



## Configuring HTTP Compression

---

You can enable the CSS to compress HTTP text data returned to clients through the SSL compression module. Compressing HTTP response data improves performance for clients using dialup and other slow speed links and reduces the amount of bandwidth that the server data consumes. By having the CSS compress HTTP response data, it offloads the server from performing this task.

This chapter describes how to configure HTTP data compression on a CSS service. The information in this chapter applies to all CSS models containing SSL functionality.

This chapter contains the following major sections:

- [CSS Compression Overview](#)
- [Compression-Only Service Quick Start](#)
- [Configuring Compression on the CSS](#)
- [Configuring TCP Options for a Compression-Only Service](#)
- [Displaying Compression Information](#)

# CSS Compression Overview

When you configure compression on a CSS service, the CSS compresses data in HTTP GET responses from the server. The CSS does not compress HTTP requests from clients or the HTTP headers in the server responses. For SSL terminated connections in which the data is encrypted, the CSS cannot compress encrypted data. It must decrypt the SSL data to cleartext data before compressing it. Then, the CSS reencrypts the compressed data before returning it to the client.

The CSS can compress the data on HTTP version 1.1 or higher connections. The CSS does not compress HTTP version 1.0 responses and passes the data without compression.

HTTP 1.1 allows different encoding of the data. The encoding values that the CSS supports are:

- gzip, the file format for compression described in RFC1952
- deflate, the data format for compression described in RFC1951
- identity, in which no compression occurs

A client may advertise its decoding capabilities through the Accept-Encoding field in HTTP request headers. The CSS uses the compression tokens in the Accept-Encoding field to determine the type of compression encoding to use on the response. If the Accept-Encoding field contains a list of compression tokens, the CSS has the following preference for applying compression:

1. deflate
2. gzip
3. identity

The CSS also allows you to configure the type of encoding for compression when there is no Accept-Encoding field in the HTTP request header. When the server returns a GET response with response data, the response indicates the compression encoding for the content.

The CSS supports compression on specific file extensions and content types. The supported file extensions are:

- .asp
- .aspx
- .css

- .htm
- .html
- .jhtml
- .js
- .jsp
- .php
- .shtml

The supported content types are:

- application/x-javascript
- text/plain
- text/html
- text/css

Also, the GET “/” condition is eligible for compression.

**Note**

---

For information on compression compatibility for your browser, refer to the website for the browser.

---

Compression on the CSS occurs on an SSL compression (SSL-C) module. When you enable compression on a CSS containing an SSL-C module, the module acts as TCP endpoints for client and server connections and compresses the returned data from the server.

This section includes the following topics on the SSL-C module and compression traffic through the CSS:

- [SSL Compression Hardware](#)
- [Compression Traffic through the CSS](#)

## SSL Compression Hardware

The SSL-module type or the CSS 11501 platform determines whether hardware or software performs compression. An SSL-C module performs compression through hardware. It contains a special chipset to optimize the compression function. In addition to compression, it can perform the full range of SSL functions, and can perform compression and SSL at the same time. The CSS performs hardware compression through an SSL-C module in a CSS 11503 or CSS 11506 chassis, or integrated SSL compression on the CSS 11501. From hereafter, this chapter refers to the SSL compression integrated hardware as the SSL-C module.

An SSL module that does not have integrated compression performs compression through software.



### Note

There is a significant difference between hardware or software compression performance capability. We highly recommend that you do not use compression on an SSL module that does not have integrated hardware compression. This module performs compression through software, but it is not optimized for performance.

Use [Table 9-1](#) to determine how the CSS performs compression based on the module type or the CSS 11501 platform.

**Table 9-1 CSS Compression Method**

		CSS Platform and SSL-Module Type				
		11501	11501S	11501S-C	11503 or 11506 with an SSL module	11503 or 11506 with an SSL-C module
Compression method	Software	Not available	Yes	No	Yes	No
	Hardware	Not available	No	Yes	No	Yes

When you want compression to occur as part of a CSS SSL configuration, you configure compression on the same service for SSL termination or initiation. This combined service of SSL and compression follow the same flow path, allowing SSL and compression to occur on the same module.

For a compression only service, you can optionally direct compression traffic to an SSL-C module by configuring its slot number. If you do not configure a slot for the compression traffic, the CSS always uses the SSL-C module in the lowest number slot for compression. If there is no SSL-C module in the CSS but there is a standard SSL module, the CSS uses the standard SSL module in the lowest number slot.

We recommend that you configure an SSL slot for a compression-only service. When the CSS has multiple SSL-C modules and all compression traffic goes to the first available SSL-C module, SSL throughput on that and on all SSL-C modules may decrease due to the load balancing of content. In this case, configuring a specific compression slot could ease any resource conflict with the SSL-C modules.

**Note**

---

On CSS 11501 with SSL compression and SSL traffic, there could be a decrease in SSL throughput when compression is enabled.

---

## Compression Traffic through the CSS

The CSS supports the following types of compression traffic:

- Compression only service
- Compression and SSL termination
- Compression and SSL initiation
- Compression and SSL termination with back-end SSL

This section contains the following topics:

- [HTTP Compression Only Service](#)
- [HTTP Compression with SSL Termination](#)
- [HTTP Compression with SSL Initiation](#)
- [HTTP Compression with SSL Termination and Back-end SSL](#)

## HTTP Compression Only Service

When traffic between a client and server does not require encryption, the CSS can use the SSL-C module to perform HTTP compression on the response data. The CSS receives cleartext data from the server and forwards compressed data to the client. For this compression configuration, you configure compression on a service and apply it to a content rule.

The following steps describes the HTTP compression flow through the CSS:

1. The client sends an HTTP GET request to the CSS.
2. For Layer 4 connections, the CSS directly forwards all client traffic to the SSL-C module.

For Layer 5 connections, the CSS performs a TCP handshake. When the CSS has received the GET request and performs Layer 5 load balancing, it forwards the Layer 5 handshake and the HTTP GET request to the SSL-C module.

3. The SSL-C module terminates the client-side TCP connection. HTTP compression will occur on this module.
4. The module establishes a backend TCP connection to the selected server and forwards the GET request. The SSL-C module acts as a TCP termination proxy for all further traffic.
5. When the server sends the GET response to the CSS, the CSS directs the response to the SSL-C module.
6. The module compresses the data and returns it to the client.

## HTTP Compression with SSL Termination

When the SSL-C module performs SSL termination on the client side connection, it decrypts data from the client and encrypts data from the server to the client. With the addition of compression, the module compresses the data from the server before encrypting and sending it to the client.

For this configuration, you configure a service with SSL termination and compression, and apply the service to a content rule. For information on configuring a service with SSL termination, see [Chapter 4, Configuring SSL Termination](#).

The following steps describe the SSL termination and HTTP compression flows through the CSS:

1. The client sends encrypted SSL data to the CSS.
2. The CSS forwards the data to an SSL-C module.
3. The SSL-C module terminates the client-side connection and decrypts the data into cleartext data.
4. The module performs HTTP header changes for compression.
5. The module initiates a TCP connection to the server and sends the request to the CSS.
6. The CSS receives the data and sets up a flow to the server and delivers the GET request.
7. When the server returns a response, the CSS directs the traffic to the SSL-C module for compression.
8. The module compresses the data and then encrypts it.
9. The module sends the encrypted compressed data to the CSS.
10. The CSS returns the encrypted compressed data to the client.

## HTTP Compression with SSL Initiation

When the SSL-C module performs SSL initiation, it encrypts data from the client and decrypts data from the server to the client. With the addition of compression, the module compresses the data from the server after decrypting and sending it to the client.

For this configuration, you configure SSL initiation and compression on the same service, and apply the service to a content rule. For information on configuring a service with SSL initiation, see [Chapter 6, Configuring SSL Initiation](#).

The following steps describes the SSL initiation and HTTP compression flows through the CSS:

1. The client sends cleartext data to the CSS.
2. The CSS forwards the data to an SSL-C module.
3. The SSL-C module terminates the client-side connection and performs HTTP header changes for compression.

4. The module initiates a back-end SSL connection to the server and encrypts the request.
5. The module initiates a TCP connection to the server and forwards the request to the CSS.
6. The CSS receives the data, sets up a flow to the server and delivers the GET request.
7. When the server returns the encrypted response, the CSS directs the traffic to the SSL-C module for compression.
8. The module decrypts the data to cleartext data and then compresses it.
9. The module sends the compressed data to the CSS.
10. The CSS returns the compressed cleartext data to the client.

## HTTP Compression with SSL Termination and Back-end SSL

When the SSL-C module performs SSL termination with back-end SSL, it decrypts data from the client and then reencrypts the data before sending it to the server. When the server returns encrypted data, the module decrypts this data and then reencrypts it before sending it to the client. With the addition of compression, the module compresses the decrypted data from the server before reencrypting and sending it to the client.

The following steps describes the SSL termination, back-end SSL, and HTTP compression flows through the CSS:

1. The client sends encrypted SSL data to the CSS.
2. The CSS forwards the data to an SSL-C module.
3. The SSL-C module terminates the client-side connection and decrypts the data into cleartext data.
4. The module performs HTTP header changes for compression.
5. The module initiates a back-end SSL connection to the server and encrypts the request.
6. The module initiates a TCP connection to the server and forwards the request to the CSS.
7. The CSS receives the data, sets up a flow to the server and delivers the GET request.

8. When the server returns the encrypted response, the CSS directs the traffic to the SSL-C module for compression.
9. The module decrypts the data to cleartext data.
10. The module compresses the data and then encrypts it.
11. The module sends the encrypted compressed data to the CSS.
12. The CSS returns the encrypted compressed data to the client.

## Compression-Only Service Quick Start

[Table 9-2](#) provides a quick overview of the steps required to configure a compression-only service and its associated content rule. Each step includes the CLI command required to complete the task. For a complete description of each feature and all the options associated with the CLI command, see the sections following [Table 9-2](#).

For information on configuring compression with an SSL termination or SSL initiation service and content rule, see [Chapter 2, SSL Configuration Quick Starts](#).

**Table 9-2 Compression-Only Service Quick Start**

---

### Task and Command Example

---

1. Enter global configuration mode.

```
# config
(config)#
```

---

2. Enter service configuration mode for the service in which you want to configure compression.

```
(config)# service server3
```

---

3. Assign the IP address for the service.

```
(config)# ip address 13.1.1.8
```

---

4. Configure the keepalive type for the service. By default, the service keepalive type is ICMP. To configure a keepalive type of none, enter:

```
(config-service[server3])# keepalive type none
```

---

**Table 9-2** *Compression-Only Service Quick Start (continued)***Task and Command Example**

5. Enable compression on the service.

```
(config-service[server3])# compress enable
```

6. (Recommended) By default, compression uses SSL-C module in the lowest slot number. To configure slot 3 to perform compression, enter:

```
(config-service[server3])# slot 3
```

For more information, see the [“Configuring an SSL Slot for a Compression-Only Service”](#) section.

7. After configuring compression on the service, activate the service.

```
(config-service[server3])# active
```

**Note** Before changing the configuration on an active service, you must suspend the service.

8. Configure the content rule for the service and activate the rule.

```
(config)# owner customer
(config-owner[customer])# content gzip
(config-owner-content[gzip])# vip address 192.168.77.40
(config-owner-content[gzip])# protocol tcp
(config-owner-content[gzip])# port 80
(config-owner-content[gzip])# url "/"
(config-owner-content[gzip])# add service server3
(config-owner-content[gzip])# active
```

The following running-configuration example shows the results of entering the commands in [Table 9-2](#).

```
!***** SERVICE *****
service server3
  ip address 13.1.1.8
  keepalive type none
  compress enable
  slot 3
  active

!***** OWNER *****
content gzip
  vip address 192.168.77.40
  protocol tcp
  port 80
```

```
url "/*"  
add service server3  
active
```

## Configuring Compression on the CSS

This section provides the following topics to configure HTTP compression:

- [Enabling Compression on a Service](#)
- [Disabling Compression on a Service](#)
- [Configuring an SSL Slot for a Compression-Only Service](#)
- [Configuring the Preferred Compression Algorithm](#)
- [Configuring the Compression Encoding Type When the Accept-Encode Field is Omitted](#)
- [Configuring Compression Data Type](#)

When you finish configuring compression on the service, you can activate the service. If you need to modify an active service, suspend the service before making any changes.



---

**Note**

You cannot configure compression with Adaptive Session Redundancy (ASR) on a service. The CSS does not allow you to configure a redundant index on a content rule that has services with compression enabled or to enable compression on a service in a rule that is configured with a redundant index.

Before changing the configuration on an active service, you must suspend the service.

---

## Enabling Compression on a Service

By default, compression is not enabled on a service. You must enable compression on a service to allow the compression of HTTP response data to the client. To configure the service for compression, use the **compress enable** command in service configuration mode. The syntax for this command is:

```
compress enable
```

For example, enter:

```
(config-service[server3])# compress enable
```

## Disabling Compression on a Service

You can reset a service to its default state of performing no compression on HTTP response data. When you disable compression on a service, the compression configuration is not removed on the service.

To disable compression, use the **compress disable** command. If the service is active, you must suspend it before changing this setting.

For example, enter:

```
(config-service[server3])# compress disable
```

## Configuring an SSL Slot for a Compression-Only Service

In a CSS 11503 and 11506, the SSL-C module performs the compression of HTTP response data to the client. If HTTP compression is part of a service configuration for SSL termination or SSL initiation, the compression traffic uses the slot number configured for the SSL service.

For a compression-only service, you can optionally direct compression traffic to an SSL-C module by configuring its slot number. We recommend that you configure an SSL-C slot number for compression traffic. If you do not configure a slot for compression, the CSS always uses the SSL-C module in the lowest number slot for compression. When the CSS has multiple SSL modules and all compression traffic goes to the first available SSL module, SSL throughput on that

and on all SSL modules may decrease due to the load balancing of content. In this case, configuring a specific compression slot could ease any resource conflict with the SSL modules.

To configure a slot where you want compression to occur, use the **slot** command in service configuration mode. The syntax of the command is:

**slot** *number*

The *number* variable is the slot of the SSL-C module.

For example, to configure the service for the SSL-C module in slot 3, enter:

```
(config-service[server3])# slot 3
```

**Note**

---

If the service is active, you must suspend it before changing this setting.

---

## Configuring the Preferred Compression Algorithm

When the CSS receives a client HTTP request for a compression service, it processes the header to determine the type of compression to use. The Accept-Encode field in the HTTP header contains the compression encoding type. By default, the CSS uses the compression algorithm as advertised in the Accept-Encoding field. However, if the Accept-Encoding field does not exist in the HTTP request, then the CSS uses the compression encoding type configured with the **compress accept-omit** command. For more information on this command, see the [“Configuring the Compression Encoding Type When the Accept-Encode Field is Omitted”](#) section.

To configure the CSS to prefer a compression encoding type provided by the Accept-Encode field or the **compress accept-omit** command, use the **compress encode** command. The CSS compares its preferred setting with the entry in the Accept-Encode field or the **compress accept-omit** command and determines how the flow will be compressed or if the flow will be bypassed for compression.

**Table 9-3** lists the compression encoding type that the CSS uses based on the Accept-Encoding field and preferred compression encoding setting through the **compress encode** command. Bypass indicates that the CSS bypasses the flow for compression. For example, if the Accept-Encoding field contains the identity entry for no compression, the CSS bypasses the flow for compression regardless

of the setting for the **compress encode** command. Also note that the **force-gzip** and **force-deflate** settings overrides the entries in the Accept-Encode field except identity.

**Table 9-3** Compression Type Based on the Accept-Encode Field and the compress encode command

		compress encode command setting				
		gzip	deflate	auto (default)	force-gzip	force-deflate
Compression type in the Accept-Encode field	identity	bypass	bypass	bypass	bypass	bypass
	deflate	bypass	deflate	deflate	gzip	deflate
	gzip	gzip	bypass	gzip	gzip	deflate

Table 9-4 lists the compression encoding type that the CSS uses based on the **compress accept-omit** command and the preferred compression encoding setting through the **compress encode** command. For example, the default setting for the **compress accept-omit** command is identity to bypass compression. The CSS bypasses compression if the **compress encode** command is set to **auto**, **gzip**, or **deflate**. But it does not bypass compression for the **force-gzip** or **force-deflate** setting.

**Table 9-4** Compression Type Based on the compress accept-omit command and the compress encode command

		compress encode command setting				
		gzip	deflate	auto (default)	force-gzip	force-deflate
compress accept-omit command setting	identity (default)	bypass	bypass	bypass	gzip	deflate
	deflate	bypass	deflate	deflate	gzip	deflate
	gzip	gzip	bypass	gzip	gzip	deflate



**Note**

The CSS compresses only HTTP GET responses. Otherwise, the CSS marks the flow to bypass compression.

The syntax for the **compress encode** command is:

```
compress encode auto|deflate|gzip|force-deflate|force-gzip
```

The keywords for this command are:

- **auto** - (Default) The CSS uses the compression algorithm token in the Accept-Encoding field in the HTTP request from the client. From a list of tokens in the Accept-Encoding field, the CSS has the following preference for applying the compression algorithm:

1. deflate
2. gzip
3. identity

If the field does not contain a compression type or does not exist, the CSS uses the setting in the **compress accept-omit** command.

- **deflate** - The CSS prefers the deflate algorithm for compression and bypasses the flow for compression for the identity or gzip compression encoding type.
- **gzip** - The CSS prefers the gzip algorithm and bypasses the flow for compression for the identity or deflate compression encoding type.
- **force-deflate** - The CSS always uses the deflate algorithm for encoding, except if the Accept-Encoding field contains an identity entry.
- **force-gzip** - The CSS always uses the gzip algorithm for encoding, except if the Accept-Encoding field contains an identity entry.

**Note**

---

If the encoding for the **force-deflate** or **force-gzip** keyword does not match the capability of the client, the transaction may fail because the client is unable to process the response.

---

For example, to configure the CSS to prefer the deflate algorithm, enter:

```
(config-service[server3])# compress encode deflate
```

To reset the preferred encoding to its default value of auto, enter:

```
(config-service[server3])# compress encode auto
```

**Note**

---

If the service is active, you must suspend it before changing this setting.

---

## Configuring the Compression Encoding Type When the Accept-Encode Field is Omitted

By default, if the HTTP header does not contain the Accept-Encode field, the CSS bypasses the flow for compression. If the CSS receives an HTTP header that omits the Accept-Encoding field, you can configure a compression encoding type that the CSS compares with the preferred compression encoding setting of the **compress encode** command. See [Table 9-4](#) for the compression encoding type that the CSS uses based on the **compress accept-omit** command and the preferred compression encoding setting of the **compress encode** command.

To specify the compression encoding type for HTTP requests that do not include the Accept-Encoding field, use the **compress accept-omit** command. The syntax for this command:

```
compress accept-omit deflate|gzip|identity
```

The keywords are:

- **deflate** - The deflate compression encoding type
- **gzip** - The gzip compression encoding type
- **identity** - (Default) The cleartext identity encoding type that bypasses the flow for compression



---

**Note**

Verify that the client is capable of accepting the configured compression encoding type. Otherwise, unexpected results could occur including the failure of the transaction.

---

For example, to configure the deflate encoding type, enter:

```
(config-service[server3])# compress accept-omit deflate
```

To return the encoding type to its default setting of identity, enter:

```
(config-service[server3])# compress accept-omit identity
```



---

**Note**

If the service is active, you must suspend it before changing this setting.

---

## Configuring Compression Data Type

By default, the CSS used the ASCII alpha-numeric optimized Huffman code for traffic that is mostly text. To optimize the compression for different traffic types containing mostly numbers or mixed traffic, you can specify the Huffman code type. Use the **compress type** command. The syntax for this command is:

```
compress type ascii|default|numeric
```

The keywords are:

- **ascii** - (Default) ASCII alpha-numeric optimized Huffman code for traffic that is mostly text (for example, HTML, XML, TXT)
- **default** - Default Huffman code defined by the deflate algorithm for mixed traffic
- **numeric** - ASCII numeric optimized Huffman code for traffic that is mostly ASCII numbers (for example, 0 through 9)

For example, to configure the ASCII numeric optimized Huffman code for traffic, enter:

```
(config-service[server3])# compress type numeric
```

To return the Huffman code to its default setting of ascii, enter:

```
(config-service[server3])# compress type ascii
```

**Note**

---

If the service is active, you must suspend it before changing this setting.

---

## Configuring TCP Options for a Compression-Only Service

You can configure the TCP connections between the client or server and the SSL-C module for a compression-only service.

**Note**

---

Do not configure the **compress tcp** commands for an SSL service type of ssl-accel, backend, or init. The CSS will generate errors.

---

To configure the TCP connections, use the **compress tcp** command in service configuration mode. The topics in this section includes:

- [Configuring Client TCP Connection Timeout Values](#)
- [Configuring the Server TCP Connection Timeout Values](#)
- [Configuring the Acknowledgement Delay for TCP Connections](#)
- [Configuring the Nagle Algorithm for TCP Connections](#)
- [Configuring the TCP Buffering for TCP Connections](#)
- [Configuring the TCP Retransmission Timer](#)

## Configuring Client TCP Connection Timeout Values

The TCP connection between the CSS and a client is terminated when the specified time interval elapses. The TCP timeout functions enable you to have more control over the TCP connection between the CSS SSL module and a client. To configure the termination of a TCP connection with the client, see the following sections:

- [Configuring the Client TCP SYN Timeout Value](#)
- [Configuring the Client TCP Inactivity Timeout Value](#)

## Configuring the Client TCP SYN Timeout Value

The CSS SYN timer counts the delta between the CSS sending the SYN/ACK and the client replying with an ACK as the means to terminate the TCP three-way handshake. Use the **compress tcp virtual syn-timeout** *seconds* command in service configuration mode to specify a timeout value that the CSS uses to terminate a TCP connection with a client that has not successfully completed the TCP three-way handshake prior to transferring data.

Enter a TCP SYN inactivity timeout value in seconds, from 1 to 3600 (1 hour). The default is 30 seconds.



### Note

---

The connection timer should always be less than the retransmit termination time for new TCP connections.

---

For example, to configure a TCP SYN timeout of 30 minutes (1800 seconds), enter:

```
(config-service[server3])# compress tcp virtual syn-timeout 1800
```

To reset the TCP SYN timeout to the default of 30 seconds, enter:

```
(config-service[server3])# no compress tcp virtual syn-timeout
```

**Note**

---

If the service is active, you must suspend it before changing this setting.

---

## Configuring the Client TCP Inactivity Timeout Value

The TCP inactivity timeout begins once the CSS receives an ACK from the client to terminate the TCP three-way handshake. The inactivity timer resumes immediately following where the SYN timer stops, with regard to traffic flow. Use the **compress tcp virtual inactivity-timeout** *seconds* command in service configuration mode to specify a timeout value that the CSS uses to terminate a TCP connection with the client when there is little or no activity occurring on the connection.

Enter a TCP inactivity timeout value in seconds, from 0 (TCP inactivity timeout disabled) to 3600 (1 hour). The default is 240 seconds.

For example, to configure a TCP inactivity time of 30 minutes (1800 seconds), enter:

```
(config-service[server3])# compress tcp virtual inactivity-timeout 1800
```

To reset the TCP inactivity timer to the default of 240 seconds, enter:

```
(config-service[server3])# no compress tcp virtual inactivity-timeout
```

**Note**

---

If the service is active, you must suspend it before changing this setting.

---

## Configuring the Server TCP Connection Timeout Values

The TCP connection between the CSS and a server is terminated when the specified time interval elapses. The TCP timeout functions enable you to have more control over TCP connections between the CSS SSL module and a server.

To configure the termination of a TCP connection with the server, see the following sections:

- [Configuring the Server TCP SYN Timeout Value](#)
- [Configuring the Server TCP Inactivity Timeout Value](#)

### Configuring the Server TCP SYN Timeout Value

The TCP SYN timer counts the delta between the CSS initiating the back-end TCP connection by transmitting a SYN and the server replying with a SYN/ACK. Use the **compress tcp server syn-timeout** *seconds* command in service configuration mode to specify a timeout value that the CSS uses to end a TCP connection with a server that has not successfully completed the TCP three-way handshake prior to transferring data.

Enter a TCP SYN timeout value in seconds, from 1 to 3600 (1 hour). The default is 30 seconds.

**Note**

---

The connection timer should always be less than the retransmit termination time for new TCP connections.

---

For example, to configure a TCP SYN timeout of 30 minutes (1800 seconds), enter:

```
(config-service[server3])# compress tcp server syn-timeout 1800
```

To reset the TCP SYN timeout to the default of 30 seconds, enter:

```
(config-service[server3])# no compress tcp server syn-timeout
```

**Note**

---

If the service is active, you must suspend it before changing this setting.

---

## Configuring the Server TCP Inactivity Timeout Value

The TCP inactivity timeout begins once the CSS receives a SYN/ACK from the server. The inactivity timer resumes immediately following where the SYN timer stops, with regard to traffic flow. Use the **compress tcp server inactivity-timeout** *seconds* command in service configuration mode to specify a timeout value that the CSS uses to terminate a TCP connection with a server when there is little or no activity occurring on the connection.

Enter a TCP inactivity timeout value in seconds, from 0 (TCP inactivity timeout disabled) to 3600 (1 hour). The default is 240 seconds.

For example, to configure a TCP inactivity time of 30 minutes (1800 seconds), enter:

```
(config-service[server3])# compress tcp server inactivity-timeout 1800
```

To reset the TCP inactivity timer to the default of 240 seconds, enter:

```
(config-service[server3])# no compress tcp server inactivity-timeout
```

**Note**

If the service is active, you must suspend it before changing this setting.

## Configuring the Acknowledgement Delay for TCP Connections

By default, the acknowledgement delay on a client or server connection is 200 milliseconds (ms). You can disable or adjust the TCP timer length for delayed acknowledgements by using the **compress tcp virtualserver ack-delay** command in service configuration mode:

```
compress tcp virtualserver ack-delay value
```

The *value* variable is the timer length in milliseconds (ms) for delayed acknowledgements. The default value is 200. Enter a value from 0 to 10000. A value of 0 disables the acknowledgement delay in receiving traffic from the client or server.

For example, to set an acknowledgement delay of 400 ms for the client TCP connection, enter:

```
(config-service[server3])# compress tcp virtual ack-delay 400
```

To set an acknowledgement delay of 400 ms for the server TCP connection, enter:

```
(config-service[server3])# compress tcp server ack-delay 400
```

To reset the acknowledgement delay to the default of 200 ms for the client TCP connection, enter:

```
(config-service[server3])# no compress tcp virtual ack-delay
```

To reset the acknowledgement delay to the default of 200 ms for the server TCP connection, enter:

```
(config-service[server3])# no compress tcp server ack-delay
```


**Note**

If the service is active, you must suspend it before changing this setting.

## Configuring the Nagle Algorithm for TCP Connections

The TCP Nagle algorithm automatically concatenates a number of small buffer messages transmitted over the TCP connection between a client or server and the SSL-C module. This process increases the throughput of your CSS by decreasing the number of packets sent over each TCP connection. However, the interaction between the Nagle algorithm and the TCP delay acknowledgment may increase latency in your TCP connection. Disable the Nagle algorithm when you observe an unacceptable delay in a TCP connection.

To disable or reenble the Nagle algorithm for the client or server TCP connection, use the **compress tcp virtualserver nagle** command in service configuration mode. The syntax for this command is:

```
compress tcp virtualserver nagle enable|disable
```

The keywords are:

- **enable** - Reenables the Nagle algorithm. By default, the Nagle algorithm is enabled for each TCP connection.
- **disable** - Disables the Nagle algorithm when you observe a delay on the TCP connection.

For example, to disable the Nagle algorithm for the client TCP connection, enter:

```
(config-service[server3])# compress tcp virtual nagle disable
```

To disable the Nagle algorithm for the server TCP connection, enter:

```
(config-service[server3])# compress tcp server nagle disable
```

To reenale the Nagle algorithm for the client TCP connection, enter:

```
(config-service[server3])# compress tcp virtual nagle enable
```

To reenale the Nagle algorithm for the server TCP connection, enter:

```
(config-service[server3])# compress tcp server nagle enable
```

**Note**

If the service is active, you must suspend it before changing this setting.

## Configuring the TCP Buffering for TCP Connections

If the network is slow and congested, you can increase the buffer size that the CSS buffers for a given TCP connection before shutting down the TCP window to 0. Increasing the buffer size decreases latency on slow connections to a client but increases buffering in the CSS. Use this feature with caution, because if there are many slow clients, the CSS memory may run low.

Use the **compress tcp buffer-share** command in service configuration mode to set the TCP buffering from the client or server on a given TCP connection. The syntax for this command is:

```
compress tcp buffer-share [rx bytes][tx bytes]
```

The keywords and variables are:

- **rx bytes** - Sets the amount of data in bytes that a given connection can buffer from the client traffic. Enter a number from 16400 to 262144. By default, the buffer size is 32768.
- **tx bytes** - Sets the amount of data in bytes that a given connection can buffer from the server. Enter a number from 16400 to 262144. By default, the buffer size is 65536.

For example, to set the buffer size for client traffic to 65536 bytes, enter:

```
(config-service[server3])# compress tcp buffer-share rx 65536
```

To reset the buffer size for client traffic to the default of 32768, enter:

```
(config-service[server3])# no compress tcp buffer-share rx
```

To set the buffer size for server traffic to 131072, enter:

```
(config-service[server3])# compress tcp buffer-share tx 131072
```

To reset the reset the buffer size for server traffic to the default of 65536, enter:

```
(config-service[server3])# no compress tcp buffer-share tx
```


**Note**


---

If the service is active, you must suspend it before changing this setting.

---

## Configuring the TCP Retransmission Timer

On networks that experience a lot of packet loss, TCP transaction can take a long time. To adjust the retransmission timer for TCP transactions, use the **compress tcp virtualserver retrans** command. The syntax for this command is:

```
compress tcp virtualserver retrans milliseconds
```

The *milliseconds* variable is the minimum time in milliseconds for retransmission of TCP transactions. Enter a number from 50 to 500. The default value is 500.

For example, to set the retransmission timer to 100 milliseconds for client transactions, enter:

```
(config-service[server3])# compress tcp virtual retrans 100
```

To set the retransmission timer to 100 milliseconds for server transactions, enter:

```
(config-service[server3])# compress tcp server retrans 100
```

To reset the default value of 500 milliseconds for client transactions, enter:

```
(config-service[server3])# no compress tcp virtual retrans
```

To reset the default value of 500 milliseconds for server transactions, enter:

```
(config-service[server3])# no compress tcp server retrans
```


**Note**


---

If the service is active, you must suspend it before changing this setting.

---

# Displaying Compression Information

You can display the CSS compression statistics and compression coprocessor information. To display compression statistics, use the **show service** command.

[Table 9-5](#) describes the compression-related fields in the **show service** output. To set the compression statistics for the CSS to zero in these fields, use the **zero service compression** command.

**Table 9-5** *Compression Statistic Field Descriptions for the show service Command Output*

Field	Description
HTTP Responses In	The total number of uncompressed HTTP responses into the SSL-C module.
Compressed HTTP Responses Out	The total number of compressed HTTP responses passing through the SSL-C module.
Bypassed HTTP Responses Out	The total number of responses when the HTTP header did not allow compression.
HTTP Response Bytes In	The total number of uncompressed HTTP response bytes into the SSL-C module.
Compressed HTTP Response Bytes Out	The total number of compressed HTTP response bytes passing through the SSL-C module.
Bypassed HTTP Response Bytes Out	The total number of bytes when the HTTP header did not allow compression.
Compression Ratio Percentage	The percentage of the original HTTP responses reduced in size by compression (compressed HTTP response bytes out/HTTP response bytes in). A low percentage indicates that the responses were not very compressible. A high percentage indicates that the responses were very compressible.

To display whether the coprocessor for hardware compression is available in the CSS, use the **show chassis coprocessors** command. This command displays compression hardware information in the Coprocessor Inventory fields. You can also display the Coprocessor Inventory fields by using the **show chassis verbose** command.

[Table 9-6](#) describes the Coprocessor Inventory fields in the **show chassis coprocessor** output.

**Table 9-6** *Compression Inventory Field Descriptions for the show chassis coprocessor Command*

Field	Description
Slot/SubSlot	The slot and subslot where the compression coprocessors are located.
Coprocessor	The type of coprocessor, as indicated by COMPRESSION.
Version	The software version that the CSS loaded from the flash device into the coprocessor.
Active	The state of the flash location for the image that the CSS loaded onto the coprocessor. The states are LOCKED, OPERATIONAL, or UNKNOWN if an error occurred while the CSS loaded the image.