IP address, port, and domain name.

Using virtual Web hosting, you may configure:

- Services with either a range of IP addresses or a range of ports.
- Content rules with either a range of VIPs or a DQL (but not both). This would allow the CSS to map the range of VIPs or the domain names in the DQL to the servers.
- Content rules with either a range of VIPS or a DQL (but not both) that would map to a server without a range. This allows the CSS to map many domain names to one server.

You can configure the CSS to load balance the Web sites by configuring port ranges, VIP ranges, or DQLs. For more information on the service and content rule commands required, see Chapter 1, Configuring Services and Chapter 3, Configuring Content Rules.

See Table 5-11 for the steps required to configure virtual Web hosting.

Table 5-11 Virtual Web Hosting Configuration Quick Start

Task and Command Example
1. Enter config mode by typing **config**. `(config)#`

2. Create a service.
`(config)# service serv1`
`(config-service[serv1])#`

3. Assign an IP address to the service and define the IP address range. Enter a number from 1 to 65535. When using the **ip address range** command, use IP addresses that are within the subnet you are using. The CSS does not use ARP for IP addresses that are not on the circuit subnet. `(config-service[serv1])# ip address 10.3.6.1 range 200`

4. Configure other service rules as needed (for example, protocol, keepalive parameters).
`(config-service[serv1])# protocol tcp`
`(config-service[serv1])# keepalive type http`
`(config-service[serv1])# keepalive method get`
`(config-service[serv1])# keepalive uri "/index.html"`

5. Activate the service. `(config-service[serv1])# active`

6. Create the content rule.
`(config-owner[arrowpoint])# content rule1`
`(config-owner-content[arrowpoint-rule1])#`

7. Configure a VIP. You can define a VIP range only if you do not plan to configure a DQL. `(config-owner-content[arrowpoint-rule1])# vip address 192.168.3.6 range 10` When using the **vip address range** command, use IP addresses that are within the subnet you are using. The CSS does not use ARP for IP addresses that are not on the circuit subnet.

Table 5-11 Virtual Web Hosting Configuration Quick Start (continued)

Task and Command Example

8. Configure other content rule commands as needed (for example, port, protocol, and add a service).
`(config-owner-content[arrowpoint-rule1])# port 80`
`(config-owner-content[arrowpoint-rule1])# protocol tcp`
`(config-owner-content[arrowpoint-rule1])# add service serv1`

9. Activate the content rule. `(config-owner-content[arrowpoint-rule1])# active`

10. If you have not configured a VIP range, you can create a DQL.
`(config)# dql pet_domains`
`(config-dql[pet_domains])#`

11. Add domains to the DQL you created.
`(config-dql[pet_domains])# domain www.birds.com index 1 "idaho-based"`
`(config-dql[pet_domains])# domain www.cats.com index 2 "worldwide"`
`(config-dql[pet_domains])# domain www.horses.com index 3 "florida-based"`

12. Add the DQL to the content rule using the **url** command. `(config-owner-content[arrowpoint-rule1])# url "/*" dql pet_domains`

Where to Go Next

You can configure HTTP header load balancing by creating an HTTP header field group and configuring HTTP header fields. For information, see Chapter 6, Configuring HTTP Header Load Balancing.

Configuring HTTP Header Load Balancing

This chapter describes how to configure HTTP header load balancing by creating an HTTP header field group and configuring HTTP header fields. Information in this chapter applies to all CSS models except where noted.

This chapter contains the following sections:

Note You must enable service remapping for HTTP header load balancing to work properly. For information on the service remapping feature, refer to Chapter 3, Configuring Content Rules.

HTTP Header Load Balancing Overview

Configuring HTTP header load balancing enables the CSS to inspect incoming content requests for HTTP header fields. This allows the CSS to make load balancing decisions based on the HTTP header field information and then direct content requests to the servers designed to handle the type of content being requested.

The CSS can direct content requests to specific servers based on different types of browsers or different representations of the same content that has been modified for end users. For example, a client running a hand-held personal organizer may want the same content as a client using a PC, but with fewer graphics. Or German users may want to see content in German only.

Using HTTP header load balancing eliminates the need to duplicate various forms of the same content across all of the servers, thus freeing up valuable server space. In addition to dividing the server farm for different types of clients, you can also use HTTP header load balancing to bypass non-cacheable traffic and prioritize client browser traffic from search engine services.

Using HTTP Header Load Balancing in a Content Rule

Using an HTTP header field group in a Layer 5 content rule enables a rule to be more specific than if the rule just defined a URL. The HTTP header field group makes the content match more specific. Because content rules are hierarchical, if a request for content matches more than one rule, the characteristics of the most specific rule apply to the flow. This hierarchy for Layer 5 rules is defined below. The CSS uses this order of precedence to process requests for the content, with 1 being the highest match and 4 being the lowest match.

1. Domain name, IP address, protocol, port, URL, HTTP header field group
2. IP address, protocol, port, URL, HTTP header field group
3. Domain name, protocol, port, URL, HTTP header field group
4. Protocol, port, URL, HTTP header field group

HTTP Header Load Balancing Configuration Quick Start

Table 6-1 provides a quick overview of the steps required to create and configure HTTP header load balancing. Each step includes the CLI command required to complete the task. For a complete description of each feature and all the HTTP header load balancing configuration options, see the sections following Table 6-1.

Ensure that you have already created and configured a service and owner for the content rules. The command examples in Table 6-1 create HTTP load balancing for owner *arrowpoint* and content rule *rule1*.

Table 6-1 HTTP Load Balancing Configuration Quick Start

Task and Command Example
1. Enter into config mode by typing **config**. `(config)#`

2. Create a header field group. This example creates the group *ppilot*.
`(config)# header-field-group ppilot`
`(config-header-field-group[ppilot])#`

3. Describe the header field group (optional). `(config-header-field-group[ppilot])# description "ppilot content"`
4. Configure header field entries by defining a header, field, name, field type, and operator. `(config-header-field-group[ppilot])# header-field palm1 user-agent contain "MSIE" 20`
5. Associate the header field group to a content rule. `(config-owner-content[arrowpoint-rule1])# header-field-rule ppilot`
6. Display the header field group to verify your configuration (optional). `(config)# show header-field-group`

Creating a Header Field Group

Header field group configuration mode allows you to create a *header field group*. A header field group contains a list of user-defined header field entries used by the CSS content rule lookup process. A group can contain several header-field entries.

Note The CSS supports a maximum number of 1024 header field groups, with a maximum of 4096 header field entries.

Note When there is more than one header field entry in a group, each header field entry must be successfully matched before the CSS uses the associated content rule.

To create a header field group or to access header field group configuration mode, use the **header-field-group** command from all configuration modes except boot and RMON modes.

The prompt changes to (config-header-field-group [group_name]). You can also use this command in header-field-group mode to access another group.

The syntax for this mode transition command is:

header-field-group *group_name*

Enter the *group_name* of the header-field group you want to create. You must define a unique name for each header field group so different content rules can use the groups. Enter a text string with a maximum of 32 characters. To see an existing list of header-field groups, use the **header-field-group ?** command.

For example, enter:

```
(config)# header-field-group ppilot
(config-header-field-group[ppilot])#
```

To remove a header-field group, use the **no header-field-group** command. For example, enter:

```
(config)# no header-field-group ppilot
```

Describing the Header Field Group

Use the **description** command to provide a description for a header field group. The syntax for this command is:

description "*text***"**

Enter the text as a quoted text string with a maximum length of 64 characters.

For example,

```
(config-header-field-group[ppilot])# description "ppilot content"
```

To remove a description for a header-field group, enter:

```
(config-header-field-group[ppilot])# no description
```

Configuring a Header Field Entry

Use the **header-field** command to define a header field entry in a header field group. A header field entry contains a header field name, field type to be used, an operation to be performed, the header-string to be searched for, and an optional search length.

If a header field group contains multiple header field entries, a content request must match each entry for the rule to be used.

Note The CSS supports a maximum number of 1024 header field groups, with a maximum of 4096 header field entries.

The syntax for this command is:

header-field *name field_type operator* {*header_string* {*search_length*}}

The variables and options are:

- *name* - The name uniquely identifies the header field entry. Enter the name as a string from 1 to 31 characters. You must define a header field entry name because the CSS can use the same field type multiple times in a header field group.
- *field_type* - The field type includes one of the following:
  - **user-agent**
  - **language**
  - **host**
  - **cache-control**
  - **pragma**
  - **encoding**
  - **charset**
  - **connection**
  - **referer**
  - **accept**
  - **request-line**
  - **cookies**
  - **msisdn** - The header field type for Wireless Application Protocol (WAP). HTTP requests from some wireless gateways contain the MSISDN field in the HTTP header. By configuring the msisdn header field type in a header field group, you can load balance wireless requests. See "Example 3. Wireless configuration that load balances HTTP requests based on the MSISDN header field" later in this chapter.

Note You can use this feature alone or with the **advanced-balance wap-msisdn** sticky command. Refer to Chapter 4, Configuring Sticky Parameters for Content Rules, "Specifying an Advanced Load-Balancing Method for Sticky Content".

- *operator* - Enter one of the following operators:
  - **exist|not-exist** - Use the **exist** and **not-exist** operators to check whether or not a specified header field exists in a content request header.
  - **equal|not-equal** {**"***header_string***"**} - Use the **equal** and **not-equal** operators to match a defined *header_string* to the contents of the specified header field, and determine whether or not it is equal to the header string. Enter the *header_string* as a quoted text string with a maximum of 31 characters including spaces.
  - **contain|not-contain** {**"***header_string***"** {*search_length*}} - Use the **contain** and **not-contain** operators to match the configured *header_string* to a substring in the contents of the specified field type, and determine whether or not its contents contain the *header_string*. Enter the *header_string* as a quoted text string with a maximum of 31 characters including spaces.

    You may include an optional *search_length* to define the header field portion to be used for the operation. If you do not define a search length, the CSS uses the entire header field (delimited by a CR and LF) for the operation. To define the search length, enter a number from 0 to 1024.

For example, enter:

```
(config-header-field-group[ppilot])# header-field palm1 user-agent
contain "MSIE" 20

(config-header-field-group[ppilot])# header-field palm2 user-agent
contain "palm"
```

To remove a header field entry, use the **no header-field** command. For example, enter:

```
(config-header-field-group[ppilot])# no header-field palm1
```

Associating a Header Field Group to a Content Rule

Use the **header-field-rule** command to associate a header field group with a content rule, and optionally assign a weight value to the header field group. Use weights to allow the CSS to prefer one content rule over a similar content rule. For example, you want to load balance French clients to a specific server, and you also want to differentiate French Internet Explorer clients from French Netscape clients. If it is more important to direct the French clients to a specific server than to direct them to a server based on whether they are using Internet Explorer or Netscape, then you need to weight the "French" content rule higher than the "Internet Explorer/Netscape" content rule.

Note The CSS supports only one header field group for each content rule.

The syntax for this content mode command is:

header-field-rule *name* {*weight number*}

The variables are:

- *name* - The name of the header field group used with the content rule. To see a list of groups, use the **header-field-rule ?** command.
- **weight** *number* - The weight you want to assign to the header field group. Enter a number from 0 to 1024. The default weight is 0.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# header-field-rule french
weight 3
```

To remove the header field group from the content rule, enter:

```
(config-owner-content[arrowpoint-rule1])# no header-field-rule
```

Showing a Content Rule Header Field Group Configuration

Use the **show rule header-field** command to display information about the header field group associated with a content rule. The syntax is:

show rule header-field

For example, to display information about the header-field rule and group associated with a specific content rule, enter:

```
(config-owner-content[arrowpoint-rule1])# show rule header-field
```

Showing Header Field Groups

Use the **show header-field-group** command to display the configuration for all header field groups or a specific group. This command is available in all modes.

The syntax and options for this command are:

- **show header-field-group** - Displays a summary of all configured header field groups
- **show header-field-group all** - Displays detailed information about all configured header field groups
- **show header-field-group** *name* - Displays detailed information about a specific header field group

For example, to show a summary of all configured header field groups, enter:

```
(config)# show header-field-group
```

Table 6-2 describes the fields in the **show header-field-group** output.

Table 6-2 Field Descriptions for the show header-field-group Command

Field	Description
Header field group	The name of the header-field group
Description	The configured description for the header-field group

Header Field Group Configuration Examples

When configuring header field groups, it is good practice to configure rules to be specific in rule matching (as shown in configuration example 2). If the rules are not specific enough, the CSS may match a client request to the first rule it finds and the first matched rule could change on subsequent requests.

Configuration examples 1 and 2 show the header field group and owner portions of a running-config. Configuration example 3 shows a wireless configuration.

Example 1. Header field group configuration that is ambiguous in rule-matching capabilities.

Example 1 shows a configuration that is ambiguous. If a client request specifies the language as French and the user-agent as Netscape, this request could match equally to ruleA2 or ruleA3. In this example, the rule matching may not be consistent. One method to solve the ambiguity between ruleA2 and ruleA3 is to use different weight values. If you assign a weight value of 10 to header field group B when you associate it with ruleA2, the CSS will always use ruleA2 as a match to the example client request. Another method is to configure more specific rules as shown in configuration example 2.

```
! ***************** HEADER FIELD GROUP ********************

header-field-group A
    header-field ua1 language equal "en"

header-field-group B
    header-field ua2 language equal "fr"

header-field-group C
    header-field-group ua3 user-agent contain "Netscape"

! ********************** OWNER ****************************

owner arrowpoint
    content ruleA
        protocol tcp
        vip address 192.168.128.151
        port 80
        url "/*"
        add service server1
        add service server2
```

```
content ruleA1
    protocol tcp
    vip address 192.168.128.151
    port 80
    url "/*"
    header-field-rule A
    add service server11
    add service server12

content ruleA2
    protocol tcp
    vip address 192.168.128.151
    port 80
    url "/*"
    header-field-rule B
    add service server21
    add service server22

content ruleA3
    protocol tcp
    vip address 192.168.128.151
    port 80
    url "/*"
    header-field-rule C
    add service server31
    add service server32
```

Example 2. Header field group configuration that broadens the rule-matching capabilities.

Example 2 shows the same configuration as Example 1 only modified to broaden the rule-matching capabilities. Each content rule is specific. The client request specifying the language as French and the user-agent as Netscape will match only on Rule A2.

```
! ***************** HEADER FIELD GROUP ********************

header-field-group A
    header-field ua1 language equal "en"
    header-field ua2 user-agent contain "Netscape"

header-field-group B
    header-field ua3 language equal "fr"
    header-field ua4 user-agent contain "Netscape"

header-field-group C
    header-field ua5 language equal "en"
    header-field ua6 user-agent not-contain "Netscape"
```

```
header-field-group D
    header-field ua7 language equal "fr"
    header-field ua8 user-agent not-contain "Netscape"

! ********************** OWNER ***************************

owner arrowpoint
    content ruleA
        protocol tcp
        vip address 192.168.128.151
        port 80
        url "/*"
        add service server1
        add service server2

    content ruleA1
        protocol tcp
        vip address 192.168.128.151
        port 80
        url "/*"
        header-field-rule A
        add service server11
        add service server12

    content ruleA2
        protocol tcp
        vip address 192.168.128.151
        port 80
        url "/*"
        header-field-rule B
        add service server21
        add service server22

    content ruleA3
        protocol tcp
        vip address 192.168.128.151
        port 80
        url "/*"
        header-field-rule C
        add service server31
        add service server32

    content ruleA4
        protocol tcp
        vip address 192.168.128.151
        port 80
        url "/*"
```

```
    header-field-rule D
    add service server41
    add service server42
```

Example 3. Wireless configuration that load balances HTTP requests based on the MSISDN header field

Example 3 shows a configuration that makes load-balancing decisions based on whether or not a client is a wireless client. Wireless devices use the Wireless Application Protocol (WAP). When a wireless client sends a request for content, the WAP protocol gateway (a device that translates requests from the WAP protocol stack to the WWW protocol stack) generates the MSISDN field and adds it to the HTTP header. You can test for the presence of the MSISDN header field using the **exist** and **not-exist** operators in the header field entry of a header field group. Then, you can make load-balancing decisions based on the presence or absence of the MSISDN header field. For details on configuring the MSISDN header field type, see "Configuring a Header Field Entry" earlier in this chapter.

In the following example, any TCP port 80 traffic destined for VIP 192.168.128.151 that has the MSISDN field in the HTTP header will hit the content rule ruleWap. Any TCP port 80 traffic destined for 192.168.128.151 that does not have the MSISDN field in the HTTP header will hit the content rule ruleNoWap.

```
header-field-group wap
   header-field 1 msisdn exist

owner arrowpoint
   content ruleWap
     vip address 192.168.128.151
     protocol tcp
     port 80
     url "/*"
     add service server1
     add service server2
     header-field-rule wap
     active

   content ruleNoWap
     vip address 192.168.128.151
     protocol tcp
     port 80
     url "/*"
     add service server21
     add service server22
     active
```

Note You can use the MSISDN header field with the **advanced-balance wap-msisdn** command to configure wireless users for e-commerce applications. For details on configuring a wireless user, refer to Chapter 4, Configuring Sticky Parameters for Content Rules, "Configuring Wireless Users for E-Commerce Applications".

Where to Go Next

You can configure the CSS for content caching using content rules and a service type that supports caching. For information about configuring the CSS for content caching, refer to Chapter 7, Configuring Caching.

Configuring Caching

This chapter provides an overview of the CSS caching feature and describes how to configure it for operation. Information in this chapter applies to all CSS models, except where noted.

The chapter includes the following sections:

Caching Overview

Increasing demand for information on the Internet causes congestion and long delays in retrieving information. Because much of the same information is retrieved over and over again, saving and storing this information can satisfy subsequent requests with more efficiency and less bandwidth.

Saving and storing information locally is known as *caching*. With Web caching, copies of recently requested content are stored temporarily on a cache server in locations that are topologically closer to the client. The content is then readily available to be reused for subsequent client requests for the same content.

By storing content locally, you:

- Optimize network resources
- Conserve network bandwidth
- Reduce Internet congestion
- Improve network response time and overall service quality

Content Caching

You can make Web caching cost-effective and more reliable by deploying Content Caching in your network. By creating content rules to utilize your cache servers, the CSS acts as a cache front end device by:

- Examining network traffic for Web content requests
- Bypassing the cache automatically for non-cacheable content
- Distributing content requests to maximize cache hits on services
- Bypassing the cache or redistributing content requests among the remaining cache services if a cache service fails

When a client requests content, the CSS:

- Intercepts the request for content
- Applies content intelligence by parsing the HTTP request header to distribute content requests to the cache servers

The CSS then either:

- Directs the request to the appropriate cache based on the load-balancing method you specify in the content rule (for example, destination IP address)
- Bypasses the cache servers and forwards the request to the origin server if the content is non-cacheable

When the CSS directs the request to the cache server, the cache server either returns the requested content (if it has a local copy) or sends a new request for the content through the CSS to the origin server hosting the content. When the cache sends a new request for content and receives a reply from the origin server, it returns the response to the client. If the content is cacheable, the cache saves a copy of the content for future requests.

When the requested content is found on a local cache server, the request is known as a *cache hit*. When the requested content is not local and the cache initiates a new request for the content, the request is known as a *cache miss*.

The following sections provide CSS examples of:

- Using Proxy Caching
- Using Reverse Proxy Caching
- Using Transparent Caching
- Using Cache Clustering

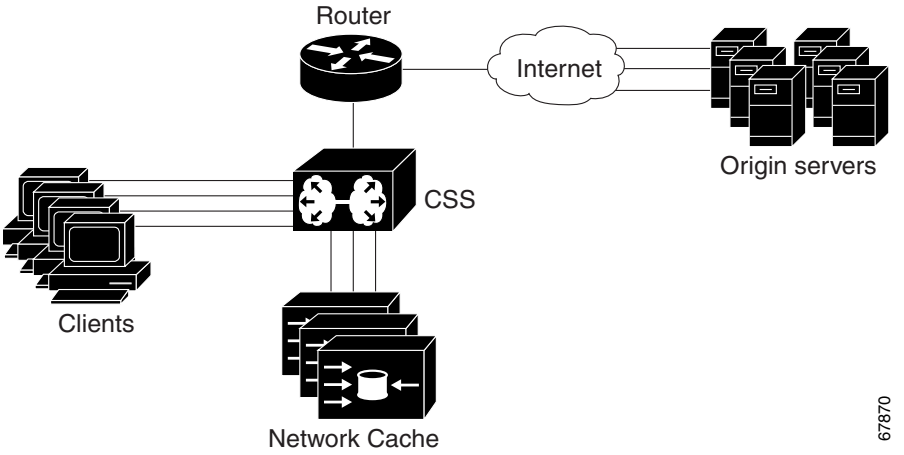
Using Proxy Caching

With proxy caching, each client is configured with the IP address of the proxy cache to which clients send content requests. You may also configure a URL for browsers to identify the location of the proxy configuration file for automatic proxy configuration. Each client's content request is sent directly to the proxy cache IP address. The cache either returns the requested content if it has a local copy or sends a new request to the origin server for the information.

If all cache servers are unavailable in a proxy cache configuration, the client request does not pass to the origin server because clients are configured with the proxy cache VIP.

Figure 7-1 shows an example of using a CSS in a proxy cache configuration.

Figure 7-1 Proxy Cache Configuration Example



Using Reverse Proxy Caching

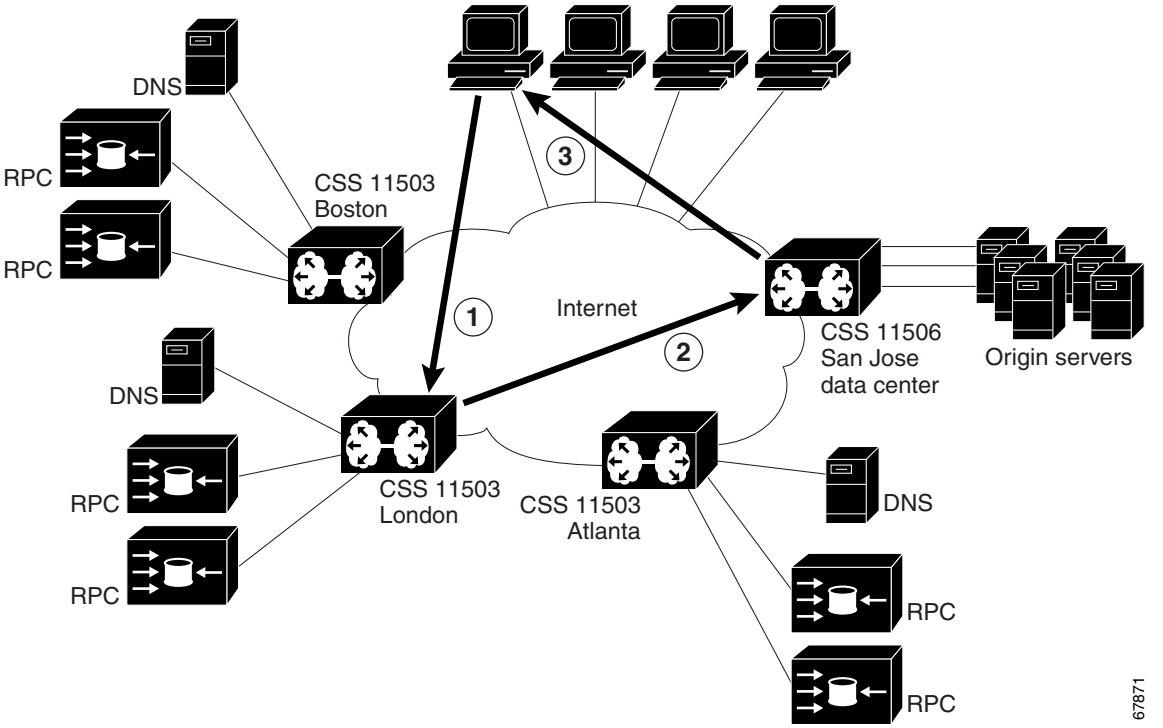
In a reverse proxy cache configuration, the proxy server is configured with an Internet-routable IP address. Clients are directed to the proxy server based on a Domain Name System (DNS) resolution of a domain name. To a client, the reverse proxy server appears like a Web server.

In a regular proxy cache configuration, the proxy server acts as a proxy for the client. In the reverse proxy configuration, the reverse proxy server acts as a proxy for the server. Also, a reverse proxy cache caches specific content, whereas proxy and transparent caches cache frequently requested content. Reverse proxy caches serve two primary functions:

- Replication of content to geographically dispersed areas
- Replication of content for load balancing

Figure 7-2 shows an example of a CSS 11506 and CSS 11503s in a reverse proxy cache configuration.

Figure 7-2 Reverse Proxy Cache Configuration Example



Using Transparent Caching

Transparent caching deploys cache servers that are transparent to the browsers. You do not have to configure browsers to point to a cache server. Cache servers duplicate and store inbound Internet data previously requested by clients.

When you configure transparent caching on the CSS, the CSS intercepts and redirects outbound client requests for Internet data to the cache servers on your network. The cache either returns the requested content if it has a local copy or sends a new request to the origin server for the information.

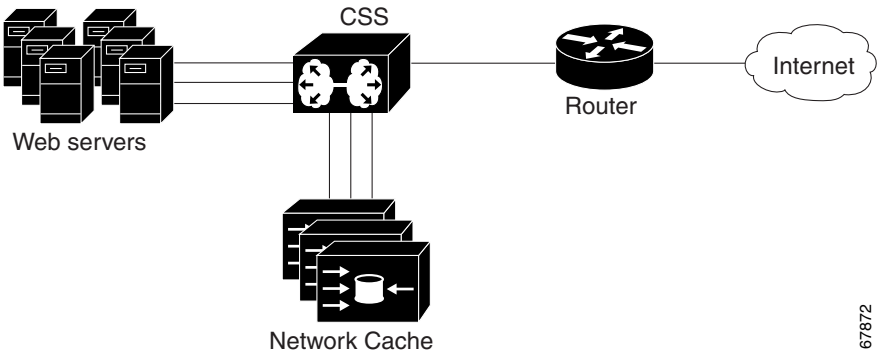
If all cache servers are unavailable in a transparent cache configuration, the CSS allows all client requests to progress to the origin servers.

A transparent caching configuration:

- Reduces network congestion caused by HTTP traffic
- Increases network efficiency
- Decreases the time required to fulfill a client request by accessing locally stored information rather than obtaining the same information across the Internet

Figure 7-3 shows an example of a typical transparent cache configuration.

Figure 7-3 Transparent Cache Configuration Example



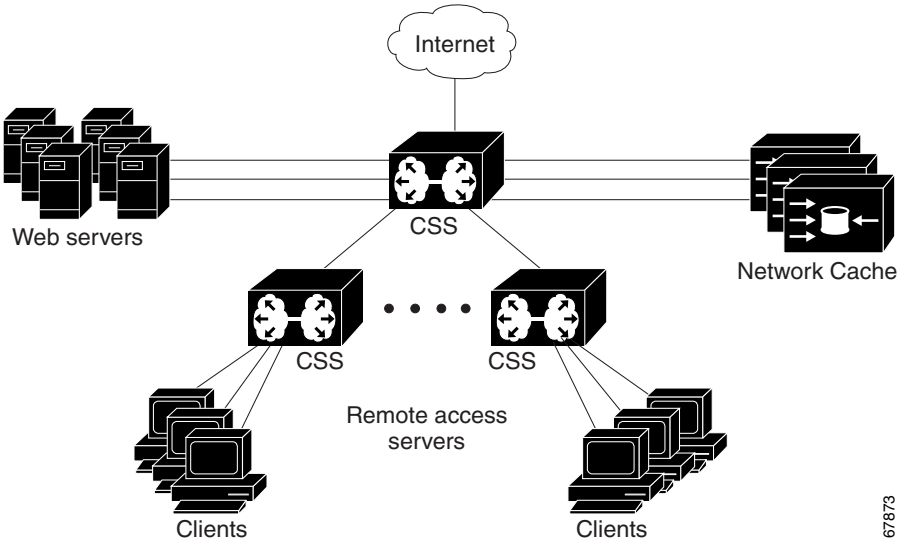
Using Cache Clustering

Multiple caches deployed at a single location is referred to as cache clustering. Cache clustering provides:

- Scalability
- Redundancy
- Transparency
- Simplified administration

Figure 7-4 shows an example of using Content Caching in a cache cluster configuration.

Figure 7-4 Cache Cluster Configuration Example



Caching Configuration Quick Start

Table 7-1 provides the steps to configure service *serv1* as a caching service. Each step includes the CLI command required to complete the task. Ensure that you have configured services, owners, and content rules prior to configuring CSS caching.

Note When using Content Caching, the keepalive type must be **ICMP** (default setting).

For a complete description of each caching command, see the sections following Table 7-1.

Table 7-1 Caching Configuration Quick Start

Task and Command Example
1. Specify a service type (**type local**, **type proxy-cache**, **type redirect**, **type transparent-cache**). Default is local. `(config-service[serv1])# type transparent-cache`

2. Create an Extension Qualifier List (EQL) where you specify which content types the CSS caches.
`(config)# eql graphics`
 	
`(config-eql[graphics])#`

3. Describe the EQL by entering a quoted text string with a maximum length of 63 characters. `(config-eql[graphics])# description "This EQL specifies cacheable graphic files"`

4. Specify the **extension** for content you want the CSS to cache. Enter a text string from 1 to 8 characters.
`(config-eql[graphics])# extension jpeg`
 	
Optionally, you may provide a description of the extension type. Enter a quoted text string with a maximum length of 64 characters.
`(config-eql[graphics])# extension gif "This is a graphics file"`
`(config-eql[graphics])# exit`
`(config)#`

Table 7-1 Caching Configuration Quick Start (continued)

Task and Command Example
5. Specify the EQL in a content rule to match all content requests with the desired extensions. `(config-owner-content[arrowpoint.com-rule1])# url "/*" eql graphics`
6. Configure the load balancing method for the cache content rule. The default is roundrobin. `(config-owner-content[arrowpoint.com-rule1])# balance domain`
7. Specify a failover type to define how the CSS handles content requests when a service fails (**bypass**, **next**). The default is linear. `(config-owner-content[arrowpoint.com-rule1])# failover bypass`
8. Display the EQL configuration. `(config-owner-content[arrowpoint.com-rule1])# show eql`
9. Display the content rule to show the cache configuration. `(config-owner-content[arrowpoint.com-rule1])# show rule`

Configuring Caching

Configure caching using content rules. When creating caching content rules, the additional configuration requirements involve:

- Specifying a service type that supports caching
- Specifying a failover type for the cache servers
- Configuring a load-balancing algorithm that supports caching
- Configure EQLs to identify file extensions that the CSS should direct to the cache services

Note If you are running the Inktomi® Traffic Server™ on a system that does not listen in promiscuous mode and want to bypass the Inktomi Adaptive Redirect module (that is, send traffic directly to port 8080 instead of port 80), specify the CSS service type as **type proxy-cache**. Configuring the CSS service type to **type proxy-cache** causes the CSS to perform full Network Address Translation (NAT) when directing traffic to the Traffic Server.

Specifying a Service Type

The CSS enables you to specify the following cache-specific service types using the **type** command. The default service type is local.

- **type nci-direct-return** - Specifies the service as NAT Channel indication for direct return. Use with reverse proxy cache and NAT peering.
- **type nci-info-only** - Specifies the service as NAT Channel indication for information only. Use with reverse proxy cache and NAT peering.
- **type proxy-cache** -Specifies the service as a proxy cache. This option bypasses content rules for requests coming *from* the cache server. In this case, bypassing content rules prevents a loop between the cache and the CSS.
- **type rep-cache** - Specifies the service as a replication cache.
- **type rep-cache-redir** - Specifies the service as a replication cache with redirect.
- **type transparent-cache** - Specifies the service as a transparent cache. No content rules are applied to requests from this service type. Bypassing content rules in this case prevents a loop between the cache and the CSS.

  For example, to specify service *serv1* as a proxy cache, enter:

  ```
  (config-service[serv1])# type proxy-cache
  ```

  The CSS recognizes and forwards the following HTTP methods directly to the destination server in a transparent caching environment. However, the CSS does not load balance these methods.

  - RFC-2068: OPTIONS, TRACE
  - RFC-2518: PROPFIND, PROPPATCH, MKCOL, MOVE, LOCK, UNLOCK, COPY, DELETE

Note To enable the CSS to redirect a request to a remote service when a request for content matches the rule, you must specify a URL for the content rule.

Specifying a Failover Type

To define how the CSS handles content requests when a cache service fails or is suspended, use the **failover** command. For the CSS to use this setting, ensure that you configure a keepalive for each service; that is, do not set keepalive type to none (default keepalive is ICMP). The CSS uses the keepalive settings to monitor the cache services to determine server health and availability. Refer to Chapter 1, Configuring Services for more information on the **keepalive** command.

By default, the CSS uses a linear failover method, which distributes the content requests to the failed service evenly among the remaining services.

Note If you remove a service (using the **remove service** command) the CSS rebalances the remaining services. The CSS does not use the failover setting.

This command supports the following options:

- **failover bypass** - Bypass all failed services and send the content request directly to the origin server. This option is used in a proxy or transparent cache environment when you want to bypass the failed cache and send the content request directly to the server that contains the content.
- **failover linear** (default) - Distribute the content request evenly between the remaining services.
- **failover next** - Send the content requests to the cache service next to the failed service. The CSS selects the service to redirect content requests to by referring to the order in which you configured the services.

For example, enter:

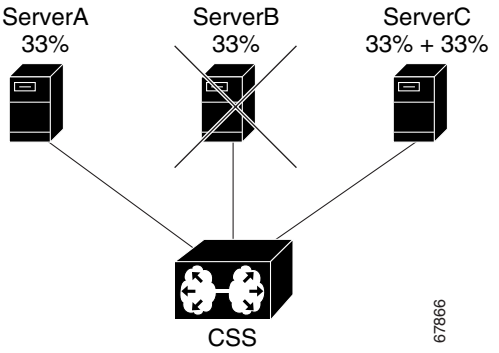
```
(config-owner-content[arrowpoint.com-rule1])# failover bypass
```

To restore the default failover method of linear, enter:

```
(config-owner-content[arrowpoint.com-rule1])# no failover
```

Figure 7-5 shows three cache services configured for failover **next**. If ServerB fails, the CSS sends ServerB content requests to ServerC, which was configured after ServerB in the content rule.

Figure 7-5 Cache Services Configured for Failover Next



As shown in Figure 7-6, if ServerC fails, the CSS sends ServerC content requests to ServerA because no other services were configured after ServerC.

Figure 7-6 Cache Services Configured for Failover Next

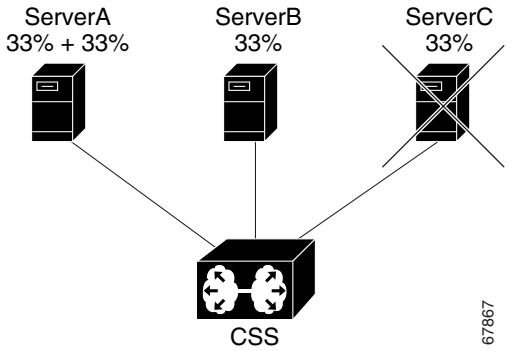


Figure 7-7 shows three cache services configured for **failover linear (the default)**. If you *suspend* ServerB or if it *fails*, the CSS does not rebalance the services. It evenly distributes ServerB cache workload between servers A and C.

Note that Figure 7-7 and Figure 7-8 use the alphabet to illustrate division balance.

Figure 7-7 Suspended or Failed Cache Service Configured for Failover Linear

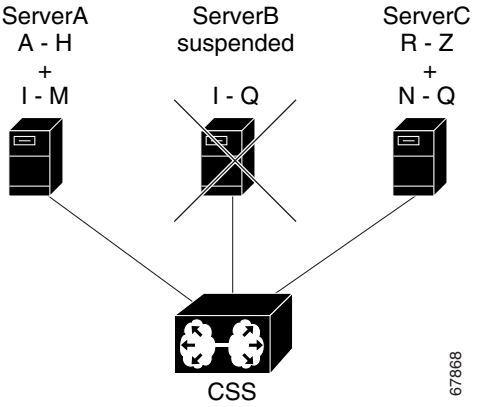
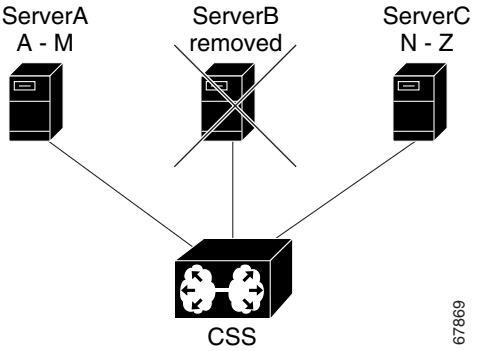


Figure 7-8 also shows three cache services configured for failover **linear**, but in this example, you *remove* ServerB using the **remove service** command from owner-content mode. Because the CSS does not apply the failover setting when you remove a service, it rebalances the remaining services.

Figure 7-8 Removing a Cache Service Configured for Failover Linear



Configuring Load Balancing

To specify the load-balancing algorithm for a content rule, use the **balance** command. This command is available in content configuration mode. The options are:

- **balance aca** - ArrowPoint Content Awareness load-balancing algorithm (refer to Chapter 1, Configuring Services, "Using ArrowPoint Content Awareness Based on Server Load and Weight"). ACA balances the traffic over the services based on load or on server weight and load.
- **balance destip** - Destination IP address division algorithm. The CSS directs all client requests with the same destination IP address to the same service. This option is typically used in a caching environment.
- **balance domain** - Domain name division algorithm. The CSS divides the alphabet evenly across the number of caches. It parses the host tag for the first four letters following the first dot and then uses these characters of the domain name to determine to which server it should forward the request. This option is typically used in a caching environment.
- **balance domainhash** - Internal CSS hash algorithm based on the domain string. The CSS parses the host tag and does an XOR hash across the entire host name. It then uses the XOR hash value to determine to which server to forward the request. This method guarantees that all requests with the same host tag will be sent to the same server in order to increase the probability of a cache hit. This option is typically used in a caching environment.

  Note If you are using the domainhash load-balancing method with proxy cache services, you may see duplicate sites across caches because the CSS balances on the first GET request in a persistent connection unless the subsequent GET request does not match a rule with the same proxy service specified. If you are concerned with duplicate hits across caches, reset persistence to remap and disable persistence on the rule. Issue the **(config) persistence reset remap** command globally and the **(config-owner-content) no persistent** command on the content rule.

- **balance leastconn** - Least connection algorithm. This balance method chooses a running service that has the least number of connections.

- **balance roundrobin** - Roundrobin algorithm (default). The CSS resolves the request by evenly distributing the load to resolve domain names among local and remote content domain sites.
- **balance srcip** - Source IP address division algorithm. The CSS directs all client requests coming from the same source IP address to the same service. This option is generally used in a caching configuration.
- **balance url** - URL division algorithm. The CSS divides the alphabet evenly across the number of caches. It then parses the URL for the first four characters located after the portion of the URL matched on by the rule. For example, if the URL in a content rule is configured for "/news/*", the CSS it will balance on the first four characters following "/news/". This option is typically used in a caching environment.
- **balance weightedrr** - Weighted roundrobin algorithm. The CSS uses roundrobin but weighs some services more heavily than others depending on the server's configured weight. All servers have a default weight of 1. To set a server weight, use the **add service weight** command in owner-content mode.
- **balance urlhash** - Internal CSS hash algorithm based on the URL string. The CSS parses the URL and performs an XOR hash across the URL. It then uses the XOR hash value to determine to which server to forward the request. This method guarantees that all requests for the same URL will be sent to the same server in order to increase the probability of a cache hit. This option is typically used in a caching environment.

Note A Layer 5 content rule supports the HTTP CONNECT, GET, HEAD, POST, PUSH, and PUT methods. The CSS recognizes and forwards the following HTTP methods directly to the destination server in a transparent caching environment. Note that the CSS does not load balance these HTTP methods. RFC-2068: OPTIONS, TRACE; RFC-2518: PROPFIND, PROPPATCH, MKCOL, MOVE, LOCK, UNLOCK, COPY, DELETE.

In a transparent caching environment (for example, no VIP address on a Layer 5 content rule), the CSS bypasses these HTTP methods, and they are forwarded to the destination server.

For example, to specify weightedrr load balancing, enter:

```
(config-owner-content[arrowpoint-rule1])# balance weightedrr
```

To revert the balance type to the default of roundrobin, enter:

```
(config-owner-content[arrowpoint-rule1])# no balance
```

Configuring a Double-Wildcard Caching Content Rule

When you want to optimize L3 and L4 TCP/IP traffic, configure a content rule for transparent caching without specifying the VIP address and port number. This configuration may be particularly useful in a wireless environment where there is intelligence built into the backend server.

If all other matching criteria in the content rule are met by the client request, a request with any VIP or port will match the rule. This is called a double-wildcard caching rule. You still need to specify the protocol in the rule. Tyically, use this type of rule when you are load-balancing services of **type transparent-cache**. However, you can configure this type of rule with other service types as well.

Note If you have a configuration that requires a double-wildcard rule, be aware that the client request will match on this rule when the client attempts to connect directly to a server IP address.

Enabling Content Requests to Bypass Caches

This section covers:

- Using the param-bypass Command
- Using the cache-bypass Command
- Using the bypass-hosttag Command

Using the param-bypass Command

Use the **param-bypass** command to enable content requests to bypass transparent caches when the CSS detects special terminators in the requests. These terminators include "#" and "?" which indicate that the content is dependent on the arguments that follow the terminators. Because the content returned by the server is dependent on the content request itself, the returned content is not cacheable.

This command contains the following options:

- **param-bypass disable** (default) - Content requests with special terminators do not bypass transparent caches.
- **param-bypass enable** - Content requests with special terminators bypass transparent caches and are forwarded to the origin server.

For example, to enable the **param-bypass** command, enter:

```
(config-owner-content[arrowpoint-rule1])# param-bypass enable
```

Using the cache-bypass Command

By default, a CSS does not apply content rules to requests from a proxy or transparent-cache type service going to the origin server when the cache does not contain the requested content. Use the **no cache-bypass** command to allow the application of content rules to requests originating from a proxy or transparent cache. Use the **cache-bypass** command to restore the default behavior of the CSS after you have issued the **no cache-bypass** command.

For example, to allow the CSS to apply content rules to requests from a proxy or transparent-cache type service, enter:

```
(config-service[serv1])# no cache-bypass
```

To restore the CSS default behavior after issuing the **no cache-bypass** command, enter:

```
(config-service[serv1])# cache-bypass
```

Using the bypass-hosttag Command

Use the **bypass-hosttag** command to allow a CSS configured as a Client Side Accelerator (CSA) to bypass a cache farm and establish a connection with the origin server to retrieve non-cacheable content. The domain name from the host tag field is used to look up the origin IP address on the CSA.

Note Use the **bypass-hosttag** command only with a CSS operating in a CSA environment. For details on CSA, refer to the *Cisco Content Services Switch Advanced Configuration Guide*.

For example, enter:

```
(config-service[serv1])# bypass-hosttag
```

To disable bypassing cache for non-cacheable content, enter:

```
(config-service[serv1])# no bypass-hosttag
```

Configuring Network Address Translation for Transparent Caches

Use the **transparent-hosttag** command to enable destination Network Address Translation (NAT) for the transparent cache service type. This command NATs the destination address of the client's packet (forwarded by the CSS to the cache) to the origin server IP address for the requested domain. Using this command ensures that the cache always has the current origin server IP address based on periodic DNS lookups that the CSS performs for all accelerated domains.

The alternative is to manually configure all origin server IP addresses on the cache, which may or may not support static configuration. Also, statically configured IP addresses can become obsolete if the origin server IP address changes. For caches that suppport DNS resolution and use the DNS response to fetch content or that support configuration of origin server IP addresses, **transparent-hosstag** is not required, but recommended.

Note You can use the **transparent-hosttag** command only with a CSS operating in a Client Side Accelerator (CSA) environment. For details on CSA, refer to the *Cisco Content Service Switch Advanced Configuration Guide*.

For example, enter:

```
(config-service[serv1])# transparent-hosttag
```

To disable destination NATing for the transparent cache service type, enter:

```
(config-service[serv1])# no transparent-hosttag
```

Configuring Network Address Translation Peering

Network Address Translation (NAT) peering allows clients to connect to remote Web sites through CSSs and have the return traffic use the shortest network path back to the client. The forward path from the client to the server is through TCP connections between two CSSs, but the reverse path from the server to the client may take the shortest network route rather than traversing back through the CSSs.

Note NAT peering is part of the CSS Enhanced feature set.

NAT peering allows the CSS to:

- Forward client connections to a remote CSS
- Perform the final translation at the remote CSS, which allows return traffic packets to flow to the client through any network path
- Preserve the client IP address when forwarding traffic to the origin server

Note Adaptive Session Redundancy (ASR) does not support NAT Peering. For details on ASR, refer to the *Cisco Content Services Switch Advanced Configuration Guide*, Chapter 6, Configuring VIP and Virtual IP Interface Redundancy.

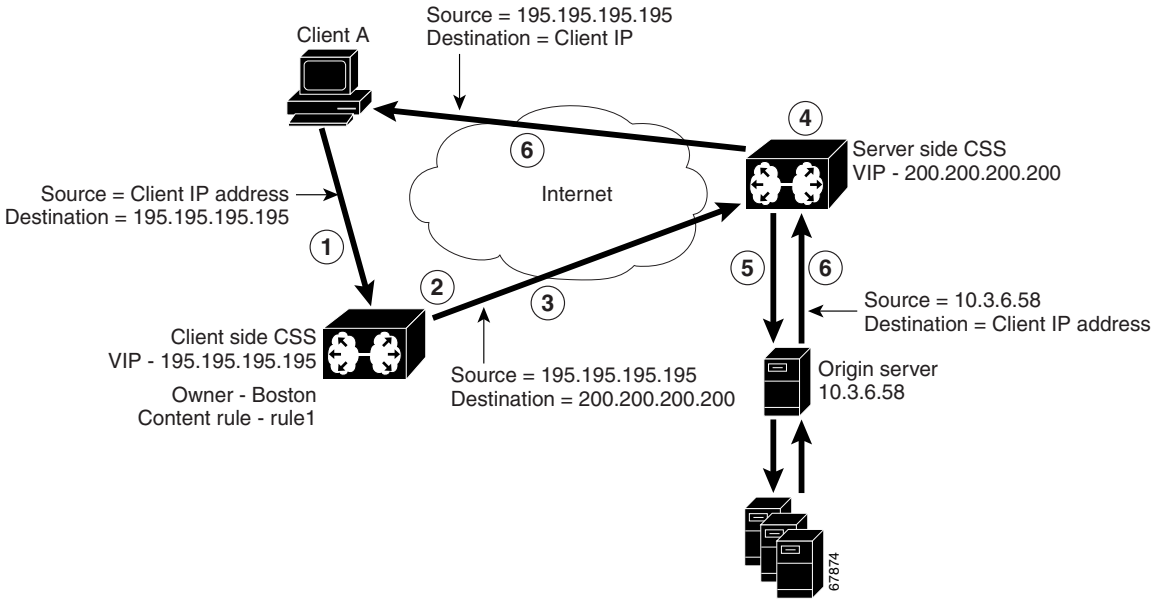
To perform NAT transformations on a TCP flow, the client-side CSS forwards traffic to the server-side CSS through a NAT channel. This channel uses a special TCP option called the NAT Channel Indication (NCI) option. This option indicates to the server-side CSS that NAT parameters are in use, and contains the original source and destination IP addresses, and TCP port numbers. This option also has a spoof bit to indicate that part of the flow has been spoofed and the rest of the forward path must be established before the destination CSS can use the information in the packet to perform the NAT transformations for the reverse path.

Note Spoofing occurs when a CSS requires information from the HTTP request, (such as host tag, file name, file extension) in order to make a load balancing decision.

The server-side CSS preserves the client address and port. This allows the origin server to maintain statistics based on the original traffic source addressing data, and allows the return path to be independent of the forwarding path.

Figure 7-9 shows an example of NAT peering. The steps that follow describe this example.

Figure 7-9 NAT Peering Configuration Example



1. Client A sends a content request for */bostonInfo.html* from the client-side CSS (CSS1, VIP 195.195.195.195).
2. The client-side CSS matches the request to its content rule, which specifies a service located on the server-side CSS (CSS2, VIP2 200.200.200.200). The server-side CSS service is configured for service type **nci-direct-return**. This service type informs the client-side CSS to include the NCI option in the TCP packet sent to server-side CSS. If a Layer 5 rule is matched, the spoof bit in the NCI option is set.
3. The client-side CSS sends the TCP packet to the server-side CSS. Source address group mapping maps the Client A source address and port to those from the client-side CSS. The TCP packet contains the client-side CSS source information,the server-side CSS destination information, and the original source and destination information from Client A.
4. The server-side CSS determines whether or not the spoof bit has been set in the packet. If the bit is set, the CSS stores the NAT information until the connection is spoofed. The server-side CSS sets up the forward and return paths. The server-side CSS then matches the request from the client-side CSS on a content rule.

Note The server-side CSS (in Figure 7-9) would use the NCI option in a packet if the VIP rule is directed at a local, proxy-cache, or transparent cache service.

5. The server-side CSS sends the request to the origin server with the destination IP address translated to the origin server IP address and the source IP address translated to the client IP address.
6. The origin server responds directly back to Client A. As the packet flows through the server-side CSS, that CSS translates the source IP address to the CSS1 VIP. The destination IP address is the client IP address.

Configuring NAT Peering

All NAT peering configuration occurs on the client-side CSS. During the configuration consider the following:

- When you configure the NCI service as **nci-direct-return**, the service must be directed to the VIP on the server-side CSS to indicate an endpoint for the connection. The server-side CSS always uses the **nci-direct-return** option to modify the source address and port that the server sees. When the nci-direct-return service is used on the client-side, the return path is modified to directly return to the client.
- When you are specifying an NCI service type, you must specify:
  - **type nci-direct-return** to represent a VIP on another CSS
  - **type nci-info-only** for any Web server

Table 7-2 describes the steps necessary to configure NAT peering using command examples based on the configuration in Figure 7-9. Because NAT peering applies to Layer 3 as well as Layer 5 rules, the port, protocol, and URL rule examples shown in Table 7-2 are optional.

Table 7-2 NAT Configuration Quick Start

Task and Command Example
1. On the client-side CSS (CSS1), create content rules to configure the server-side CSS (CSS2) as a service.
a. Create service CSS2. `CSS1 (config)# service CSS2`
b. Configure CSS2 VIP as the service IP address. `CSS1 (config-service[CSS2])# ip address 200.200.200.200`
c. Configure CSS2 as a service type nci-direct-return. `CSS1 (config-service[CSS2])# type nci-direct-return`
d. Activate the content rule. `CSS1 (config-service[CSS2])# active`

Table 7-2 NAT Configuration Quick Start (continued)

Task and Command Example

2. On the client-side CSS (CSS1), create content rules with the criteria required for the client-side CSS (CSS1) to forward traffic to the server-side CSS (CSS2).
a. Create an owner.
`CSS1 (config)# owner boston.com`
b. Name the content rule and assign it the owner.
`CSS1 (config-owner[boston.com])# content rule1`
c. Configure the CSS1 VIP.
`CSS1 (config-owner-content[boston.com-rule1])# vip 195.195.195.195`
d. Configure port and protocol.
`CSS1 (config-owner-content[boston.com-rule1])# port 80`
`CSS1 (config-owner-content[boston.com-rule1])# protocol tcp`
e. Define the URL.
`CSS1 (config-owner-content[boston.com-rule1])# url "//bostoninfo.html/"`
f. Add CSS2 as the service.
`CSS1 (config-owner-content[boston.com-rule1])# service CSS2`
g. Activate the rule.
`CSS1 (config-owner-content[boston.com-rule1])# active`

Table 7-2 NAT Configuration Quick Start (continued)

Task and Command Example

3. On the client-side CSS (CSS1), create a source group for the client traffic. CSS1 will translate the Client A IP address to the IP address defined in the source group. To configure a source group:
a. Create the source group.
`CSS1 (config)# group boston`
`CSS1 (config-group[boston])#`
b. Define the CSS1 VIP as the IP address into which the Client A IP address will be translated.
`CSS1 (config-group[boston])# vip 195.195.195.195`
c. Activate the source group.
`CSS1 (config-group[boston])# active`

4. On the client-side CSS (CSS1), create an Access Control List (ACL) clause to specify which source IP addresses use the source group. Note that clause 20 is a required clause that permits all other traffic. Without clause 20, all traffic not defined in clause 10 is denied.
`CSS1 (config)# acl 1`
`CSS1 (config-acl[1])# clause 10 permit tcp any destination content boston.com/rule1 sourcegroup boston`
`CSS1 (config-acl[1])# clause 20 permit any any destination any apply circuit-(VLAN1)`

5. On the server-side CSS (CSS2), configure the origin server connected to CSS2.
a. Create origin server *serv1*.
`CSS2 (config)# service serv1`
b. Configure an IP address for serv1.
`CSS2 (config-service[serv1])# ip address 10.3.6.58`
c. Activate the server.
`CSS2 (config-service[serv1])# active`

Table 7-2 NAT Configuration Quick Start (continued)

Task and Command Example

6. On the server-side CSS (CSS2), configure content rules with the criteria required to forward content requests to *serv1*.
a. Create an owner.
`CSS2 (config)# owner boston.com`
b. Name the content rule and assign it the owner.
`CSS2 (config-owner[boston.com])# content rule1`
c. Configure the CSS2 VIP.
`CSS2 (config-owner-content[boston.com-rule1])# vip 200.200.200.200`
d. Configure port and protocol.
`CSS2 (config-owner-content[boston.com-rule1])# port 80`
`CSS2 (config-owner-content[boston.com-rule1])# protocol tcp`
e. Add *serv1* as the service.
`CSS2 (config-owner-content[boston.com-rule1])# service serv1`
f. Define a URL.
`CSS2 (config-owner-content[boston.com-rule1])# url "/*"`
g. Activate the rule.
`CSS2 (config-owner-content[boston.com-rule1])# active`

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X