Understanding Web Internals

HTTP

The Definitive Guide

David Gourley & Brian Totty
with Marjorie Sayer, Sailu Reddy & Anshu Aggarwal

O'REILLY®

HTTP

The Definitive Guide

David Gourley and Brian Totty
with Marjorie Sayer, Sailu Reddy, and Anshu Aggarwal

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

CHAPTER 6

Proxies

Web proxy servers are intermediaries. Proxies sit between clients and servers and act as "middlemen," shuffling HTTP messages back and forth between the parties.This chapter talks all about HTTP proxy servers, the special support for proxy features, and some of the tricky behaviors you'll see when you use proxy servers.

In this chapter, we:

- Explain HTTP proxies, contrasting them to web gateways and illustrating how proxies are deployed.
- Show some of the ways proxies are helpful.
- Describe how proxies are deployed in real networks and how traffic is directed to proxy servers.
- Show how to configure your browser to use a proxy.
- Demonstrate HTTP proxy requests, how they differ from server requests, and how proxies can subtly change the behavior of browsers.
- Explain how you can record the path of your messages through chains of proxy servers, using Via headers and the TRACE method.
- Describe proxy-based HTTP access control.
- Explain how proxies can interoperate between clients and servers, each of which may support different features and versions.

Web Intermediaries

Web proxy servers are middlemen that fulfill transactions on the client's behalf. Without a web proxy, HTTP clients talk directly to HTTP servers. With a web proxy, the client instead talks to the proxy, which itself communicates with the server on the client's behalf. The client still completes the transaction, but through the good services of the proxy server.

HTTP proxy servers are both web servers and web clients. Because HTTP clients send request messages to proxies, the proxy server must properly handle the requests and the connections and return responses, just like a web server. At the same time, the proxy itself sends requests to servers, so it must also behave like a correct HTTP client, sending requests and receiving responses (see Figure 6-1). If you are creating your own HTTP proxy, you'll need to carefully follow the rules for both HTTP clients and HTTP servers.

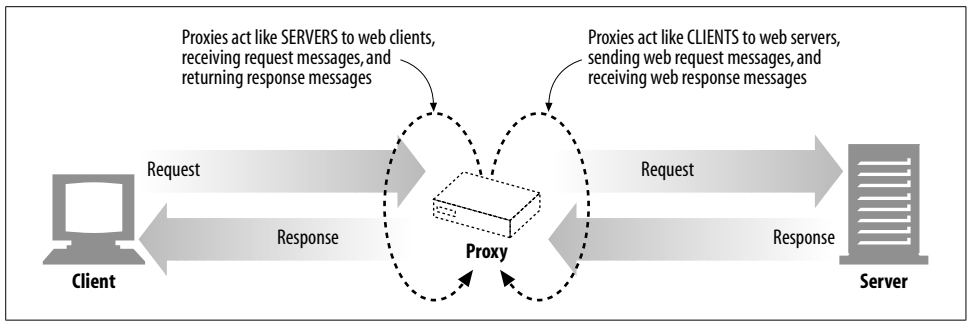

Figure 6-1. A proxy must be both a server and a client

Private and Shared Proxies

A proxy server can be dedicated to a single client or shared among many clients. Proxies dedicated to a single client are called *private proxies*. Proxies shared among numerous clients are called *public proxies*.

Public proxies

Most proxies are public, shared proxies. It's more cost effective and easier to administer a centralized proxy. And some proxy applications, such as caching proxy servers, become more useful as more users are funneled into the same proxy server, because they can take advantage of common requests between users.

Private proxies

Dedicated private proxies are not as common, but they do have a place, especially when run directly on the client computer. Some browser assistant products, as well as some ISP services, run small proxies directly on the user's PC in order to extend browser features, improve performance, or host advertising for free ISP services.

Proxies Versus Gateways

Strictly speaking, proxies connect two or more applications that speak the same protocol, while gateways hook up two or more parties that speak different protocols. A gateway acts as a "protocol converter," allowing a client to complete a transaction with a server, even when the client and server speak different protocols.

Figure 6-2 illustrates the difference between proxies and gateways:

- The intermediary device in Figure 6-2a is an HTTP proxy, because the proxy speaks HTTP to both the client and server.
- The intermediary device in Figure 6-2b is an HTTP/POP gateway, because it ties an HTTP frontend to a POP email backend. The gateway converts web transactions into the appropriate POP transactions, to allow the user to read email through HTTP. Web-based email programs such as Yahoo! Mail and MSN Hotmail are HTTP email gateways.

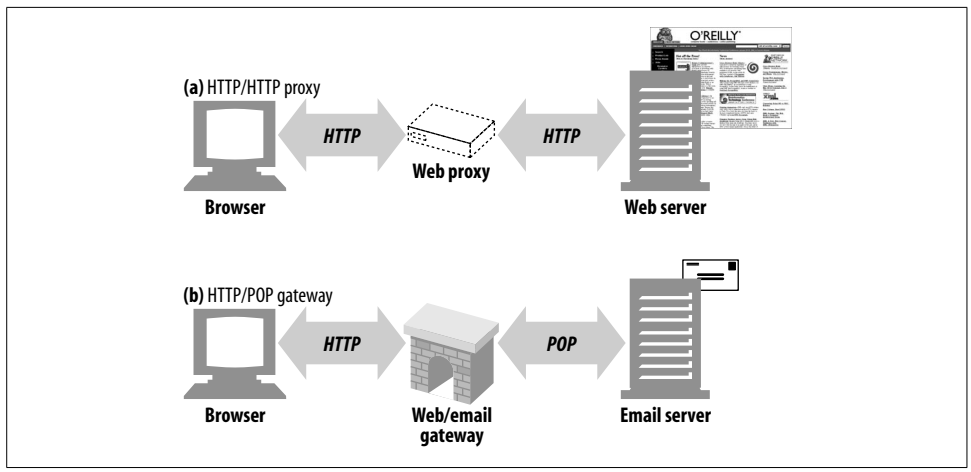

Figure 6-2. Proxies speak the same protocol; gateways tie together different protocols

In practice, the difference between proxies and gateways is blurry. Because browsers and servers implement different versions of HTTP, proxies often do some amount of protocol conversion. And commercial proxy servers implement gateway functionality to support SSL security protocols, SOCKS firewalls, FTP access, and web-based applications. We'll talk more about gateways in Chapter 8.

Why Use Proxies?

Proxy servers can do all kinds of nifty and useful things. They can improve security, enhance performance, and save money. And because proxy servers can see and touch all the passing HTTP traffic, proxies can monitor and modify the traffic to implement many useful value-added web services. Here are examples of just a few of the ways proxies can be used:

Child filter (Figure 6-3)
: Elementary schools use filtering proxies to block access to adult content, while providing unhindered access to educational sites. As shown in Figure 6-3, the

proxy might permit unrestricted access to educational content but forcibly deny access to sites that are inappropriate for children.*

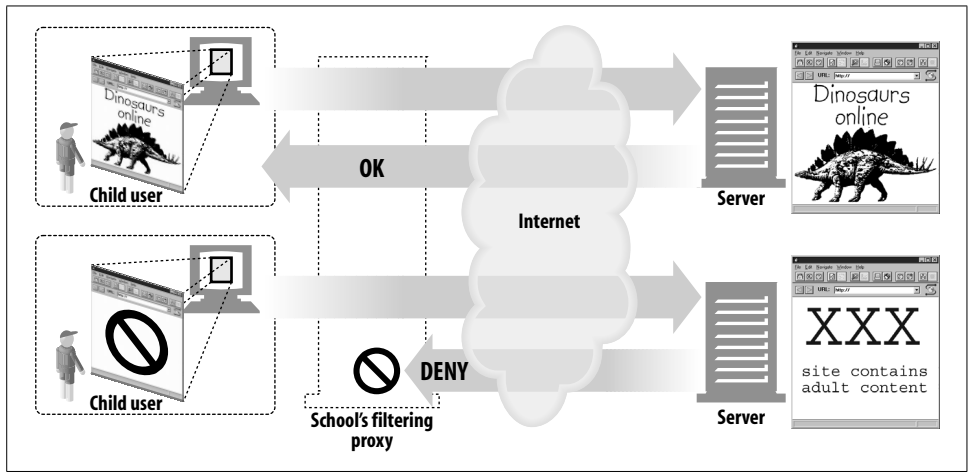

Figure 6-3. Proxy application example: child-safe Internet filter

Document access controller (Figure 6-4)

Proxy servers can be used to implement a uniform access-control strategy across a large set of web servers and web resources and to create an audit trail. This is useful in large corporate settings or other distributed bureaucracies.

All the access controls can be configured on the centralized proxy server, without requiring the access controls to be updated frequently on numerous web servers, of different makes and models, administered by different organizations.†

In Figure 6-4, the centralized access-control proxy:

- Permits client 1 to access news pages from server A without restriction
- Gives client 2 unrestricted access to Internet content
- Requires a password from client 3 before allowing access to server B

Security firewall (Figure 6-5)

Network security engineers often use proxy servers to enhance security. Proxy servers restrict which application-level protocols flow in and out of an organization, at a single secure point in the network. They also can provide hooks to scrutinize that traffic (Figure 6-5), as used by virus-eliminating web and email proxies.

* Several companies and nonprofit organizations provide filtering software and maintain "blacklists" in order to identify and restrict access to objectionable content.

† To prevent sophisticated users from willfully bypassing the control proxy, the web servers can be statically configured to accept requests only from the proxy servers.

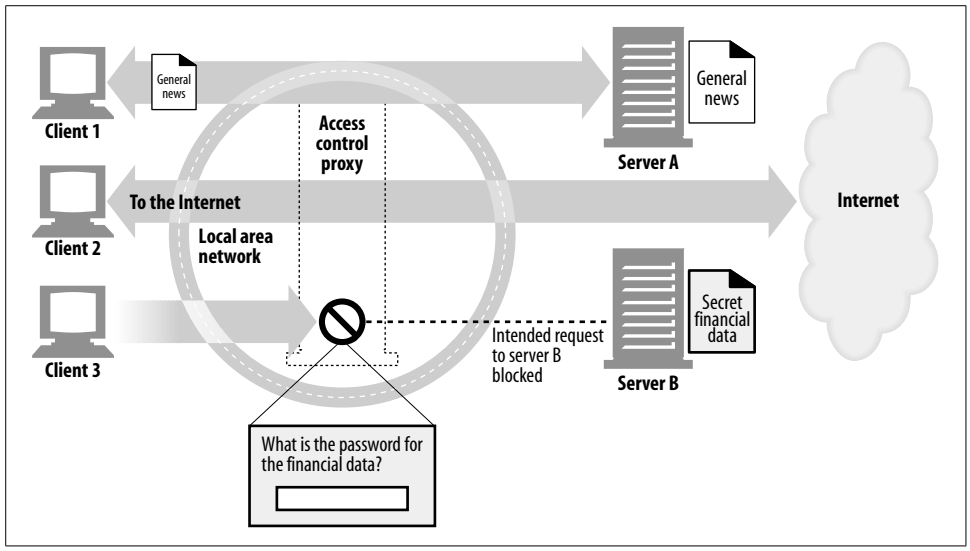

Figure 6-4. Proxy application example: centralized document access control

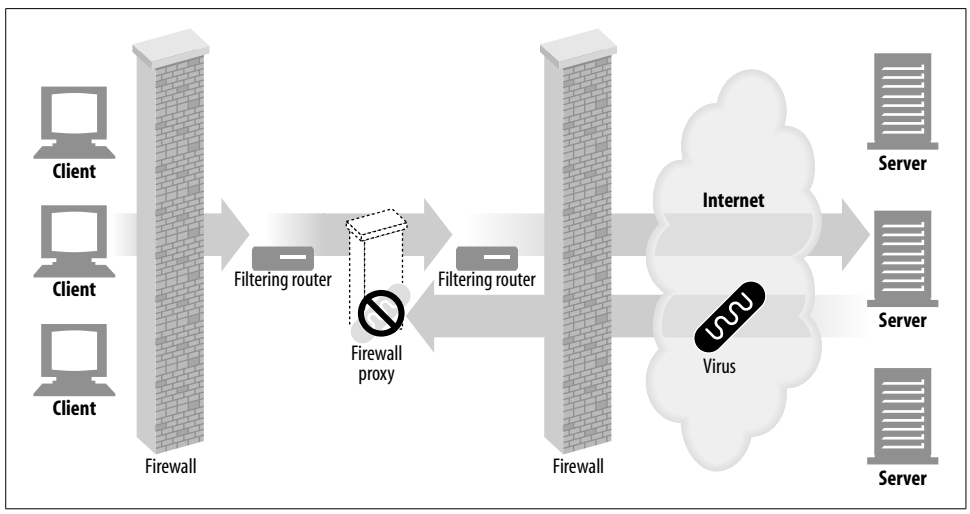

Figure 6-5. Proxy application example: security firewall

Web cache (Figure 6-6)

Proxy caches maintain local copies of popular documents and serve them on demand, reducing slow and costly Internet communication.

In Figure 6-6, clients 1 and 2 access object A from a nearby web cache, while clients 3 and 4 access the document from the origin server.

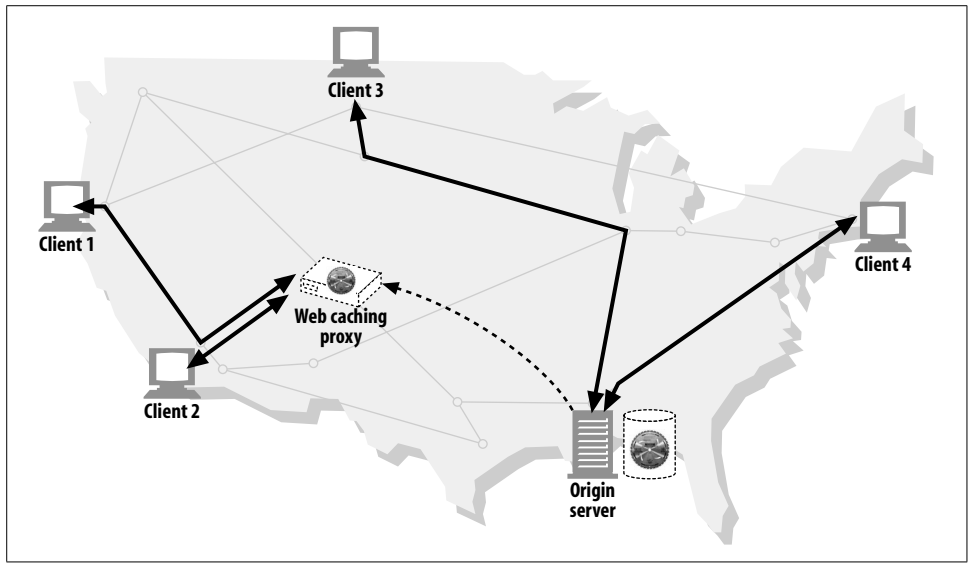

Figure 6-6. Proxy application example: web cache

Surrogate (Figure 6-7)

Proxies can masquerade as web servers. These so-called *surrogates* or *reverse proxies* receive real web server requests, but, unlike web servers, they may initiate communication with other servers to locate the requested content on demand.

Surrogates may be used to improve the performance of slow web servers for common content. In this configuration, the surrogates often are called *server accelerators* (Figure 6-7). Surrogates also can be used in conjunction with content-routing functionality to create distributed networks of on-demand replicated content.

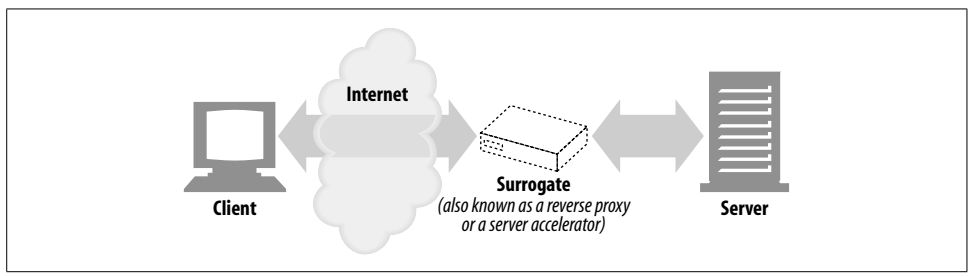

Figure 6-7. Proxy application example: surrogate (in a server accelerator deployment)

Content router (Figure 6-8)

Proxy servers can act as "content routers," vectoring requests to particular web servers based on Internet traffic conditions and type of content.

Content routers also can be used to implement various service-level offerings. For example, content routers can forward requests to nearby replica caches if the

user or content provider has paid for higher performance (Figure 6-8), or route HTTP requests through filtering proxies if the user has signed up for a filtering service. Many interesting services can be constructed using adaptive content-routing proxies.

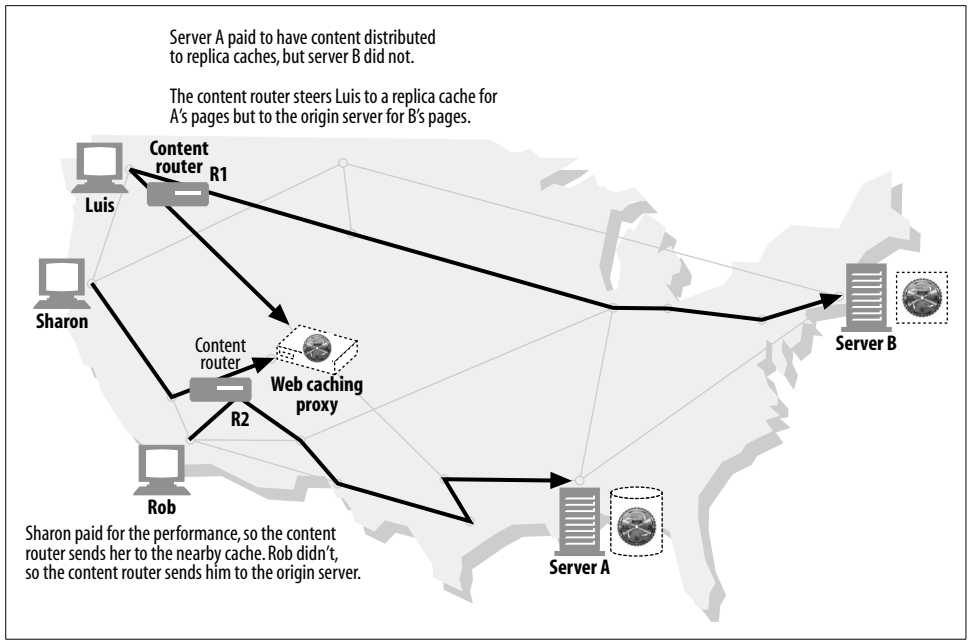

Figure 6-8. Proxy application example: content routing

Transcoder (Figure 6-9)

Proxy servers can modify the body format of content before delivering it to clients. This transparent translation between data representations is called *transcoding*.*

Transcoding proxies can convert GIF images into JPEG images as they fly by, to reduce size. Images also can be shrunk and reduced in color intensity to be viewable on television sets. Likewise, text files can be compressed, and small text summaries of web pages can be generated for Internet-enabled pagers and smart phones. It's even possible for proxies to convert documents into foreign languages on the fly!

Figure 6-9 shows a transcoding proxy that converts English text into Spanish text and also reformats HTML pages into simpler text that can displayed on the small screen of a mobile phone.

* Some people distinguish "transcoding" and "translation," defining transcoding as relatively simple conversions of the encoding of the data (e.g., lossless compression) and translation as more significant reformatting or semantic changes of the data. We use the term transcoding to mean any intermediary-based modification of the content.

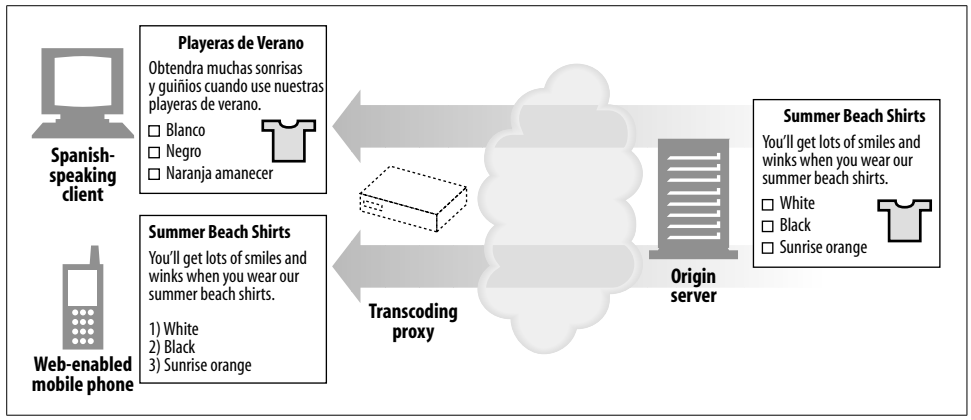

Figure 6-9. Proxy application example: content transcoder

Anonymizer (Figure 6-10)

Anonymizer proxies provide heightened privacy and anonymity, by actively removing identifying characteristics from HTTP messages (e.g., client IP address, From header, Referer header, cookies, URI session IDs).*

In Figure 6-10, the anonymizing proxy makes the following changes to the user's messages to increase privacy:

- The user's computer and OS type is removed from the User-Agent header.
- The From header is removed to protect the user's email address.
- The Referer header is removed to obscure other sites the user has visited.
- The Cookie headers are removed to eliminate profiling and identity data.

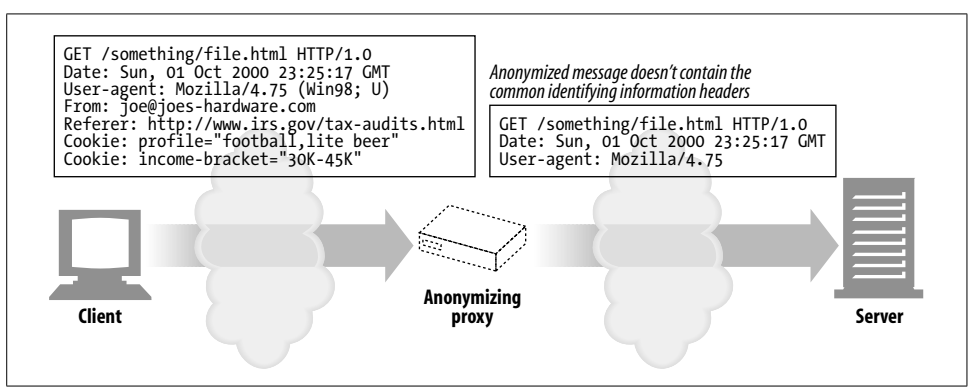

Figure 6-10. Proxy application example: anonymizer

* However, because identifying information is removed, the quality of the user's browsing experience may be diminished, and some web sites may not function properly.

Where Do Proxies Go?

The previous section explained what proxies do. Now let's talk about where proxies sit when they are deployed into a network architecture. We'll cover:

- How proxies can be deployed into networks
- How proxies can chain together into hierarchies
- How traffic gets directed to a proxy server in the first place

Proxy Server Deployment

You can place proxies in all kinds of places, depending on their intended uses. Figure 6-11 sketches a few ways proxy servers can be deployed.

Egress proxy (Figure 6-11a)
: You can stick proxies at the exit points of local networks to control the traffic flow between the local network and the greater Internet. You might use egress proxies in a corporation to offer firewall protection against malicious hackers outside the enterprise or to reduce bandwidth charges and improve performance of Internet traffic. An elementary school might use a filtering egress proxy to prevent precocious students from browsing inappropriate content.

Access (ingress) proxy (Figure 6-11b)
: Proxies are often placed at ISP access points, processing the aggregate requests from the customers. ISPs use caching proxies to store copies of popular documents, to improve the download speed for their users (especially those with high-speed connections) and reduce Internet bandwidth costs.

Surrogates (Figure 6-11c)
: Proxies frequently are deployed as surrogates (also commonly called reverse proxies) at the edge of the network, in front of web servers, where they can field all of the requests directed at the web server and ask the web server for resources only when necessary. Surrogates can add security features to web servers or improve performance by placing fast web server caches in front of slower web servers. Surrogates typically assume the name and IP address of the web server directly, so all requests go to the proxy instead of the server.

Network exchange proxy (Figure 6-11d)
: With sufficient horsepower, proxies can be placed in the Internet peering exchange points between networks, to alleviate congestion at Internet junctions through caching and to monitor traffic flows.*

* Core proxies often are deployed where Internet bandwidth is very expensive (especially in Europe). Some countries (such as the UK) also are evaluating controversial proxy deployments to monitor Internet traffic for national security concerns.

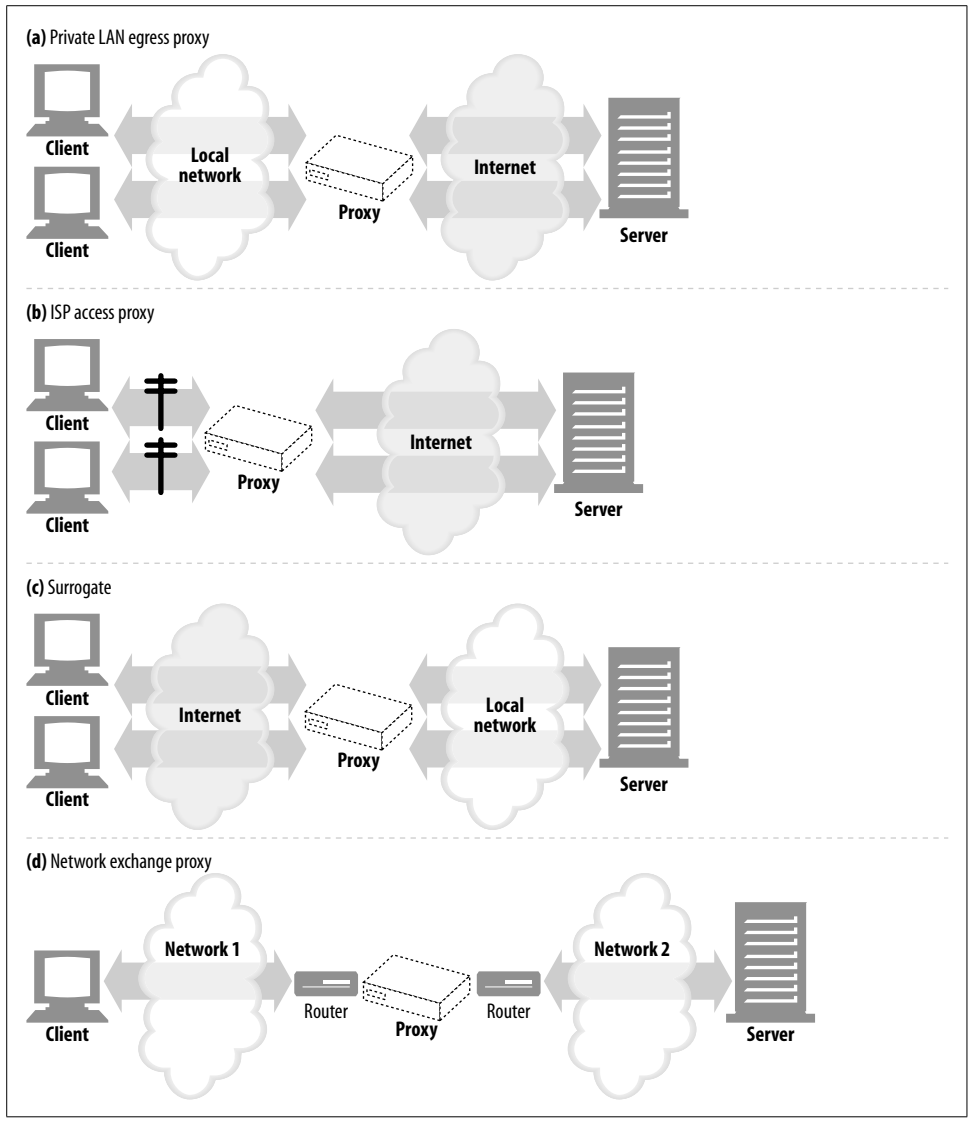

Figure 6-11. Proxies can be deployed many ways, depending on their intended use

Proxy Hierarchies

Proxies can be cascaded in chains called *proxy hierarchies*. In a proxy hierarchy, messages are passed from proxy to proxy until they eventually reach the origin server (and then are passed back through the proxies to the client), as shown in Figure 6-12.

Proxy servers in a proxy hierarchy are assigned *parent* and *child* relationships. The next *inbound* proxy (closer to the server) is called the parent, and the next *outbound* proxy (closer to the client) is called the child. In Figure 6-12, proxy 1 is the child

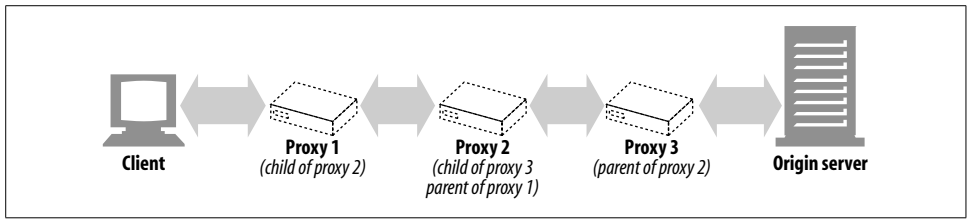

Figure 6-12. Three-level proxy hierarchy

proxy of proxy 2. Likewise, proxy 2 is the child proxy of proxy 3, and proxy 3 is the parent proxy of proxy 2.

Proxy hierarchy content routing

The proxy hierarchy in Figure 6-12 is static—proxy 1 always forwards messages to proxy 2, and proxy 2 always forwards messages to proxy 3. However, hierarchies do not have to be static. A proxy server can forward messages to a varied and changing set of proxy servers and origin servers, based on many factors.

For example, in Figure 6-13, the access proxy routes to parent proxies or origin servers in different circumstances:

- If the requested object belongs to a web server that has paid for content distribution, the proxy could route the request to a nearby cache server that would either return the cached object or fetch it if it wasn't available.
- If the request was for a particular type of image, the access proxy might route the request to a dedicated compression proxy that would fetch the image and then compress it, so it would download faster across a slow modem to the client.

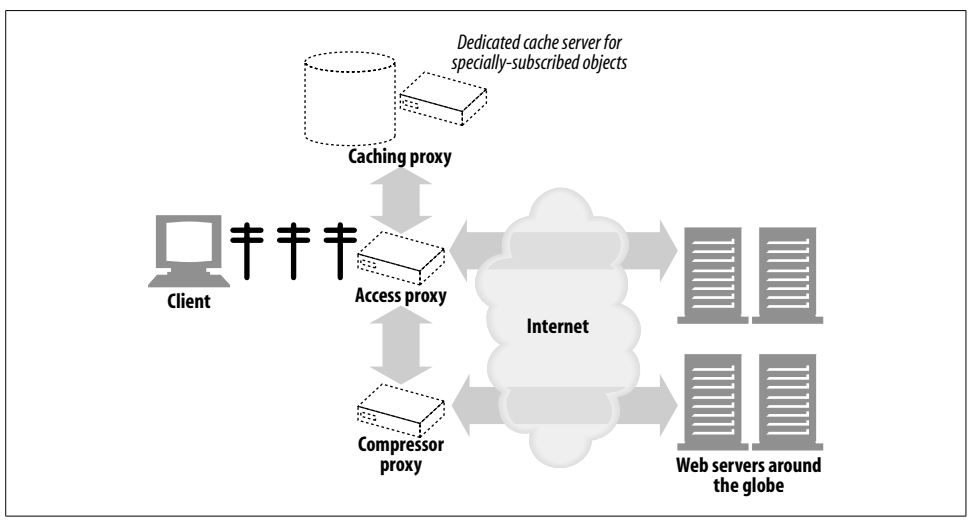

Figure 6-13. Proxy hierarchies can be dynamic, changing for each request

Here are a few other examples of dynamic parent selection:

Load balancing
: A child proxy might pick a parent proxy based on the current level of workload on the parents, to spread the load around.

Geographic proximity routing
: A child proxy might select a parent responsible for the origin server's geographic region.

Protocol/type routing
: A child proxy might route to different parents and origin servers based on the URI. Certain types of URIs might cause the requests to be transported through special proxy servers, for special protocol handling.

Subscription-based routing
: If publishers have paid extra money for high-performance service, their URIs might be routed to large caches or compression engines to improve performance.

Dynamic parenting routing logic is implemented differently in different products, including configuration files, scripting languages, and dynamic executable plug-ins.

How Proxies Get Traffic

Because clients normally talk directly to web servers, we need to explain how HTTP traffic finds its way to a proxy in the first place. There are four common ways to cause client traffic to get to a proxy:

Modify the client
: Many web clients, including Netscape and Microsoft browsers, support both manual and automated proxy configuration. If a client is configured to use a proxy server, the client sends HTTP requests directly and intentionally to the proxy, instead of to the origin server (Figure 6-14a).

Modify the network
: There are several techniques where the network infrastructure intercepts and steers web traffic into a proxy, without the client's knowledge or participation. This interception typically relies on switching and routing devices that watch for HTTP traffic, intercept it, and shunt the traffic into a proxy, without the client's knowledge (Figure 6-14b). This is called an *intercepting* proxy.*

Modify the DNS namespace
: Surrogates, which are proxy servers placed in front of web servers, assume the name and IP address of the web server directly, so all requests go to them instead

* Intercepting proxies commonly are called "transparent proxies," because you connect to them without being aware of their presence. Because the term "transparency" already is used in the HTTP specifications to indicate functions that don't change semantic behavior, the standards community suggests using the term "interception" for traffic capture. We adopt this nomenclature here.

of to the server (Figure 6-14c). This can be arranged by manually editing the DNS naming tables or by using special dynamic DNS servers that compute the appropriate proxy or server to use on-demand. In some installations, the IP address and name of the real server is changed and the surrogate is given the former address and name.

Modify the web server
: Some web servers also can be configured to redirect client requests to a proxy by sending an HTTP redirection command (response code 305) back to the client. Upon receiving the redirect, the client transacts with the proxy (Figure 6-14d).

The next section explains how to configure clients to send traffic to proxies. Chapter 20 will explain how to configure the network, DNS, and servers to redirect traffic to proxy servers.

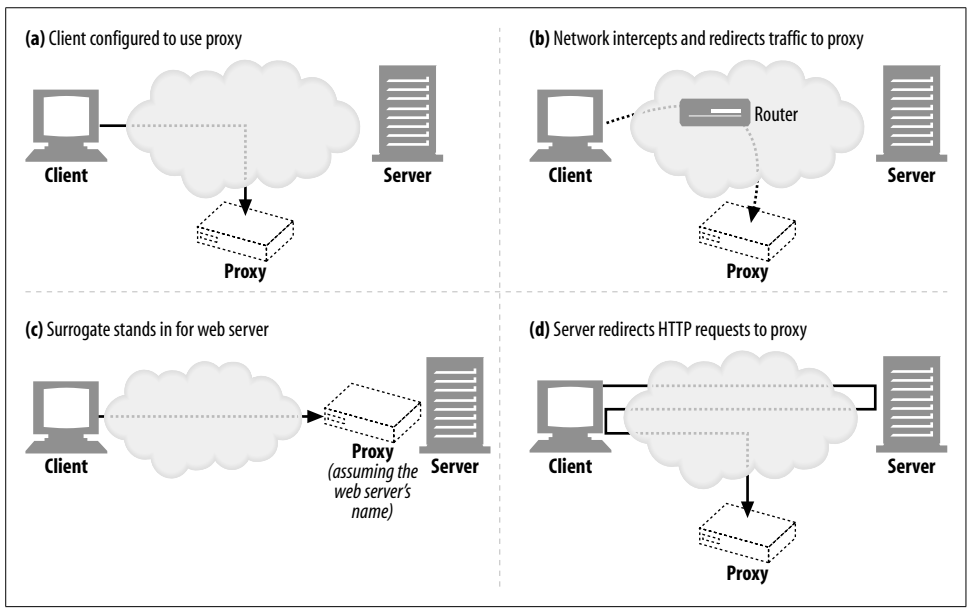

Figure 6-14. There are many techniques to direct web requests to proxies

Client Proxy Settings

All modern web browsers let you configure the use of proxies. In fact, many browsers provide multiple ways of configuring proxies, including:

Manual configuration
: You explicitly set a proxy to use.

Browser preconfiguration
: The browser vendor or distributor manually preconfigures the proxy setting of the browser (or any other web client) before delivering it to customers.

Proxy auto-configuration (PAC)
: You provide a URI to a JavaScript *proxy auto-configuration* (PAC) file; the client fetches the JavaScript file and runs it to decide if it should use a proxy and, if so, which proxy server to use.

WPAD proxy discovery
: Some browsers support the Web Proxy Autodiscovery Protocol (WPAD), which automatically detects a "configuration server" from which the browser can download an auto-configuration file.*

Client Proxy Configuration: Manual

Many web clients allow you to configure proxies manually. Both Netscape Navigator and Microsoft Internet Explorer have convenient support for proxy configuration.

In Netscape Navigator 6, you specify proxies through the menu selection Edit → Preferences → Advanced → Proxies and then selecting the "Manual proxy configuration" radio button.

In Microsoft Internet Explorer 5, you can manually specify proxies from the Tools → Internet Options menu, by selecting a connection, pressing "Settings," checking the "Use a proxy server" box, and clicking "Advanced."

Other browsers have different ways of making manual configuration changes, but the idea is the same: specifying the host and port for the proxy. Several ISPs ship customers preconfigured browsers, or customized operating systems, that redirect web traffic to proxy servers.

Client Proxy Configuration: PAC Files

Manual proxy configuration is simple but inflexible. You can specify only one proxy server for all content, and there is no support for failover. Manual proxy configuration also leads to administrative problems for large organizations. With a large base of configured browsers, it's difficult or impossible to reconfigure every browser if you need to make changes.

Proxy auto-configuration (PAC) files are a more dynamic solution for proxy configuration, because they are small JavaScript programs that compute proxy settings on the fly. Each time a document is accessed, a JavaScript function selects the proper proxy server.

To use PAC files, configure your browser with the URI of the JavaScript PAC file (configuration is similar to manual configuration, but you provide a URI in an "automatic configuration" box). The browser will fetch the PAC file from this URI and use

* Currently supported only by Internet Explorer.

the JavaScript logic to compute the proper proxy server for each access. PAC files typically have a *.pac* suffix and the MIME type "application/x-ns-proxy-autoconfig."

Each PAC file must define a function called `FindProxyForURL(url,host)` that computes the proper proxy server to use for accessing the URI. The return value of the function can be any of the values in Table 6-1.

Table 6-1. Proxy auto-configuration script return values

FindProxyForURL return value	Description
DIRECT	Connections should be made directly, without any proxies.
PROXY host:port	The specified proxy should be used.
SOCKS host:port	The specified SOCKS server should be used.

The PAC file in Example 6-1 mandates one proxy for HTTP transactions, another proxy for FTP transactions, and direct connections for all other kinds of transactions.

Example 6-1. Example proxy auto-configuration file

```
function FindProxyForURL(url, host) {
    if (url.substring(0,5) == "http:") {
        return "PROXY http-proxy.mydomain.com:8080";
    } else if (url.substring(0,4) =="ftp:") {
        return "PROXY ftp-proxy.mydomain.com:8080";
    } else {
        return "DIRECT";
    }
}
```

For more details about PAC files, refer to Chapter 20.

Client Proxy Configuration: WPAD

Another mechanism for browser configuration is the Web Proxy Autodiscovery Protocol (WPAD). WPAD is an algorithm that uses an escalating strategy of discovery mechanisms to find the appropriate PAC file for the browser automatically. A client that implements the WPAD protocol will:

- Use WPAD to find the PAC URI.
- Fetch the PAC file given the URI.
- Execute the PAC file to determine the proxy server.
- Use the proxy server for requests.

WPAD uses a series of resource-discovery techniques to determine the proper PAC file. Multiple discovery techniques are used, because not all organizations can use all techniques. WPAD attempts each technique, one by one, until it succeeds.

The current WPAD specification defines the following techniques, in order:

- Dynamic Host Discovery Protocol (DHCP)
- Service Location Protocol (SLP)
- DNS well-known hostnames
- DNS SRV records
- DNS service URIs in TXT records

For more information, consult Chapter 20.

Tricky Things About Proxy Requests

This section explains some of the tricky and much misunderstood aspects of proxy server requests, including:

- How the URIs in proxy requests differ from server requests
- How intercepting and reverse proxies can obscure server host information
- The rules for URI modification
- How proxies impact a browser's clever URI auto-completion or hostname-expansion features

Proxy URIs Differ from Server URIs

Web server and web proxy messages have the same syntax, with one exception. The URI in an HTTP request message differs when a client sends the request to a server instead of a proxy.

When a client sends a request to a web server, the request line contains only a partial URI (without a scheme, host, or port), as shown in the following example:

```
GET /index.html HTTP/1.0
User-Agent: SuperBrowserv1.3
```

When a client sends a request to a proxy, however, the request line contains the full URI. For example:

```
GET http://www.marys-antiques.com/index.html HTTP/1.0
User-Agent: SuperBrowser v1.3
```

Why have two different request formats, one for proxies and one for servers? In the original HTTP design, clients talked directly to a single server. Virtual hosting did not exist, and no provision was made for proxies. Because a single server knows its own hostname and port, to avoid sending redundant information, clients sent just the partial URI, without the scheme and host (and port).

When proxies emerged, the partial URIs became a problem. Proxies needed to know the name of the destination server, so they could establish their own connections to

the server. And proxy-based gateways needed the scheme of the URI to connect to FTP resources and other schemes. HTTP/1.0 solved the problem by requiring the full URI for proxy requests, but it retained partial URIs for server requests (there were too many servers already deployed to change all of them to support full URIs).*

So we need to send partial URIs to servers, and full URIs to proxies. In the case of explicitly configured client proxy settings, the client knows what type of request to issue:

- When the client *is not* set to use a proxy, it sends the partial URI (Figure 6-15a).
- When the client *is* set to use a proxy, it sends the full URI (Figure 6-15b).

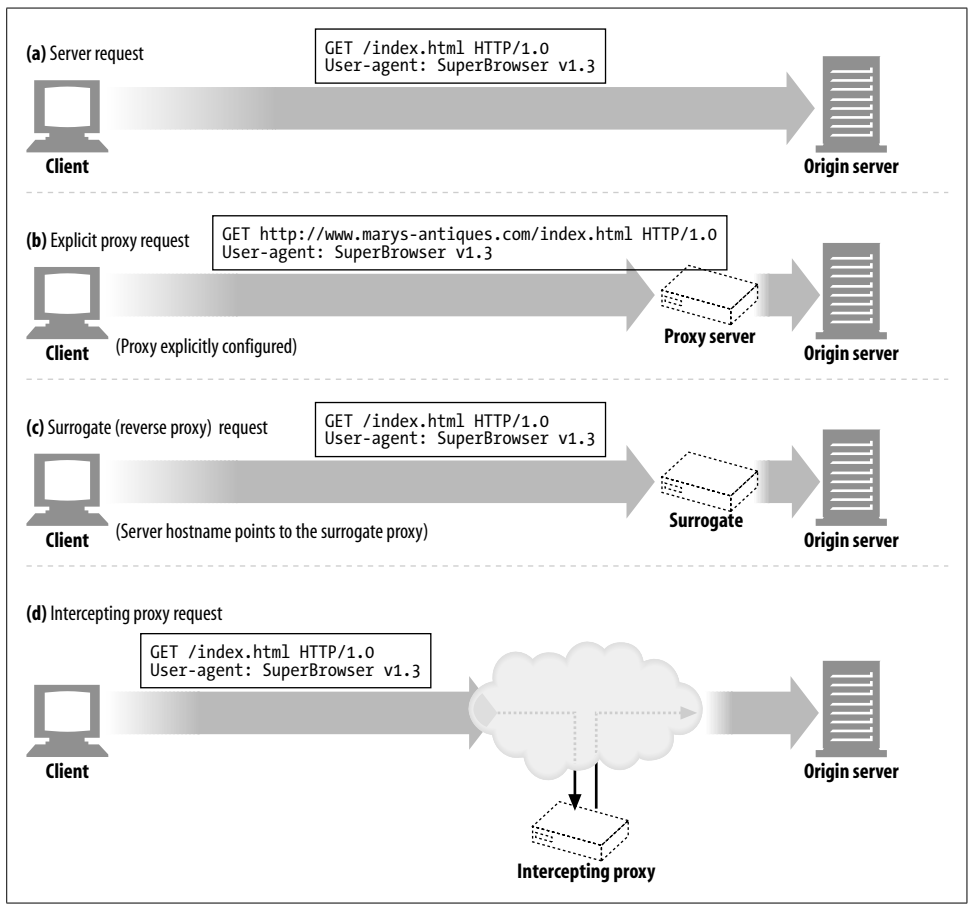

Figure 6-15. Intercepting proxies will get server requests

* HTTP/1.1 now requires servers to handle full URIs for both proxy and server requests, but in practice, many deployed servers still accept only partial URIs.

The Same Problem with Virtual Hosting

The proxy "missing scheme/host/port" problem is the same problem faced by virtually hosted web servers. Virtually hosted web servers share the same physical web server among many web sites. When a request comes in for the partial URI */index.html*, the virtually hosted web server needs to know the hostname of the intended web site (see "Virtually hosted docroots" in Chapter 5 and "Virtual Hosting" in Chapter 18 for more information).

In spite of the problems being similar, they were solved in different ways:

- Explicit proxies solve the problem by requiring a full URI in the request message.
- Virtually hosted web servers require a Host header to carry the host and port information.

Intercepting Proxies Get Partial URIs

As long as the clients properly implement HTTP, they will send full URIs in requests to explicitly configured proxies. That solves part of the problem, but there's a catch: a client *will not always know* it's talking to a proxy, because some proxies may be invisible to the client. Even if the client is not configured to use a proxy, the client's traffic still may go through a surrogate or intercepting proxy. In both of these cases, the client will think it's talking to a web server and won't send the full URI:

- A *surrogate*, as described earlier, is a proxy server taking the place of the origin server, usually by assuming its hostname or IP address. It receives the web server request and may serve cached responses or proxy requests to the real server. A client cannot distinguish a surrogate from a web server, so it sends partial URIs (Figure 6-15c).
- An *intercepting proxy* is a proxy server in the network flow that hijacks traffic from the client to the server and either serves a cached response or proxies it. Because the intercepting proxy hijacks client-to-server traffic, it will receive partial URIs that are sent to web servers (Figure 6-15d).*

Proxies Can Handle Both Proxy and Server Requests

Because of the different ways that traffic can be redirected into proxy servers, general-purpose proxy servers should support both full URIs and partial URIs in request messages. The proxy should use the full URI if it is an explicit proxy request or use the partial URI and the virtual Host header if it is a web server request.

* Intercepting proxies also might intercept client-to-proxy traffic in some circumstances, in which case the intercepting proxy might get full URIs and need to handle them. This doesn't happen often, because explicit proxies normally communicate on a port different from that used by HTTP (usually 8080 instead of 80), and intercepting proxies usually intercept only port 80.

The rules for using full and partial URIs are:

- If a full URI is provided, the proxy should use it.
- If a partial URI is provided, and a Host header is present, the Host header should be used to determine the origin server name and port number.
- If a partial URI is provided, and there is no Host header, the origin server needs to be determined in some other way:
  - If the proxy is a surrogate, standing in for an origin server, the proxy can be configured with the real server's address and port number.
  - If the traffic was intercepted, and the interceptor makes the original IP address and port available, the proxy can use the IP address and port number from the interception technology (see Chapter 20).
  - If all else fails, the proxy doesn't have enough information to determine the origin server and must return an error message (often suggesting that the user upgrade to a modern browser that supports Host headers).*

In-Flight URI Modification

Proxy servers need to be very careful about changing the request URI as they forward messages. Slight changes in the URI, even if they seem benign, may create interoperability problems with downstream servers.

In particular, some proxies have been known to "canonicalize" URIs into a standard form before forwarding them to the next hop. Seemingly benign transformations, such as replacing default HTTP ports with an explicit ":80", or correcting URIs by replacing illegal reserved characters with their properly escaped substitutions, can cause interoperation problems.

In general, proxy servers should strive to be as tolerant as possible. They should not aim to be "protocol policemen" looking to enforce strict protocol compliance, because this could involve significant disruption of previously functional services.

In particular, the HTTP specifications forbid general intercepting proxies from rewriting the absolute path parts of URIs when forwarding them. The only exception is that they can replace an empty path with "/".

URI Client Auto-Expansion and Hostname Resolution

Browsers resolve request URIs differently, depending on whether or not a proxy is present. Without a proxy, the browser takes the URI you type in and tries to find a corresponding IP address. If the hostname is found, the browser tries the corresponding IP addresses until it gets a successful connection.

* This shouldn't be done casually. Users will receive cryptic error pages they never got before.

But if the host isn't found, many browsers attempt to provide some automatic "expansion" of hostnames, in case you typed in a "shorthand" abbreviation of the host (refer back to "Expandomatic URLs" in Chapter 2):*

- Many browsers attempt adding a "www." prefix and a ".com" suffix, in case you just entered the middle piece of a common web site name (e.g., to let people enter "yahoo" instead of "www.yahoo.com").
- Some browsers even pass your unresolvable URI to a third-party site, which attempts to correct spelling mistakes and suggest URIs you may have intended.
- In addition, the DNS configuration on most systems allows you to enter just the prefix of the hostname, and the DNS automatically searches the domain. If you are in the domain "oreilly.com" and type in the hostname "host7," the DNS automatically attempts to match "host7.oreilly.com". It's not a complete, valid hostname.

URI Resolution Without a Proxy

Figure 6-16 shows an example of browser hostname auto-expansion without a proxy. In steps 2a–3c, the browser looks up variations of the hostname until a valid hostname is found.

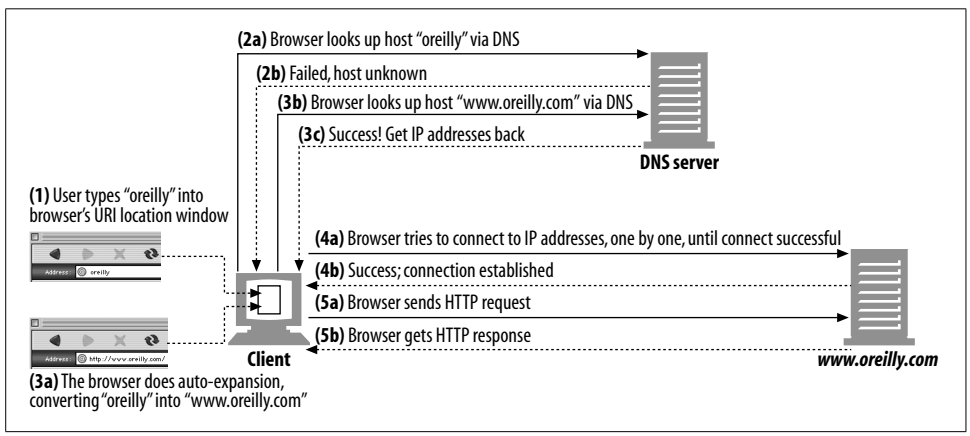

Figure 6-16. Browser auto-expands partial hostnames when no explicit proxy is present

Here's what's going on in this figure:

- In Step 1, the user types "oreilly" into the browser's URI window. The browser uses "oreilly" as the hostname and assumes a default scheme of "http://", a default port of "80", and a default path of "/".
- In Step 2a, the browser looks up host "oreilly." This fails.

* Most browsers let you type in "yahoo" and auto-expand that into "www.yahoo.com." Similarly, browsers let you omit the "http://" prefix and insert it if it's missing.

- In Step 3a, the browser auto-expands the hostname and asks the DNS to resolve "www.oreilly.com." This is successful.
- The browser then successfully connects to *www.oreilly.com*.

URI Resolution with an Explicit Proxy

When you use an explicit proxy the browser no longer performs any of these convenience expansions, because the user's URI is passed directly to the proxy.

As shown in Figure 6-17, the browser does not auto-expand the partial hostname when there is an explicit proxy. As a result, when the user types "oreilly" into the browser's location window, the proxy is sent "http://oreilly/" (the browser adds the default scheme and path but leaves the hostname as entered).

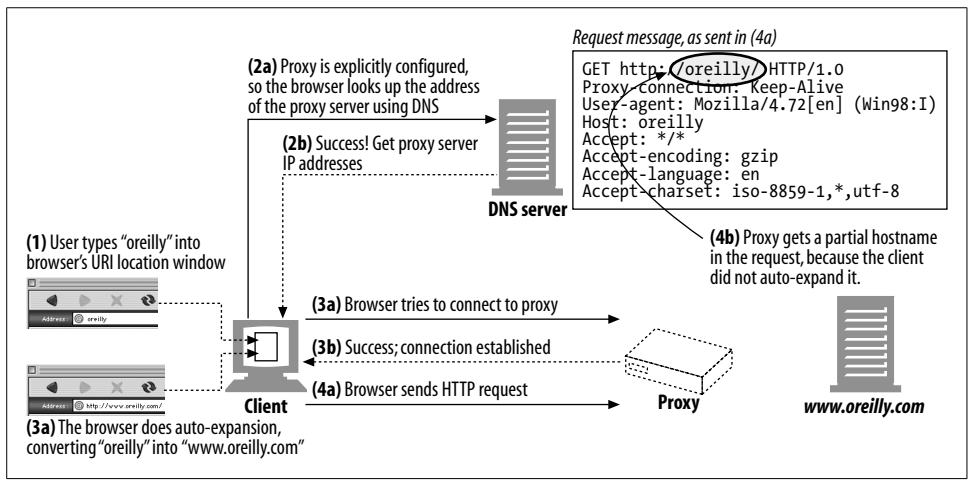

Figure 6-17. Browser does not auto-expand partial hostnames when there is an explicit proxy

For this reason, some proxies attempt to mimic as much as possible of the browser's convenience services as they can, including "www...com" auto-expansion and addition of local domain suffixes.*

URI Resolution with an Intercepting Proxy

Hostname resolution is a little different with an invisible intercepting proxy, because as far as the client is concerned, there is no proxy! The behavior proceeds much like the server case, with the browser auto-expanding hostnames until DNS success. But a significant difference occurs when the connection to the server is made, as Figure 6-18 illustrates.

* But, for widely shared proxies, it may be impossible to know the proper domain suffix for individual users.

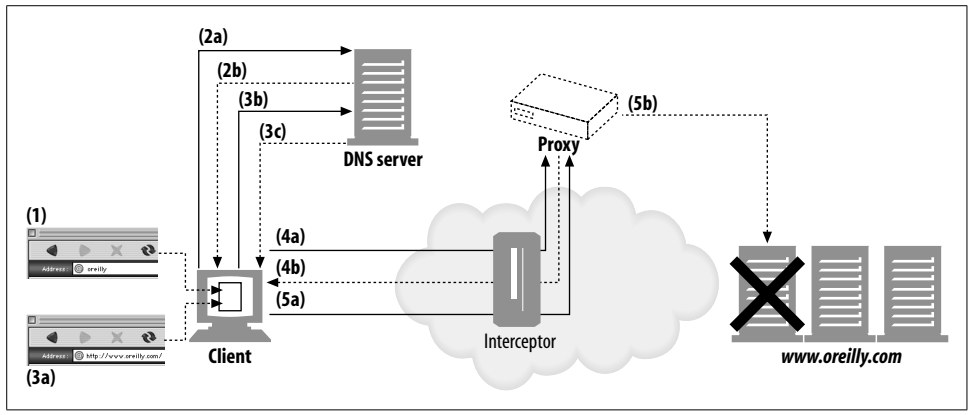

Figure 6-18. Browser doesn't detect dead server IP addresses when using intercepting proxies

Figure 6-18 demonstrates the following transaction:

- In Step 1, the user types "oreilly" into the browser's URI location window.
- In Step 2a, the browser looks up the host "oreilly" via DNS, but the DNS server fails and responds that the host is unknown, as shown in Step 2b.
- In Step 3a, the browser does auto-expansion, converting "oreilly" into "www.oreilly.com." In Step 3b, the browser looks up the host "www.oreilly.com" via DNS. This time, as shown in Step 3c, the DNS server is successful and returns IP addresses back to the browser.
- In Step 4a, the client already has successfully resolved the hostname and has a list of IP addresses. Normally, the client tries to connect to each IP address until it succeeds, because some of the IP addresses may be dead. But with an intercepting proxy, the first connection attempt is terminated by the proxy server, not the origin server. The client believes it is successfully talking to the web server, but the web server might not even be alive.
- When the proxy finally is ready to interact with the real origin server (Step 5b), the proxy may find that the IP address actually points to a down server. To provide the same level of fault tolerance provided by the browser, the proxy needs to try other IP addresses, either by reresolving the hostname in the Host header or by doing a reverse DNS lookup on the IP address. It is important that both intercepting and explicit proxy implementations support fault tolerance on DNS resolution to dead servers, because when browsers are configured to use an explicit proxy, they rely on the proxy for fault tolerance.

Tracing Messages

Today, it's not uncommon for web requests to go through a chain of two or more proxies on their way from the client to the server (Figure 6-19). For example, many

corporations use caching proxy servers to access the Internet, for security and cost savings, and many large ISPs use proxy caches to improve performance and implement features. A significant percentage of web requests today go through proxies. At the same time, it's becoming increasingly popular to replicate content on banks of surrogate caches scattered around the globe, for performance reasons.

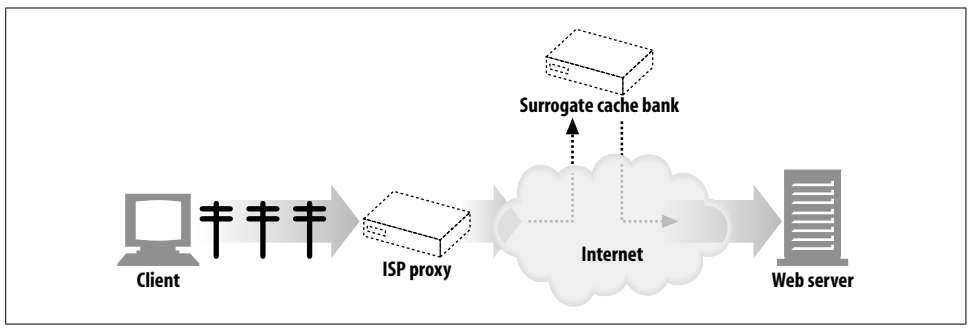

Figure 6-19. Access proxies and CDN proxies create two-level proxy hierarchies

Proxies are developed by different vendors. They have different features and bugs and are administrated by various organizations.

As proxies become more prevalent, you need to be able to trace the flow of messages across proxies and to detect any problems, just as it is important to trace the flow of IP packets across different switches and routers.

The Via Header

The Via header field lists information about each intermediate node (proxy or gateway) through which a message passes. Each time a message goes through another node, the intermediate node must be added to the end of the Via list.

The following Via string tells us that the message traveled through two proxies. It indicates that the first proxy implemented the HTTP/1.1 protocol and was called *proxy-62.irenes-isp.net*, and that he second proxy implemented HTTP/1.0 and was called *cache.joes-hardware.com*:

```
Via: 1.1 proxy-62.irenes-isp.net, 1.0 cache.joes-hardware.com
```

The Via header field is used to track the forwarding of messages, diagnose message loops, and identify the protocol capabilities of all senders along the request/response chain (Figure 6-20).

Proxies also can use Via headers to detect routing loops in the network. A proxy should insert a unique string associated with itself in the Via header before sending out a request and should check for the presence of this string in incoming requests to detect routing loops in the network.

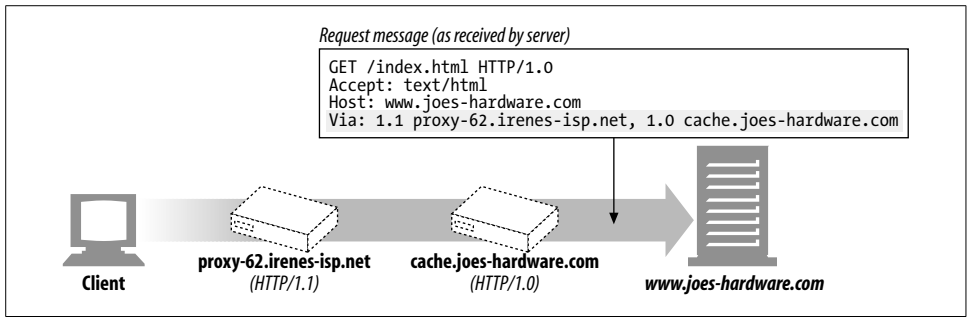

Figure 6-20. Via header example

Via syntax

The Via header field contains a comma-separated list of *waypoints*. Each waypoint represents an individual proxy server or gateway hop and contains information about the protocol and address of that intermediate node. Here is an example of a Via header with two waypoints:

```
Via = 1.1 cache.joes-hardware.com, 1.1 proxy.irenes-isp.net
```

The formal syntax for a Via header is shown here:

```
Via               = "Via" ":" 1#( waypoint )
waypoint          = ( received-protocol received-by [ comment ] )
received-protocol = [ protocol-name "/" ] protocol-version
received-by       = ( host [ ":" port ] ) | pseudonym
```

Note that each Via waypoint contains up to four components: an optional protocol name (defaults to HTTP), a required protocol version, a required node name, and an optional descriptive comment:

Protocol name
: The protocol received by an intermediary. The protocol name is optional if the protocol is HTTP. Otherwise, the protocol name is prepended to the version, separated by a "/". Non-HTTP protocols can occur when gateways connect HTTP requests for other protocols (HTTPS, FTP, etc.).

Protocol version
: The version of the message received. The format of the version depends on the protocol. For HTTP, the standard version numbers are used ("1.0", "1.1", etc.). The version is included in the Via field, so later applications will know the protocol capabilities of all previous intermediaries.

Node name
: The host and optional port number of the intermediary (if the port isn't included, you can assume the default port for the protocol). In some cases an organization might not want to give out the real hostname, for privacy reasons, in which case it may be replaced by a pseudonym.

Node comment

An optional comment that further describes the intermediary node. It's common to include vendor and version information here, and some proxy servers also use the comment field to include diagnostic information about the events that occurred on that device.*

Via request and response paths

Both request and response messages pass through proxies, so both request and response messages have Via headers.

Because requests and responses usually travel over the same TCP connection, response messages travel backward across the same path as the requests. If a request message goes through proxies A, B, and C, the corresponding response message travels through proxies C, B, then A. So, the Via header for responses is almost always the reverse of the Via header for responses (Figure 6-21).

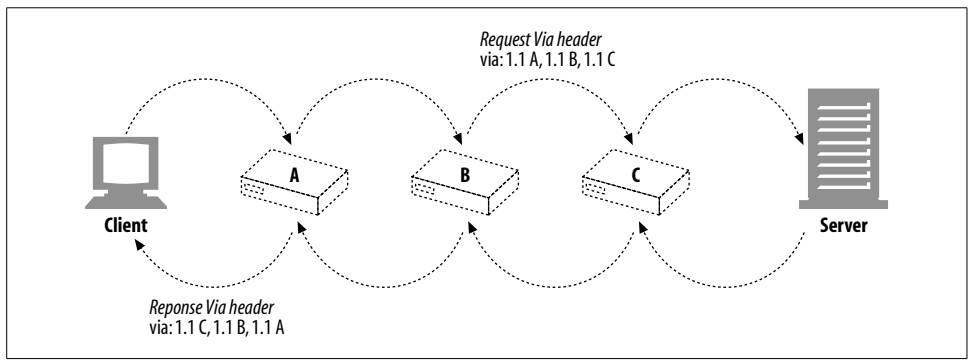

Figure 6-21. The response Via is usually the reverse of the request Via

Via and gateways

Some proxies provide gateway functionality to servers that speak non-HTTP protocols. The Via header records these protocol conversions, so HTTP applications can be aware of protocol capabilities and conversions along the proxy chain. Figure 6-22 shows an HTTP client requesting an FTP URI through an HTTP/FTP gateway.

The client sends an HTTP request for *ftp://http-guide.com/pub/welcome.txt* to the gateway *proxy.irenes-isp.net*. The proxy, acting as a protocol gateway, retrieves the desired object from the FTP server, using the FTP protocol. The proxy then sends the object back to the client in an HTTP response, with this Via header field:

```
Via: FTP/1.0 proxy.irenes-isp.net (Traffic-Server/5.0.1-17882 [cMs f ])
```

* For example, caching proxy servers may include hit/miss information.

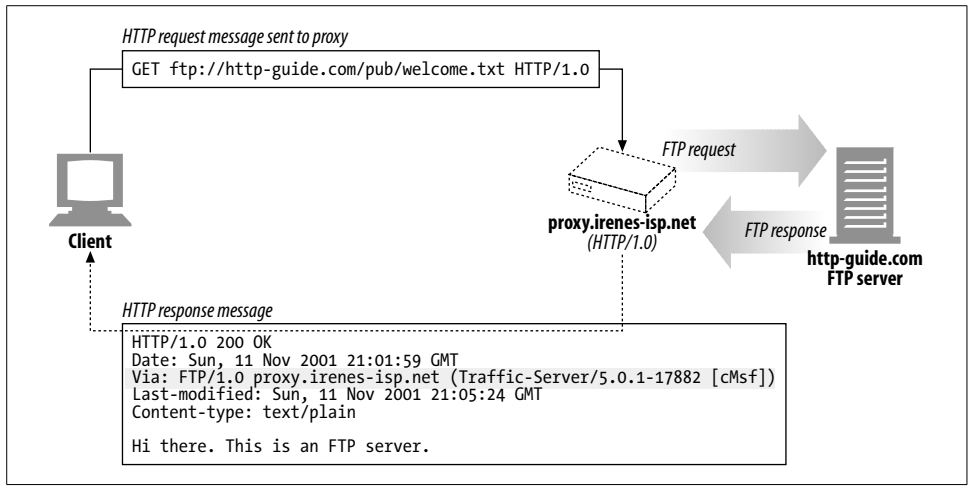

Figure 6-22. HTTP/FTP gateway generates Via headers, logging the received protocol (FTP)

Notice the received protocol is FTP. The optional comment contains the brand and version number of the proxy server and some vendor diagnostic information. You can read all about gateways in Chapter 8.

The Server and Via headers

The Server response header field describes the software used by the origin server. Here are a few examples:

```
Server: Apache/1.3.14 (Unix) PHP/4.0.4
Server: Netscape-Enterprise/4.1
Server: Microsoft-IIS/5.0
```

If a response message is being forwarded through a proxy, make sure the proxy does not modify the Server header. The Server header is meant for the origin server. Instead, the proxy should add a Via entry.

Privacy and security implications of Via

There are some cases when we want don't want exact hostnames in the Via string. In general, unless this behavior is explicitly enabled, when a proxy server is part of a network firewall it should not forward the names and ports of hosts behind the firewall, because knowledge of network architecture behind a firewall might be of use to a malicious party.*

* Malicious people can use the names of computers and version numbers to learn about the network architecture behind a security perimeter. This information might be helpful in security attacks. In addition, the names of computers might be clues to private projects within an organization.

If Via node-name forwarding is not enabled, proxies that are part of a security perimeter should replace the hostname with an appropriate pseudonym for that host. Generally, though, proxies should try to retain a Via waypoint entry for each proxy server, even if the real name is obscured.

For organizations that have very strong privacy requirements for obscuring the design and topology of internal network architectures, a proxy may combine an ordered sequence of Via waypoint entries (with identical received-protocol values) into a single, joined entry. For example:

```
Via: 1.0 foo, 1.1 devirus.company.com, 1.1 access-logger.company.com
```

could be collapsed to:

```
Via: 1.0 foo, 1.1 concealed-stuff
```

Don't combine multiple entries unless they all are under the same organizational control and the hosts already have been replaced by pseudonyms. Also, don't combine entries that have different received-protocol values.

The TRACE Method

Proxy servers can change messages as the messages are forwarded. Headers are added, modified, and removed, and bodies can be converted to different formats. As proxies become more sophisticated, and more vendors deploy proxy products, interoperability problems increase. To easily diagnose proxy networks, we need a way to conveniently watch how messages change as they are forwarded, hop by hop, through the HTTP proxy network.

HTTP/1.1's TRACE method lets you trace a request message through a chain of proxies, observing what proxies the message passes through and how each proxy modifies the request message. TRACE is very useful for debugging proxy flows.*

When the TRACE request reaches the destination server,† the entire request message is reflected back to the sender, bundled up in the body of an HTTP response (see Figure 6-23). When the TRACE response arrives, the client can examine the exact message the server received and the list of proxies through which it passed (in the Via header). The TRACE response has Content-Type message/http and a 200 OK status.

Max-Forwards

Normally, TRACE messages travel all the way to the destination server, regardless of the number of intervening proxies. You can use the Max-Forwards header to limit

* Unfortunately, it isn't widely implemented yet.

† The final recipient is either the origin server or the first proxy or gateway to receive a Max-Forwards value of zero (0) in the request.

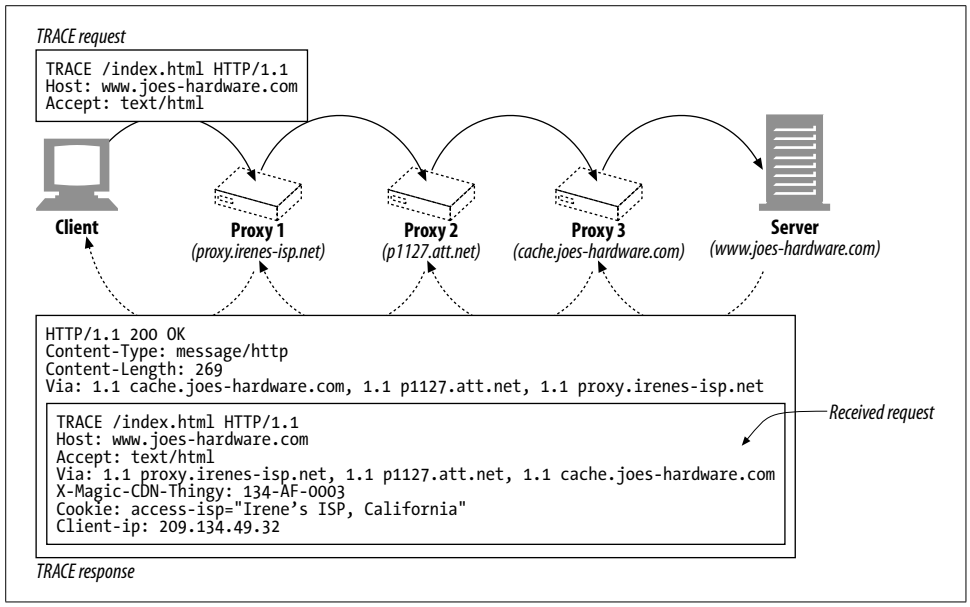

Figure 6-23. TRACE response reflects back the received request message

the number of proxy hops for TRACE and OPTIONS requests, which is useful for testing a chain of proxies forwarding messages in an infinite loop or for checking the effects of particular proxy servers in the middle of a chain. Max-Forwards also limits the forwarding of OPTIONS messages (see "Proxy Interoperation").

The Max-Forwards request header field contains a single integer indicating the remaining number of times this request message may be forwarded (Figure 6-24). If the Max-Forwards value is zero (Max-Forwards: 0), the receiver must reflect the TRACE message back toward the client without forwarding it further, even if the receiver is not the origin server.

If the received Max-Forwards value is greater than zero, the forwarded message must contain an updated Max-Forwards field with a value decremented by one. All proxies and gateways should support Max-Forwards. You can use Max-Forwards to view the request at any hop in a proxy chain.

Proxy Authentication

Proxies can serve as access-control devices. HTTP defines a mechanism called *proxy authentication* that blocks requests for content until the user provides valid access-permission credentials to the proxy:

- When a request for restricted content arrives at a proxy server, the proxy server can return a 407 Proxy Authorization Required status code demanding access

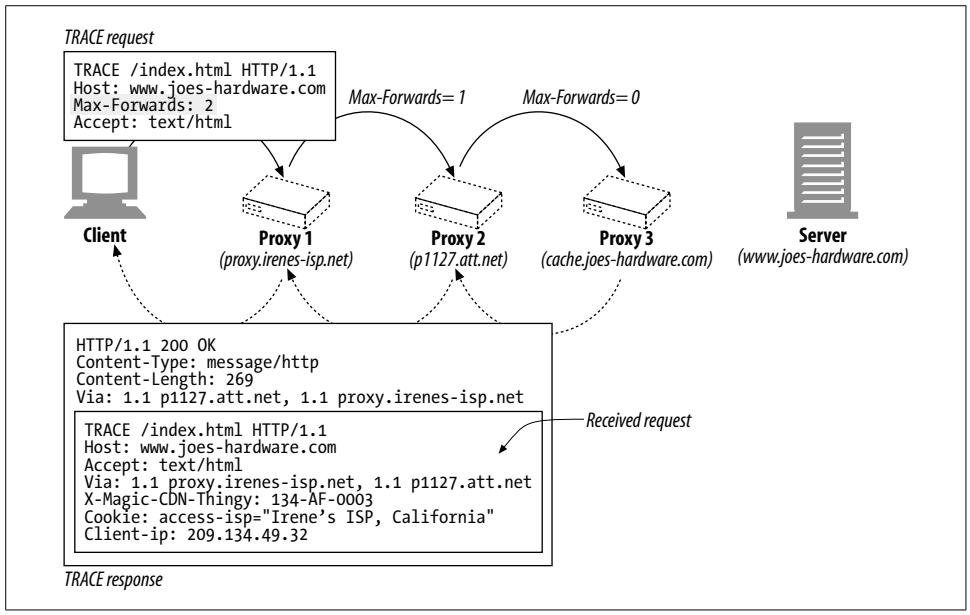

Figure 6-24. You can limit the forwarding hop count with the Max-Forwards header field

credentials, accompanied by a Proxy-Authenticate header field that describes how to provide those credentials (Figure 6-25b).

- When the client receives the 407 response, it attempts to gather the required credentials, either from a local database or by prompting the user.
- Once the credentials are obtained, the client resends the request, providing the required credentials in a Proxy-Authorization header field.
- If the credentials are valid, the proxy passes the original request along the chain (Figure 6-25c); otherwise, another 407 reply is sent.

Proxy authentication generally does not work well when there are multiple proxies in a chain, each participating in authentication. People have proposed enhancements to HTTP to associate authentication credentials with particular waypoints in a proxy chain, but those enhancements have not been widely implemented.

Be sure to read Chapter 12 for a detailed explanation of HTTP's authentication mechanisms.

Proxy Interoperation

Clients, servers, and proxies are built by multiple vendors, to different versions of the HTTP specification. They support various features and have different bugs. Proxy servers need to intermediate between client-side and server-side devices, which may implement different protocols and have troublesome quirks.

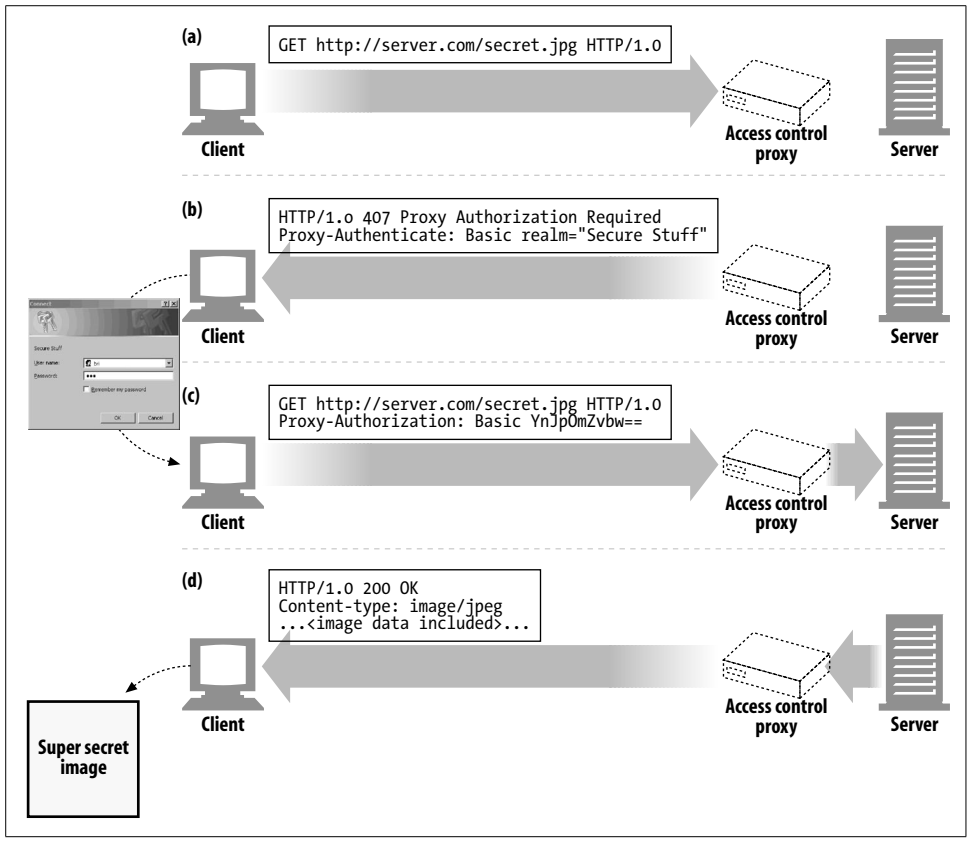

Figure 6-25. Proxies can implement authentication to control access to content

Handling Unsupported Headers and Methods

The proxy server may not understand all the header fields that pass through it. Some headers may be newer than the proxy itself; others may be customized header fields unique to a particular application. Proxies must forward unrecognized header fields and must maintain the relative order of header fields with the same name.* Similarly, if a proxy is unfamiliar with a method, it should try to forward the message to the next hop, if possible.

Proxies that cannot tunnel unsupported methods may not be viable in most networks today, because Hotmail access through Microsoft Outlook makes extensive use of HTTP extension methods.

* Multiple message header fields with the same field name may be present in a message, but if they are, they must be able to be equivalently combined into a comma-separated list. The order in which header fields with the same field name are received is therefore significant to the interpretation of the combined field value, so a proxy can't change the relative order of these same-named field values when it forwards a message.

OPTIONS: Discovering Optional Feature Support

The HTTP OPTIONS method lets a client (or proxy) discover the supported functionality (for example, supported methods) of a web server or of a particular resource on a web server (Figure 6-26). Clients can use OPTIONS to determine a server's capabilities before interacting with the server, making it easier to interoperate with proxies and servers of different feature levels.

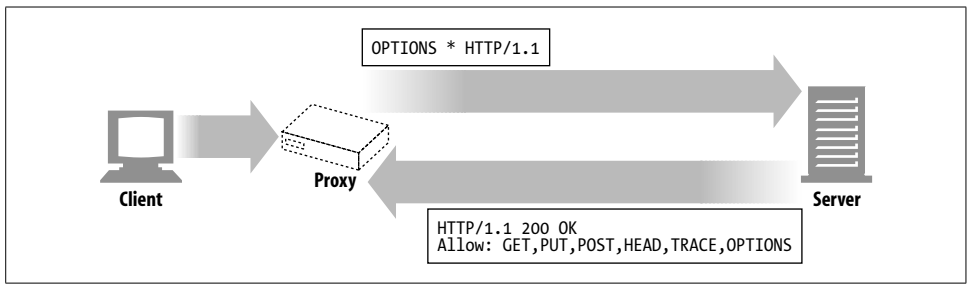

Figure 6-26. Using OPTIONS to find a server's supported methods

If the URI of the OPTIONS request is an asterisk (*), the request pertains to the entire server's supported functionality. For example:

```
OPTIONS * HTTP/1.1
```

If the URI is a real resource, the OPTIONS request inquires about the features available to that particular resource:

```
OPTIONS http://www.joes-hardware.com/index.html HTTP/1.1
```

On success, the OPTIONS method returns a 200 OK response that includes various header fields that describe optional features that are supported on the server or available to the resource. The only header field that HTTP/1.1 specifies in the response is the Allow header, which describes what methods are supported by the server (or particular resource on the server).* OPTIONS allows an optional response body with more information, but this is undefined.

The Allow Header

The Allow entity header field lists the set of methods supported by the resource identified by the request URI, or the entire server if the request URI is *. For example:

```
Allow: GET, HEAD, PUT
```

The Allow header can be used as a request header to recommend the methods to be supported by the new resource. The server is not required to support these methods

* Not all resources support every method. For example, a CGI script query may not support a file PUT, and a static HTML file wouldn't accept a POST method.

and should include an Allow header in the matching response, listing the actual supported methods.

A proxy can't modify the Allow header field even if it does not understand all the methods specified, because the client might have other paths to talk to the origin server.

For More Information

For more information, refer to:

http://www.w3.org/Protocols/rfc2616/rfc2616.txt
: RFC 2616, "Hypertext Transfer Protocol," by R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Mastinter, P. Leach, and T. Berners-Lee.

http://search.ietf.org/rfc/rfc3040.txt
: RFC 3040, "Internet Web Replication and Caching Taxonomy."

Web Proxy Servers
: Ari Luotonen, Prentice Hall Computer Books.

http://search.ietf.org/rfc/rfc3143.txt
: RFC 3143, "Known HTTP Proxy/Caching Problems."

Web Caching
: Duane Wessels, O'Reilly & Associates, Inc.