CHAPTER 1

Content Distribution Overview

What is Content Distribution?

In this guide content is downloadable media on a network. That media may come in many forms, for example, video files, text files, presentations, pictures, or any other file type. Consider for a moment the life span of a specific type of content. It is created at some unique point in time, it is given a filename, and it is generally saved to a storage device. In that sense, this piece of content is unique in the world. If that piece of content is considered interesting to large audiences, it is eventually replicated on many different storage devices, typically personal computers and the like. Without a content distribution mechanism, the content must be downloaded from a web server each time a client makes a request for it, taxing the WAN (Wide Area Network) link linearly and exacerbating or even creating congestion issues on the network. Problems also arise when the original piece of content is modified. The copies of the content are effectively out of date from that point on.

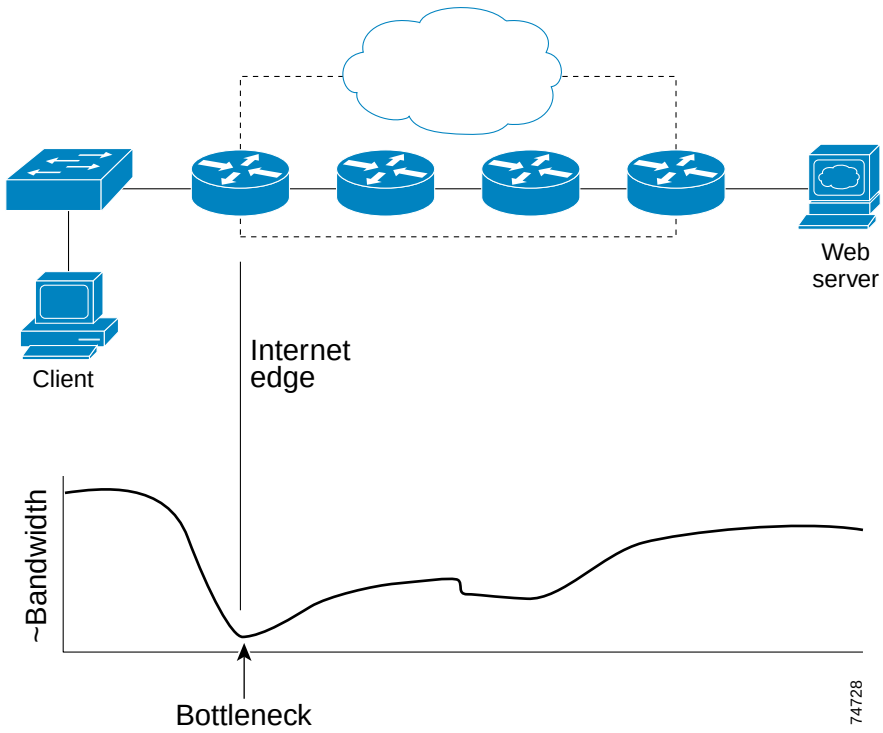
The concept of content distribution was born out of a simple need to have content placed closer to those clients making requests for it. Content distribution technologies have become forerunners in addressing some of the issues that have been generated as a result of the Web's rapid growth, such as slow content download times, WAN link congestion, and content freshness.

The Bottleneck Theory

A data stream can only traverse through a network as quickly as the slowest link in its path allows. In theory, every path through a network has a potential bottleneck. A bottleneck is defined as a point in the network that is responsible for the greatest level of packet latency. The bottleneck is not always immediately identifiable by simply looking at a network diagram. Lots of factors weigh into the equation but the primary suspect is network bandwidth. This chapter is meant to address how to design and deploy content distribution technologies to mitigate the effects of network bottlenecks and their effects on the applications that run across them.

In enterprises, the network bottleneck is typically located at the Internet edge. Most networks were originally designed using the 80/20 rule. The 80/20 rule states that 80% of network traffic is generally on the LAN and 20% of network traffic is on the WAN. Due to the advent of the Internet, that rule has flipped. Now most of the traffic from the network transverses the WAN. However, due to cost constraints, enterprises can not afford to increase the bandwidth of their WAN links to keep up with the growing demand. Since the majority of enterprise web traffic traverses the WAN links that comprise the Internet edge, this point in the network becomes the point of focus for bottlenecks. Figure 1-1identifies the typical drop in bandwidth experienced between the LAN and the WAN. Figure 1-1illustrates The Bottleneck Theory.

Figure 1-1 The Bottleneck Theory



Early Content Distribution Solutions

The Content distribution concept has been in existence for some time. Consider the lone piece of content described earlier. Prior to any elegant content distribution mechanisms, the choices for distribution applications were e-mail, File Transfer Protocol (FTP), UNIX-to-UNIX Copy Protocol (UUCP), Trivial File Transfer Protocol (TFTP), and Network File System (NFS). In a crude fashion, an e-mail system can serve as a content distribution solution. A person simply needs to attach a piece of content to an e-mail destined for a group alias and a single piece of content is sent to a large, dispersed audience. Enterprises have been using FTP to transfer files to and from remote servers for years. A script could be written so that once a day it copies content from one place to another. Again, a crude, but real content distribution method. The problem, of course, with these kinds of methods are that they are not always the most efficient uses of network resources and in some cases present content management and control issues.

Current Content Distribution Solutions

Today, content distribution is multifaceted, including caching, pre-positioning, revision management, and request routing solutions. Each one is described in the following sections.

Caching Solution

As the Web grows, more and more content is being placed on web servers. One of the nicest things about moving content onto a web server is that in most cases any browser can access the content, regardless of the client operating system. The need for proprietary application protocols is also reduced since most web traffic is over the industry standard, HTTP (Hyper Text Transfer Protocol). And since most content is at least a few router hops away from a client's perspective, and most likely across a WAN link, the network bottleneck becomes a significant factor in the amount of time it takes a piece of content to traverse the network. Furthermore, if lots of individuals make requests for the same content over the same network path, where the network is taxed linearly for each request made, network bottlenecks become an issue.

Caching solves this problem by serving content from a point that is closer to the client than the bottleneck is, hence avoiding the slowest link altogether. The reciprocal benefit is that the bottleneck is not taxed with additional traffic. The question now becomes how does the caching device get the interesting piece of content in the first place?

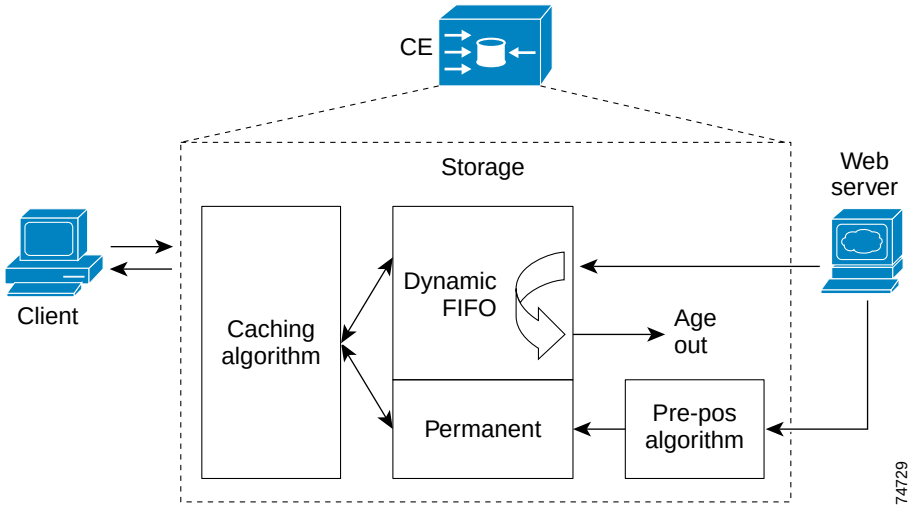
A caching device, such as a Cisco Content Engine, is placed strategically in the path of the request for content. The request for content is sent to a content engine that checks to see if that piece of content is in local storage. If it is, the content engine serves the request instead the client having to contact the Web server directly. If the content engine does not have the content stored locally, it makes a request to the web site for the piece of content, places it in local storage (with an expire timestamp), and delivers it to the client. Note that the very first request for a piece of content incurs normal fetch delays, however, while that same piece of content is still fresh (consistent with the original copy), the content engine services all subsequent client requests from local storage. The results can be drastic improvements in overall client experience and network bottleneck congestion.

Caching devices have a finite amount of local storage that they use to serve content from. Once that storage is full, the content that has been in storage the longest is overwritten by the most recently requested content. In other words, content has a shelf-life within a cache device. A caching device will also reclaim storage space from a piece of content if the content becomes stale, or outdated.

Pre-Positioning Solution

As content files continue to become larger, the need for alternative caching mechanisms becomes a reality. Consider the balance between the size of a piece of content, the frequency in which it is requested, and the amount of time it takes for a content engine to fetch it from the web server. Recall that in a traditional caching algorithm, the first request for a piece of content incurs download delays subject to bottlenecks between themselves and the web server. Download times may become very long for large pieces of content that are traversing a particularly slow bottleneck, so much so that the client experience is unacceptable. Also consider that the download of a large piece of content will consume a lot of bandwidth, potentially inflicting unacceptable delays on other network traffic contending for bandwidth across the bottleneck. There is a point where it makes a lot of sense to place larger pieces of content on the content engine prior to any clients actually requesting it. Furthermore, it makes sense to pin that content in place indefinitely, or at least until it is explicitly removed or updated. Therefore, pre-positioned content has its own place in storage on a content engine, separate from the space used by the caching algorithm, as shown in Figure 1-2.

Figure 1-2 Cisco-Content Engine



Getting Content to the Edge of the Network

In a traditional caching algorithm, the content engine reacts to a client request by fetching content from the origin server and storing it locally. This achieves the goal of getting content closer to the client pool and is in fact one form of content distribution. However, in the case of pre-positioned content, the content distribution mechanism is slightly different. Rather than being reactive, as in the case of caching, the pre-positioning mechanism is proactive in nature. A network administrator must schedule the replication of content out to the individual content engines prior to making the content available to the clients. This introduces a number of additional system considerations.

Pre-positioning content can be a costly task in terms of network utilization, time, and management, which dictates that not all content should be pre-positioned. Primary candidates for pre-positioning are streaming media files. Streaming media files are good candidates for pre-positioning for the following two reasons:

- First, streaming media files tend to be quite large, which can be a resource monopolizer if passed through a caching device.
- Second, there is an element of user experience directly related to the quality of the stream.
  - If streaming media is deprived of ample network bandwidth due to a bottleneck, the client notices a degradation in the overall quality of the stream.
  - This implies that, whenever possible, streaming media files should be pre-positioned somewhere between the client and the network bottleneck, typically on the enterprise LAN (Local Area Network).
  - The network bottleneck for enterprises is commonly the last mile, which ends at the enterprise edge. This supports the need for content distribution even more.

The de facto method of getting content to the edge of the network is by HTTP. Content engines are given instructions either via the Command Line Interface (CLI) or from a Content Distribution Manager (CDM). When a content engine receives this instruction, it makes a request for the content from the web server using HTTP. The request looks like just another client request from the server's perspective.

The end result is a single network device that is responding to client requests for content in two different ways, although the client is unaware as to which way is used or, in many cases, that the content engine has even intercepted their request. All the client perceives is an increase in download speed. Correspondingly, the network bottleneck is relieved of the download traffic.

Change Management Solution

Recall that, in general, content is created once. Content distribution technologies address the challenges associated with scaling that content to a large, widely dispersed audiences. As soon as a piece of content is copied to some other point in the network, it is subject to revision issues. Consider the case where the original piece of content is altered. This immediately renders all of the other copies of the original content out-of-date, and not fresh. A major challenge in any content distribution solution is ensuring that the content that is being replicated somehow remains fresh.

Traditional caching algorithms use the freshness facilities that are inherent in HTTP. When a content engine fetches a piece of content from the origin server, it gives the content a timestamp and records the expiration date tag that the server has given to that piece of content. The expiration date is configured by the content author and can be tuned low for frequently modified content, such as weather and news, and tuned high for infrequently modified content, such as movie trailers. The most common method is for the content engine to send an If-Modified-Since (IMS) message to the server when it sees that a piece of content has expired. If the content has indeed changed, it fetches the modified version of the content. If not, it simply resets the expiration date and begins to age the content out again as before. This process continues dynamically and indefinitely.

In the case of a pre-positioning model, maintaining content freshness is a manual process. When an administrator modifies a piece of content, it is up to them to copy the latest version of the file to the various interested content engines. This potentially immense operational overhead required in keeping pre-positioned content fresh, contributed to the birth of Content Distribution Networks.

Dynamic Content and Content Distribution

Dynamic content is created at the time it is requested. This is appealing because a single web site can appear to have many different faces, although it is administered as a single site. This type of site requires advanced programming beyond simple Hypertext Markup Language (HTML), for example, Java and Perl programming. Content may take different forms based on user identification or some other user-based criteria such as cookie information. This technology allows the Web site to respond with personalized content rather than generic, which, from a marketing perspective, is quite valuable.

The problem with dynamic content, from a content distribution perspective, is that it is not cacheable by a content engine due to its unpredictable nature. It also is not capable of being pre-positioned due to its requirement that some external application must create the content before it can be served. Therefore, dynamic content and content distribution are for the most part disjointed technologies today.

Request Routing Solutions

The previous sections made note of the fact that for content distribution methods to work, a device known as a content engine must receive the intercepted client request for content. The reality is that there are a few ways for a client request to find its way to a Cisco Content Engine. The most basic way is to configure each client browser to point to a specific content engine, known as proxy caching solution. All other solutions where a client finds its way to a content engine use request routing methods. Whatever the method to route the request, the end goal is always to get the client to make a request to a content engine, whether it is explicitly or via some redirection mechanism.

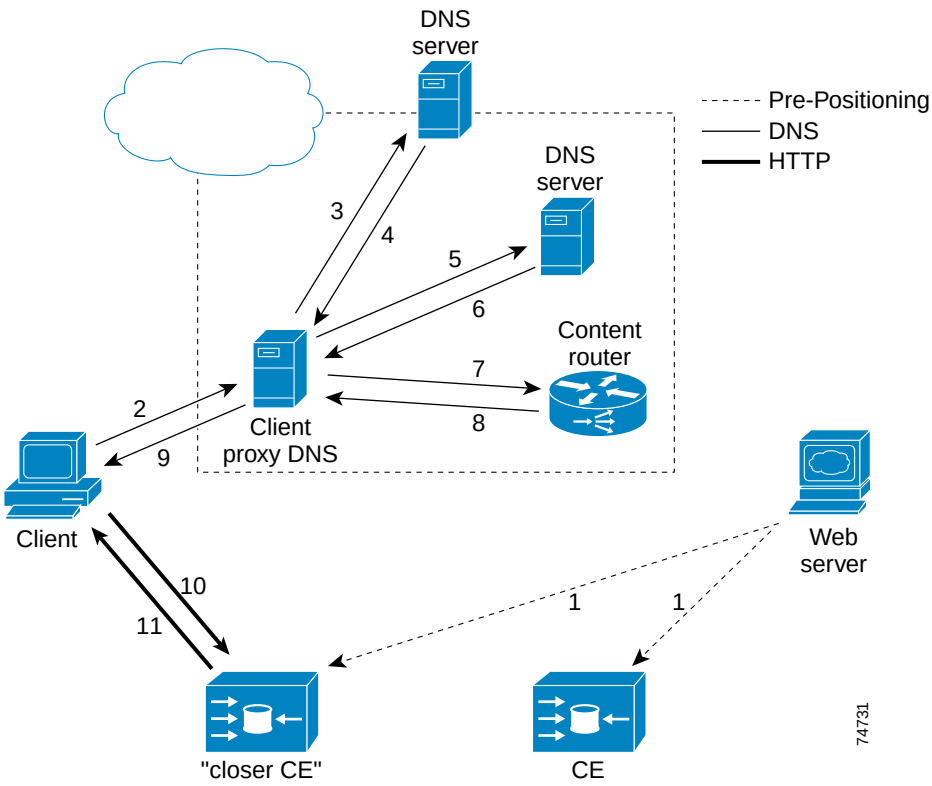
Request routing is the act of manipulating a client request so that it ultimately ends up at the proper content location, which in this case is a content engine. This is also known as redirection. Request routing algorithms vary but share this same goal. These algorithms may reside within a Cisco router, a content switch, or a device referred to as a Cisco Content Router. A Content Router is a request routing appliance that resides somewhere in the network. It redirects client requests to appropriate Content engines. The Content engines serve strictly as the caching or content pre-positioning devices when necessary. There is functional autonomy assigned to each component in the overall content distribution solution.

This begs the question, where is the request routing mechanism injected? Furthermore, when is a client subjected to it? There are three basic choices: during the DNS resolution process, at the access router using Web Cache Control Protocol (WCCP), or during the HTTP request. Each technology varies considerably and has its own pros and cons.

DNS-based Request Routing

Before a client can make an HTTP request for content, it must first resolve the domain name of the Web server it intends to make the HTTP request to. During that process, a DNS-based request routing mechanism intercepts the client's request for a DNS A-Record and hands back a content engine's IP address. The request routing algorithm will use proximity information to decide which of the available content engines in the network can best service that particular client's request. Proximity is defined by a metric, typically the round-trip-time between a content engine and the client DNS name server. Figure 1-3 displays DNS-based request routing.

Figure 1-3 DNS-based Request Routing Model



1. Content is replicated to content engines.
2. Client sends DNS request to Client Proxy DNS name server for domain name resolution.
3. Client proxy DNS name server sends DNS request to root DNS name servers.
4. Root DNS name server responds with Name Server (NS) record.
5. Client proxy DNS name server sends DNS request to name server specified in NS record.
6. DNS name server responds with NS record pointing to the Content Router.
7. Client proxy DNS name server sends DNS request to Content Router specified in NS record.
8. Content Router responds with Address Record[1] of closer content engine.
9. Client proxy DNS name server caches Address Record and responds to client request by handing off Address Record.
10. Client uses IP address from Address Record to make HTTP request to content engine for content.
11. Content engine services request by delivering content to client.

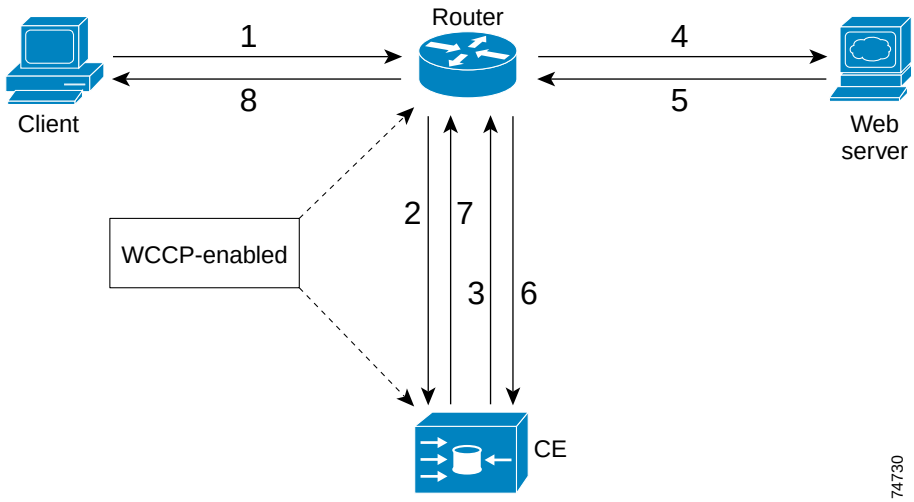
DNS-based request routing implies that the actual domain name that is resolved contains a subdomain delegated to the content router. Therefore, this is typically accomplished in conjunction with the modification of domain names within web pages.

1. In Cisco's currently shipping iCDN product, the content router returns an name server record pointing to a content engine and the local DNS name server must make one more requests to the content engine for the actual address record.

WCCP-Based Request Routing

When a client sends a request for content, the request eventually crosses an access router. That access router is configured to use WCCP as its redirection mechanism. When the router notices that the client is sending a request for content (based on TCP port, which is usually 80), it will forward that request directly to an attached content engine. All other traffic is forwarded out the appropriate router interface using normal routing procedures. WCCP Version 1 will only allow for TCP port 80 traffic to be intercepted. WCCP Version 2 affords the network administrator the ability to intercept traffic on any TCP or UDP port. Figure 1-4 illustrates WCCP based request routing.

Figure 1-4 WCCP Model



1. Client sends HTTP request out towards Internet using well-known TCP port (typically port 80).
2. WCCP-enabled router intercepts request and redirects to local Content engine.
3. If Content engine does not have content, it spawns HTTP request to web server and send to WCCP-enabled router.

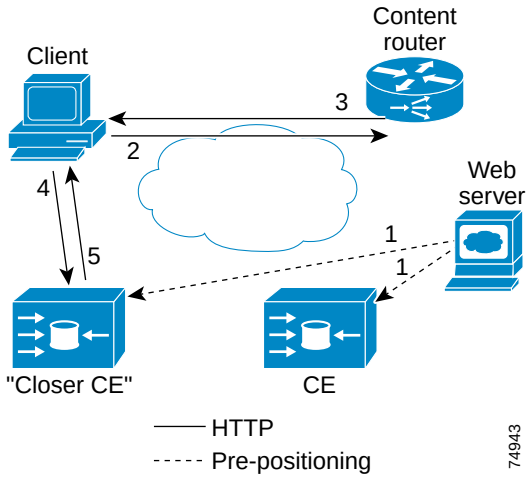
Note If Content engine already has content in local storage, skip to Step 7.

4. Router forwards request to web server.
5. Web server responds by sending content.
6. Router forwards content-to-content engine.
7. Content engine responds to clients HTTP request with content.
8. Router forwards content to client.

HTTP-Based Request Routing

HTTP-based request routing is accomplished by having the initial client request go to a request routing mechanism that possesses an HTTP server interface. The request routing mechanism will hand back an HTTP 302 redirect to the client's browser that points it to the Content engine in the network that is best suited to service the request. This decision is based on a subnet-matching scheme where content engines are given responsibility for their local subnets, called Coverage Zones.

Figure 1-5 HTTP-based Request Routing Model



1. Content is replicated to Content engines.
2. Client sends HTTP request to Content Router for content.
3. Content Router responds with HTTP 302 redirect.
4. Client sends HTTP request to Content engine specified in 302 redirect.
5. Content engine responds by sending content.

Applications of Content Distribution

The basic ideas covered in the previous sections can be applied in many ways. They serve as conceptual building blocks for addressing certain requirements, which include web application acceleration, improved client response times, and improved server performance. Content distribution solutions may include caching, pre-positioning, or both.

Improving Client Response Times (Proxy/Transparent Caching)

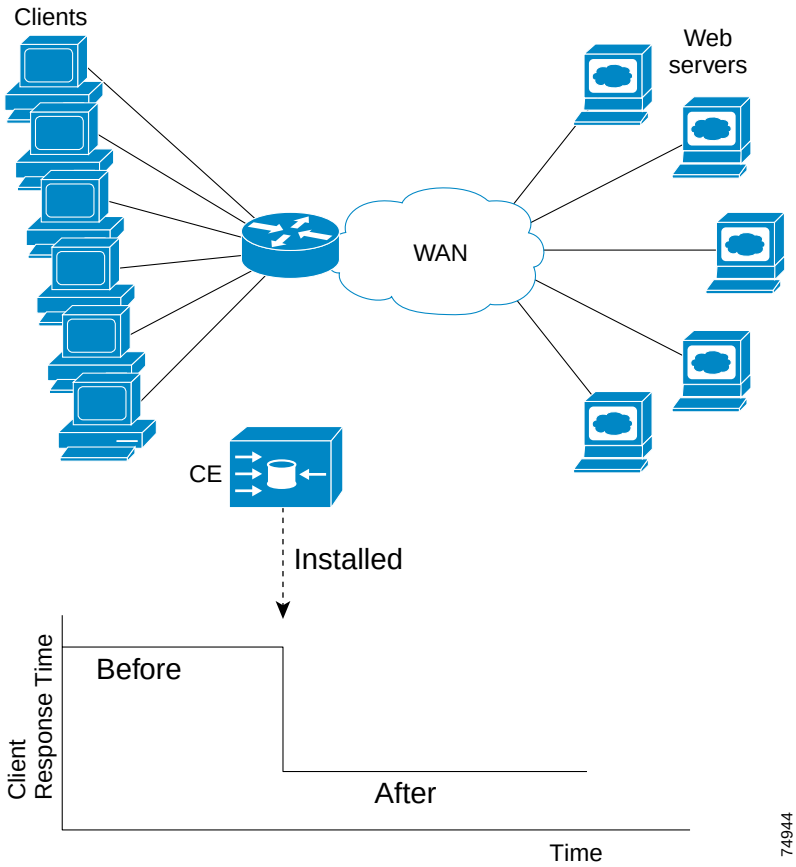
Probably the most recognizable forms of content distribution is proxy or transparent caching. This simply means that for a given client pool, client requests for content are serviced by local content engines and treated by the caching algorithm. The distinction between proxy and transparent caching is as follows:

- Proxy Caching – client's browser is explicitly configured to point to a content engine

- Transparent Caching – router or switch is configured to intercept client requests automatically and redirect them to a content engine using WCCP

Figure 1-6 illustrates the benefits of an ideal proxy/transparent caching deployment. Assume that before a content engine is deployed, average client response time for Internet web-based traffic is known. After a content engine is deployed, the average client response time for the same network under the same conditions will be reduced, resembling the step function improvement depicted in the graph.

Figure 1-6 Proxy/Transparent Caching Model



The result of implementing Proxy/Transparent Caching on an enterprise is a step function improvement in the client experience (as compared to the same network without this technology). Clients being serviced by a content engine will experience overall web response time improvements. The existing network will also breathe a bit easier due to the reduction in overall content traversing the WAN link.

Advanced techniques include hierarchical placement of content engines throughout larger enterprises. See the Cisco "*Caching in the Enterprise: Overview, Enterprise Proxy Caching,*" and "*Enterprise Transparent Caching*" documents for a detailed explanation of this technology.

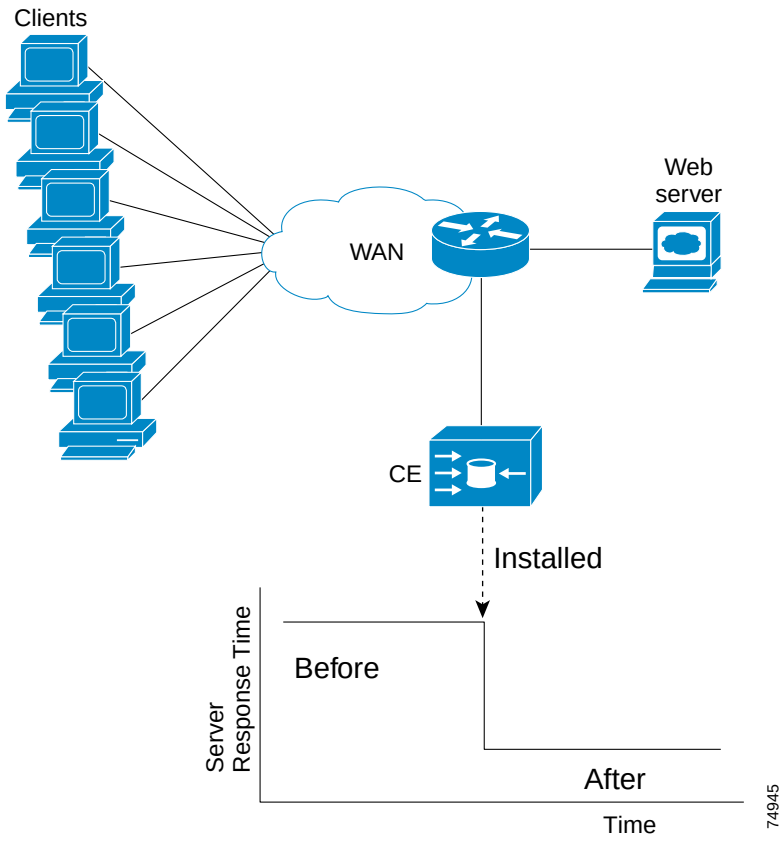
Web Server Scalability (Reverse Proxy Caching)

When faced with the fact that their current web server is not meeting client demands, a network administrator has a couple of immediate options. The most obvious option is to simply mirror the content on the original web server to a second server (or more). If the network administrator chooses this route, they must now purchase new server hardware and administer the second server along with the original, introducing added management complexities and cost. The other way an administrator can mitigate the traffic on the original server is to place one or more content engines in front of the server and have the engines intercept all client requests destined for content on the real server.

This method of front-ending web servers is known as Reverse Proxy Caching (RPC). RPC is so named due to the fact that the content engines use the same basic caching algorithm as in the Proxy/Transparent Caching model, but are configured as proxies on behalf of the server, not the client. In this case, it is the web servers that are targeted as beneficiaries.

Figure 1-7 illustrates the benefits of an ideal reverse proxy caching deployment. Assume that before a content engine is deployed, average server response time is known. After a content engine is deployed, the average server response time for the same network under the same conditions will be reduced, resembling the step function improvement depicted in the graph.

Figure 1-7 Reverse Proxy Caching Model



The result of implementing Reverse Proxy Caching is that the load on the web servers is reduced. This extends the scalability of the web server and concurrently decreases its response time under heavy loads. A reciprocal effect is that the clients accessing this particular web server experience potentially better server responsiveness as well, which is a win-win situation.

Targeted Acceleration for Specific Content (Content Delivery Networks)

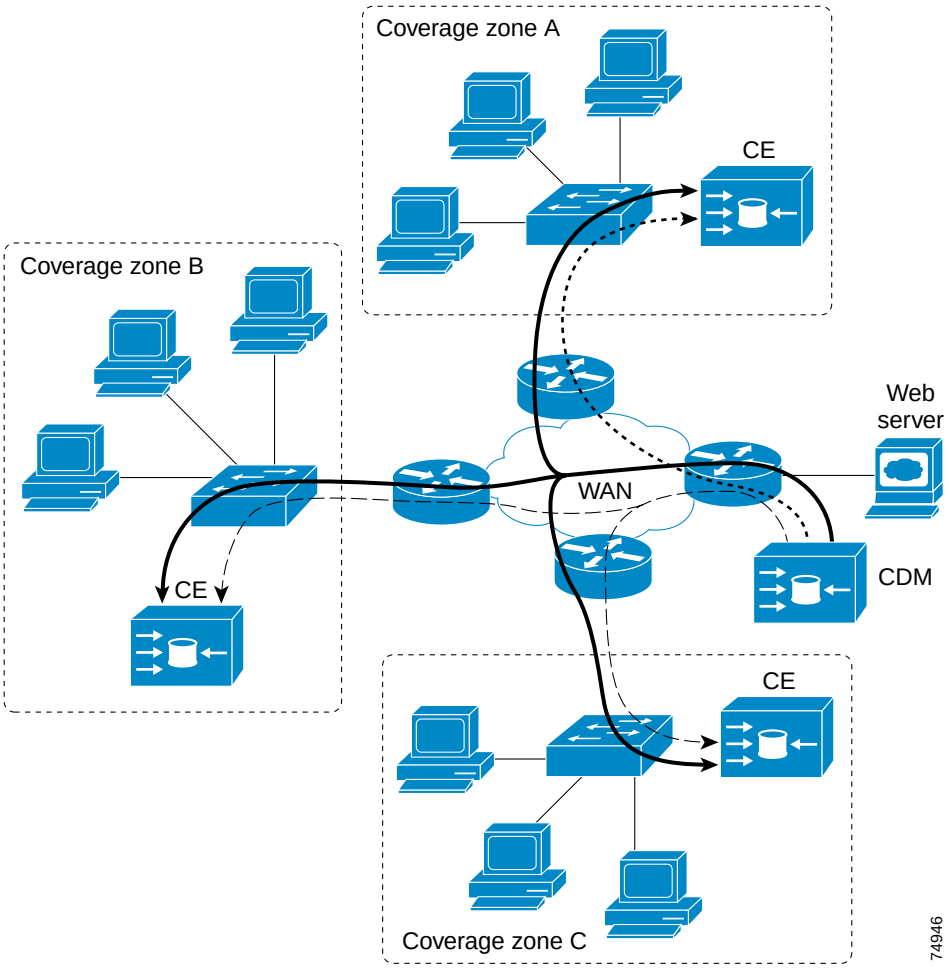
As discussed previously, clients subject to traditional caching algorithms used by proxy/transparent experience delays on occasion when the content request is not served from the content engine. This “first requestor” characteristic of traditional caching algorithms may not be tolerable in certain situations. Content pre-positioning addresses this issue. However, for large scale enterprises, managing lots of pre-positioned content can quickly become an unruly exercise.

This problem and others are addressed with a concept known as Content Delivery Networks (CDNs). They address content distribution, request routing, and content management issues by consolidating them into a single deployment architecture. Their value is most evident in large-scale deployments where an enterprise requires centralized control over all aspects of its web content and current content distribution techniques.

CDN capabilities continue to evolve. To date, the primary goal of CDNs has been to address the issues associated with large scale content pre-positioning. Certain content can be pre-positioned for a certain group of clients, whereas some other content can be pre-positioned for some other group of clients. The result is the ability of CDNs to target certain pieces of content for an audience based on some group criteria or logical mapping. Content engines are tied to client pools via CDN Coverage Zones, and they obtain pre-positioned content by subscribing to CDN Channels. Coverage Zone and Channel assignments are easily manipulated to flexibly facilitate this many to many relationship between content and clients.

Note As depicted in Figure 1-8, content engines may subscribe to multiple channels and multiple content engines can subscribe to the same Channel.

Figure 1-8 Content Distribution Model



	Channel		
Coverage Zone	**One**	**Two**	**Three**
A		Dotted Line	Solid Line
B	Dashed Line		Solid Line
C	Dashed Line		Solid Line

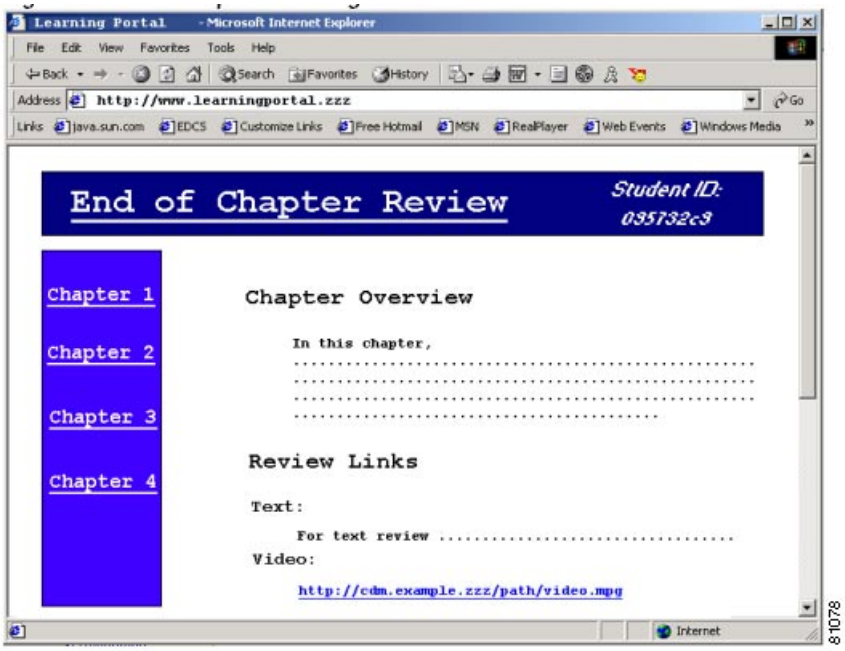
Cisco offers a Content Distribution Manager (CDM) to address the content management aspects of pre-positioning content. Cisco Content Routers (CR) address request routing aspects. Content engines (CE) address the content delivery aspect of CDNs. Depending on solution requirements, the CR may be omitted from the architecture due to the fact that the CDM may possess an embedded request routing algorithm.

Application Services on Content Distribution Infrastructure

Layering application services on top of a content distribution network requires an understanding of the interface between the application and the network. Cisco products address the immediate network infrastructure requirements by offering AVVID networking technologies such as high availability, security, multicast, QoS, and the content distribution technologies discussed in this chapter. Higher-level solutions such as E-learning require a number of other technologies in addition to those discussed in this chapter.

In many cases it is imperative to employ content distribution technologies since E-learning solutions may naturally rely heavily on them. For example, an E-learning solution may require that a student view training videos in preparation for end of chapter exams. The current state of that student's academic progression is tracked via a learning management system application. When it is time for the student to review the chapter, they are prompted to view the applicable training video. The link may have been dynamically generated as part of the curriculum that particular student has chosen to follow. The link points to a video that has been pre-positioned on a content engine that is close to the student. When the student clicks on the link to view the video, the Request Routing mechanism redirects the student's browser to the content engine that is responsible for servicing the student's coverage zone.

Figure 1-9 Example E-learning Portal



There are a number of applications that use the content distribution technologies discussed in this chapter, including learning management systems and content authoring and publishing software. For more on E-learning, see the *E-Learning Overview* document.

Conclusion

Content Distribution is a concept that multiple technologies address in a number of solution areas. The concept is simple – to scale a single piece of content to a large audience by replicating it to different points on the network. Those points are strategically better suited to serve individual client pools. Content is either reactively populated out to content engines via the traditional caching algorithms or proactively pre-positioned out at content engines. Choosing one method over the other depends on what kind of content is to be published and what portion of the network is targeted as the primary beneficiary. Getting to the content once it is on a content engine is a function of the request routing algorithm in place.

Content Distribution Networks are prevalent in larger, more elegant types of solutions. Content distribution technologies coupled with application services such as learning management systems comprise well-rounded E-learning solutions.

Content distribution is a viable, lasting consideration for most network administrators, where the benefits in most cases quickly justify its implementation.