Configuring Sticky Parameters for Content Rules

This chapter describes how to configure sticky parameters for content rules. The information in this chapter applies to all CSS models, except where noted. This chapter contains the following major sections:

Sticky Overview

During a session, the CSS maintains an association between a client and a server. This association is referred to as *stickiness*. Stickiness enables transactions over the Web when the client must remain on the same server for the entire session. Depending on the content rule, the CSS "sticks" a client to an appropriate server after the CSS has determined which load-balancing method to use.

If the CSS determines that a client is already stuck to a particular service, then the CSS places the client request on that service, regardless of the load balancing criteria specified by the matched content rule. If the CSS determines that the client is not stuck to a particular service, it applies normal load balancing to the content request.

Client *cookies* uniquely identify clients to the services providing content. A cookie is a small data structure used by a server to deliver data to a Web client and request that the client store the information. In certain applications, the client returns the information to the server to maintain the state between the client and the server.

When the CSS examines a request for content and determines through content rule matching that the content is sticky, it examines any cookie or URL present in the content request. The CSS uses this information to place the content request on the appropriate server.

The total number of entries in the CSS sticky table depends on the size of the CPU memory.

- The CSS 11501 supports a 128K sticky table (288 MB of CPU memory).
- The CSS 11503 and 11506 supports either a 128K or 32K sticky table, depending on whether the System Control module (SCM) has 288 MB or 144 MB of memory. With 288 MB of memory, the CSS supports a 128K sticky table. With 144 MB of memory, the CSS supports a 32K sticky table.

The size of the sticky table means that once 128K or 32K simultaneous users are on the site, the table wraps and the first users become “unstuck.”

The following sections describe stickiness and its uses:

Why Use Stickiness?

When customers visit an e-commerce site, they usually start out by browsing the site, the Internet equivalent of window shopping. Depending on the application, the site may require that the customer become “stuck” to one server once the connection is established, or the application may not require this until the customer starts to build a shopping cart.

In either case, once the customer adds items to the shopping cart, it is important that all of the customer’s requests get directed to the same server so that all the items are contained in one shopping cart on one server. An instance of a customer's shopping cart is typically local to a particular Web server and is not duplicated across multiple servers.

E-commerce applications are not the only types of applications that require stickiness. Any Web application that maintains client information may require stickiness, such as banking applications or online trading.

Because the application must distinguish each user or group of users, the CSS needs to determine how a particular user is stuck to a specific Web server. The CSS can use a variety of methods, including:

- Source IP address
- Source IP address and destination port
- String found in a cookie or a URL
- SSL session ID

The e-commerce application itself dictates which of these methods is appropriate for a particular e-commerce vendor.

Using Layer 3 Sticky

If an application requires that a user be stuck for the entire session, use Layer 3 sticky, which sticks a user to a server based on the user's IP address. The total number of entries in the sticky table depends on the size of the CPU memory (128K sticky table with 288 MB of CPU memory or a 32K sticky table with 144 MB of CPU memory).

If the volume of your site is such that you will have more than 128K or 32K users at a time, or if a large percentage of your customers come to you through a mega-proxy, then consider using either a different sticky method (for example, the advanced-balance method **cookies**, **cookieurl**, or **url**), or increasing your sticky mask.

Note If you use the **sticky-inact-timeout** command to specify the inactivity timeout period on a sticky connection, when the sticky table becomes full and none of the entries have expired from the sticky table, the CSS rejects subsequent needed sticky requests.

The default sticky mask is 255.255.255.255, which means that each entry in the sticky table is an individual IP address. Some mega-proxies allow one user to use several different IP addresses in a range of addresses over the life of one session. This use of multiple addresses for one session can cause some of the TCP connections to get stuck to one server, and other TCP connections to a different server for the same transaction. The result is possibly losing some items from the shopping cart. To avoid this problem, use one of the more advanced methods of sticking. If you cannot, we recommend using a sticky mask of 255.255.240.0.

Using Layer 4 Sticky

Layer 4 sticky functions identically to Layer 3 sticky, except that it sticks based on a combination of source IP address, protocol, and destination port. Layer 4 sticky also uses a sticky table and has the same limitations as Layer 3 sticky.

If the CSS sees the same IP address with two different destination ports, it will use two entries. You can also apply sticky mask to Layer 4 sticky.

If you are concerned about whether your site can handle all of the simultaneous sessions, then consider using the Layer 5 advanced-balanced methods of **arrowpoint-cookie**, **cookie**, **cookieurl**, or **url**.

Using Layer 5 Sticky

Layer 5 sticky uses a combination of destination IP address, protocol, port, and URL that may or may not contain an HTTP cookie or a domain name. Layer 5 sticky can function based on a sticky string in a cookie or URL, or based on an SSL version 3 session ID. The advanced-balanced methods such as **arrowpoint-cookie**, **cookie**, **cookieurl**, and **url** do not use a sticky table to keep track of IDs. The **advanced-balance ssl** method for SSL sticky does use a sticky table.

Note If you use the **sticky-inact-timeout** command to specify the inactivity timeout period on a sticky connection, when the sticky table becomes full and none of the entries have expired from the sticky table, the CSS rejects subsequent new sticky requests. If the **sticky-inact-timeout** command is specified for a Layer 5 content rule using SSL sticky, the SSL sessions continue even if the sticky table is full but the CSS does not maintain stickiness on the new sessions.

Configuring Sticky on the CSS

Configuring sticky on the CSS requires you to:

- Determine the sticky method you want to use according to the requirements of the site (for example, Layer 3, Layer 4, or one of the string methods)
- Configure a failover method

If you use advanced-balance methods **cookies**, **url**, or **cookieurl**, you must also:

- Determine whether you want to use an exact string match or a hash, and then configure that function.
- Determine how you want to delimit (configure) the string.

To configure sticky on the CSS:

1. Configure the sticky method using the **advanced-balance** command and its options. The **advanced-balance** command options are described in "Specifying an Advanced Load-Balancing Method for Sticky Content" later in this chapter.
    - To configure Layer 3 sticky, use **advanced-balance sticky-srcip** in the content rule. If necessary, change the sticky mask from the default of 255.255.255.255.

- To configure Layer 4 sticky, use **advanced-balance sticky-srcip-dstport** in the content rule. If necessary, change the sticky mask from the default of 255.255.255.255.
- To configure sticky cookies, use **advanced-balance cookies** in the content rule.
- To configure sticky URL, use **advanced-balance url** in the content rule.
- To configure sticky cookies with URLs, use **advanced-balance cookieurl** in the content rule.

2. Configure a failover method. Use the **sticky-serverdown-failover** command to define what will happen if a sticky string is found but the associated service has failed or is suspended. The sticky failover default is for the CSS to use the configured load-balancing method. The **sticky-serverdown-failover** options are described in the "Configuring Sticky Serverdown Failover" section later in this chapter.

   If you configured an advanced-balance method of **sticky-srcip** or **sticky-srcip-dstport**, no further steps are required.

   If you configured the advanced-balance methods **cookies**, **url**, or **cookieurl**, complete Steps 3 and 4.

3. If you are using **advanced-balance cookies**, **url**, or **cookieurl**, determine whether you want to use an exact string match or a hash.

   To use an exact string match:

   a. Enter the **string operation match-service-cookie** command (this is the default for the **string operation** command).

   b. For each service configuration, use the service mode **string** command to configure the unique string that you want to use for matching each server.

      For example, you have three servers and you want the string matching to be serverid111 for service1, serverid112 for service2, and serverid113 for service3. Configure the Web server applications to use these strings when they set cookies or pass parameters.

   For information on the **string operation match-service-cookie** command, see the "Specifying a String Operation" section later in this chapter.

   To use the hash algorithm:

   a. Enter the **string operation** command in the content rule.

b. Select an option (**hash-a**, **hash-crc32**, or **hash-xor**) depending on the hash method you wish to use. Hashing requires that each server can accept cookies set by all other servers.

   We recommend using either **hash-xor** or **hash-crc32,** depending on your string possibilities. If the strings are completely dissimilar, use **hash-xor**. If the strings are similar, use **hash-crc32**. For example, if your string values are abc1, abc2, and abc3, the **hash-xor** method cannot provide you with enough variance in the hash values (that is, abc1 and abc2 may end up on the same server because they may hash to the same value).

   For information on the string operation hash options, see the "Specifying a String Operation" section later in this chapter.

4. If you are using **advanced-balance cookies**, **url**, or **cookieurl**, determine how you want to delimit (configure) the string. Use the following owner-content **string** commands to delimit the string:

   - **string range** - Defining the string range enables you to limit the size of the search. By default the CSS searches the first 100 bytes of the cookie, URL, or parameters in the URL depending on the method. If you know where in the cookie or URL the string is likely to appear, define the string range accordingly. The range is from 1 to 2000. The default is 1 to 100. The string range options are described in the "Configuring String Range" section later in this chapter.
   - **string eos-char** - A maximum of 3 ASCII characters that delimit the end of the string within the string range. Use this option when the string length varies. Note that **string process-length** overrides **string eos-char**. If you do not configure either option, the CSS uses a maximum of 100 bytes for the delimiter.
   - **string prefix** - The CSS uses the string prefix (maximum of 30 characters) to locate the string within the string range of the cookie or URL. If the string prefix is specified, but not found, the CSS uses the normal balance method.
   - **string process-length** - Specifies the number of bytes within the string range after the end of the prefix plus the skip-length that is used to determine the string. Use this option when the string length is fixed.
   - **string skip-length** - Specifies the number of bytes to skip after the end of the prefix within the string range. The range is 0 to 64.

For example, if you are using ipaddr=192.168.3.6&, then use the **string prefix** “ipaddr=” and the **string eos-char** “&” because the IP addresses vary in length.

For example, if you are using server ID=server111, then use the **string prefix** “server ID=” and a **string process-length** of 8 because the string length does not vary in length.

Table 11-1 describes sticky rules and how they apply to content rules.

Table 11-1 Applying Sticky Rules to Content Rules

Rule Type	Sticky Configuration	Stickiness Based on...
Layer 3 content rule	**advanced-balance sticky-srcip**	Source IP address using a sticky mask.
Layer 4 content rule	**advanced-balance sticky-srcip-dstport**	Source IP address and destination port using a sticky mask.
Layer 5 content rule not using a sticky string	**advanced-balance sticky-srcip-dstport**	Source IP address and destination port using a sticky mask.
Layer 5 content rule using a sticky string	**advanced-balance cookies** or **advanced-balance cookieurl**	Searching for a sticky string in the cookie or URL. If the CSS does not find the sticky string in the cookie or URL, the CSS load-balances each request among the available servers.
Layer 5 content rule with SSL	**advanced-balance ssl**	SSL v3 session ID. If no session ID is present, the CSS uses the source IP address and destination port to maintain stickiness.

Note In some environments, URL, cookie strings, or HTTP header information can span over multiple packets. In these environments, the CSS can parse multiple packets for Layer 5 information before making load-balancing decisions. Through the global configuration mode **spanning-packets** command, the CSS can parse up to 20 packets; the default is 6. The CSS makes the load-balancing decision as soon as it finds a match and does not require parsing of all of the configured number of spanned packets. Because parsing multiple packets does impose a longer delay in connection, performance can be impacted by longer strings that span multiple packets. For information on using the **spanning-packets** command, see Chapter 10, Configuring Content Rules.

Specifying an Advanced Load-Balancing Method for Sticky Content

A content rule is "sticky" when additional sessions from the same user or client are sent to the same service as the first connection, overriding normal load balancing. By default, the advanced balancing method is disabled.

Use the **advanced-balance** command to specify an advanced load-balancing method for a content rule that includes stickiness. The **advanced-balance** command options **cookies**, **cookieurl**, and **url** use strings for sticking clients to servers. These options are beneficial when the sticky table limit is too small for your application requirements because the string methods do not use the sticky table.

Note We strongly recommend that you configure an **advanced-balance** method content rule that requires the TCP protocol for Layer 5 (L5) spoofing with a default URL string, such as **url "/*"**. The addition of the URL string forces the content rule to become an L5 rule and ensures L5 load balancing or stickiness. If you do not configure a default URL string, unexpected results can occur.

For example, if you configure an L3 content rule with an L5 advanced-balance method, L5 stickiness will not work as expected.

```
content testing
vip address 192.168.128.131
add service s1
advanced-balance arrowpoint-cookie
active
```

The **advanced-balance arrowpoint-cookie** method causes the CSS to spoof the connection, however, the CSS still marks it as an L3 rule. Thus, the CSS does not insert the generated cookie and the rule defaults to L3 stickiness (sticky-srcip). You must configure a URL like **url "/*"** to promote this rule to L5, ensuring that L5 stickiness works as expected.

The syntax and options for this content mode command are:

- **advanced-balance arrowpoint-cookie** - Enables the content rule to stick a client to a server based on the unique service identifier information of the selected server in the arrowpoint cookie. Configure the service identifier by using the **(config-service) string** command. For information on configuring the arrowpoint cookie, see the "Configuring an Arrowpoint Cookie" section later in this chapter. You can use this option with any Layer 5 content rule.

  Note If you are using the **arrowpoint-cookie** option of the **advanced-balance** command, do not configure string match criteria or use the **sticky-no-cookie-found-action** or **sticky-serverdown-failover** commands.

  Note When you configure the **arrowpoint-cookie expiration** command and the **advanced-balance arrowpoint-cookie** command, the CSS CPU may spike and the CSS may experience a degradation in its performance.

- **advanced-balance cookies** - Enables the content rule to stick a client to a server based on the configured string found in the HTTP cookie header. You must specify a port in the content rule to use this option. The CSS then spoofs the connection. A content rule with a sticky configuration set to **advanced-balance cookies** requires all clients to enable cookies on their browser.

  When a client makes an initial request, they do not have a cookie. But once they go to a server that is capable of setting cookies, they receive the cookie from the server. Each subsequent request contains the cookie until the cookie expires. A string in a cookie can be used to stick a client to a server. The service mode **string** command enables you to specify where the CSS should locate the string within the cookie.

The CSS processes the cookie using:

- An exact match that you set up when you configure the services.
- Data for a hash algorithm. For more information, see the "Comparing Hash Method with Match Method" section later in this chapter.

- **advanced-balance cookieurl** - Same as the **advanced-balance cookies** command, but if the CSS cannot find the cookie header in the HTTP packet, this type fails over to look up the URL extensions (that is, the portion after the "?" in the URL) based on the same string criteria. You must specify a port in the content rule to use this option. The CSS then spoofs the connection.

  This option is useful if a Microsoft IIS web server is used with Cookie Munger, which dynamically places the session state information in the cookie header or URL extension, depending on whether the client can accept cookies.

  Some client applications do not accept cookies. When a site depends upon the information in the cookie, administrators sometimes modify the server application so that it appends the cookie data to the parameters section of the URL. The parameters typically follow a "?" at the end of the main data section of the URL.

  The **advanced-balance cookieurl** command sticks a client to a server based on locating the configured string in the:

  - Cookie, if a cookie exists
  - Parameters section of the URL, if no cookie exists

  The string can either be an exact match or be hashed.

- **advanced-balance none** - Disables the advanced-balancing method for a content rule (default).
- **advanced-balance sticky-srcip** - Enables the content rule to stick a client to a server based on the client IP address, also known as Layer 3 stickiness. You can use this option with Layer 3, Layer 4, or Layer 5 content rules.
- **advanced-balance sticky-srcip-dstport** - Enables the content rule to stick a client to a server based on both the client IP address and the server destination port number, also known as Layer 4 stickiness. You can use this option with Layer 4 or Layer 5 content rules.

- **advanced-balance sip-call-id** - Enables the content rule to stick a client to a server based on Session Initiation Protocol (SIP) Call-ID. The application type must be **sip** for the content rule and the protocol must be UDP. For more information about SIP, see the "Configuring Session Initiation Protocol Load Balancing" section.
- **advanced-balance ssl** - Enables the content rule to stick the client to the server based on the Secure Socket Layer (SSL) version 3 session ID assigned by the server. The application type must be SSL for the content rule. You must specify a port in the content rule to use this option. The CSS then spoofs the connection.

  Sites where encryption is required for security purposes often use SSL. SSL contains session IDs, and the CSS can use these session IDs to stick the client to a server. For the CSS to successfully provide SSL stickiness, the application must be using SSL version 3 session IDs. Sticky SSL uses the sticky table. If you are concerned about the number of concurrent sessions, and not concerned about security, you should consider using the **cookies**, **cookieurl**, or **url** options.

  Note Use the **ssl-l4-fallback disable** command when you want to disable the CSS from inserting the Layer 4 hash value, which is based on the source IP address and destination address pair, into the sticky table. This may be necessary in a lab environment when testing SSL with a small number of clients and servers, where some retransmissions might occur. In this case, you would not want to use the Layer 4 hash value because it will skew the test results. See the "Configuring SSL-Layer 4 Fallback" section later in this chapter for details.

  Do not issue the **ssl-l4-fallback disable** command if SSL version 2 is in use on the network.

- **advanced-balance url** - Enables the content rule to stick a client to a server based on a configured string found in the URL of the HTTP request. You must specify a port in the content rule to use this option. The CSS then spoofs the connection.

  The **advanced-balance url** command is similar to the **advanced-balance cookies** command. It can use either an exact match method or a hash algorithm. The string can exist anywhere in the URL.

- **advanced-balance wap-msisdn** - Enables a Layer 5 content rule to stick a client to a server based on the MSISDN header field in an HTTP request. MSISDN is the header field for wireless clients using the Wireless Application Protocol (WAP). The MSISDN field value can contain the client's telephone number or user ID, which uniquely identifies the client. This command is especially useful for clients using e-commerce applications.

Note We recommend that you configure **advanced-balance wap-msisdn** only on a Layer 5 content rule (a rule configured with a URL statement).

If the MSISDN header is present in an HTTP request, the CSS generates a hash value (key) based on the value in the MSISDN header field. The CSS uses the key to look up an entry in the sticky table. If an entry exists in the sticky table, the CSS sends the client to the sticky server indicated by the table entry.

If an entry does not exist in the sticky table, the CSS:

a. Generates a new entry in the sticky table (similar to Layer 3, Layer 4, and SSL sticky)

b. Load balances the request to a server

c. Stores the selected server and the key (hashed value of the MSISDN header) in the sticky entry

The CSS looks up the same table entry and sends the client to the same server for subsequent requests from the same client.

If the MSISDN header field is not present in an HTTP request, the CSS load-balances the client request based on the configured load-balancing method. The default load-balancing method is roundrobin.

In the following example, TCP port 80 traffic destined for 192.168.128.151 is stuck to either server1 or server2 based on the contents of the MSISDN HTTP header field.

```
owner arrowpoint
   content ruleWapSticky
     vip address 192.168.128.151
     protocol tcp
     port 80
     url "/*"
     add service server1
```

```
add service server2
advanced-balance wap-msisdn
active
```

For example, to specify **advanced-balance wap-msisdn** for content rule *rule1*, enter:

```
(config-owner-content[arrowpoint-rule1])# advanced-balance
wap-msisdn
```

Note You can use the **advanced-balance wap-msisdn** command alone or with the MSISDN header field type. For a configuration example using both, see the "Configuring Wireless Users for E-Commerce Applications" section later in this chapter.

To disable the advanced load-balancing method, enter:

```
(config-owner-content[arrowpoint-rule1])# advanced-balance none
```

Configuring SSL-Layer 4 Fallback

Insertion of the Layer 4 hash value into the sticky table occurs when more than three frames are transmitted in either direction (client-to-server, server-to-client) or if SSL version 2 is in use on the network. If either condition occurs, the CSS inserts the Layer 4 hash value into the sticky table, overriding the further use of the SSL version 3 session ID. Use the **ssl-l4-fallback disable** command when you want to prevent the CSS from inserting the Layer 4 hash value, based on the source IP address and destination address pair, into the sticky table (the default CSS operation).

The **ssl-l4-fallback** command is applicable only when the **advanced-balance ssl** method is specified for a content rule, which forces the content rule to stick to a server based on SSL version 3 session ID. The use of the **ssl-l4-fallback** command may be necessary in a lab environment when testing SSL with a small number of clients and servers, where some retransmissions might occur. In this case, you would not want to use the Layer 4 hash value because it will skew the test results.

Note The **ssl-l4-fallback** command is a global configuration mode command and affects all contents rules using the **advanced-balance ssl** method.

The options for this global configuration mode command include:

- **ssl-l4-fallback enable** - The CSS inserts the Layer 4 hash value into the sticky table (default setting).
- **ssl-l4-fallback disable** - The CSS does not insert the Layer 4 hash value into the sticky table and continues to look for SSL version 3 session IDs.

Note Do not issue the **ssl-l4-fallback disable** command if SSL version 2 is in use on the network.

For example, to disable the CSS from inserting the Layer 4 hash value into the sticky table, enter:

```
(config)# ssl-l4-fallback disable
```

To reset the CSS back the default action of inserting a Layer 4 hash value into the sticky table, enter:

```
(config)# ssl-l4-fallback enable
```

Configuring Sticky Serverdown Failover

The sticky failover default method is for the CSS to use the configured load-balancing method. Use the **sticky-serverdown-failover** command to define what will happen if a sticky string is found but the associated service has failed or is suspended.

Note If you are using the **arrowpoint-cookie** option of the **advanced-balance** command, do not configure string match criteria, the **sticky-no-cookie-found-action** command, or the **sticky-serverdown-failover** command.

The syntax and options for this content mode command are:

- **sticky-serverdown-failover balance** - Sets the failover method to use a service based on the configured load-balancing method.
- **sticky-serverdown-failover redirect** - Sets the failover method to use the redirect string configured on a content rule. This command supports a 252-character redirect string (URL). For more information on redirect strings, see the "Redirecting Requests for Content" section in Chapter 10, Configuring Content Rules. If you do not configure a redirect string on a content rule, the load-balancing method is used.
- **sticky-serverdown-failover reject** - Rejects the content request.
- **sticky-serverdown-failover sticky-srcip** - Sets the failover method to use a service based on the client source IP address.
- **sticky-serverdown-failover sticky-srcip-dstport** - Sets the failover method to use a service based on the client source IP address and the server destination port.

For example, to set the sticky failover method to **sticky-srcip**, enter:

```
(config-owner-content[arrowpoint-rule1])sticky-serverdown-failover
sticky-srcip
```

To set the sticky failover method to its default setting of using the configured load-balancing method, enter:

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

Configuring Sticky Mask

A client IP address uniquely identifies the client to the CSS. During normal client-server sessions, the IP address is maintained throughout the connection. However, if the connection is lost (for example, due to a dense proxy failover) and the client reconnects with a different IP address, the CSS needs to reconnect the client to the same server that is preserving the client information (for example, information from a shopping cart or financial session).

Use the **sticky-mask** command to mask a group of client IP addresses in order to preserve the client connection state when the client's source IP address changes. The sticky mask specifies which portion of the client IP address the CSS will mask. The default sticky subnet mask is 255.255.255.255.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# sticky-mask 255.255.255.0
```

To restore the sticky subnet mask to the default of 255.255.255.255, enter:

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

Configuring Sticky Inactive Timeout

By default, new sticky connection uses the oldest used sticky entry. A sticky association could exist for a time depending on the sticky traffic load on the CSS. Use the **sticky-inact-timeout** command to specify the inactivity timeout period on a sticky connection for a content rule before the CSS removes the sticky entry from the sticky table. When you configure this period, the CSS keeps the sticky entry in the sticky table for the specified amount of time. The CSS does not reuse this entry until the time expires. If the sticky table is full and none of the entries has expired, the CSS rejects the new sticky request. If the **sticky-inact-timeout** command is specified for a Layer 5 content rule using SSL sticky, the SSL sessions continue even if the sticky table is full; however, the CSS does not maintain stickiness on the new sessions.

When the sticky connection expires, the CSS uses the configured load-balancing method to choose an available server for the request.

When this feature is disabled, the new sticky connection uses the oldest used sticky entry. A sticky association could exist for a time depending on the sticky traffic load on the CSS.

The syntax for this command is:

sticky-inact-timeout *minutes*

Enter the number of minutes of inactivity, from 0 to 65535. The default value is 0, which means this feature is disabled. For example, enter:

```
(config-owner-content[arrowpoint-rule1])# sticky-inact-timeout 9
```

Note The timer for each sticky entry begins at the start of a new connection and increments while the connection is open. Only a new connection that matches the entry causes its timer to reset. An entry with a timeout less than the life of an open connection may result in a new connection opening to a new server and a new sticky entry to the new server.

To disable the sticky connection inactivity timeout feature, enter:

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

Configuring Sticky Content for SSL

To use stickiness based on SSL version 3 session ID, configure a specific SSL Layer 5 rule for a service. To configure an SSL Layer 5 rule for a service:

- Set the port to 443 using the **(config-owner-content) port** command.
- Enable the content rule to be sticky based on SSL using the **(config-owner-content) advanced-balance ssl** command.
- Specify the SSL application type using the **(config-owner-content) application ssl** command.

Note We recommend that the **application ssl** command always be configured in conjunction with the **advanced-balance ssl** command. The **application ssl** command causes the CSS to spoof a connection so that you see the response come back from the server. The **advanced-balance ssl** command causes the CSS to look for the SSL session ID coming from the server and stick the client to the server based on that session ID. Once a flow is set up, the **application ssl** command then causes the CSS to treat the flow as a Layer 4 flow and does not inspect the flow for Layer 5 data in order to prevent the CSS from misinterpreting encrypted data.

For example, the following owner portion of a startup-config shows a content rule configured for SSL. Note that **url "/*"** command in this example is optional. The combination of the **application ssl** and **advanced-balance ssl** commands promotes the rule to Layer 5.

```
!************************** OWNER ***************************!
owner arrowpoint
content L5sslsticky
vip address 192.3.6.58
add service server87
add service server88
balance aca
protocol tcp
port 443
url "/*"
advanced-balance ssl
application ssl
active
```

Configuring String Range

By specifying the starting and ending byte positions within a cookie, URL, or URL extension that the CSS uses to search for the specified string, the CSS processes the information located only within this range. This limits the amount of information that the CSS has to process when examining each cookie, URL, or URL extension, enhancing its performance. By default, the string range is the first 100 bytes of the cookie, URL, or parameters in the URL.

Note If the starting position is beyond the cookie, URL, or URL extension, the CSS does not perform the string function. When the ending position is beyond the cookie, URL, or URL extension, the string processing stops at the end of the corresponding header.

Use the **string-range** command to specify the starting and ending byte positions within a cookie, URL, or URL extension that the CSS uses to search for the specified string. Enter the *start_byte* variable as the starting byte position of the cookie, URL, or URL extension after the header. Enter an integer from 1 to 1999. The default is 1. Ensure that the starting byte position is less than the end byte.

Enter the *end_byte* variable as the ending byte position of the cookie, URL, or URL extension. Enter an integer from 2 to 2000. The default is 100. Ensure that the ending byte position is more than the start byte position.

If you are using **advanced-balance**:

- **cookies** - The CSS starts counting after "Cookie: " (including the space after the colon).
- **url** - The CSS starts counting after the "/".
- **cookieurl** - The CSS starts counting after the "Cookie: " string. If the CSS does not find "Cookie: " in the HTTP request, it starts counting after the "?" in the URL of the same request.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# string-range 35
to 55
```

To restore the string range to the default of 1 to 100, enter:

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

Specifying a String Operation

To determine the method to choose a destination server for a string result, use the **string operation** command. The CSS derives the string result from the settings of the **string** criteria commands within the string range. You can choose a server by using the configured balance method or by using the hash key generated by the specified sticky hash type. If the Web servers:

- Are only capable of accepting the cookies that they set, then you must use the exact match method
- Can accept any cookies that are set by either a cookie server or other servers, then you may use the hash method

Note If you are using the **arrowpoint-cookie** option of the **advanced-balance** command, do not configure string match criteria, the **sticky-no-cookie-found-action** command, or the **sticky-serverdown-failover** command.

Comparing Hash Method with Match Method

When an application uses the exact match method, once a client makes a request to a particular server, the server is responsible for providing the client with a string unique to the server to use for future requests. Typically, if a server receives a string in a request that was set by another server, that string causes an error. In an exact match, the CSS looks for the unique string. If it finds an exact match, then the server is used. If no match is found, the CSS uses the configured load-balancing method to select a server for the client.

When an application uses one of the hash algorithms, all of the servers are capable of accepting any strings set by other servers. The model was designed so you could set up a site where the initial login would send a client to a Web server that assigns cookies to clients. When the CSS receives the first request from a client with the cookie string, it performs the hash operation on the string and chooses a server accordingly. The hash algorithm ensures that a particular string is always sent to a specific server, but it does not have to be a predefined server, as with an exact match.

Using the string operation hash algorithms may allow the Web server application to be used without being modified. When you use the **string operation match-service-cookie** method, you must modify the Web server application so that each server generates a unique string. The hash algorithms may be able to take advantage of strings already generated by the servers.

The syntax and options for this content mode command are:

- **string operation match-service-cookie** - Chooses a server by matching a service cookie in the sticky string. This is the default setting. When a match is not found, the CSS chooses the server by using the configured balance method (for example, roundrobin).
- **string operation** [**hash-a**|**hash-crc32**|**hash-xor**] - Chooses a server by using the hash key generated by the designated hash method. When using **advanced balance cookies** with a hash algorithm, all servers in the same domain must accept cookies regardless of which server created the cookie. This enables all servers configured on the Layer 5 rule to process cookies passed in an HTTP request.

  The hash method keywords are:

  - **hash-a** - Apply a basic hash algorithm on the hash string to generate the hash key

- **hash-crc32** - Apply the CRC32 algorithm on the hash string to generate a hash key
- **hash-xor** - Exclusive OR (XOR) each byte of the hash string to derive the final hash key

If the selected server is out of service, the CSS performs a rehash to choose another server.

We recommend using either **hash-xor** or **hash-crc32** depending on your string possibilities. If the strings are completely dissimilar, use **hash-xor**. If the strings are similar, use **hash-crc32**. For example, if your string values are abc1, abc2, and abc3, the **hash-xor** method cannot provide you with enough variance in the hash values (that is, abc1 and abc2 may end up on the same server because they may hash to the same value).

For example, to set the string operation to choose a server by using the string operation **hash-crc32** algorithm, enter:

```
(config-owner-content[arrowpoint-rule1])# string operation hash-crc32
```

To reset the string operation to its default setting of choosing a server by matching a service cookie in the sticky string, enter:

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

The CSS derives a string result from the following string criteria commands:

- **string ascii-conversion**
- **string match**
- **string eos-char**
- **string prefix**
- **string process-length**
- **string skip-length**

Enabling or Disabling String ASCII Conversion

By default, ASCII conversion of escaped special characters within the specified sticky string range before applying any processing to the string is enabled. Use the **string ascii-conversion** command to enable or disable the ASCII conversion.

For example, to disable ASCII conversion of escaped special characters, enter:

```
(config-owner-content[arrowpoint-rule1])# string ascii-conversion
disable
```

To reenable the ASCII conversion of escaped special characters to its default setting, use the **no** form of the command or the **enable** option. For example, enter:

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

(config-owner-content[arrowpoint-rule1])# string ascii-conversion
enable
```

Configuring the Handling of Multiple String Matches

By default, if the CSS determines that the incoming string matches multiple configured service strings, the CSS matches the most specific (longest) string. In the following example, the CSS service configuration is:

```
service s1
string pear

service s2
string grape

service s3
string banana
```

If the incoming string is grapebananapear, the string match of the most specific string is banana.

Using the **string match** command, you can also configure the CSS to:

- Match the first string in the incoming string by using the **first-string-match** keyword. Enter:

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

In the case of the previous example, the string match is grape.

- Look at each service in the order of its index entry until there is a match by using the **first-service-match** keyword. Enter:

```
(config-owner-content[arrowpoint-rule1])# string match
first-service-match
```

In the case of the previous example, the first-service-string match is pear.

Note Use **string match** command with the **advanced-balance cookies|cookiesurl|url** command.

To reset the default behavior of matching the most specific string, enter:

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

Specifying End-of-String Characters

To specify up to three ASCII characters as the delimiters for the sticky string within the string range, use the **string eos-char** command. For example, in a cookie header, a semicolon (;) is usually used as a delimiter; in a URL extension, an ampersand (&) is often used as a delimiter.

The CSS uses the **string eos-char** value if the **(config-owner-content) string process-length** command is not configured. The **(config-owner-content) string process-length** command has higher precedence. If neither command is configured, the CSS uses the maximum of 100 bytes for the final string length. Enter the sticky string end-of-string characters as a quoted text string with a maximum of three characters.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# string eos-char ";"
```

To clear the end of string characters, enter:

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

Specifying a String Prefix

If you do not configure the string prefix, the string functions start from the beginning of the string range for the cookie, URL, or URL extension, depending on the sticky type. By default, the string range is the first 100 bytes of the cookie, URL, or parameters in the URL. If the string prefix is configured but is not found in the string range, the CSS uses the load-balancing method you defined in the **sticky-serverdown-failover** command.

Use the **string prefix** command to specify the string prefix located in the string range. Enter the string prefix as a quoted text string with a maximum of 30 characters. The default is no prefix (“”).

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# string prefix “UID=”
```

To clear the string prefix, enter:

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

Specifying a String Process Length

To specify how many bytes, after the end of the prefix within the string range designated by the **string prefix** command and skipping the bytes designated by the **string skip-length** command, the string action will use, use the **string process-length** command. This command has higher precedence than the **string eos-char** command. If neither command is configured, the CSS uses the maximum of 100 bytes for the final string action. Enter the number of bytes from 0 to 252. The default is 0.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# string process-length 16
```

To set the number of bytes to its default setting of 0, enter:

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

Specifying a String Skip Length

To specify how many bytes to skip after the end of prefix within the string range to find the string result, use the **string skip-length** command. Enter the number of bytes from 0 to 64. The default is 0. For example, enter:

```
(config-owner-content[arrowpoint-rule1])# string skip-length 3
```

To set the number of bytes to its default setting of 0, enter:

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

Configuring Sticky-No-Cookie-Found-Action

To specify the action the CSS should take for a sticky cookie content rule when it cannot locate the cookie header or the specified cookie string, use the **sticky-no-cookie-found-action** command.

Note If you intend to use the **advanced-balance arrowpoint-cookie** command, do not configure the **sticky-no-cookie-found-action** command. They are not compatible.

The options for the **sticky-no-cookie-found-action** command are:

- **loadbalance** (default) - The CSS uses the configured balance method when no cookie is found in the client request.
- **redirect** *"URL"* - Redirects the client request to a specified URL string when no cookie found in the client request. When using this option, you must also specify a redirect URL. Specify the redirect URL as a quoted text string from 0 to 252 characters.
- **reject** - Rejects the client request when no cookie is found in the request.
- **service** *name* - Sends the no cookie client request to the specified service when no cookie is found in the request.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])#
sticky-no-cookie-found-action redirect
"http://www.lml.com/nocookie.html"
```

To reset **sticky-no-cookie-found-action** to the default of **loadbalance**, enter:

```
(config-owner-content[arrowpoint-rule1])# no
sticky-no-cookie-found-action
```

Configuring Sticky Parameters for E-Commerce and Other Internet Applications

By configuring sticky parameters for e-commerce applications, you can instruct the CSS how to process client requests that do not contain cookies when the requests are destined to a content rule that is sticking based on a string within a cookie. You can also instruct the CSS how to process wireless users by integrating HTTP header load balancing with the **advanced-balance wap-msisdn** command. For applications that use the CSS sticky table, you can remove a sticky table entry after a defined period of activity.

The following sections describe how to configure sticky parameters for e-commerce and other Internet applications:

- Configuring an advanced-balance arrowpoint-cookie
- Configuring an Arrowpoint Cookie
- Reinserting an Arrowpoint Cookie in an HTTP Server Response
- Configuring a Location Cookie
- Configuring Wireless Users for E-Commerce Applications
- Configuring Session Initiation Protocol Load Balancing

Configuring an advanced-balance arrowpoint-cookie

To enable the content rule to stick the client to the server based on the unique service identifier of the selected server in the arrowpoint-generated cookie, use the **advanced-balance arrowpoint-cookie** command. Configure the service identifier by using the **(config-service) string** command.

Note If you are using the **arrowpoint-cookie** option of the **advanced-balance** command, do not configure string match criteria, the **sticky-no-cookie-found-action** command, or the **sticky-serverdown-failover** command. They are not compatible with **advanced-balance arrowpoint-cookie** command.

You do not need to configure string match criteria. For information on configuring the arrowpoint-generated cookie, see the "Configuring an Arrowpoint Cookie" section. You can use this option with any Layer 5 content rule.

Note If you configure an ArrowPoint cookie on a content rule using the **advanced-balance arrowpoint-cookie** command and the CSS receives a subsequent GET with no ArrowPoint cookie on a persistent HTTP connection, the CSS ignores all persistence settings in the running-config, remaps the back-end connection to a new server, and inserts a new ArrowPoint cookie.

For example, to specify **advanced-balance arrowpoint-cookie** for content rule1, enter:

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

To disable the advanced load-balancing method, enter:

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

Configuring an Arrowpoint Cookie

The CSS generates the arrowpoint cookie transparently for a client, the client stores it and returns it in subsequent requests, and the CSS later uses it to maintain the client-server stickiness. This cookie contains the sticky information itself and does not refer to a sticky table.

Use the **arrowpoint-cookie** command to configure the arrowpoint cookie name, path, and expiration time. If you configure the **arrowpoint-cookie** method in a content rule, the CSS always checks for the existence of the arrowpoint cookie when it receives a client request. If this cookie does not exist, the CSS performs server load balancing and generates an arrowpoint cookie.

Note If you configure the CSS to insert an arrowpoint cookie in a server response, it does not expect packets to arrive out of order. If the packets arrive out of order with the CSS receiving the FIN first, when the CSS finally receives the server data packet, it forwards the packet without inserting the cookie. Enabling the **flow persist-span-ooo** command allows the CSS to detect that the FIN packet arrived out of order and discards it. Then the server retransmits the packets and the CSS processes them in the correct order and inserts the arrowpoint cookie in the server data packet.

When the CSS finds the cookie in the client request, it unscrambles the cookie data and then validates it. Then, the CSS checks the cookie expiration time. If the cookie has expired, the CSS sends a new cookie containing the information about the server where the client was stuck. This appears as an uninterrupted connection.

If the cookie format is valid, the CSS ensures the consistency between the cookie and the CSS configuration. When all the validations are passed, the CSS forwards the client request to the server indicated by the server identifier. Otherwise, the CSS treats this request as an initial request.

The options for this content mode command are:

- **arrowpoint-cookie name** - Specifies a unique cookie identifier.
- **arrowpoint-cookie path** - Sets the cookie path to a configured path.
- **arrowpoint-cookie expiration** - Sets an expiration time, which the CSS compares with the time associated with the cookie.

  Note When you configure the **arrowpoint-cookie expiration** command and the **advanced-balance arrowpoint-cookie** command, the CSS CPU may spike and the CSS may experience a degradation in its performance. Configure the **arrowpoint-cookie expiration** command only when necessary.

- **arrowpoint-cookie browser-expire -** Allows the browser to expire the cookie.
- **arrowpoint-cookie expire services -** Expires the service information when the cookie expires.

Configuring an Arrowpoint Cookie Name

To configure a unique cookie identifier with a maximum of four alphanumeric characters, use the **arrowpoint-cookie name** command. With this option, you can configure multiple CSSs to inject cookies without the potential of one CSS overwriting another CSS cookie.

The syntax of this owner-content configuration mode command is:

arrowpoint-cookie name *name*

Enter a unique string consisting of 1 to 31 alphanumeric characters for the *name* variable. The default is ARPT.

Caution When you configure a new cookie name on a content rule, the CSS no longer recognizes any pre-existing cookie name configured on that rule. Therefore, any existing stickiness is lost.

For example:

```
(config-owner-content[arrowpoint-rule1])# arrowpoint-cookie name
abc5678
```

The cookie resulting from this command would appear as:

```
abc5678=000000SservicenameT0x_0xC1234
```

To reset the Arrowpoint cookie name to the default of ARPT, enter:

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

Configuring an Arrowpoint Cookie Path

By default, the CSS sets the default path attribute of the cookie to a slash (/). Use the **arrowpoint-cookie path** command to set the arrowpoint-cookie path to a configured path.

The syntax of this owner-content configuration mode command is:

arrowpoint-cookie path "*path_name***"**

Enter the *path_name* where you want to send the cookie. Enter a quoted text string with a maximum of 99 characters. The default path of the cookie is "/".

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# arrowpoint-cookie path
"/cgi-bin/"
```

To reset the cookie path to its default of "/", enter:

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

Configuring an Arrowpoint-Cookie Expiration Time

If the arrowpoint cookie has expired, the CSS sends a new cookie that includes the server where the client was stuck. The sending of the new cookie allows for the appearance of an uninterrupted connection. By default, if you do not set an expiration time, the cookie expires when the client exits the browser.

Use the **arrowpoint-cookie expiration** command to set an expiration time, which the CSS compares with the time associated with the arrowpoint cookie.The syntax of this owner-content mode configuration command is:

arrowpoint-cookie expiration *dd:hh:mm:ss*

The variables are:

- *dd* - Number of days. Valid numbers are from 00 to 99.
- *hh* - Number of hours. Valid numbers are from 00 to 99.
- *mm* - Number of minutes. Valid numbers are from 00 to 99.
- *ss* - Number of seconds. Valid numbers are from 00 to 99.

Note Do not use all zeros for days, hours, minutes, and seconds. This value is invalid.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# arrowpoint-cookie expiration
08:04:03:06
```

To reset the expiration time to when the client exits the browser, enter:

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

Configuring Arrowpoint-Cookie Browser Expire

To allow the browser to expire the arrowpoint cookie based on the expiration time, use the **arrowpoint-cookie browser-expire** command. To configure the expiration time, see the previous section. The syntax of this owner-content configuration mode command is:

arrowpoint-cookie browser-expire

For example, enter:

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

To allow the CSS to expire the cookie, enter:

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

Note When the cookie expires, all sticky information is lost.

Configuring Arrowpoint-Cookie Expire Services

By default, when the arrowpoint cookie expires, the CSS sends a new cookie with the server information from the expired cookie. Use the **arrowpoint-cookie expire-services** command to expire service information when the cookie expires before sending a new cookie. The syntax of this owner-content configuration mode command is:

arrowpoint-cookie expire-services

For example, enter:

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

To reset the default behavior, enter:

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

Configuring an Arrowpoint Cookie Domain

Use the **cookie-domain** command to configure a domain name for the arrowpoint cookie. For details, see the "Configuring a Domain Name for the Location Cookie" section later in this chapter.

Reinserting an Arrowpoint Cookie in an HTTP Server Response

By default, the CSS always inserts an Arrowpoint (ARPT) cookie in the first server response packet that begins with HTTP. More than likely during POST processing, the packet may contain a 100 Continue response instead of a 200 OK response. When the client receives the 100 Continue response with the inserted ARPT cookie, it may discard the response along with the cookie. Because the CSS does not reinsert the cookie when it receives a following 200 OK response, the client never uses the cookie and stickiness is broken.

To reinsert the Arrowpoint cookie in an HTTP server response if the previous packet contains a 100 Continue response, use the **arpt-lct http-100-reinsert** command. Use this command on a content rule configured with the **advanced-balance arrowpoint-cookie** command.

For example, enter:

```
(config-owner-content[arrowpoint-rule1])# arpt-lct http-100-reinsert
```

To reset the default behavior of inserting the Arrowpoint cookie in the first server response packet that begins with HTTP, enter:

```
(config-owner-content[arrowpoint-rule1])# no arpt-lct
http-100-reinsert
```

Configuring a Location Cookie

Occasionally, when a client is stuck to a particular CSS and server in a multisite network configuration, a subsequent DNS request may resolve to a different IP address, which sends the client to a different CSS and server. This different resolution can be a problem in an e-commerce application, especially when a client already has items in a shopping cart. To ensure that a CSS returns the client to the original CSS in a multisite environment, configure a Location Cookie.

Note The CSS recognizes content requests that include a location cookie as part of a sticky session. Therefore, even if you add a service with a configured weight of zero as a location service to a content rule, the CSS continues to direct to that service any requests that contain location cookies originating from the service.

Overview

Location Cookie injects a user-defined NAME=VALUE cookie pair string (configured on the CSS specific to a site) into the response packet from the server. On subsequent connections after cookie injection, if a new DNS resolution sends the client to a different CSS, the new CSS attempts to match the NAME=VALUE pair to the values in a configured content rule. If the CSS cannot match the NAME=VALUE pair, the CSS compares the VALUE portion of the cookie to information in location services configured in a content rule. The CSS uses the information in the location services to return the client to the original site.

There are two methods that a CSS can use to return a client to the original site, depending on the type of location service that you configured on the CSS. The first method uses a standard service (pass-through service) to pass all traffic through the current CSS, then back to the original CSS. You must configure this same service as a destination service in a source group to ensure that all return traffic from the original CSS is NATed through the current CSS before going to the client.

The second method uses a redirect service to send a 302 redirect to the client, thereby forcing the client back to the original site. The 302 redirect uses any URL PUT and appends it with http:// unless you configure the service configuration mode **no prepend http** command. The URL points to a specific file or default file in a directory. Redirected services can also use the service configuration mode domain command to redirect from HTTP to HTTPS. Note that a 302 redirect changes all HTTP methods to HTTP GETs. For example, if a client is performing a POST operation and the current CSS redirects the client to the original site, the client will lose all data in the POST method.

Note In addition to the IP address of the main site, each site must also have a unique DNS entry; for example, site1.work.com. These site-specific DNS entries guarantee that a client will receive a unique response from the DNS server for each site and prevent a client from being redirected to an IP address.

Location Cookie Quick Start

Table 11-2, Table 11-3, and Table 11-4 provide a quick overview of the steps required to configure the Location Cookie feature on each CSS in a multisite configuration. This section includes quick start procedures for three sample sites. Each step includes the CLI command required to complete the task. For a complete description of each feature and all the options associated with the CLI commands, see the sections following the tables.

Table 11-2 Location Cookie Configuration Quick Start for Site1

Task and Command Example

1. Enter global configuration mode.
`# config`
`(config)#`

2. Configure local services as needed. For details on configuring services, see Chapter 3, Configuring Services. For example:
`(config)# service localServ1`
`(config-service[localServ1])# ip address 192.168.2.3`
`(config-service[localServ1])# active`
`(config-service[localServ1])# service localServ2`
`(config-service[localServ2])# ip address 192.168.2.4`
`(config-service[localServ2])# active`

3. Configure a redirect service that the CSS will use as a location service.
`(config-service[localServ2])# service site2`
`(config-service[site2])# ip address 192.158.128.209`
`(config-service[site2])# string site2`
`(config-service[site2])# type redirect`
`(config-service[site2])# active`

4. Configure a standard service that the CSS will use as a location service.
`(config-service[site2])# service site3`
`(config-service[site3])# ip address 192.148.128.209`
`(config-service[site3])# string site3`
`(config-service[site3])# active`

5. Configure an owner. For details on configuring an owner, see Chapter 9, Configuring Owners.
`(config)# owner ArrowPoint`
`(config-owner[ArrowPoint])#`

Table 11-2 Location Cookie Configuration Quick Start for Site1 (continued)

Task and Command Example

6. Create a content rule for the owner. For details on configuring a content rule, see Chapter 10, Configuring Content Rules.
`(config-owner[ArrowPoint])# content locCookie`
`(config-owner-content[ArrowPoint-locCookie])#`

7. Configure the following commands on the content rule.
`(config-owner-content[ArrowPoint-locCookie])# vip address 192.168.128.209`
`(config-owner-content[ArrowPoint-locCookie])# add service localServ1`
`(config-owner-content[ArrowPoint-locCookie])# add service localServ2`
`(config-owner-content[ArrowPoint-locCookie])# protocol tcp`
`(config-owner-content[ArrowPoint-locCookie])# port 80`
`(config-owner-content[ArrowPoint-locCookie])# url "/*"`
`(config-owner-content[ArrowPoint-locCookie])# location-cookie name work value site1`
`(config-owner-content[ArrowPoint-locCookie])# cookie-domain ".work.com"`
`(config-owner-content[ArrowPoint-locCookie])# add location-service site2`
`(config-owner-content[ArrowPoint-locCookie])# add location-service site3`
`(config-owner-content[ArrowPoint-locCookie])# active`

8. Create a source group and add service site3 as a destination service. For details on configuring a source group, see Chapter 3, Configuring Services.
`(config)# group site1`
`(config-group[site1])# add destination service site3`
`(config-group[site1])# vip address 192.168.128.210`
`(config-group[site1])# active`

9. Use the **show rule sticky** command to verify the location cookie configuration. `# show rule sticky`

The following running-config example shows the results of entering the commands described in Table 11-2 for site 1.

```
!************************** SERVICE **************************
```

```
service localServ1
  ip address 192.168.2.3
  active

service localServ2
  ip address 192.168.2.4
  active

service site2
  ip address 192.158.128.209
  string site2
  type redirect
  active

service site3
  ip address 192.148.128.209
  string site3
  active

!************************** OWNER ***************************
owner ArrowPoint

  content locCookie
    vip address 192.168.128.209
    add service localServ1
    add service localServ2
    protocol tcp
    port 80
    url "/*"
    location-cookie name work value site1
    cookie-domain ".work.com"
    add location-service site2
    add location-service site3
    active

!************************** GROUP ***************************
group site1
  add destination service site3
  vip address 192.168.128.210
  active
```

Table 11-3 Location Cookie Configuration Quick Start for Site2

Task and Command Example

1. Enter global configuration mode.
`# config`
`(config)#`

2. Configure local services as needed. For example:
`(config)# service localServ1`
`(config-service[localServ1])# ip address 192.158.2.3`
`(config-service[localServ1])# active`
`(config-service[localServ1])# service localServ2`
`(config-service[localServ2])# ip address 192.158.2.4`
`(config-service[localServ2])# active`

3. Configure a standard service for each of the other sites that the CSS will use as location services.
`(config-service[localServ2])# service site1`
`(config-service[site1])# ip address 192.168.128.209`
`(config-service[site1])# string site1`
`(config-service[site1])# active`
`(config-service[site1])# service site3`
`(config-service[site3])# ip address 192.148.128.209`
`(config-service[site3])# string site3`
`(config-service[site3])# active`

4. Configure an owner.
`(config)# owner ArrowPoint`
`(config-owner[ArrowPoint])#`

5. Create a content rule for the owner.
`(config-owner[ArrowPoint])# content locCookie`
`(config-owner-content[ArrowPoint-locCookie])#`

Table 11-3 Location Cookie Configuration Quick Start for Site2 (continued)

Task and Command Example

6. Configure the following commands on the content rule.
`(config-owner-content[ArrowPoint-locCookie])# vip address 192.158.128.209`
`(config-owner-content[ArrowPoint-locCookie])# add service localServ1`
`(config-owner-content[ArrowPoint-locCookie])# add service localServ2`
`(config-owner-content[ArrowPoint-locCookie])# protocol tcp`
`(config-owner-content[ArrowPoint-locCookie])# port 80`
`(config-owner-content[ArrowPoint-locCookie])# url "/*"`
`(config-owner-content[ArrowPoint-locCookie])# location-cookie name work value site2`
`(config-owner-content[ArrowPoint-locCookie])# cookie-domain ".work.com"`
`(config-owner-content[ArrowPoint-locCookie])# add location-service site1`
`(config-owner-content[ArrowPoint-locCookie])# add location-service site3`
`(config-owner-content[ArrowPoint-locCookie])# active`

7. Create a source group and add service site1 and service site3 as destination services.
`(config)# group site2`
`(config-group[site2])# add destination service site1`
`(config-group[site2])# add destination service site3`
`(config-group[site2])# vip address 192.158.128.210`
`(config-group[site2])# active`

8. Use the **show rule sticky** command to verify the location cookie configuration. `# show rule sticky`

The following running-config example shows the results of entering the commands described in Table 11-3 for site 2.

```
!************************** SERVICE **************************
service localServ1
  ip address 192.158.2.3
  active

service localServ2
  ip address 192.158.2.4
  active

service site1
  ip address 192.168.128.209
  string site1
  active

service site3
  ip address 192.148.128.209
  string site3
  active

!************************** OWNER ****************************
owner ArrowPoint

  content locCookie
    vip address 192.158.128.209
    add service localServ1
    add service localServ2
    protocol tcp
    port 80
    url "/*"
    location-cookie name work value site2
    cookie-domain ".work.com"
    add location-service site1
    add location-service site3
    active

!************************** GROUP ****************************
group site2
  add destination service site1
  add destination service site3
  vip address 192.158.128.210
  active
```

Table 11-4 Location Cookie Configuration Quick Start for Site3

Task and Command Example

1. Enter global configuration mode.
`# config`
`(config)#`

2. Configure local services as needed. For example:
`(config)# service localServ1`
`(config-service[localServ1])# ip address 192.148.2.3`
`(config-service[localServ1])# active`
`(config-service[localServ1])# service localServ2`
`(config-service[localServ2])# ip address 192.148.2.4`
`(config-service[localServ2])# active`

3. Configure a standard service that the CSS will use as a location service.
`(config-service[localServ2])# service site1`
`(config-service[site1])# ip address 192.168.128.209`
`(config-service[site1])# string site1`
`(config-service[site1])# active`

4. Configure a redirect service that the CSS will use as a location service.
`(config-service[site1])# service site2`
`(config-service[site2])# ip address 192.158.128.209`
`(config-service[site2])# string site2`
`(config-service[site2])# type redirect`
`(config-service[site2])# active`

5. Configure an owner.
`(config)# owner ArrowPoint`
`(config-owner[ArrowPoint])#`

6. Create a content rule for the owner.
`(config-owner[ArrowPoint])# content locCookie`
`(config-owner-content[ArrowPoint-locCookie])#`

Table 11-4 Location Cookie Configuration Quick Start for Site3 (continued)

Task and Command Example

7. Configure the following commands on the content rule.
`(config-owner-content[ArrowPoint-locCookie])# vip address 192.148.128.209`
`(config-owner-content[ArrowPoint-locCookie])# add service localServ1`
`(config-owner-content[ArrowPoint-locCookie])# add service localServ2`
`(config-owner-content[ArrowPoint-locCookie])# protocol tcp`
`(config-owner-content[ArrowPoint-locCookie])# port 80`
`(config-owner-content[ArrowPoint-locCookie])# url "/*"`
`(config-owner-content[ArrowPoint-locCookie])# location-cookie name work value site3`
`(config-owner-content[ArrowPoint-locCookie])# cookie-domain ".work.com"`
`(config-owner-content[ArrowPoint-locCookie])# add location-service site1`
`(config-owner-content[ArrowPoint-locCookie])# add location-service site2`
`(config-owner-content[ArrowPoint-locCookie])# active`

8. Create a source group and add service site1 as a destination service.
`(config)# group site3`
`(config-group[site1])# add destination service site1`
`(config-group[site1])# vip address 192.148.128.210`
`(config-group[site1])# active`

9. Use the **show rule sticky** command to verify the location cookie configuration. `# show rule sticky`

The following running-config example shows the results of entering the commands described in Table 11-4 for site 3.

```
!************************** SERVICE **************************
service localServ1
  ip address 192.148.2.3
  active

service localServ2
  ip address 192.148.2.4
  active
service site1
  ip address 192.168.128.209
  string site1
  active

service site2
  ip address 192.158.128.209
  string site2
  type redirect
  active

!************************** OWNER ****************************
owner ArrowPoint

  content locCookie
    vip address 192.148.128.209
    add service localServ1
    add service localServ2
    protocol tcp
    port 80
    url "/*"
    location-cookie name work value site3
    cookie-domain ".work.com"
    add location-service site1
    add location-service site2
    active

!************************** GROUP ****************************
group site3
  add destination service site1
  vip address 192.148.128.210
  active
```

Configuring the location-cookie Command

To configure the NAME=VALUE cookie string and expiration time for the local site, use the **location-cookie** command. A CSS injects this cookie string into responses from a server.

Note Location cookie requires a Layer 5 (L5) content rule. At a minimum, you need to configure url "/*" on the rule, as shown in the "Location Cookie Quick Start" section.

The syntax for this owner-content configuration mode command is:

location-cookie name *text* **value** *text* {**expiration** *dd:hh:mm:ss*}

The options and variables for this command are:

- **name** *text* - The first part of the NAME=VALUE cookie string. Enter an unquoted text string from 1 to 31 characters.
- **value** *text* - The second part of the NAME=VALUE cookie string. Enter an unquoted text string from 1 to 31 characters.
- **expiration** *dd:hh:mm:ss* - (Optional) Expiration date and time of the Location Cookie. This value indicates to the client browser when the cookie will expire based on a relative time from cookie generation. Enter a date and time in the following format:
  - *dd* - Number of days. Valid numbers are from 00 to 99.
  - *hh* - Number of hours. Valid numbers are from 00 to 99.
  - *mm* - Number of minutes. Valid numbers are from 00 to 99.
  - *ss* - Number of seconds. Valid numbers are from 00 to 99.

Note When you configure the expiration time and date for the Location Cookie, the CSS CPU may spike and the CSS may experience a degradation in its performance. Configure the **expiration** option only when necessary.

For example:

```
(config-owner-content[ArrowPoint-rule1])# location-cookie name work
value site1 00:02:30:00
```

To remove the location cookie, enter:

```
(config-owner-content[ArrowPoint-rule1])# no location-cookie name
```

Configuring a Domain Name for the Location Cookie

The cookie domain name allows your browser to send the cookie back to any site that ends with the domain name that you specify. For example, if you specify a cookie domain name of .work.com, the browser will send back the Location Cookie to all sites that end with .work.com, including site1.work.com, site2.work.com, and site3.work.com.

Use the **cookie-domain** command to configure a domain name for the Location Cookie. The syntax for this owner-content configuration mode command is:

cookie-domain *name*

The *name* variable specifies the name of the domain for the Location Cookie. Enter a quoted text string from 1 to 64 characters.

For example:

```
(config-owner-content[ArrowPoint-rule1)# cookie-domain ".work.com"
```

To remove the cookie domain name, enter:

```
(config-owner-content[ArrowPoint-rule1)# no cookie-domain
```

Configuring Location Services

To add services to the content rule that the CSS uses to locate the site where the client was originally stuck, use the **add location-service** command.

The syntax for this owner-content configuration mode command is:

add location-service *service_name*

The *name* variable specifies the name of a standard service or redirect service that you want to add to the content rule for Location Cookie matching. Enter a name from 1 to 31 characters.

For example, enter:

```
(config-owner-content[ArrowPoint-rule1)# add location-service site1
```

To remove the site1 location service, enter:

```
(config-owner-content[ArrowPoint-rule1)# remove location-service site1
```

You can configure a maximum of 10 location services; either standard services or redirect services. These services do not count toward the 64-service maximum per content rule and do not participate in the load-balancing algorithm of the content rule.

Note You cannot add a location service to a content rule if another location service on the rule is configured with the same cookie string.

A redirect service configured as a location service must have an IP address that is the same as the VIP address of the location cookie content rule configured on the redirected site. You must also define a string on any service used as a location service. This string must match the VALUE portion of the Location Cookie. For information on configuring a string on a service, see the "Configuring an Advanced Load-Balancing String" section.

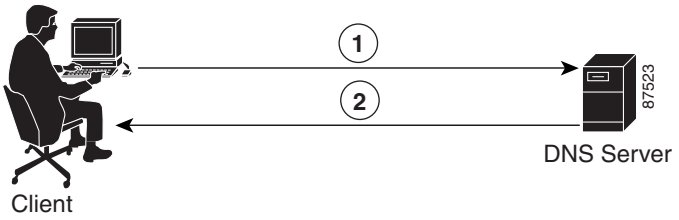
Also, you must configure as a destination service in a source group any standard service that you configure as a location service in the content rule, as shown in the "Location Cookie Quick Start" section.

Examples of Location Cookie Flow

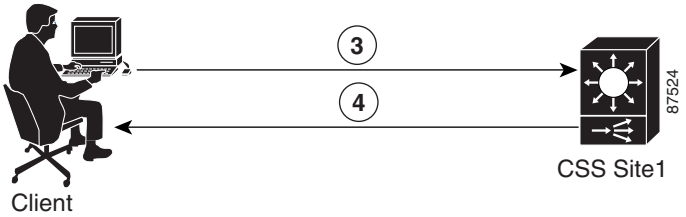
The following examples illustrate the two mechanisms that a CSS uses to return a client to the CSS and server that serviced the original request. See the "Location Cookie Quick Start" section for specific CSS site configuration information.

Example 1 - Returning a client to the original site using a location service (pass-through method)

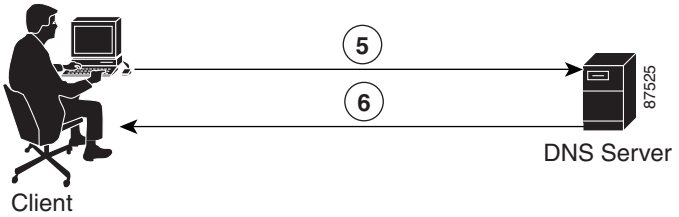
1. The client sends a lookup request for my.work.com to the DNS server.
2. The DNS server responds with the CSS Site1 VIP address of 192.168.128.209.



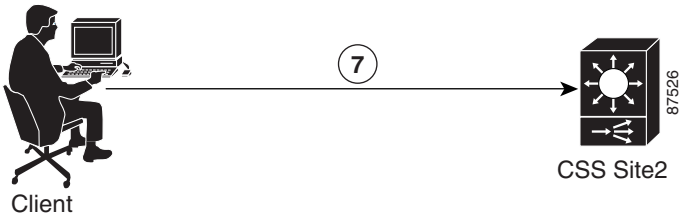
3. The client sends a GET request to 192.168.128.209 with no cookie.
4. The CSS injects the location cookie work=site1 to the response from the local server. Client-server interaction proceeds normally.



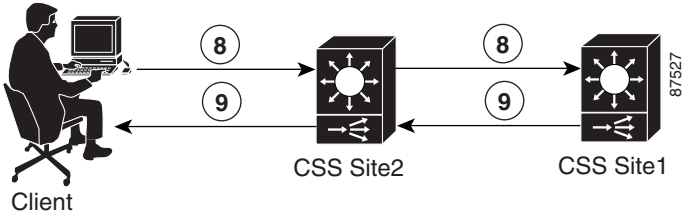
5. At some point in the future, the client sends another lookup request for my.work.com to the DNS server.
6. This time the DNS server responds with the CSS Site2 VIP address of 192.158.128.209.



7. The client sends a GET request to 192.158.128.209 (CSS Site2) with a location cookie of work=site1. This cookie string matches the configured location cookie name (work), but not the cookie value (site2). CSS Site2 searches through the list of location services checking the configured strings against the value in the cookie. In this case, a match is made on service site1.

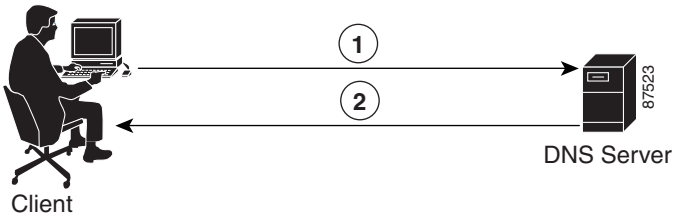


8. Service site1 is configured as a destination service on the source group site2. This service matches on the locCookie content rule configured on CSS Site2. CSS Site2 forwards packets from the client to CSS Site1.
9. Because the client already sent a cookie with the GET request in Step 7, there is no need to inject another cookie. Content rule processing continues with no changes. The server response goes back through CSS Site1 to CSS Site2 to the client. The client is now stuck to the original site (site1.work.com) through CSS Site2.

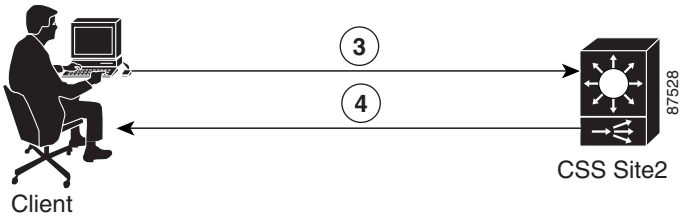


Example 2 - Returning a client to the original site using a redirect service

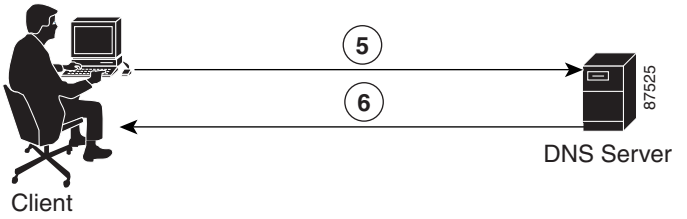
1. The client sends a lookup request for my.work.com to the DNS server.
2. The DNS server responds with the CSS Site2 VIP address of 192.158.128.209.



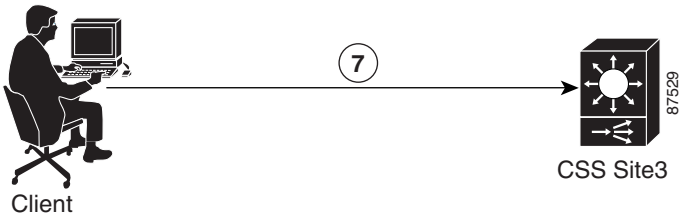
3. The client sends a GET request to 192.158.128.209 (Site2) with no cookie.
4. The CSS injects the location cookie work=site2 to the response from the local server. Client-server interaction proceeds normally.



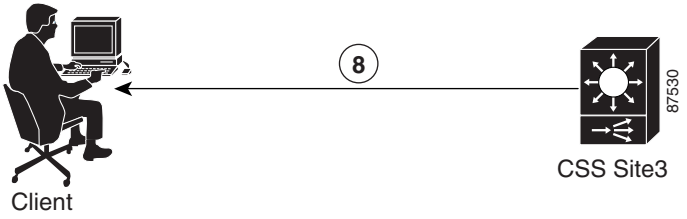
5. At some point in the future, the client sends another lookup request for my.work.com to the DNS server.
6. This time, the DNS server responds with the CSS Site3 VIP address of 192.148.128.209.



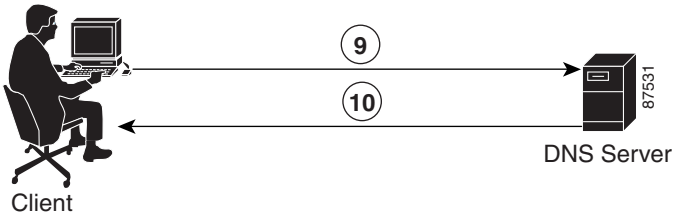
7. The client sends a GET request to 192.148.128.209 (CSS Site3) with a location cookie of work=site2. This cookie string matches the configured location cookie name (work) on CSS Site3, but not the cookie value (site3). CSS Site3 searches through the list of location services checking the configured strings against the value in the cookie. In this case, a match is made on service site2.



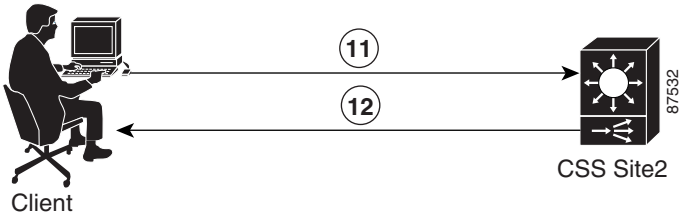
8. Service site2 is a redirect service, so CSS Site3 sends the client a 302 redirect to site2.work.com.



9. The client sends a lookup request for site2.work.com to the DNS server.
10. The DNS server responds with the CSS Site2 VIP address of 192.158.128.209.



11. The client sends a GET request to 192.158.128.209 (CSS Site2) with the cookie work=site2.
12. The cookie name and value match on the locCookie content rule configured on CSS Site2. Because the client already sent a cookie with the GET request in Step 11, there is no need to inject another cookie. Content rule processing continues with no changes. The client is now permanently stuck to the original site (site2.work.com).



Displaying Location Cookie Information

To display location cookie information, use the **show rule sticky** command. For details, see the "Showing Sticky Attributes" section.

Configuring Wireless Users for E-Commerce Applications

Wireless clients use the Wireless Application Protocol (WAP) to access Internet content. When a wireless client sends a request for content, the WAP protocol gateway (a device that translates requests from the WAP protocol stack to the WWW protocol stack) generates the MSISDN field and adds it to the HTTP header.

Use the **advanced-balance wap-msisdn** command with the MSISDN header field to configure wireless users for e-commerce applications. For details on the **advanced-balance wap-msisdn** command, see the "Specifying an Advanced Load-Balancing Method for Sticky Content" section earlier in this chapter. For details on the MSISDN header field, see the "Configuring a Header-Field Entry" section in Chapter 12, Configuring HTTP Header Load Balancing.

In the following example, TCP port 80 traffic destined for VIP 192.168.128.151 that contains the string "012" in the MSISDN HTTP header field will match content rule rule012. The CSS will stick this traffic to either server1 or server2 based on the entire contents of the MSISDN field.

TCP port 80 traffic destined for 192.168.128.151 that does not contain the string "012" in the MSISDN HTTP header field, but has the field in the header, will match content rule ruleNo012. The CSS will use roundrobin to load balance the traffic across server21 and server22.

TCP port 80 traffic destined for 192.168.128.151 that does not contain the MSISDN HTTP header field will match content rule ruleNoWap. The CSS will use roundrobin to load balance the traffic across server31 and server32.

```
header-field-group wap012
   header-field 1 wap-msisdn contain "012"

header-field-group wapNo012
   header-field 1 wap-msisdn not-contain "012"
```

```
owner arrowpoint
   content rule012
     vip address 192.168.128.151
     protocol tcp
     port 80
     url "/*"
     add service server1
     add service server2
     header-field-rule wap012
     advanced-balance wap-msisdn
     active
   content ruleNo012
     vip address 192.168.128.151
     protocol tcp
     port 80
     url "/*"
     add service server21
     add service server22
     header-field-rule wapNo012
     active

   content ruleNoWap
     vip address 192.168.128.151
     protocol tcp
     port 80
     url "/*"
     add service server31
     add service server32
     active
```

Configuring Session Initiation Protocol Load Balancing

Session Initiation Protocol (SIP) is an application-layer control protocol that functions as a signaling mechanism between user devices and media servers. SIP is a peer-to-peer protocol where end-devices (the User Agent Clients) initiate interactive communications sessions with SIP servers. These sessions can include Internet multimedia conferences, Internet telephone calls (voice-over-IP), and multimedia distribution. Examples of client devices include hardware, software, handheld IP telephones, and personal digital assistants (PDAs).

The session Call-ID is a unique call identifier that is contained in the SIP messages sent from the client to the SIP proxy server. Stickiness by Call-ID is particularly important for call stateful services that use the Call-ID to identify current SIP sessions and make decisions based on the content of the message.

If the CSS finds the SIP Call-ID in the SIP messages sent from the client to the SIP server, the CSS generates a key (hash value) based on the SIP Call-ID. The CSS uses the key to look up an entry in the sticky table. If the entry exists, the CSS sends the client to the sticky server indicated by the table entry. If the entry does not exist, the CSS creates a new sticky entry, hashes the SIP Call-ID value into a key, and saves the key in the entry.

The CSS supports the following SIP methods:

- INVITE - Indicates that the user or service is being invited to participate in a SIP session
- ACK - Confirms that the client has received a final reponse to an INVITE request
- OPTIONS - The server is being queried about its capabilities
- BYE - Indicates to the server that the client wants to release the call
- CANCEL - Cancels a pending request with the same Call-ID, To, From, and Cseq header field values, but does not affect a completed request
- REGISTER - Registers the address listed in the To header field with a SIP server

Configuration Requirements and Restrictions

The following requirements and restrictions apply to SIP load-balancing configurations on a CSS:

- The CSS supports SIP over UDP only.
- If you want UDP responses from the SIP proxy server to return to the client through the CSS and be NATed, configure destination services in the source group. Destination services NAT the client IP address to the CSS VIP, forcing return packets from the server to flow through the CSS and back to the client.
- When you enter the **application sip** content configuration mode command, the CSS automatically configures the SIP port as 5060 and the protocol as UDP. If you remove the SIP port from your configuration, the activation of the SIP content rule fails and the CSS sends an error message explaining the reason for the failure. To ensure that the port is configured, enter the **show rule** command.
- The **application sip** command is not compatible with the **url** command and the **url**, **urlhash**, **domain**, and **domainhash** load-balancing methods.

- By default, the CSS sets up flows for SIP. You can disable SIP flows using the **flow-state 5060 udp flow-disable nat-enable** global configuretion mode command. Doing so prohibits the concept of different call-IDs in the same flow. Stickiness behaves the same as in a flow-enabled configuration. To restore SIP flows, enter the **no flow-state 5060 udp** command.

SIP Load Balancing Configuration Quick Start

Table 11-5 provides a quick overview of the steps required to configure the SIP load balancing feature on each CSS in a multisite configuration. Each step includes the CLI command required to complete the task. For a complete description of each feature and all the options associated with the CLI commands, see the sections following the table.

Table 11-5 SIP Configuration Quick Start

Task and Command Example

1. Enter global configuration mode.
`# config`
`(config)#`

2. Configure a service for the SIP proxy server. For details on configuring services, see Chapter 3, Configuring Services. For example:
`(config)# service sipServer`
`(config-service[sipServer])# ip address 192.168.2.3`
`(config-service[sipServer])# active`

3. Configure an owner. For details on configuring an owner, see Chapter 9, Configuring Owners.
`(config)# owner sipOwner`
`(config-owner[sipOwner])#`

4. Create a SIP content rule for the owner. For details on configuring a content rule, see Chapter 10, Configuring Content Rules.
`(config-owner[sipOwner])# content sipRule`
`(config-owner-content[sipOwner-sipRule])#`

Table 11-5 SIP Configuration Quick Start (continued)

Task and Command Example

5. Configure the following commands on the content rule. For information about the **application sip** command, see the "Specifying an Application Type" section in Chapter 10, Configuring Content Rules. For information about the **advanced-balance sip-call-id** command, see the "Specifying an Advanced Load-Balancing Method for Sticky Content" section earlier in this chapter.
 	
Note When you enter the **application sip** command, the CSS automatically configures the protocol as UDP and the port number as 5060 if you have not already configured a protocol and a port.
 	
`(config-owner-content[sipOwner-sipRule])# vip address 192.168.128.191`
`(config-owner-content[sipOwner-sipRule])# application sip`
`(config-owner-content[sipOwner-sipRule])# advanced-balance sip-call-id`
`(config-owner-content[sipOwner-sipRule])# add service sipServer`
`(config-owner-content[sipOwner-sipRule])# active`

6. If your application requires that the client receive the server response from the VIP (CSS), then you need to configure a source group. In this case, source groups work only if the server is configured to use the "received=" field. Otherwise, you can skip this step. For details on configuring a source group, see Chapter 5, Configuring Source Groups for Services.
 	
`(config)# group sipGroup`
`(config-group[sipGroup])# vip address 192.168.1.228`
`(config-group[sipGroup])# add destination service sipServer`
`(config-group[sipGroup])# active`

7. (Recommended) Use the **show rule sticky, show sticky-table all-sticky**, **show sticky-table call-id-sticky**, and the **show sticky-stats** commands to verify your SIP configuration.
 	
`# show rule sticky`
`# show sticky-table all-sticky`
`# show sticky-table call-id-sticky`
`# show sticky-stats`

The running-config example below shows the results of entering the commands described in Table 11-5.

```
!************************** CIRCUIT **************************
circuit VLAN1

  ip address 192.168.1.191 255.255.255.0
  ip address 192.168.2.191 255.255.254.0
!************************** SERVICE **************************
service sipServer
  ip address 192.168.2.3
  active

!************************** OWNER ****************************
owner sipOwner

  content sipRule
    vip address 192.168.128.191
    protocol udp
    port 5060
    application sip
    advanced-balance sip-call-id
    add service sipServer
    active
!************************** GROUP ****************************
group sipGroup
  vip address 192.168.1.228
  add destination service sipServer
  active
```

Showing Sticky Attributes

To display the sticky attributes for a content rule, use the **show rule** command with the **sticky** option. The syntax for the **show rule** command is:

show rule {*owner_name* {*content_rule_name*
{**acl**|**all**|**dns**|**header-field**|**hot-list**|**services**|**statistics**|**sticky**}}}

This command is available in SuperUser, User, global configuration mode, or owner configuration mode.

For example, enter:

```
(config)# show rule sipOwner sipRule sticky
```

For example, in content configuration mode, enter:

```
(config-owner-content[sipOwner-sipRule])# show rule sticky
```

Table 11-6 describes the fields in the **show rule sticky** command output.

Table 11-6 Field Descriptions for the show rule sticky Command Output

Field	Description
Balance	The load-balancing algorithm for the content rule. The possible values are: • **ACA** - Arrowpoint Content Awareness algorithm. The CSS correlates content request frequency with the server's cache sizes to improve cache hit rates for that server. • **destip** - Destination IP address division. The CSS directs all client requests with the same destination IP address to the same service. • **domain** - Domain name division. The CSS uses the domain name in the request URI to direct the client request to the appropriate service. • **domainhash** - Internal CSS hash algorithm based on the domain string. The CSS uses the algorithm to hash the entire domain string. Then, the CSS uses the hash result to choose the server.

Table 11-6 Field Descriptions for the show rule sticky Command Output (continued)

Field	Description
Balance (continued)	• **leastconn** - Least connections. The CSS chooses a running service that has the least number of connections. • **roundrobin** - Roundrobin algorithm (default). • **srcip** - Source IP address division. The CSS directs all client requests with the same source IP address to the same service. • **url** - URL division. The CSS uses the URL (omitting the leading slash) in the redirect URL to direct the client requests to the appropriate service. • **urlhash** - Internal CSS hash algorithm based on the URL string. The CSS uses the algorithm to hash the entire URL string. The CSS uses the hash result to choose the server. • **weightedrr** - Weighted roundrobin algorithm. The CSS uses the roundrobin algorithm but weighs some services more heavily than others, depending on the weight configured on the service. You can configure or change the weight of a service when you add it to the rule. The content rule-configured service weight overrides the service-configured weight only for that content rule.

Table 11-6 Field Descriptions for the show rule sticky Command Output (continued)

Field	Description
Advanced Balance	The advanced load-balancing method for the content rule, including stickiness. The possible values are: • **arrowpoint-cookie** - Enables the content rule to stick the client to the server based on the unique service identifier information of the selected server in the cookie. • **cookies** - Enables the content rule to stick the client to the server based on the configured string found in the HTTP cookie header. You must specify a port in the content rule to use this option. The CSS then spoofs the connection.

Table 11-6 Field Descriptions for the show rule sticky Command Output (continued)

Field	Description
Advanced Balance (continued)	• **cookieurl** - This is the same as the **advanced-balance cookies** option, but if the CSS cannot find the cookie header in the HTTP packet, this type of failover looks up the URL extensions (that is, the portion after the "?" in the URL) based on the same string criteria. You can use this option with any Layer 5 HTTP content rule. • **none** - Disables the advanced-balancing method for the rule. This is the default setting. • **sip call-ID** - Enables the content rule to stick a client to a server based on Session Initiation Protocol (SIP) session Call-ID. The application type must be SIP for the content rule and the protocol must be UDP. • **sticky-srcip** - Enables the content rule to stick a client to a server based on the client IP address, also known as Layer 3 stickiness. You can use this option with Layer 3, 4, or 5 content rules. • **sticky-srcip-dstport** - Enables the content rule to stick a client to a server based on both the client IP address and the server destination port number; also known as Layer 4 stickiness. You can use this option with Layer 4 or 5 content rules. • **ssl** - Enables the content rule to stick the client to the server based on the Secure Socket Layer (SSL) version 3 session ID assigned by the server. The application type must be SSL for the content rule. You must specify a port in the content rule to use this option. The CSS then spoofs the connection.

Table 11-6 Field Descriptions for the show rule sticky Command Output (continued)

Field	Description
Advanced Balance (continued)	• **url** - Enables the content rule to stick a client to a server based on a configured string found in the URL of the HTTP request. You must specify a port in the content rule to use this option. The CSS then spoofs the connection.
Sticky Mask	The subnet mask used for stickiness. The default is 255.255.255.255.
Sticky Inactivity timeout	The inactivity timeout period on a sticky connection for a content rule before the CSS removes the sticky entry from the sticky table. The range is from 0 to 65535 minutes. The default value is 0, which means this feature is disabled.
Sticky No Cookie Found Action	The action the CSS should take for a sticky cookie content rule when it cannot locate the cookie header or the specified cookie string in the client request. The possible values are: • **loadbalance** - The CSS uses the configured balanced method when no cookie is found in the client request. This is the default setting. • **redirect "URL"** - The CSS redirects the client request to a specified URL string when no cookie is found in the client request. When using this option, you must also specify a redirect URL. Enter the redirect URL as a quoted text string from 0 to 64 characters. • **reject** - The CSS rejects the client request when no cookie is found in the request. • **service name** - The CSS sends the no cookie client request to the specified service when no cookie is found in the request.

Table 11-6 Field Descriptions for the show rule sticky Command Output (continued)

Field	Description
Sticky Server Down Failover	The action that the CSS should take when a sticky string is found but the associated service has failed or is suspended. The possible values are: • **Balance** - The failover method uses a service based on the configured load balancing method (default). • **Redirect** - The failover method uses a service based on the currently configured redirect string. If a redirect string is not configured, the load balancing method is used. • **Reject** - The failover method rejects the content request. • **Sticky-srcip** - The failover method uses a service based on the client IP address. This is dependent on the sticky configuration. • **Sticky-srcip-dstport** - The failover method uses a service based on the client IP address and the server destination port. This is dependent on the sticky configuration.
ArrowPoint Cookie Path	The pathname where you want to send the cookie. The default path of the cookie is "/".
ArrowPoint Cookie Expiration	The expiration time that the CSS compares with the time associated with the cookie. If you do not set an expiration time, the cookie expires when the client exits the browser.
ArrowPoint Cookie CSS/Browser Expired	Indicates whether the **arrowpoint-cookie browser-expire** command is enabled to allow the browser to expire the cookie based on the expiration time. If the command is enabled, the field displays "Browser" in place of "CSS." The default is "CSS."

Table 11-6 Field Descriptions for the show rule sticky Command Output (continued)

Field	Description
ArrowPoint Cookie Service	Specifies whether the **arrowpoint-cookie expire-services** command has been entered to expire service information when the cookie expires before sending a new cookie. By default, when the cookie expires, the CSS sends a new cookie with the server information from the expired cookie.
ArrowPoint Cookie Advanced	Specifies whether the **advanced-balance arrowpoint-cookie** command has been entered to enable the content rule to stick the client to the server based on the unique service identifier of the selected server in the arrowpoint cookie.
ArrowPoint Cookie Format	Specifies the format of the arrowpoint-cookie expiration time, whether the RFC 2822-compliant format is enabled or disabled. The **arrowpoint-cookie rfc2822-compliant** command configures the arrowpoint-cookie expiration time syntax to be RFC 2822-compliant. This command causes the arrowpoint-cookie expiration time syntax to be only three-character days of the week (for example, “Tue” rather than “Tues”) and to capitalize only the first character of the month (for example, “Jan” rather than “JAN”).

Table 11-6 Field Descriptions for the show rule sticky Command Output (continued)

Field	Description
String Match Criteria	The string criteria to derive string results and the method to choose a destination server for the result. The string result is a sticky string in the cookie header, URL, or URL extension based on a sticky type being configured. See the following fields.
String Range	The starting and ending byte positions within a cookie, URL, or URL extension from a client. By specifying the range of bytes, the CSS processes the information located only within the range. • The range is from 1 to 1999. The default starting byte position is 1. • The range is from 2 to 2000. The default ending byte position is 100.
String Prefix	The string prefix located in the sticky range. If you do not configure the string prefix, the string functions start from the beginning of the cookie, URL, or URL extension, depending on the sticky type. If the string prefix is configured but is not found in the specified sticky range, load balancing defaults to the roundrobin method. The default has no prefix (“”).
String Eos-Char	The ASCII characters as the delimiters for the sticky string.
String Ascii-Conversion	Specifies whether to enable or disable the ASCII conversion of Escape-sequence special characters within the specified sticky range before applying any processing to the string. By default, ACSII conversion is enabled.
String Skip-Len	The number of bytes to skip after the end of the prefix to find the string result. The default is 0. The range is from 0 to 64.

Table 11-6 Field Descriptions for the show rule sticky Command Output (continued)

Field	Description
String Process-Len	The number of bytes, after the end of the prefix designated by the **string prefix** command and skipping the bytes designated by the **string skip-length** command, that the string operation will use. The range is from 0 to 64. The default is 0.
String Operation	The method to choose a destination server for a string result as derived from the settings of the string criteria commands. The possible values are: • **match-service-cookie** - Choose a server by matching a service cookie in the sticky string. This is the default setting. When a match is not found, the server is chosen by using the configured balance method (for example, roundrobin). This is the default method. • **hash-a** - Apply a basic hash algorithm on the hash string to generate the hash key. • **hash-crc32** - Apply the CRC32 algorithm on the hash string to generate a hash key. • **hash-xor** - Exclusive OR (XOR) each byte of the hash string to derive the final hash key.
Location-Cookie	The format (NAME=VALUE) of the location cookie string.
Location-Cookie Expiration	The expiration date and time (dd:hh:mm:ss) of the location cookie. This value tells the client browser the time the cookie expires.
Cookie-Domain	A domain name for the location cookie (for example, .site.com). The cookie domain name allows your browser to send the cookie back to any site that ends with the domain name that you specify.

Showing Sticky Table Configurations

The **show sticky-table** command displays the contents of the CSS sticky table based on the advanced load-balancing method for a content rule. Use the following **show sticky-table** commands from any mode to display the sticky information contained in the CSS sticky table:

- **show sticky-table all-sticky** - Displays all Layer 3, Layer 4, SIP Call-ID, SSL, and WAP MSISDN sticky entries
- **show sticky-table l3-sticky** - Displays the Layer 3 entries
- **show sticky-table l4-sticky** - Displays the Layer 4 entries
- **show sticky-table sip-callid-sticky** - Displays the SIP Call-ID entries
- **show sticky-table ssl-sticky** - Displays the SSL entries
- **show sticky-table wap-sticky** - Displays the WAP MSISDN entries

To display sticky configurations for content, use the **show rule sticky** command in content mode. For details on the **show rule sticky** command, see the "Showing Sticky Attributes" section.

To display all sticky entries contained in the CSS sticky table, enter:

```
(config)# show sticky-table all-sticky
```

Table 11-7 describes all of the fields in the **show sticky-table all-sticky** command output.

Table 11-7 Field Descriptions for the show sticky-table all-sticky Command Output

Field	Description
All Sticky List on Slot *n*, Subslot *n*	Identifies the slot and subslot numbers of the SP in the CSS. If there are multiple SPs, the sticky table information is divided by slot number.
Entries for Page	Indicates the number of the page of information from the sticky table. The show screen displays a maximum of 100 sticky table entries for a page. The default is page 1.
Entry Number	The row number of the entry displayed from the sticky table.

Table 11-7 Field Descriptions for the show sticky-table all-sticky Command Output (continued)

Field	Description
Hash Value	A key generated by the CSS for the different sticky data types. The hash value is a representation of the client-specific information inserted into the sticky table, functioning as an index or entry key into the sticky table. The CSS generates the following hash values for the various sticky data types: • Layer 3 - Source IP address • Layer 4 - Combination of source IP address and destination port • SIP - Session Initiation Protocol (SIP) Call-ID (CID) • SSL - SSL version 3.0 session ID (SID) • WAP - MSISDN header field
Rule Index	A CSS-assigned unique numeric index for the rule. This is the index displayed using the **show rule summary** command.
Rule State	The state of the rule: ACT (active) or SUSP (suspended).
Srv Index	The CSS-assigned unique numeric index for the service. This is the index displayed in the **show service summary** command.
Srv State	The state of the service. The State field displays the service as Alive, Dying, Down, or Suspended.
Time (Sec) Elapsed	Indicates the elapsed time since the entry in the sticky table was last referenced and has been idle since that point in time. The counter starts at 0 and increments until the sticky table entry is used again.
Hit Cnt	The number of times the CSS received a transaction from the client for the entry in the CSS sticky table.

Table 11-7 Field Descriptions for the show sticky-table all-sticky Command Output (continued)

Field	Description
Col Cnt	The number of times the CSS received a transaction from different clients for an entry in the CSS sticky table with the same hash value. This field is used only with the **show sticky-table ssl-sticky** command.
Elem Type	The sticky type associated with the content rule. The possible element types include: • Layer 3 • Layer 4 • SIP - Session Initiation Protocol (SIP) Call-ID (CID) • SSL - SSL version 3.0 session ID (SID) • WAP - MSISDN header field
Inact Cfg (Min)	The inactivity timeout period configured for the content rule associated with the entry. This field indicates the length of idle time the sticky entry is held in the sticky table. A value of 0 (the default, which means the feature is disabled) indicates that the entry is not timed out of the sticky table. The CSS removes the entry from the sticky table if it is the least used entry in the table, the sticky table becomes full, and a new entry needs to be added to the table.
Total Number of Entries Found	The total number of the queried entries found in the sticky table. This total value can also be based on a specific sticky data type (Layer 3, Layer 4, SIP, SSL, or WAP).

Showing Layer 3 Sticky Table Information

Use the **show sticky-table l3-sticky** command to display the Layer 3 sticky entries contained in the CSS sticky table. Layer 3 sticks a user to a server based on the source IP address.

The syntax for this global configuration mode command is:

show sticky-table l3-sticky [**page** {*value*}|**ipaddress** {*ip_address sticky_mask*}]

The **show sticky-table l3-sticky** command supports the following options and variables:

- **page** *value* - Shows Layer 3 sticky entries for a specific page in the sticky table, at 100 entries per page. Enter a value from 1 to 5000 to select the page of entries you want to view from the sticky table. To determine the page you want to display, take the Total Number of Used Entries Found value in the **show sticky-stats** command output and divide by 100 (entries per page).
- **ipaddress** *ip_address sticky_mask* - The IP address of the Layer 3 sticky table entry to be shown. Enter the IP address in dotted-decimal notation (for example, 192.168.2.5). Specify the sticky mask from the content rule for this IP address in dotted-decimal notation (for example, 255.255.255.0). The default sticky mask of a content rule is 255.255.255.255.

For example, to display Layer 3 sticky entries from page 60 in the sticky table, enter:

```
(config)# show sticky-table l3-sticky page 60
```

For example, to display Layer 3 sticky entries from a specific IP address and sticky mask in the sticky table, enter:

```
(config)# show sticky-table l3-sticky ipaddress 192.168.2.5
255.255.255.255
```

See Table 11-7 for a description of the fields in the **show sticky-table l3-sticky** command output.

Showing Layer 4 Sticky Table Information

Use the **show sticky-table l4-sticky** command to display the Layer 4 sticky entries contained in the CSS sticky table. Layer 4 sticky functions identically to Layer 3 sticky, except that it sticks a user to a server based on a combination of source IP address and destination port.

The syntax for this global configuration mode command is:

show sticky-table l4-sticky [**page** {*value*}|**ipaddress** {*ip_address sticky_mask*} {*port*}]

The **show sticky-table l4-sticky** command supports the following options and variables:

- **page** *value* - Shows Layer 4 sticky entries for a specific page in the sticky table, at 100 entries per page. Enter a value from 1 to 5000 to select the page of entries you want to view from the sticky table. To determine the page you want to display, take the Total Number of Used Entries Found value in the **show sticky-stats** command output and divide by 100 (entries per page).
- **ipaddress** *ip_address sticky_mask* - The IP address of the Layer 3 sticky table entry to be shown. Enter the IP address in dotted-decimal notation (for example, 192.168.2.5). Specify the sticky mask from the content rule for this IP address in dotted-decimal notation (for example, 255.255.255.0). The default sticky mask of a content rule is 255.255.255.255.
- *port* - Destination port of the entry to be shown.

For example, to display Layer 4 sticky entries from a specific IP address and sticky mask in the sticky table for destination port 80, enter:

```
(config)# show sticky-table l4-sticky ipaddress 192.168.2.5
255.255.255.255 80
```

See Table 11-7 for a description of the fields in the **show sticky-table l4-sticky** command output.

Showing SIP Call-ID Sticky Table Information

Use the **show sticky-table sip-callid-sticky** command to display the entries contained in the sticky table based on session Call-ID. Call-ID is a unique call identifier contained in the SIP messages sent from the client to the SIP server.

The syntax for this global configuration mode command is:

show sticky-table sip-callid-sticky [**page** {*value*}|**Call-ID** {*sip_callid*}]

The **show sticky-table sip-callid-sticky** command supports the following options and variables:

- **page** *value* - Shows SIP Call-ID sticky entries for a specific page in the sticky table, at 100 entries per page. Enter a value from 1 to 5000 to select the page of entries you want to view from the sticky table. To determine the page you want to display, take the Total Number of Used Entries Found value in the **show sticky-stats** command output and divide by 100 (entries per page).
- **Call-ID** *sip_callid* - Specifies a specific Call-ID to display from the sticky table. You can locate the Call-ID number by performing a packet trace.

For example, to display sticky entries for a specific Call-ID in the sticky table, enter:

```
(config)# show sticky-table sip-calliD-sticky 12345600@here.com
```

See Table 11-7 for a description of the fields in the **show sticky-table sip-callid-sticky** command output.

Showing SSL Sticky Table Information

Use the **show sticky-table ssl-sticky** command to display the SSL entries contained in the sticky table.

The syntax for this global configuration mode command is:

show sticky-table ssl-sticky [**rule** {*index*}{**page** {*value*}}|**time** {*number*} {**page** {*value*}}|**sid** {*text*}|**collision**|**page** {*value*}]

The **show sticky-table ssl-sticky** command supports the following options and variables:

- **rule** *index* - Displays the SSL entries in the sticky table for the content rule. Enter the index number for the SSL sticky content rule. You can locate the index number for the content rule in the **show rule summary** command.
- **page** *value* - Shows entries for a specific page in the sticky table, at 100 entries per page. Enter a value from 1 to 5000 to select the page of entries you want to view from the sticky table. To determine the page you want to display, take the Total Number of Used Entries Found value list in the **show sticky-stats** command output and divide by 100 (entries per page).
- **time** *number* - Specifies the window of elapsed time (in seconds) in which to display entries from the sticky table. All sticky entries in the table that were referenced within the specified time appear in the show output. Enter the time in seconds.
- **sid** *text* - Displays the entries in the sticky table based on SSL Session ID (SID). Enter the SID value as a hexadecimal ASCII string without the 0x prefix. You can locate the SID number by performing a packet trace.
- **collision** - Displays the entries in the sticky table that have a collision count (Col Cnt) greater than 0.

For example, to show SSL entries in the sticky table based on content rule index number and page number in the sticky table, enter:

```
(config)# show sticky-table ssl-sticky rule 4 page 33
```

See Table 11-7 for a description of the fields in the **show sticky-table ssl-sticky** command output.

Showing WAP Sticky Table Information

Use the **show sticky-table wap-sticky** command to display the entries contained in the sticky table based on the MSISDN header field. MSISDN is the header field for wireless clients using the Wireless Application Protocol (WAP).

The syntax for this global configuration mode command is:

show sticky-table wap-sticky [**page** {*value*}|**msisdn** {*msisdn_header*}]

The **show sticky-table wap-sticky** command supports the following options and variables:

- **page** *value* - Shows MSISDN sticky entries for a specific page in the sticky table, at 100 entries per page. Enter a value from 1 to 5000 to select the page of entries you want to view from the sticky table. To determine the page you want to display, take the Total Number of Entries Found value list in the **show sticky-table all-sticky** command output or the **show sticky-stats** command output and divide by 100 (entries per page).
- **msisdn** *msisdn_header* - Specifies the MSISDN header field to display from the sticky table. Enter the *msisdn_header* as a text string. The MSISDN header field typically contains the wireless phone numbers. You can locate the MSISDN header by performing a packet trace.

For example, to show MSISDN sticky entries in the sticky table based on MSISDN header, enter:

```
(config)# show sticky-table wap-sticky msisdn 6079979410
```

See Table 11-7 for a description of the fields in the **show sticky-table wap-sticky** command output.

Showing Sticky Connection Statistics

The **show sticky-stats** command displays a summary of sticky connection statistics for the CSS.

To display sticky configurations for content, you can also use the **show rule sticky** command in content mode. For details on the **show rule sticky** command, see the "Showing Sticky Attributes" section.

For example:

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

Table 11-8 describes the fields in the **show sticky-stats** command output.

Table 11-8 Field Descriptions for the show sticky-stats Command Output

Field	Description
Total Number of New Sticky Entries	The total number of unique entries used in the sticky table. Every time an entry is created in the sticky table that is unique, the counter increments.
Total Number of Sticky Table Hits	The total number of times the CSS received a request from a client that matches an entry in the CSS sticky table. Every time the CSS receives a client entry and the hash value exists in the sticky table, the counter increments. The CSS performs a lookup in the sticky table. If no match is found, the entry is considered to be a new sticky count.
Total Number of Sticky Rejects (No Entry)	The total number of times that the CSS rejects sticky requests. When the sticky table becomes full and none of the entries have expired from the sticky table, the CSS rejects subsequent sticky requests.
Total Number of Sticky Collisions	The total number of times the CSS receives a request from a client for an entry in the CSS sticky table with the same hash value and the load-balancing server cannot be resolved.
Total Number of Available Sticky Entries	The total number of available sticky entries in the sticky table.

Table 11-8 Field Descriptions for the show sticky-stats Command Output (continued)

Field	Description
Total Number of Used Sticky Entries	The total number of entries currently used in the sticky table. The CSS supports a 128K sticky table (with 288 MB of CPU memory) or a 32K sticky table (with 144 MB of CPU memory).
Total Number of L3 Sticky Entries	The total number of Layer 3 sticky entries in the sticky table.
Total Number of L4 Sticky Entries	The total number of Layer 4 sticky entries in the sticky table.
Total Number of SSL Sticky Entries	The total number of SSL session ID sticky entries in the sticky table.
Total Number of WAP Sticky Entries	The total number of WAP MSISDN header sticky entries in the sticky table.
Total Number of SIP Sticky Entries	The total number of SIP Call-ID sticky entries in the sticky table.