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:

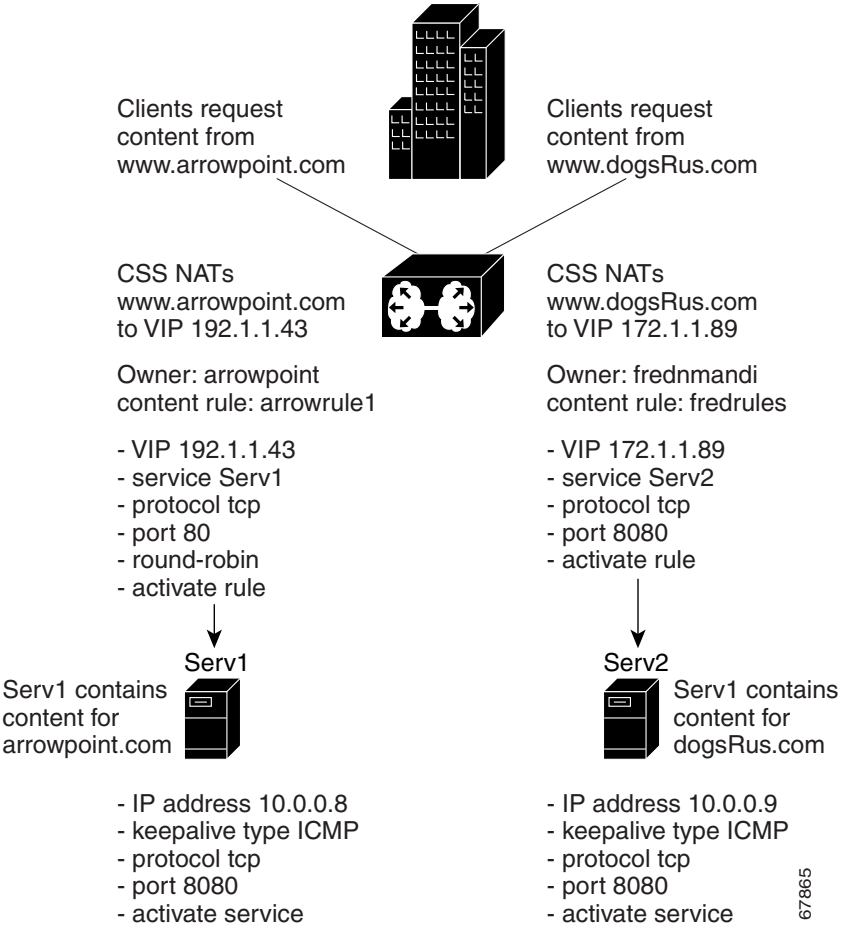
  - Where the content physically resides, whether local or remote
  - Where to direct the request for content (which service or services)
  - Which load balancing method to use

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.

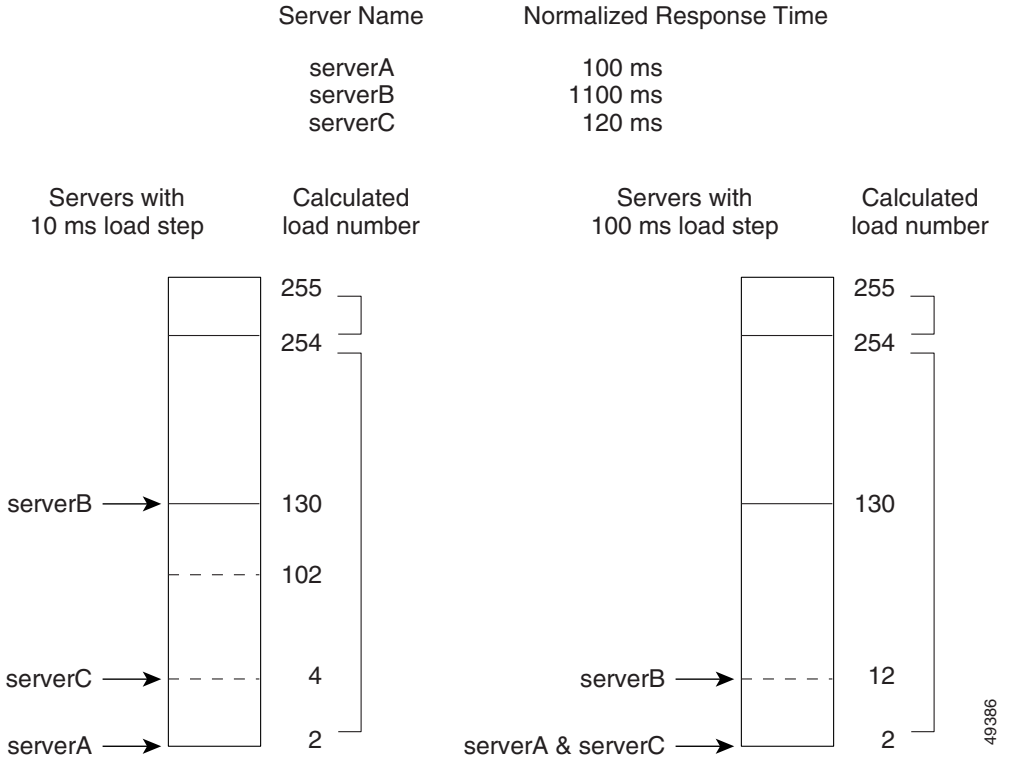
A service becomes ineligible to receive flows when its load number exceeds the configured load threshold. The CSS uses the configured ageout timer value to return the service to the eligible state.

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.

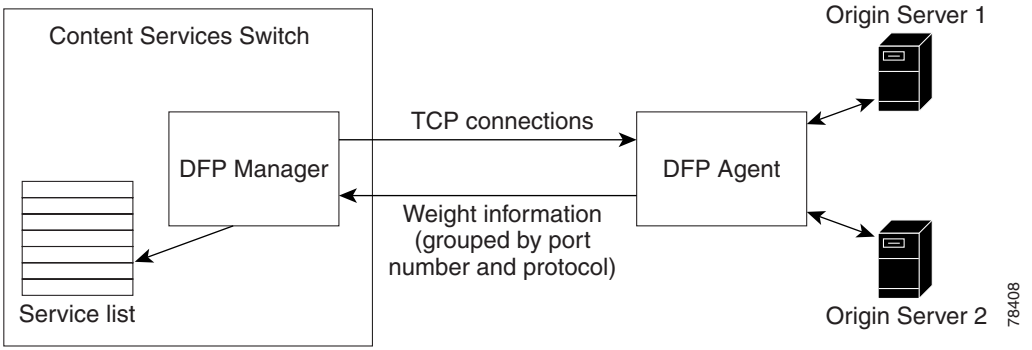
If a DFP agent is responsible for collecting information from multiple servers, the servers are grouped by their port number and protocol-type, and a separate load vector is required for each grouping. A DFP agent can report weights for a maximum of 128 servers in a single weight report. Upon receiving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