



Deployment Scenarios for Standalone Content Engines

This chapter introduces some sample scenarios for deploying standalone Content Engines in enterprise and service provider environments. This chapter contains the following sections:

- [Deciding Which Services to Deploy, page 3-2](#)
- [Deploying Caching and Streaming Services in Nontransparent Mode, page 3-3](#)
- [Deploying Caching and Streaming Services in Transparent Mode, page 3-6](#)
- [Deploying Streaming Media Services on Standalone Content Engines, page 3-14](#)
- [Deploying Filtering and Access Control Services, page 3-15](#)



Note

For information about deploying a Content Engine as a device that is registered with a Content Distribution Manager, see the *Cisco ACNS Software Configuration Guide for Centrally Managed Deployments, Release 5.5*.

Deciding Which Services to Deploy

By pushing content to the edges of a network, you can accelerate content delivery and optimize WAN bandwidth usage. The process used to accomplish this is called *content caching*. Content caching is also referred to as *network caching*. Because of the special position of the Content Engine as an *in-line device* between the web clients and the Internet, you can easily deploy Content Engines as content caching engines. Bandwidth usage and web latency are significantly reduced, because frequently accessed Internet content is being locally cached and served by the Content Engine at each location.


Note

To integrate with existing proxy infrastructures, the ACNS software supports a number of proxy protocols, including FTP, HTTPS, HTTP 1.0, and HTTP 1.1. See [Table B-1](#) for a list of supported network protocols.

The types of supported services that you can deploy with a Content Engine vary depending on the method used to route the request to the Content Engine. Content requests can be routed to the Content Engine directly from clients (direct proxy routing) or through a WCCP router or Layer 4 CSS switch (transparent redirection).


Note

See [Table 1-5](#) and [Table 1-6](#) for a list of caching and streaming services supported when direct proxy routing or transparent redirection is used to direct client requests to a standalone Content Engine.

The *direct proxy routing method* is the simplest, most straightforward routing method. With direct proxy routing, you must configure client desktops (client browsers and media players) to send their content requests directly to a specific Content Engine that is functioning as their proxy server. Client requests are sent directly to the Content Engine that is configured as the client's proxy server. This routing method is typically used when user desktops are tightly controlled.

The *transparent redirection method* requires an understanding of network topology and traffic patterns. Organizations generally prefer to use the transparent redirection method because it does not require any configuration changes to client desktops.

However, even though direct proxy routing requires changes to client desktops, there may be a legacy requirement for using direct proxy routing. There also may be cases in which organizations need to use direct proxy routing for a particular service (for example, HTTPS proxy caching) because they are not allowed to make the necessary configuration changes to the WCCP router or switch at the branch office.

This section introduces the routing methods and the associated caching and streaming services that are supported by standalone Content Engines that are running the ACNS 5.x software:

- [Deploying Caching and Streaming Services in Nontransparent Mode, page 3-3](#)
- [Deploying Caching and Streaming Services in Transparent Mode, page 3-6](#)


Note

A standalone Content Engine supports direct proxy routing and transparent redirection through WCCP routing and Layer 4 switching. However, if you want to use content routing as a routing method, then you must register the Content Engine with a Content Distribution Manager. Standalone Content Engines are not registered with Content Distribution Managers and therefore cannot support Content Routing. For information about Content Routing, see the *Cisco ACNS Software Configuration Guide for Centrally Managed Deployments, Release 5.5*.

Deploying Caching and Streaming Services in Nontransparent Mode

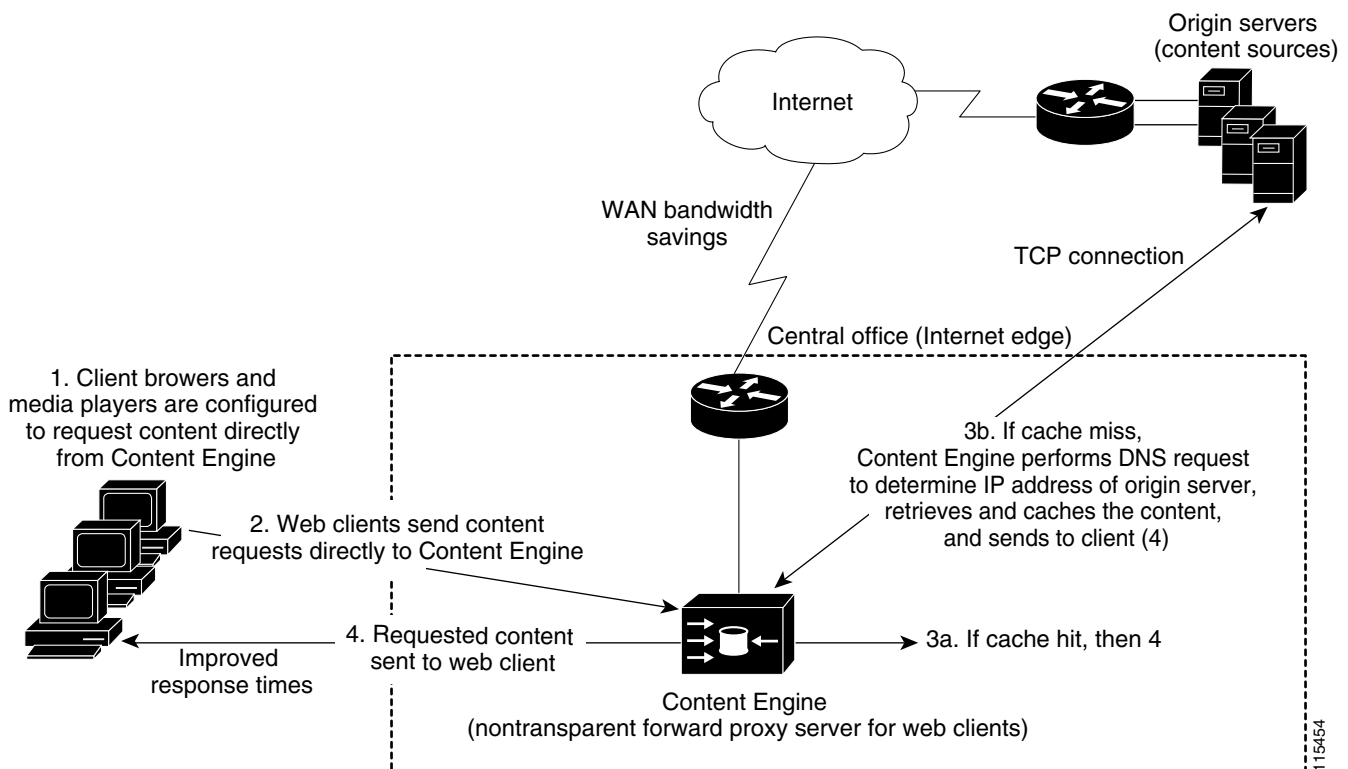
With direct proxy routing, the standalone Content Engine is the destination of all browser or media player requests for web content. Such requests are called *proxy-style requests*. A proxy-style request arrives with the same destination IP address as that of the Content Engine; the request is specifically routed directly to the Content Engine (the forward proxy server) by the web client.

Overview of Nontransparent Forward Proxy Caching

In deployments that use direct proxy routing to route content requests to the Content Engine (see [Figure 3-1](#)), the Content Engine acts a network gateway device that is optimized to retrieve content on behalf of web clients.

- If the requested content is already in the Content Engine's local storage (cache hit), the Content Engine sends the content to the web client.
- If the requested content is not already stored in the Content Engine's local cache (cache miss), the Content Engine retrieves the requested content from the origin server, stores a local copy of the content if the content is cacheable, and sends the requested content to the web client. When the Content Engine receives subsequent requests for the same content, it serves the content from its local storage.

Figure 3-1 Nontransparent Forward Proxy Caching with Standalone Content Engines



Typically, direct proxy routing is implemented in enterprise environments as opposed to service provider environments because this type of caching requires modifications to client desktops. With direct proxy routing, you must explicitly configure client browsers and media players to point to the Content Engine that is functioning as a forward (nontransparent) proxy for these clients. For more information on this topic, see the “[Configuring Client Browsers and Media Players for Direct Proxy Routing](#)” section on page 4-35.

Some significant advantages to deploying nontransparent (proxy) caching services on standalone Content Engines are:

- Internet access for user populations can be regulated by the Content Engine that is acting as a gateway device for these end users.
- Internet requests all appear to be sourced from the proxy cache (Content Engine), which hides internal network addressing.
- Frequently requested cacheable content is cached locally, which results in significant WAN bandwidth savings and accelerated delivery of content to web clients.

In the ACNS 5.2 software release, the Setup utility was added to expedite the basic configuration (device network settings, disk configuration, and a set of commonly used caching services) of standalone Content Engines. This basic configuration includes a set of commonly used caching services. See [Table 4-2](#) for a list of caching services that you can configure with the Setup utility.

Configuring Incoming Proxy Ports for Nontransparent Mode Services

In proxy mode, the Content Engine that is functioning as a forward proxy server supports up to eight incoming ports each for FTP, HTTP, HTTPS, and RTSP requests. The term *proxy port* is used to refer to the port from which the Content Engine receives the proxy-style request and sends the requested content back to the client. The incoming proxy ports can be the same ports that are used by transparent mode services (for example, HTTP transparent caching). You can change the incoming proxy ports on the Content Engine without stopping any WCCP services that are running on the Content Engine.

As part of configuring a nontransparent mode service (for example, HTTP proxy caching, FTP-over-HTTP proxy caching, RealMedia proxy caching, and WMT proxy caching), you must do the following:

- Configure the client browser or media player to direct any requests for request to the incoming proxy port on the Content Engine.
- Configure the Content Engine to listen on the incoming proxy port for client requests.

A standalone Content Engine accepts nontransparent (proxy-style) requests directly from a client browser or media player when the incoming proxy ports are configured on the client browser and media player and on the Content Engine.

For information about how to point client browsers or media players to a specific Content Engine, see [Table 3-1](#).

Table 3-1 Configuring Client Browsers and Media Players to Support Direct Proxy Routing of Content Requests

Nontransparent Caching	More Information
HTTP proxy caching	See the “ Pointing Client Browsers Directly to a Standalone Content Engine ” section on page 4-36.
WMT proxy caching of WMT RTSP requests	See the “ Pointing Windows Media 9 Players Directly to a Standalone Content Engine for WMT RTSP Requests ” section on page 4-43.
RealMedia proxy caching	See the “ Pointing RealMedia Players Directly to a Standalone Content Engine ” section on page 4-46.

The **proxy incoming** option of the **http**, **https**, **ftp**, and **rtsp** global configuration commands supports up to eight ports per protocol. You can specify up to eight incoming proxy ports on a single command line or on multiple command lines. Proxy-style requests in HTTP, FTP, HTTPS, and RTSP protocols can be received on the same incoming proxy port.



Note

Both transparent and proxy-style requests can be serviced on the same port.

In this example, the standalone Content Engine is configured to accept HTTP, HTTPS, and FTP-over-HTTP proxy requests directly from client browsers on ports 81, 8080, and 8081:

```
ContentEngine(config)# http proxy incoming 81 8080 8081
ContentEngine(config)# https proxy incoming 81 8080 8081
ContentEngine(config)# ftp-over-http proxy incoming 81 8080 8081
```

In the ACNS 5.3.1 software release, the **ftp** keyword was replaced with the **ftp-over-http** and **ftp-native** keywords to clearly differentiate between FTP-over-HTTP caching and FTP native caching. Consequently, the **ftp proxy incoming** global configuration command was replaced with the **ftp-over-http proxying incoming** and **ftp-native proxy incoming** global configuration commands in the ACNS 5.3.1 software release. The **ftp-native proxy incoming** command, which was added in the ACNS 5.3.1 software release, is used to configure an incoming port for FTP native requests from FTP clients (for example, Reflection X or WS-FTP clients) to support nontransparent FTP native caching. For more information about nontransparent FTP native caching, see the “[Configuring Nontransparent FTP Native Caching](#)” section on page 7-42.

The following example shows how to configure an incoming proxy port a standalone Content Engine to listen for incoming WMT traffic on a specific port. These WMT requests are being sent directly to the Content Engine (nontransparent forward proxy) from client Windows Media players that are configured to point directly to the Content Engine. The default WMT port is 1755. Valid port numbers are from 1 to 65535.

```
ContentEngine(config)# wmt port incoming portnumber
```

The following example shows how to configure an incoming proxy port on a standalone Content Engine to listen for incoming RTSP traffic on a specific port. These RTSP requests are being sent directly to the Content Engine (nontransparent forward proxy) from client media players that are configured to point directly to the Content Engine. The default RTSP port is 554. Valid port numbers are from 1 to 65535.

```
ContentEngine(config)# rtsp port incoming portnumber
```

To disable HTTP, HTTPS, FTP, and RTSP incoming proxy services, use the **no protocol proxy incoming** global configuration command (for example, the **no wmt proxy incoming** global configuration command). To add or remove ports in proxy mode, enter a new command that specifies all the ports to be used.

SSL Tunneling and Nontransparent Caching Deployments

The SSL tunneling protocol allows a proxy server to act as a tunnel between the end user and the origin server. The client asks for an SSL tunnel through an HTTP request. This allows the Content Engine to service CONNECT method requests in the form `https://url` for tunneling SSL over HTTP.



Tip

Browsers only initiate HTTPS-over-HTTP requests when they are explicitly configured for a proxy. If a browser is not explicitly configured for a proxy, the browser initiates an HTTP-over-SSL connection itself, and because this is on TCP port 443, the request is only intercepted by the Content Engine if the Content Engine is running the ACNS 5.1.5 software and later releases.

SSL on port 443 uses end-to-end encryption, and any transparent device between the client and the origin server sees nothing more than a stream of random bytes.

Deploying Caching and Streaming Services in Transparent Mode

In the ACNS 5.x software, standalone Content Engines can handle transparently redirected content requests from a WCCP router and a Layer 4 switch. With transparent redirection, the standalone Content Engine serves as a transparent (invisible) proxy server to web clients or web servers (for example, web servers in a server farm at a company headquarters). When the Content Engine is proxying for web clients, it is a transparent forward proxy server. When it is proxying for a web server, the Content Engine is a transparent reverse proxy server.

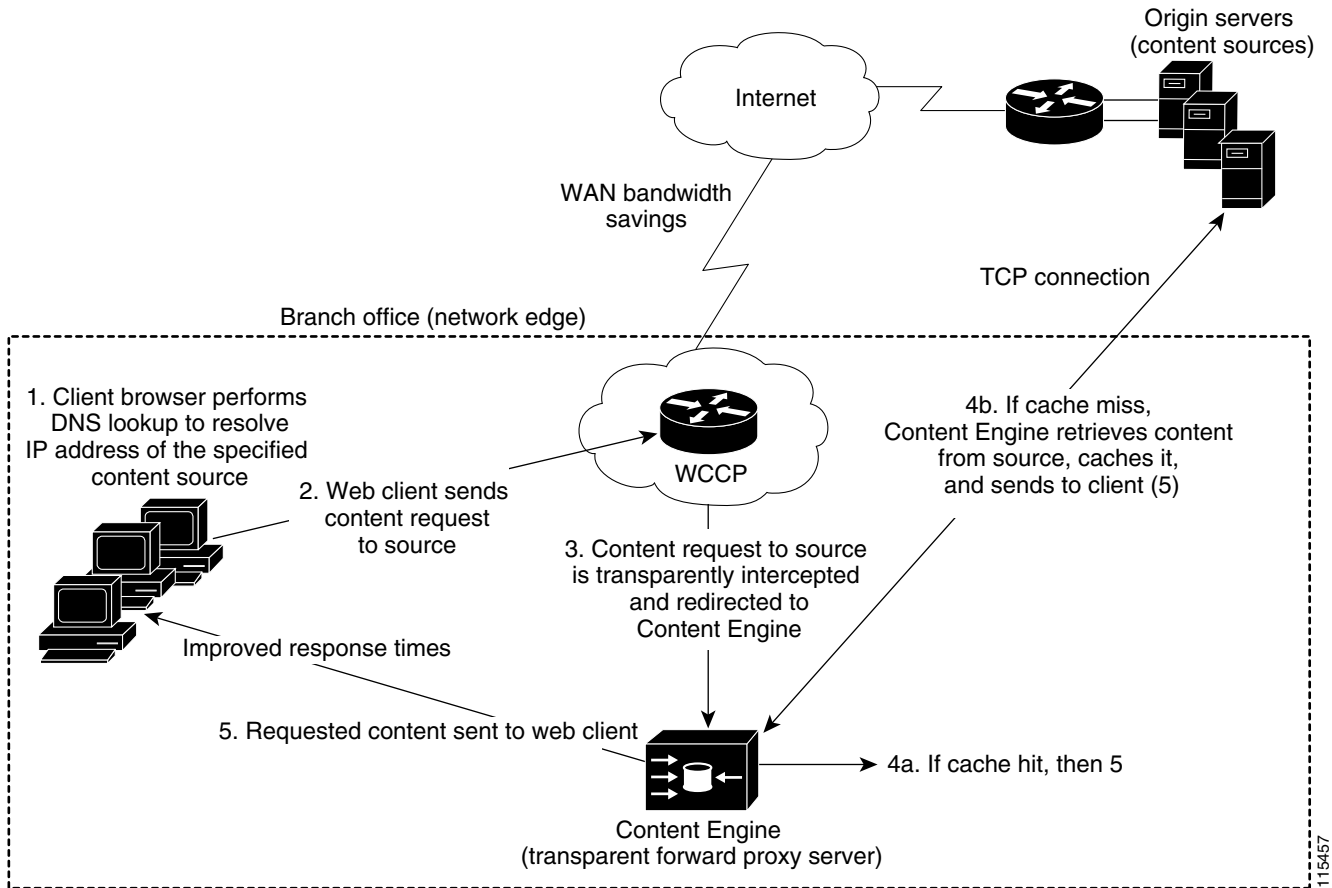
Overview of Transparent Forward Proxy Caching

When you deploy a standalone Content Engine as a transparent cache (a transparent forward proxy server), you place the Content Engine in the middle between web clients and the origin servers; close to the user population. The transparent cache is placed in one of these places, in the path of the network traffic where all egress traffic is guaranteed to pass:

- At the network edge (branch offices)
- At the Internet edge (regional offices or headquarters)

With transparent redirection, web clients request content directly from the source (origin servers). However, these requests are transparently intercepted at a network egress point. A network device (a WCCP router or a Layer 4 Cisco Content Services Switch [CSS] switch) transparently intercepts and redirects these requests to the standalone Content Engine that is functioning as a transparent caching engine for these clients. [Figure 3-2](#) shows an example of transparent forward proxy caching with a WCCP router.

Figure 3-2 Transparent Forward Proxy Caching with a Standalone Content Engine and WCCP Router

**Note**

For more information about implementing forward proxy caching through a Layer 4 switch or WCCP-enabled router, see the [“Overview of Transparent Reverse Proxy Caching”](#) section on page 3-10 and the [“Overview of Transparent Reverse Proxy Caching”](#) section on page 3-10.

By supporting WCCP Version 2 or by interoperating with a Layer 4 CSS switch, standalone Content Engines can achieve a basic level of transparency, which includes transparent receipt of content traffic, fault tolerance, and scalable clustering.

Because a Content Engine can be transparent to the client and to network operation, you can easily place standalone Content Engines in several network locations in a hierarchical fashion. For example, an ISP can deploy a standalone Content Engine (Content Engine A) at their main point of access to the Internet so that all of their points of presence (POPs) benefit, because requested content can be available at this main point of access without going through the Internet.

To further improve service to certain web clients, the ISP can deploy a standalone Content Engine (Content Engine B, C, and D) at each POP. Then, when a client accesses the Internet, the request is first redirected to the POP Content Engine. If the POP Content Engine (Content Engine B, C, or D) is unable to fulfill the request from local storage, it makes a normal web request to the origin server. Upstream, this request is redirected to Content Engine A. If the request is fulfilled by Content Engine A, traffic on

the main Internet access link is avoided, the origin servers experience lower demand, and the client experiences better network response times. Enterprise networks can apply this hierarchical transparent architecture in the same way.

Although transparent caching deployments require an understanding of the network topology and traffic patterns, there are some significant advantages to deploying transparent mode services on standalone Content Engines:

- No end user configuration—No desktop configuration changes are required.
- Fail-safe operation—Caches are automatically fault-tolerant and fail-safe. Any cache failure does not cause denial of service to the end user.
- Scalability—Cache service can be scaled by deploying multiple caches (cache clusters or hierarchical caches).
- Automatic bypass—Sites that depend on end user authentication or that fail to conform to HTTP standards will automatically bypass a transparent cache.

Transparent Redirection and Forward Proxy Caching with a Layer 4 Switch

When Layer 4 switching is used for transparent redirection, a Layer 4 CSS switch transparently intercepts and redirects content requests to the Content Engine. With transparent interception through a CSS switch, the user is unaware that the request made to an origin web server is redirected to the Content Engine by the Layer 4 CSS switch. The Layer 4 CSS switch can be configured to dynamically analyze the request and determine if the requested content is cacheable or not. If the requested content is not cacheable, the Layer 4 CSS switch sends the request directly to the origin server. If the requested content is cacheable, the Layer 4 CSS switch directs the request to the Content Engine. The Content Engine either returns the requested content if it has a local copy or sends a new request to the origin web server for the content.

When Layer 4 switching is used to redirect content requests transparently to standalone Content Engines, if the TCP SYN packet passes through the Layer 4 CSS switch that has the Layer 4 redirection feature turned on, the packet is diverted to a Content Engine that is attached to the switch. In this case, the Layer 4 CSS switch changes the MAC address of the TCP SYN packet, so that instead of going out to the gateway or the origin server, the MAC address is changed to that of the Content Engine. The Layer 4 CSS switch then sends the packet to the Content Engine. All of these actions occur in the hardware.

The Content Engine must be prepared to accept requests that are sent to it even if the IP address of the packet is not the address of the Content Engine. The Content Engine then handles the TCP SYN packet similarly to how it handles packets redirected through WCCP.

Transparent forward proxy caching with a standalone Content Engine and a Layer 4 CSS switch works as follows:

1. A user (web client) requests a web page from a browser.
2. The Layer 4 CSS switch analyzes the request and determines whether the requested content is cacheable or not. If the requested content is cacheable, the Layer 4 CSS switch transparently redirects the request to a Content Engine.



Note

If all the Content Engines are unavailable in a transparent cache configuration, the Layer 4 CSS switch allows all client requests to progress to the origin web servers.

3. If the Content Engine has the requested content already stored in its local cache, it returns the requested content to the client.
4. If the Content Engine does not have the requested content, the following events occur:
 - a. The Content Engine sets up a separate TCP connection to the origin web server to retrieve the content.
 - b. The content returns to, and is stored on, the Content Engine.
5. The Content Engine sends the requested content to the web client. Upon subsequent requests for the same content, the Content Engine transparently fulfills the request from its local storage (cache).

Transparent Redirection and Forward Proxy Caching with a WCCP Router

A router with the proper configuration is required for transparent caching services. The router must be running a version of Cisco IOS software that supports WCCP Version 1 or Version 2. When caching support is enabled on the router and WCCP support is enabled on the Content Engines, the devices can communicate and deliver the services for which they are configured. To use a WCCP-enabled router, an IP address must be configured on the interface connected to the Internet and the interface must be visible to the Content Engine on the network.



Tip

To suspend caching services, you can disable caching support on the router rather than powering off or otherwise disabling individual Content Engines. (For instance, use the **no ip wccp** router command to disable caching.)

Using WCCP, the router transparently redirects requests to the specified TCP ports on the Content Engine rather than to the intended host sites.

WCCP runs on UDP port 2048 within a generic routing encapsulation (GRE) tunnel between the WCCP-enabled router and the Content Engine. When a WCCP-enabled router receives an IP packet, the router determines whether the packet is a request that should be directed to a Content Engine.

A WCCP Version 1 router can look for TCP as the protocol field in the IP header and for port 80 as the destination port in the TCP header. If the packet meets these criteria, it is redirected to a Content Engine. With WCCP Version 1, web-cached information can only be redirected to a Content Engine if it was destined for TCP port 80. Many applications require packets intended for other ports to be redirected, for example, proxy web cache handling, FTP proxy caching, web caching for ports other than port 80, RealAudio, and video.

If a router is configured for WCCP Version 2 as opposed to WCCP Version 1, then the router can be configured to redirect traffic to the Content Engine on TCP ports other than port 80. For example, if you configure the custom-web-cache service (service 98) on the router and Content Engine, the routers can redirect HTTP traffic to Content Engines on a port other than port 80. The Content Engine can then be configured to cache the content from these redirected HTTP requests. This type of caching is called HTTP transparent caching with WCCP.

A transparent request is a request redirected to the Content Engine from a router. The style of the URL within a transparent request is usually server-style but can be proxy-style when the Content Engine intercepts a request destined for another proxy server. Server-style requests do not include the protocol and hostname, but RTSP requests are the same for both server-style and proxy-style URLs. If a server-style URL is received, only HTTP and RTSP are supported (if the RTSP user agent criteria are met). If a proxy-style URL is received, HTTP, HTTPS, FTP, and RTSP are supported when the respective proxy services are configured.

To configure WCCP as a redirection method, you must perform the following tasks:

1. Enable WCCP redirection on the router, as described in the “[Configuring WCCP Services on a Router](#)” section on page 6-27.
2. Enable WCCP on the standalone Content Engine. Configure the specific WCCP services (for example, the web-cache service) that you want the WCCP-enabled router and Content Engine to support.

For more information on this topic, see the “[Configuring Standalone Content Engines for WCCP Transparent Redirection](#)” section on page 6-9.

**Note**

For a complete list of supported WCCP services, see [Table B-3](#).

Overview of Transparent Reverse Proxy Caching

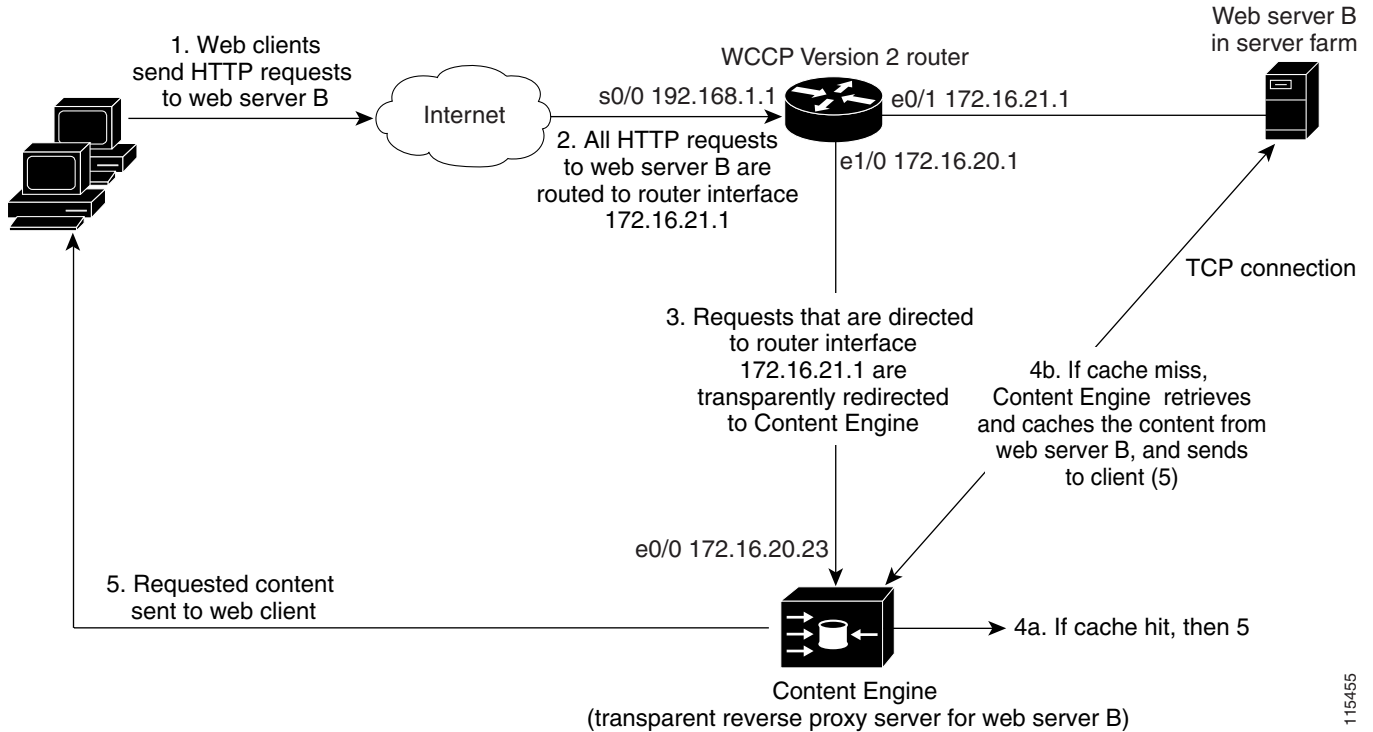
To ensure fast response times, maximized service availability, and the ability to withstand an excessive number of URL hits or an excess of bandwidth requested, Content Engines can be deployed in front of a website server farm to offload traffic from busy firewalls and servers, helping to optimize the entire website infrastructure. This type of deployment is called web server acceleration, or reverse proxying. A Content Engine deployed in this manner is called a reverse proxy cache because the Content Engine is operating at the opposite end of the transaction, in front of the origin server.

By having the Content Engine (the reverse proxy server) transparently handle inbound requests for content instead of having the origin servers (the servers in the server farms) handle these requests, web traffic is significantly reduced. Reverse proxy servers are an effective method for scaling server farms.

In a reverse proxy cache configuration, the proxy server is configured with an Internet-routable IP address. Clients are directed to the proxy server based on DNS resolution of a domain name. To a client, the reverse proxy server appears like a web server.

The ACNS 5.x software provides reverse proxy caching by allowing traffic redirection or interception to be performed by two types of devices: a WCCP Version 2-enabled router or a Layer 4 CSS switch. [Figure 3-3](#) shows a typical reverse proxy caching deployment with a WCCP-enabled router. In this type of deployment, the Content Engine interoperates with the WCCP Version 2-enabled router to bring the reverse-proxy service (service 99) within the web server environment. The Content Engine is deployed in front of a web server farm. Unlike transparent and nontransparent forward proxy servers, the reverse proxy server proxies requests on behalf of a server farm, and it only caches content from servers in the server farm.

Figure 3-3 Reverse Proxy Caching with a Standalone Content Engine and a WCCP Version 2 Router

**Note**

A redirect list on the router or a static bypass list on the Content Engine can be used to allow flows to bypass interception. These lists use criteria based on source and destination IP addresses.

For more information about implementing reverse proxy caching through a WCCP router or Layer 4 CSS switch, respectively, see the [“Example 1—Deploying Reverse Proxy Caching with WCCP Transparent Redirection”](#) section on page 3-12, and the [“Example 2—Deploying Reverse Proxy Caching with Layer 4 Switching”](#) section on page 3-13.

In [Figure 3-3](#), the router interface connected to the Internet has an IP address of 192.168.1.1. All HTTP requests destined for web server B are routed to the router interface at 172.16.21.1. Upon receiving the HTTP request at this interface, the router transparently intercepts and redirects the request to the Content Engine that has an IP address of 172.16.20.23. Thus, the Content Engine is logically in front of web server B, offloading web server HTTP traffic. Web clients who are requesting content from the origin server receive the static web pages from the Content Engine that is acting in reverse proxy mode. This frees up the back end infrastructure from processing this HTTP traffic.

The following are some significant advantages to deploying reverse proxy caching on standalone Content Engines:

- Provides an alternative to web server expansion by offloading the processing of static images from the server farm.
- Provides a possible way of replicating content to geographically dispersed areas by deploying Content Engines in these areas.
- Does not require any client configuration changes (you do not need to configure the client browsers to point to the Content Engine that is functioning as the reverse proxy server).

115455

Example 1—Deploying Reverse Proxy Caching with WCCP Transparent Redirection

This example shows how to deploy the reverse proxy caching service using WCCP transparent redirection (as opposed to Layer 4 switching).

-
- Step 1** Enable WCCP Version 2 on the standalone Content Engine. (WCCP Version 1 does not support the reverse-proxy service.)
- ```
ContentEngine(config)# wccp version 2
```
- Step 2** Configure a router list.
- ```
ContentEngine(config)# wccp router-list 1 172.16.20.1
```
- Step 3** Instruct the Content Engine to run the reverse-proxy service (service 99).
- ```
ContentEngine(config)# wccp reverse-proxy router-list-num 1
```
- Step 4** Exit global configuration mode.
- ```
ContentEngine(config)# exit
```
- Step 5** Instruct the router to run the reverse-proxy service (service 99).
- ```
Router(config)# ip wccp 99
```
- Step 6** Configure the reverse-proxy service with output redirection facing the original servers by performing these steps:
- Specify which router interface to configure. In this example, Ethernet 0/1 is the router interface to the web server.
- ```
Router(config)# interface Ethernet 0/1
```
- Instruct the router to redirect TCP port 80 traffic bound for the specified interface to Content Engines that accept reverse proxy service. In this example, there is only one router.
- ```
Router(config-if)# ip wccp 99 redirect out
```
- Step 7** Configure the reverse-proxy service with input redirection facing the standalone Content Engine by performing these steps:
- Specify which router interface to configure. In this example, s0/0 is the router interface to the Internet.
- ```
Router(config)# interface s0/0
```
- Instruct the router to redirect TCP port 80 traffic received on the specified interface to Content Engines that accept reverse proxy service.
- ```
Router(config-if)# ip wccp 99 redirect in
```
- Step 8** Exit interface configuration mode.
- ```
Router(config-if)# exit
```
-

Example 2—Deploying Reverse Proxy Caching with Layer 4 Switching

With reverse proxy caching that is based on a Layer 4 switch, the standalone Content Engine interoperates with a Layer 4 CSS switch to bring reverse proxy service within the web server environment. In this case, a set of Content Engines use their local caches to accelerate the actual web server. The Layer 4 CSS switch is a load-balancing switch that has a virtual IP address (for example, 200.200.200.1). This virtual IP address is the IP address of the web server that is seen by the rest of the world (web clients that are requesting information from that server). Client requests arrive at the Layer 4 switch first; the switch has Layer 4 redirection capabilities and it redirects the requests to the Content Engine (reverse proxy server) that is attached to the Layer 4 switch. If the request is a cache hit, then the Content Engine services the request. If the request is a cache miss, the Content Engine sends the request to the web server at the back end, caches the result, and sends the requested content to the client.

In reverse proxy caching cases, the Content Engines are transparent to the rest of the world and the clients typically are not aware of the presence of the Content Engines (reverse proxy servers).


Note

The sample solution shown here works only with particular web servers and in particular configurations.

This example shows how to configure reverse proxy caching based on a Layer 4 CSS switch (CS150) and a standalone Content Engine (ce1).

Step 1 Disable the load bypass feature to ensure that the Content Engine serves requests if it can.

```
ce1(config)# no bypass load enable
```

Step 2 Configure the Content Engine to accept traffic redirection using the Layer 4 switch.

```
ce1(config)# http 14-switch enable
```

Step 3 Configure the Layer 4 switch for reverse proxy caching.

The Layer 4 switch must be in configuration mode in order to initiate configuration changes.


Note

See the *CSS Advanced Configuration Guide* for more information regarding caching with a Layer 4 CSS switch.

a. Specify the owner of the CSS switch.

```
CS150(config)# owner cisco
```

b. Create a reverse proxy rule for the current owner.

```
CS150(config-owner[cisco])# content RPCRule
Create content RPCRule>, [y/n]:y
```

c. Add services to the reverse proxy rule. In this case, the standalone Content Engine (ce1) is added as a service to the reverse proxy rule.

```
CS150(config-owner-content[RPCRule])# add service ce1
```

d. Assign a virtual IP address to the CSS switch.

```
CS150(config-owner-content[cisco-RPCRule])# vip address 200.200.200.1
```

- e. Specify TCP as the service protocol.

```
CS150(config-owner-content[cisco-RPCRule])# protocol tcp
```

- f. Specify that requests for traffic should come through port 80.

```
CS150(config-owner-content[cisco-RPCRule])# port 80
```

- g. Define what content is cacheable.

```
CS150(config-owner-content[cisco-RPCRule])# url "/*" eq1 Cacheable
```

- Step 4** Exit configuration mode on the CSS switch.

```
CS150(config-owner-content[cisco-RPCRule])# exit
```

Using 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. For more information on this topic, see [Chapter 15, “Configuring Advanced Transparent Caching Features on Standalone Content Engines.”](#)

Deploying Streaming Media Services on Standalone Content Engines

Streaming is a technology that allows content to be accessed or viewed before all the media packets have been received, with caching, the content must be received in its entirety before it can be accessed. Streaming media can be delivered as live content or as on-demand content, such as video on demand (VOD).

Cisco ACNS software supports several types of streaming media solutions, including the Windows Media Technologies (WMT) solution from Microsoft Corporation and the RealMedia solution from RealNetworks, Inc. See [Table 1-3](#) for a list of supported streaming media solutions.

For information about how to deploy RealMedia streaming services on standalone Content Engines, see [Chapter 8, “Configuring RealMedia Services on Standalone Content Engines.”](#) For information about how to deploy WMT streaming services on standalone Content Engines, see [Chapter 9, “Configuring WMT Streaming Media Services on Standalone Content Engines.”](#)

For information about how to configure streaming media services for Content Engines that are registered with a Content Distribution Manager, see the *Cisco ACNS Software Configuration Guide for Centrally Managed Deployments, Release 5.5*.

Deploying Filtering and Access Control Services

After you configure caching and streaming services on a standalone Content Engine, you can configure specific content services (for example, rules processing and URL filtering). This section provides an example of how to configure a standalone Content Engine to control user Internet access. In this example, a company wants to deploy a standalone Content Engine as a caching engine in its enterprise. The company also has the following special requirements for processing content requests:

- The standalone Content Engine should only allow Internet access to users on selective subnets, and should limit their Internet access to specific websites.
- The standalone Content Engine should block specific users on the permitted subnets from accessing the permitted websites.
- The standalone Content Engine should block Internet access to all other users, and display a custom message informing users that their request to access the website has been denied.
- Because there are only a small number of sites and users, the company does not want to implement a third-party URL filtering solution (for example, the SmartFilter product).

This example shows how to implement a solution that meets the above requirements with a standalone Content Engine that is running the ACNS 5.x software.

Step 1 Enable rules processing on the standalone Content Engine.

```
ContentEngine(config)# rule enable
```

Step 2 Block selective users who belong to the permitted subnet from accessing the normally permitted sites. In this case, the Content Engine blocks users who belong to the subnet 192.168.1.50 and the domain .foo\com from accessing the normally permitted sites.

```
ContentEngine(config)# rule action block pattern-list 2 protocol all
ContentEngine(config)# rule pattern-list 2 group-type and
ContentEngine(config)# rule pattern-list 2 src-ip 192.168.1.50 255.255.255.0
ContentEngine(config)# rule pattern-list 2 domain .*foo\.com
```

Step 3 Define a custom message that you want the Content Engine to display to a blocked user.

- a. Create a separate directory under local1 or local2 for holding the custom message file.
- b. Create an HTML file named block.html that contains the blocking message. Make sure to copy all embedded graphics associated with the custom message HTML page to the same directory that contains the block.html file. The following is an example of a block.html file:

```
<TITLE>Cisco Content Engine example customized message for url-filtering</TITLE>
<p>
<H1>
<CENTER><B><I><BLINK>
<FONT COLOR="#800000">P</FONT>
<FONT COLOR="#FF00FF">R</FONT>
<FONT COLOR="#00FFFF">A</FONT>
<FONT COLOR="#FFFF00">D</FONT>
<FONT COLOR="#800000">E</FONT>
<FONT COLOR="#FF00FF">E</FONT>
<FONT COLOR="#00FFFF">P</FONT>
<FONT COLOR="#FF8040">'</FONT>
<FONT COLOR="#FFFF00">S</FONT>
</BLINK>
<FONT COLOR="#0080FF">Blocked Page</FONT>
</I></B></CENTER>
```

```

</H1>
<p>
<p>
<IMG src="/content/engine/blocking/url/my.gif">
<p>
This page is blocked by the Content Engine.
<p>

```

- Step 4** Configure the Content Engine to send blocked users the custom message. Specify the directory (dirname) the block.html file.

```
ContentEngine(config)# url-filter http custom-message dirname
```

In this example, the block.html file displays the following custom message when the Content Engine intercepts a request to the blocked site:

This page is blocked by the Content Engine

- Step 5** Configure the Content Engine to deny client requests for URLs that are listed in a badurl.lst file, or configure it to fulfill only requests for URLs that are in a goodurl.lst file. The use of URL lists applies to requests in HTTP, HTTPS, and FTP format as well as streaming media protocols such as MMS-over-HTTP and RTSP. The following example shows how to permit specific HTTP URLs to the exclusion of all other URLs:

- a. Create a plain text file named goodurl.lst.
- b. In the goodurl.lst file, enter the URLs that you want exclusively to allow. The list of URLs in the goodurl.lst file must be written in the form `http://www.domain.com` and be delimited with carriage returns.
- c. Copy the goodurl.lst file to the `/local1` sysfs directory of the Content Engine.



Note We recommend creating a separate directory under local1 to hold the good lists, for example, `/local1/filtered_urls`.

- Step 6** Point the Content Engine to the goodurl.lst filename.

```
ContentEngine(config)# url-filter http good-sites-allow file local/local1/goodurl.lst
```

- Step 7** Configure the Content Engine to actively permit access to only the good URLs.

```
ContentEngine(config)# url-filter http good-sites-allow enable
```

For more information about how to configure URL filtering and rules on standalone Content Engines, see [Chapter 11, “Configuring Content Preloading and URL Filtering on Standalone Content Engines”](#) and [Chapter 13, “Configuring the Rules Template on Standalone Content Engines.”](#)