



Configuring Advanced Transparent Caching Features on Standalone Content Engines

This chapter describes how to configure advanced transparent caching features on standalone Content Engines that are running ACNS software, Release 5.x or later. The chapter includes the following sections:

- [About Advanced Transparent Caching Features, page 14-1](#)
- [Configuring Bypass Settings on Standalone Content Engines, page 14-3](#)
- [Configuring WCCP Flow Protection, page 14-9](#)
- [Configuring WCCP Slow Start, page 14-10](#)
- [Configuring WCCP IP Spoofing, page 14-10](#)

About Advanced Transparent Caching Features

One of the fundamental principles of transparent network caching is that the Content Engine must remain transparent to the end user at all times. A transparent caching solution must not introduce any possible failure conditions or side effects in a network.

The ACNS software uses a WCCP-enabled router and various advanced techniques to ensure that the Content Engine remains transparent, even if client browsers are nonoperational or web servers are not HTTP-compliant.

Table 14-1 lists the so called “advanced transparent caching features.” This set of features ensures that you do not encounter unexpected problems when you deploy the Cisco caching solutions (including the deployment of standalone Content Engines as caching engines).

Table 14-1 Advanced Transparent Caching Features for Standalone Content Engines

Technology Service	Benefit
Bypass	
Authentication traffic bypass	Preserves cache transparency and avoids disruption of service by allowing the Content Engine to generate a bypass list for the selected client/server pairs. For more information, see the “Configuring Authentication Traffic Bypass on Standalone Content Engines” section on page 14-4.
Static bypass	Permits traffic from the specified sources to bypass the Content Engine. For more information, see the “Configuring Static Bypass on Standalone Content Engines” section on page 14-7.
Overload bypass	Prevents a Content Engine from becoming a bottleneck when traffic loads exceed the capacity of the Content Engine. For more information, see the “Configuring Overload Bypass on Standalone Content Engines” section on page 14-8.
WCCP flow protection	Prevents existing flows from being broken when the WCCP cluster load distribution changes because of the addition or subtraction of a Content Engine into or from a cluster. For more information, see the “Configuring WCCP Flow Protection” section on page 14-9.
WCCP slow start	Prevents cluster destabilization when a new Content Engine is added to a heavily loaded cluster. For more information, see the “Configuring WCCP Slow Start” section on page 14-10.
WCCP IP spoofing	Uses the client IP address when connecting to the origin web server. For more information, see the “Configuring WCCP IP Spoofing” section on page 14-10.

ACNS software also has a built-in bypass mechanism that is not configurable. This built-in bypass mechanism affects connections that are opened before WCCP is activated, and retransmissions of client packets after the connection has been terminated. Through this built-in bypass mechanism, the Content Engine automatically sends such traffic back to the router.

The following sections in this chapter describe how to configure these advanced transparent caching features on standalone Content Engines:

- [Configuring Bypass Settings on Standalone Content Engines](#)
- [Configuring WCCP Flow Protection](#)
- [Configuring WCCP Slow Start](#)
- [Configuring WCCP IP Spoofing](#)

Configuring Bypass Settings on Standalone Content Engines

Bypass refers to a method that a Content Engine can use to handle various error responses (including authentication failure) from an origin server. When the Content Engine receives an error response from an origin server, it adds an entry for the server to its bypass list. When it receives subsequent requests for content residing on the bypassed server, it redirects packets to the bypass gateway. If no bypass gateway is configured, then the packets are returned to the redirecting Layer 4 switch.

Bypass features can be used with a WCCP Version 2 router or with a Layer 4 switch, such as the Cisco Content Switching Module or the Content Services Switch (CSS) switch.

This section describes how to configure a standalone Content Engine to support the following types of bypass:

- [Configuring Authentication Traffic Bypass on Standalone Content Engines](#)
- [Configuring Static Bypass on Standalone Content Engines](#)
- [Configuring Overload Bypass on Standalone Content Engines](#)



Note

The bypass feature is only available when WCCP Version 2 is enabled in your local network. The Content Engine can only bypass WCCP-redirected traffic and not proxy-style requests.

To display a bypass summary that includes the number of entries in the bypass list (see sample output below), use the **show bypass summary** EXEC command.

```
ContentEngine# show bypass summary
Total number of requests bypassed = 0
    Requests bypassed due to system overload           = 0
    Requests bypassed due to authentication issues      = 0
    Requests bypassed due to facilitate error transparency = 0
    Requests bypassed due to static configuration      = 0
Total number of entries in the bypass list = 1
    Number of Authentication bypass entries = 0
    Number of Error bypass entries        = 0
    Number of Static Configuration entries = 1
L2 Bypass:
    Number of L2 bypassed packets = 0
ContentEngine#
```

To display a list of entries in the bypass list (see sample output below), use the **show bypass list** EXEC command.

```
ContentEngine# show bypass list

      Client                Server                Entry type
      -----                -
1.1.1.1:0                5.5.5.5:0                static-config
```

Configuring Authentication Traffic Bypass on Standalone Content Engines

Occasionally a website uses the client's IP address to authenticate a client. This method of client authentication is typically used only in older origin web servers. However, in such situations there must be a way for the Content Engine "to get out of the way" between the client and the origin web server so that the client can be authenticated by the origin web server. With direct proxy routing, this is not possible because the client is configured to point directly to the Content Engine as its outgoing proxy server (the Content Engine receives the client requests directly from the client browser or media player). However, with WCCP-intercepted requests, if the authentication traffic bypass feature is enabled on the Content Engine, then WCCP-intercepted requests can bypass the Content Engine so that the origin web server can authenticate the client.

Another typical use of this bypass feature, occurs with websites that do not allow the Content Engine to connect directly on behalf of the client because of IP authentication. In order to preserve cache transparency and avoid disruption of service, the Content Engine can use this bypass feature to generate a dynamic access list for selected client-server pairs. Authentication traffic bypass triggers are also propagated upstream and downstream in the case of hierarchical caching.

By default, the authentication traffic bypass feature is disabled on a Content Engine. Use the **bypass auth-traffic** global configuration command to enable the authentication traffic bypass feature on a standalone Content Engine. When a client-server pair enters authentication traffic bypass, the pair is bypassed for an amount of time set by the **bypass timer** global configuration command (20 minutes by default). For example, when a client-server pair performs authentication traffic bypass, it is bypassed for a configurable amount of time, which is set by the **bypass timer** global configuration command.

If the authentication traffic bypass feature is enabled on a Content Engine, then the following occurs:

1. The client (an end user who is using a browser to request content) sends a content request to a web server (an origin web server).



Note Only WCCP-redirectioned requests can be bypassed. With proxy-style requests, client browsers are explicitly configured to point directly to the Content Engine as their proxy server; therefore, bypass cannot be configured for proxy-style requests.

2. A WCCP router transparently intercepts the content request and forwards it to the Content Engine (transparent proxy server).
3. The Content Engine masquerades as the origin web server and responds to the client. At the same time, the Content Engine sends a request to the origin web server using its own IP address.
4. If the origin web server is performing any kind of request authentication that is based on IP addresses, it rejects the request.
5. If the Content Engine receives a 401, 403, 501, 503, 502, 503, 504, or 505 response from the origin web server, it performs the following authentication traffic bypass actions.
 - a. It sends a redirect to the client with the exact same URL.
 - b. It adds a bypass entry (the client-server pair entry) to the bypass list.
6. The retried request from the client is intercepted again by the WCCP Version 2 router and is forwarded to the Content Engine. This time, the Content Engine forwards the request to the origin web server instead of handling the request itself because of the client-server bypass entry that was just created in the bypass list.
7. The origin web server responds to the client, and this response goes directly to the client.

You can use either the Content Engine GUI or the CLI to configure authentication traffic bypass on standalone Content Engines:

- From the Content Engine GUI, choose **Caching > Bypass**. The Bypass window appears. Click the **Authentication Bypass On** radio button to enable authentication traffic bypass. In the Bypass Entry Expiration Time field, specify a value (in minutes) to set the number of minutes that an idle client-server pair remains on the bypass access list. The default value is 20 minutes. Click **Update** to save the settings.
- From the Content Engine CLI, use the **bypass auth-traffic** and **bypass gateway** global configuration commands. The parameters are described in [Table 14-2](#).
bypass {auth-traffic enable | gateway ipaddress | timer minutes}

Table 14-2 Authentication Traffic Bypass Command Parameters

Parameter	Description
auth-traffic	Sets the authenticated traffic bypass feature configuration.
enable	Enables authentication traffic bypass.
gateway	Configures a router to which bypassed packets are redirected when the Content Engine receives requests redirected by a Layer 4 switch.
<i>ipaddress</i>	IP address of the router acting as the bypass gateway.
timer	Sets the authentication bypass timer (in minutes). The bypass entry is removed from the dynamic list when the timer expires.
<i>minutes</i>	Time in minutes (1–1440).

This example forces all authenticated HTTP traffic to bypass the Content Engine for 24 hours.

```
ContentEngine(config)# bypass auth-traffic enable
ContentEngine(config)# bypass timer 1440
```

To identify the WCCP Version 2 router to which the Content Engine will direct responses when errors are received from the origin server, use the **bypass gateway ipaddress** global configuration command. Replace *ipaddress* with the IP address of a router that is a Layer 2 neighbor of the Content Engine.

To enable bypass with a Layer 4 switch, use the **http 14 switch enable** global configuration command.

To disable the authentication traffic bypass feature on a standalone Content Engine, use the **no** form of the **bypass auth-traffic** global configuration command.



Note

The **bypass auth-traffic** global configuration command is also used to enable transparent error handling on standalone Content Engines.

Scenario 1—Dynamic Bypass upon Receiving a Web Server Error

These two scenarios implement the WCCP return-path functionality, which is a mechanism whereby a Content Engine can return traffic to the WCCP-enabled router or switch, telling the router or switch to forward the packets as if the Content Engine were not present.

It is typical for about 3 percent of all HTTP traffic flows to fail. These failed flows are automatically retried using authentication bypass or dynamic client bypass, demonstrating that the failure conditions were preexisting and not due to the deployment of transparent caching.

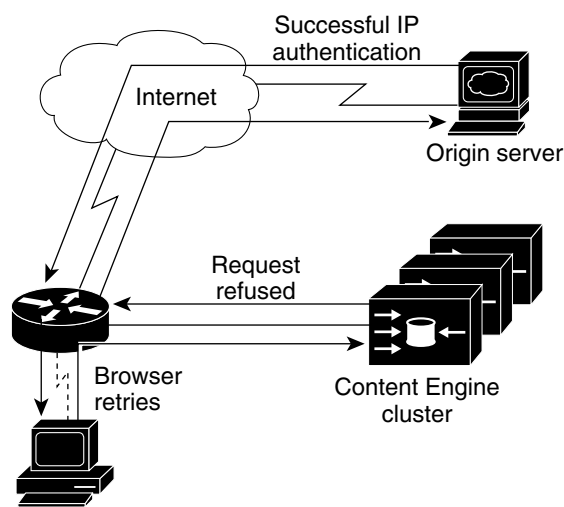
A user issues an HTTP request from a web browser. The request is transparently intercepted and redirected to the Content Engine. The Content Engine accepts the incoming TCP connection from the web browser, determines that the request is for an object not in storage (a cache miss), and issues a request for the object from the origin web server, but receives some kind of error message (for instance, a protocol or authentication error) from the web server.

The Content Engine has already accepted the TCP connection from the web browser and the three-way TCP handshake has taken place. The Content Engine detects that the transaction with the web server has failed, but it does not know the cause (for example, the origin web server is performing authentication based on user source IP address, or there is an incompatibility between the TCP stacks).

Dynamic client bypass in this case means that the Content Engine returns the HTTP response code to the browser. The response returned is an HTML page with a meta tag, requesting a refresh. The Content Engine closes the TCP connection between the web browser and the Content Engine by issuing a “Connection: close” HTTP response header to the web browser. The browser then automatically retries the connection.

On the connection retry, the Content Engine does not accept the connection. It passes the request back to the WCCP-enabled router or switch unintercepted. The router then sends the flow toward the origin web server directly from the web browser, thereby bypassing the Content Engine. (See [Figure 14-1](#).)

Figure 14-1 Dynamic Traffic Bypass



68676

Scenario 2—Dynamic Bypass upon Receiving an Unsupported Protocol

When the Content Engine receives non-HTTP requests over TCP port 80, it issues a “retry” response, closes the connection, and does not accept subsequent connections just as it does as in scenario 1. A “retry” response is a normal HTTP response which states that the response needs a refresh or another try.



Note

Non-HTTP includes nonconforming HTTP as well as different protocols such as Secure Shell (SSH), Simple Mail Transfer Protocol (SMTP), or Network News Transport Protocol (NNTP). An example of nonconforming HTTP is the failure of a web server to issue two carriage returns and line feeds at the end of the HTTP header section.

Note that all HTTP clients do not support HTML. If the client or server does not support http/html, then the client can experience problems and the Content Engine will not be able to serve the requested content to the client.

Configuring Static Bypass on Standalone Content Engines

The static bypass feature permits traffic from specified sources to bypass a Content Engine. The types of traffic sources are as follows:

- Specific web client to a specific web server
- Specific web client to any web server
- Any web client to a specific web server

To enable and configure the static bypass feature on a standalone Content Engine, use the **bypass static** global configuration command, as described in [Table 14-3](#).

bypass static {*clientip* | **any-client**} {*serverip* | **any-server**}



Note

You must not exceed 50 bypass list entries for any one Content Engine.

Table 14-3 *bypass static* Command Parameters

Parameter	Description
static	Adds a static entry to the bypass list.
<i>clientip</i>	IP address from which requests will bypass the Content Engine. Wildcards are not supported.
<i>serverip</i>	IP address to which requests will bypass the Content Engine. Wildcards are not supported.
any-server	Requests from a specified client to any server will bypass the Content Engine.
any-client	HTTP traffic from any client destined to a particular server will bypass the Content Engine.



Note

To clear all static configuration lists on a Content Engine, use the **no** form of the **bypass static** global configuration command.

The following are some examples of how to use the **bypass static** global configuration command to configure the static bypass feature on a standalone Content Engine.

This example forces HTTP traffic from a specific client to a specific server to bypass the Content Engine.

```
ContentEngine(config)# bypass static 10.1.17.1 172.16.7.52
```

This example forces all HTTP traffic destined to a specific server to bypass the Content Engine.

```
ContentEngine(config)# bypass static any-client 172.16.7.52
```

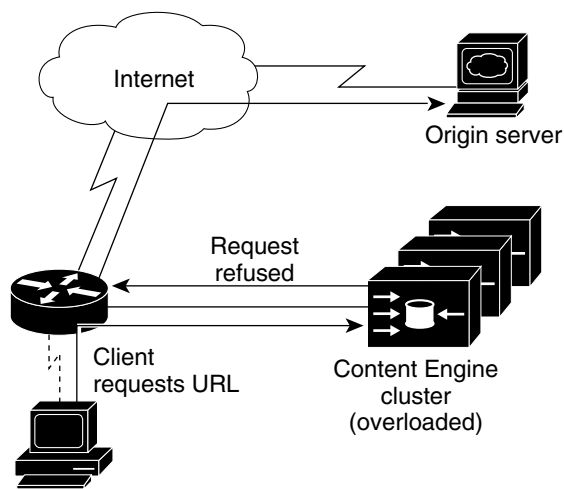
This example forces all HTTP traffic from a specific client to any web server to bypass the Content Engine.

```
ContentEngine(config)# bypass static 10.1.17.1 any-server
```

Configuring Overload Bypass on Standalone Content Engines

When a Content Engine is overloaded and the overload bypass option is enabled, the Content Engine bypasses a bucket and reroutes the overload traffic. If the load remains too high, another bucket is bypassed, and so on until the Content Engine can handle the load. (See [Figure 14-2](#).)

Figure 14-2 Overload Bypass



Note

A bucket is defined as a certain subsection of the allotted hash assigned to each Content Engine in a Content Engine cluster. If only one Content Engine exists in this environment, it has 256 buckets assigned to it.

When the first bucket bypass occurs, a time interval must elapse before the Content Engine begins to again service the bypassed buckets. The duration of this interval is set to 10 minutes by default.

When the Content Engine begins to service the bypassed traffic again, it begins with a single bypassed bucket. If the load is serviceable, the Content Engine picks up another bypassed bucket, and so on. The interval between picking up one bucket and the next is set to 60 seconds by default.

You can use the Content Engine GUI or the CLI to configure overload bypass on a standalone Content Engine:

- From the Content Engine GUI, choose **Caching > Bypass**, and use the displayed Bypass window. For more information about how to use the Bypass window to configure load bypass, click the **HELP** button in the Bypass window.
- From the Content Engine CLI, use the **bypass load** global configuration command. The parameters are described in [Table 14-4](#).

```
bypass load {enable | in-interval seconds | out-interval seconds | time-interval minutes }
```

Table 14-4 bypass load Command Parameters

Parameter	Description
load	Adds a static entry to the bypass list.
enable	Enables overload bypass on the Content Engine.

Table 14-4 *bypass load Command Parameters (continued)*

Parameter	Description
in-interval	Sets the interval between buckets coming back.
<i>seconds</i>	Time in seconds (2–600). The default is 60 seconds.
out-interval	Sets the interval between bypassing buckets.
<i>seconds</i>	Time in seconds (4–600). The default is 4 seconds.
time-interval	Sets the interval between one bucket being bypassed and the next.
<i>minutes</i>	Time in minutes (1–1440). The default is 10 minutes.

Configuring WCCP Flow Protection

WCCP flow protection is a mechanism that ensures that no existing flows are broken when a new Content Engine is brought online. When transparent traffic interception or redirection first begins, WCCP flow protection ensures that no existing HTTP flows are broken by allowing preexisting, established HTTP flows to continue on. WCCP flow protection also ensures that when a new Content Engine joins an existing Content Engine cluster, existing flows serviced by preexisting Content Engines in the cluster will continue to receive those existing flows.

The mechanisms used by WCCP flow protection result in all of the benefits of maintaining per flow state information in a centralized location but without the overhead, scaling issues, and redundancy or resiliency issues (for example, asymmetrical traffic flows) associated with keeping per flow state information in the switching layer.

Use the **wccp flow-redirect** global configuration command to implement WCCP flow protection. This command works with WCCP Version 2 only. Flow protection is designed to keep the TCP flow intact as well as to not overwhelm Content Engines when they are first started up or are reassigned new traffic. This feature also has a slow start mechanism whereby the Content Engines try to take a load appropriate for their capacity.

This example shows how to enable WCCP flow protection on a standalone Content Engine.

```
ContentEngine(config)# wccp flow-redirect enable
```



Note

When bypass is enabled, the client itself tries to reach the origin web server. You must disable all bypass options to eliminate an unnecessary burden on the network.

Configuring WCCP Slow Start

Within a cluster of Content Engines, TCP connections are redirected to other Content Engines as units are added or removed. A Content Engine can be overloaded if it is reassigned new traffic too quickly or if it is introduced abruptly into a fat pipe.

WCCP slow start performs the following tasks in order to prevent a Content Engine from being overwhelmed when it comes online or is reassigned new traffic:

- TCP flow protection when WCCP Version 2 is enabled and a Content Engine is introduced into the cluster
- TCP flow protection when WCCP Version 2 is disabled and a Content Engine is leaving the cluster
- Load assignment to the Content Engine in slow increments rather than a full load at boot up

Slow start is applicable only in the following cases:

- Initial boot up when there is no Content Engine yet present in the server farm
- When a new Content Engine is added to a cluster that is not handling the full load; for example, when there are some buckets that are being shed by the cluster

In all other cases slow start is not necessary and all the Content Engines can be assigned their share of traffic right away.

To enable slow start capability of the caching service on a standalone Content Engine, use the **wccp slow-start enable** global configuration command. To disable slow start capability, use the **no** form of this command.

Configuring WCCP IP Spoofing

With typical transparent caching, an end user issues an HTTP request from a web browser. This request is transparently intercepted and redirected to the Content Engine (acting as a transparent proxy server) by a WCCP router. The Content Engine accepts the incoming TCP connection from the WCCP router, determines that the request is for an object not in storage (cache miss), and issues a request to the origin server for the requested object. When the Content Engine contacts the origin server, it uses its own IP address instead of the IP address of the client for which it is making the request.

If IP spoofing is configured on the WCCP Version 2-enabled routers and the Content Engines, the Content Engine (acting as a transparent proxy server) can send out the client's IP address to the origin server for authentication purposes instead of sending out the request with its own IP address. The WCCP router can also intercept packets from the server that are destined for the client's IP address, and redirect these packets to the Content Engine.

By spoofing a client's IP address, the following capabilities are supported:

- The Content Engine can send out packets with the client IP (which is different from the Content Engine's own IP address)
- The Content Engine can receive packets with the client IP (which is again different from the Content Engine's own IP address), and send the packet to the correct application that is waiting for the packet
- The WCCP Version 2-enabled router can intercept the packets from both the client and the server transparently, and forward these redirected packets to the same Content Engine so that the TCP connection is not broken

ACNS software prior to Release 5.0.7 did not perform IP address spoofing for transparently intercepted proxy-style requests. However, with ACNS software, Release 5.0.7 and later, IP address spoofing is performed for transparently intercepted proxy-style requests when the Content Engine is configured to use the proxy server from the original request to fetch the content. To configure a standalone Content Engine to use the proxy server from the original request, use the **proxy-protocols transparent original-proxy** global configuration command.

With a proxy-style request, the client sends a proxy-style HTTP request if the client is configured to send HTTP requests directly to a specific Content Engine. The client sends the request to the IP address of the proxy server, with the complete destination URL, including the name of the origin server in the HTTP method (for example, GET).

In the case of a server-style HTTP request, the client sends the request directly to the destination server with the HTTP Host header containing the domain name of the origin server, and the HTTP method containing the path to the file or script that the client is requesting.

The proxy-style request for *myfile.html* located in *mydirectory* in the domain *myserver.com*, when transparently intercepted, will have the following initial HTTP line:

```
GET http://myserver.com/mydirectory/myfile.html HTTP/1.1
```

The server-style request for *myfile.html* located in *mydirectory* in the domain *myserver.com*, when transparently intercepted, will have the following initial HTTP line:

```
GET /mydirectory/myfile.html HTTP/1.1
```

To enable IP spoofing on a standalone Content Engine, enter the **wccp spoof-client-ip enable** global configuration command. To disable IP spoofing, enter the **no wccp spoof-client-ip enable** global configuration command.



Tip

The Content Engine can also use authentication traffic bypass to automatically generate a dynamic access list and to connect to a server using the client's IP address for selected client-server pairs. For more information on this topic, see the [“Configuring Authentication Traffic Bypass on Standalone Content Engines” section on page 14-4](#). You can also forward the client's IP address without turning IP spoofing on by using the **http append x-forwarded-for-header** global configuration command on the Content Engine that is serving the request.

IP spoofing is recommended in the following scenarios:

- Logging of user IP addresses
- Filtering based on user IP addresses
- Policy-based routing to provide some users better service than others

IP spoofing, by nature, can break in various subtle scenarios, and therefore to avoid potential pitfalls for ACNS users, several restrictions have been put in place. These restrictions block IP spoofing of some or all client traffic even though globally IP spoofing is enabled. The scenarios under which IP spoofing is intentionally avoided are listed below.

- The client request is not transparently redirected (meaning, proxy-style requests)
- The request is transparently redirected but the request is proxy style, and the proxy-protocols transparent original-proxy configuration is not present
- An HTTP outgoing proxy has been configured

- The client request matches any of the following rule patterns:
 - rule use-proxy
 - rule use-server
 - rule rewrite

Examples of Configuring IP Spoofing with Standalone Content Engines

To configure standalone Content Engines and WCCP Version 2-enabled routers for IP spoofing, you must configure both the Content Engine and WCCP Version 2-enabled routers for IP spoofing.

**Note**

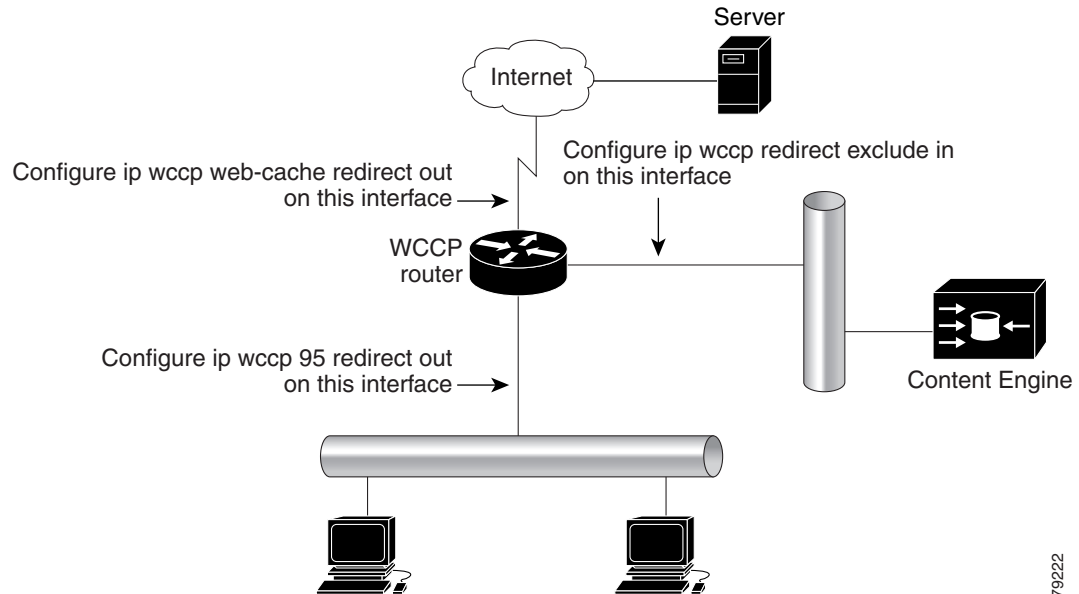
Before you can enable IP spoofing on a Content Engine, you must configure the router interfaces serving the client, the origin server, and the Content Engine. The clients, Content Engines, and origin servers must be configured on three separate interfaces on the WCCP Version 2-enabled router.

This section provides some examples of how to configure IP spoofing with standalone Content Engines and WCCP Version 2 routers:

- [Scenario 1—IP Spoofing with the Content Engine and Clients on Different Subnets](#)
- [Scenario 2—IP Spoofing with a Reverse Proxy Server](#)

Scenario 1—IP Spoofing with the Content Engine and Clients on Different Subnets

The following is a sample scenario of how to configure a standalone Content Engine and WCCP Version 2 router for IP spoofing. In this scenario, the Content Engine and the requesting clients are on different subnets, and two WCCP services (the web-cache service and service 95) are configured. One WCCP service (the web-cache service) hashes on the destination IP address, and the other WCCP service (service 95) hashes on the source IP address. (See [Figure 14-3](#).)

Figure 14-3 IP Spoofing with Content Engine and Clients on Different Subnets

7922

**Note**

The custom-web-cache service (service 98) could also be used instead of the standard web-cache service (service 0) for the WCCP service that hashes on the destination IP address. For a list of WCCP services, see [Table B-3](#).

To configure IP spoofing on a standalone Content Engine and a WCCP Version 2-enabled router when the Content Engine and clients are on different subnets, follow these steps:

- Step 1** Enable WCCP Version 2 on the Content Engine, which is on a different subnet than the clients.
- ```
ContentEngine# configure terminal
ContentEngine(config)# wccp version 2
```
- Step 2** On the Content Engine, configure a router list. In this case, the router list number 1 is created and it has only a single WCCP router, which has an IP address of 10.10.20.1.
- ```
ContentEngine(config)# wccp router-list 1 10.10.20.1
```
- Step 3** On the Content Engine, configure port list 1 to be associated with a WCCP service through port 80.
- ```
ContentEngine(config)# wccp port-list 1 80
```
- Step 4** On the Content Engine, use the **wccp service-number** global configuration command to configure a user-defined WCCP service (service 95) that hashes on the source IP address.
- Specify the WCCP service number.
  - Associate this service with the list of WCCP Version 2-enabled routers (router list number 1) and ports (port list number 1) that will be used to support this WCCP service.
  - Associate the hashing parameters with the source IP address and the source port.
  - Specify the **ip match-source-port** option that hashes on the source IP address.

```
ContentEngine(config)# wccp service-number 95 router-list-num 1 port-list-num 1
application cache hash-source-ip match-source-port
```



**Note** WCCP services number 90 to 97 are for user-defined services, as described in [Table B-3](#). A user-defined service is a WCCP service in which port numbers can be configured to redirect traffic to a Content Engine. In this scenario, service 95 is used to create a user-defined WCCP service. By specifying the **application cache** option of the **wccp service-number** global configuration command, the traffic is redirected to the Content Engine's conventional caching processes (for HTTP requests), whereas the **application streaming** option redirects traffic to the Content Engine's media caching processes (for WMT or RTSP requests).

**Step 5** Inform the WCCP router that the Content Engine is accepting redirected web traffic.

```
ContentEngine(config)# wccp web-cache router-list-num 1
```

**Step 6** Enable client IP spoofing on the Content Engine.

```
ContentEngine(config)# wccp spoof-client-ip enable
```

**Step 7** Exit global configuration mode on the Content Engine.

```
ContentEngine(config)# exit
```

**Step 8** Write the running configuration to nonvolatile memory.

```
ContentEngine# write memory
```

Now that IP spoofing is configured on the Content Engine, complete the remaining steps to configure the WCCP Version 2 router for IP spoofing.

**Step 9** Ensure that WCCP Version 2 is enabled on the router.

```
Router(config)# ip wccp version 2
```

**Step 10** Instruct the WCCP router to run the web-cache service.

```
Router(config)# ip wccp web-cache
```

**Step 11** Enable service 95 on the WCCP router.

```
Router(config)# ip wccp 95
```

**Step 12** On the WCCP router, specify an interface to configure, and enter interface configuration mode.

```
Router(config)# interface type number
```

**Step 13** Enable WCCP redirection on the WAN interface (the router interface that is connected to the origin server) with the service that hashes on the destination IP address (the web-cache service). (See [Figure 14-3](#).)

```
Router(config-if)# ip wccp web-cache redirect out
```

**Step 14** Enable WCCP redirection on the LAN interface (the router interface that is connected to the client) with the service that hashes on the source IP address (service 95). (See [Figure 14-3](#).)

```
Router(config-if)# ip wccp 95 redirect out
```

**Step 15** Disable traffic redirection on the router interface that is connected to the Content Engine. (See [Figure 14-3](#).)

```
Router(config-if)# ip wccp redirect exclude in
```

You must disable traffic redirection on the router interface that is connected to the Content Engine to avoid loopbacks as the WCCP router tries to send the packet with the source IP address back to the Content Engine.

**Step 16** Exit global configuration mode.

```
Router(config-if)# exit
```

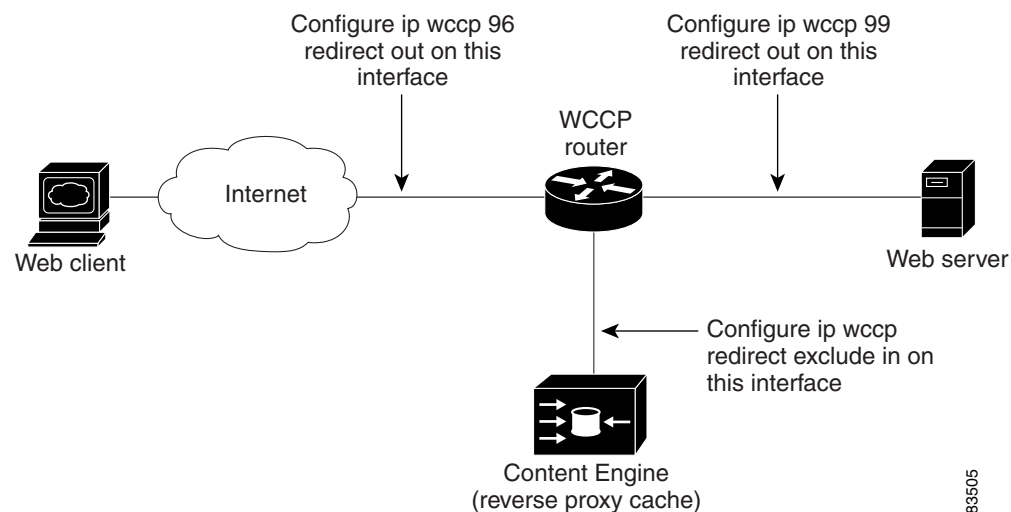
**Step 17** Save the configuration on the router.

```
Router# copy running-config startup-config
```

## Scenario 2—IP Spoofing with a Reverse Proxy Server

The following is a sample scenario of how to configure a standalone Content Engine and WCCP Version 2 router for IP spoofing. In this scenario, the Content Engine is functioning as a reverse proxy server. (See [Figure 14-4](#).)

**Figure 14-4** IP Spoofing with a Reverse Proxy Server



To configure IP spoofing on a standalone Content Engine (reverse proxy server) and a WCCP Version 2-enabled router, follow these steps:

**Step 1** Enable WCCP Version 2 on the Content Engine (that is acting as a reverse proxy server).

```
ContentEngine(config)# wccp version 2
```

**Step 2** Configure a router list on the Content Engine. In this case, there is only a single router (a WCCP Version 2-enabled router that has an IP address of 10.10.20.1) on router list number 1.

```
ContentEngine(config)# wccp router-list 1 10.10.20.1
```

**Step 3** Associate port list 1 with a WCCP service through port 80.

```
ContentEngine(config)# wccp port-list 1 80
```

**Step 4** Configure a user-defined WCCP service (service 96) on the Content Engine. Associate the router list, port list, and hashing parameters with the destination IP address and the source port for this WCCP service.

```
ContentEngine(config)# wccp service-number 96 router-list-num 1 port-list-num 1
application cache hash-destination-ip match-source-port
```




---

**Note** If you have a Content Engine cluster and you are using weight assignments within this cluster, you must make sure that the weight assignments for the service groups assigned to IP spoofing for both outbound and inbound packets are equal on all Content Engines to prevent a break in the TCP connection. Use the **wccp service-number *servnumber* router-list-num *num* port-list-num *port* application cache weight *percentage*** command to establish a weight for these service groups if needed. By default, the Content Engine cluster hashes appropriately with IP spoofing turned on, so assigning weights to service groups is not needed.

---

**Step 5** Inform the WCCP Version 2-enabled routers in the specified router list (for example, router list number 1) that the Content Engine (that is acting as a reverse proxy server) is accepting HTTP reverse proxy traffic.

```
ContentEngine(config)# wccp reverse-proxy router-list-num 1
```

**Step 6** Enable client IP spoofing on the Content Engine.

```
ContentEngine(config)# wccp spoof-client-ip enable
```

**Step 7** Exit global configuration mode.

```
ContentEngine(config)# exit
```

**Step 8** Write the running configuration to nonvolatile memory.

```
ContentEngine# write memory
```

**Step 9** Configure the router for IP spoofing, as follows:

- a. On the WCCP router, enable the reverse-proxy service (service 99).

```
Router(config)# ip wccp 99
```

- b. On the WCCP router, specify an interface to configure, and enter interface configuration mode.

```
Router(config)# interface type number
```

- c. On the router, configure redirect out for WCCP service number 96 (the service that hashes on the destination IP address) on the interface that is connected to the client.

```
Router(config-if)# ip wccp 96 redirect out
```

- d. On the router, configure redirect out for the reverse-proxy service (service 99) on the interface that is connected to the origin server.

```
Router(config-if)# ip wccp 99 redirect out
```

**Step 10** Disable WCCP redirection on the router interface that is connected to the Content Engine (that is acting as a reverse proxy server).

```
Router(config-if)# ip wccp redirect exclude in
```

**Step 11** Exit global configuration mode.

```
Router(config-if)# exit
```

---