Chapter 8
Proxy Deployment

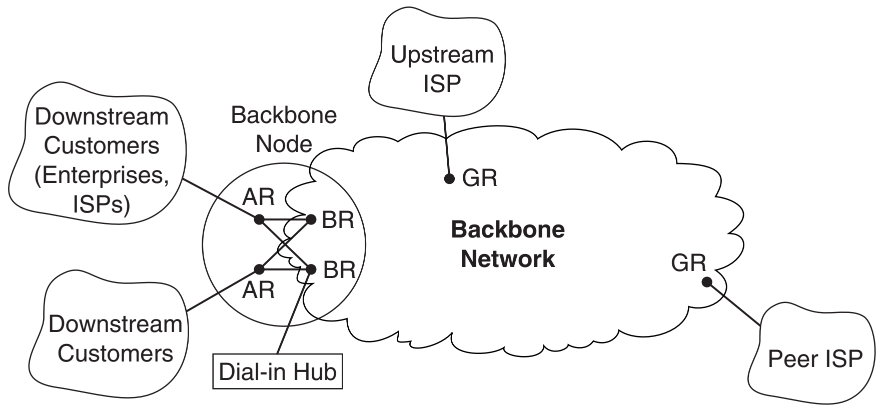
Typical proxy users include individual departments such as academic units of a university, entire enterprises, and Internet Service Providers (ISPs). In this chapter, we discuss various alternatives for deploying forward proxy caching in enterprise and ISP networks. The general problem considered here is a mechanism to deliver a client request to the proxy instead of the origin server specified in the requested URL.

8.1 Overview of Internet Connectivity Architectures

Before discussing alternatives for proxy deployment, we need to understand at a high level how an enterprise obtains its Internet connectivity. Consider Figure 8.1, which depicts at a high level a typical ISP architecture. A large ISP has a backbone network consisting of backbone nodes. Each node has some *access routers* (*ARs*) that are the backbone entry points for enterprise customers of the ISP (such as other ISPs) and *backbone routers* (*BRs*) that provide connectivity between backbone nodes. Access routers are connected to their local backbone routers via LANs. Backbone routers in different nodes are connected via *wide area network* (*WAN*) lines, creating the *backbone topology*. In addition, some backbone nodes contain *gateway routers* (*GRs*) that link the backbone to the outside world. On the ISP side, GRs are connected to their local BRs via a LAN.

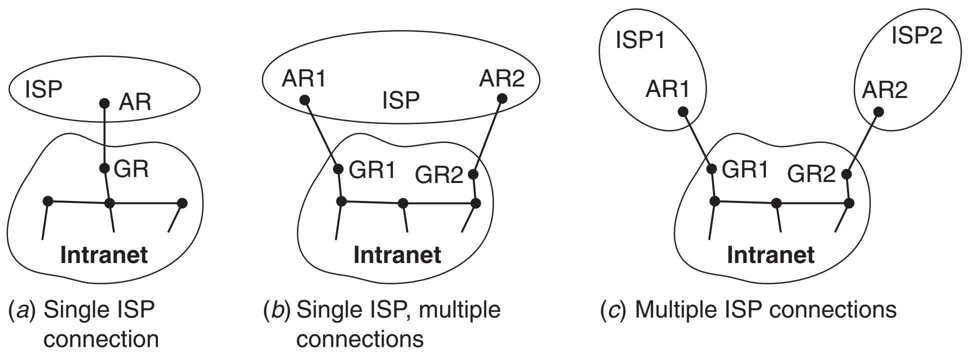
Residential dial-in customers connect to modem banks located in the ISP's *dial-in points of presence (dial-in POPs)*, otherwise known as *dial-in hubs*. These hubs are sometimes connected to backbone routers, bypassing access routers. Only the largest enterprise customers connect directly at access routers of a backbone node. Others connect at the ISP's *broadband points of presence (broadband POPs)*, which aggregate several enterprises into one connection to the backbone access router. These POPs are omitted in Figure 8.1 for simplicity.

Figure 8.2 shows typical ways in which enterprise networks connect to the Internet. An enterprise may buy its Internet connectivity from one ISP (Figures 8.2a and 8.2b) or multiple ISPs (Figure 8.2c). If using a single ISP, the enterprise network may be



AR = Access Router; BR = Backbone Router; GR = Gateway Router

Figure 8.1 A high-level ISP architecture


ISP
AR
GR
Intranet
(a) Single ISP connection
AR1
ISP
AR2
GR1
GR2
Intranet
(b) Single ISP, multiple connections
ISP1
ISP2
AR1
AR2
GR1
GR2
Intranet
(c) Multiple ISP connections

GR = Corporate Gateway Router; AR = ISP Access Router

Figure 8.2 Typical examples of enterprise network connections to the Internet

connected to its ISP at a single access router or multiple access routers (Figures 8.2a and 8.2b). The main question for the purpose of our discussion is whether there is a single element in the enterprise network (a router, switch, or link) through which all Internet traffic flows. We refer to such an element as a *focal point*. Focal points may be undesirable from the fault-tolerance perspective, since a failure at a focal point breaks the Internet connectivity of the enterprise. At the same time, they create convenient points to deploy proxies, as we show later in this chapter.

Note that many enterprises with multiple ISP connections use only one connection for normal operation, with the rest serving as backups in case the main connection fails. These enterprise networks have a focal point despite multiple Internet connections. Other enterprise networks alternate between their Internet connections to balance traffic across all available links. Doing so eliminates a focal point for Internet traffic.

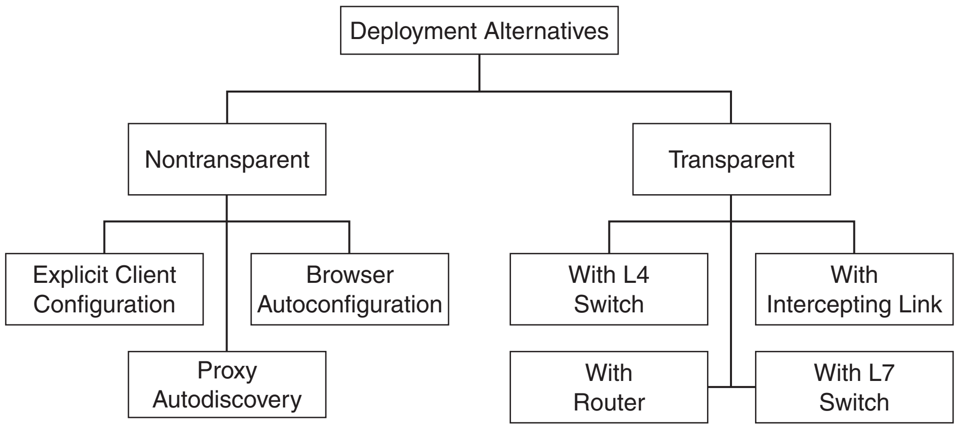


Figure 8.3 Deployment alternatives for a forward proxy

An ISP can deploy a proxy at a POP, at a backbone node to serve requests entering the backbone at this node (a *backbone entry point proxy*), or at a backbone node to serve any requests regardless of where they enter the backbone (a *backbone core proxy*). The main tradeoff here is between the amount of benefit from a cache hit versus the hit rate. A proxy at a POP is the closest the ISP can get to the customers. Therefore, a cache hit at a POP proxy can save the ISP the most bandwidth; such a hit can also reduce latency the most, compared to other ISP locations. On the other hand, a POP proxy only serves clients that connect at this POP; a limited client population can reduce the hit rate [Duska et al. 1997; Gribble and Brewer 1997]. Perhaps even more importantly, POP proxy deployment can get quite expensive, since a large ISP can have hundreds of POPs. A backbone entry point proxy can serve all clients from all POPs and enterprise networks that connect to the backbone at the corresponding backbone node, although this proxy is farther away from customers. A core backbone proxy can potentially serve all ISP clients but is the farthest away from them. Interproxy cooperation (see Chapter 9) can alleviate the tradeoff. We consider ISP proxy deployment in the form of a backbone entry point proxy. The same issues arise also in other locations, so our discussion applies to those locations as well.

The location of a proxy is less significant to enterprise networks because, with the exception of a few global corporations, enterprise networks do not have their own backbones.[1] Therefore, an enterprise or an enterprise department proxy can serve all clients in the enterprise or the department, regardless of the proxy location.

A fundamental issue in proxy deployment is how to deliver requests to the proxy. Figure 8.3 summarizes various methods for request delivery, which we consider in the

[1] Even global corporations often rely on ISPs for intracompany communication over a wide area, by means of so-called virtual private networks. For the purpose of our discussion, such a corporation can be assumed to have a collection of individual campus-size exterprise networks.

rest of this chapter. The main choice here is between nontrasparent and transparent proxy deployment.

8.2 Nontransparent Proxy Deployment

Nontransparent proxy deployment refers to a deployment in which client software is made aware in some way of the proxy's existence. Then clients send their Web requests to the proxy regardless of the origin servers specified in the URLs. In addition to document retrieval, these clients also delegate DNS resolutions to a proxy: a browser sends to its proxy the entire requested URL, including the host name part, and the proxy then, if needed, resolves the host name into an IP address and retrieves the object from that address. This is different from the actions of clients accessing the Web directly. Such a client itself obtains the origin server's IP address and sends the HTTP request containing just the path portion of the URL (see Section 4.1) to that address.

Proxies whose deployment is nontransparent to clients are often called *explicit proxies*. Let us now consider different ways of making clients aware of the explicit proxy.

8.2.1 Explicit Client Configuration

With explicit client configuration, clients (browsers or other proxies) are explicitly configured to send requests to a proxy instead of the origin servers.

There are several advantages to explicit client configuration. One advantage is its flexibility. Users can easily bypass the proxy if they choose to, by simply changing their browser configuration. Also, the proxy can be deployed at any convenient place in the network: since clients explicitly address their requests to the proxy, the requests will find their way to the proxy regardless of where it is located.

The main disadvantage of this deployment method is that clients must be explicitly configured. This causes administrative complications, especially for an ISP. In fact, the ISP often does not even know the clients at all, as, for example, when they belong to an enterprise customer or a downstream ISP. Switching the proxy off for maintenance or other reasons also becomes a problem. Still, explicit client configuration is a simple and valid alternative for proxy deployment in enterprise networks, and ISPs sometimes use it by preconfiguring browsers of their own residential customers.

8.2.2 Browser Autoconfiguration

Modern browsers implement proxy *autoconfiguration* capability [Netscape 1996]. Instead of being configured directly to use a certain proxy, a browser is configured to download a special URL every time it is started, called an *autoconfiguration file*, which identifies the proxy the browser should use. This level of indirection allows the administrator to maintain the proxy configuration information in a centralized manner. For instance, when a browser should be assigned to a different proxy, the administrator need only change the autoconfiguration file that is returned to the browser. The browser itself need not be reconfigured, though it must be restarted for the change to take effect.

In addition to simplifying administration, an autoconfiguration file can specify backup proxies and also different proxies for different Web sites and URL types. For example, the file can specify that the browser should use proxy P1 for all URLs matching the "*.gif" pattern, proxy P2 for those URLs matching the "*.html" pattern, and go directly to the Web site for all URLs that match the "*/cgi-bin/*" pattern, where the asterisk denotes an arbitrary string.

The autoconfiguration facility simplifies administration of explicit proxies but does not change the fact that every browser must be explicitly configured. The next step in automating browser configuration is a mechanism that allows the browser to *discover* automatically the autoconfiguration file URL.

8.2.3 Proxy Auto-Discovery

A mechanism for discovering an explicit proxy, called *Web Proxy Auto-Discovery (WPAD)* [Inktomi 1998], utilizes various protocols that hosts may use to discover resources on the network, such as printers, name servers, and so forth. For example, when using DNS, all browsers using this method essentially agree on a well-known relative domain name of the Web server that must provide the autoconfiguration file—such as wpad. At start-up, the browser queries its DNS server for this name (after expanding it into a fully qualified name as it normally would, like wpad.research.att.com). The DNS server returns the IP address of the Web server where the autoconfiguration file resides. The browser then constructs the autoconfiguration URL by adding a well-known path portion to the discovered IP address of the server, such as wpad.dat. If, for example, the IP address of the autoconfiguration server is 127.200.27.11, then the autoconfiguration URL would be http://127.200.27.11/wpad.dat. Once this URL is constructed, the browser can download the autoconfiguration file and start using the specified proxies.

With autodiscovery, the administrator need only maintain a Web server with the well-known domain name and an autoconfiguration file on that server with the well-known URL. The browsers could be used "out of the box," with no configuration, and still find proxies to use.

The proxy auto-discovery mechanism is in the proposal state. Moreover, it works only when the browser and the proxy are in the same administrative domain, such as an enterprise network. In other cases, *transparent deployment*, achieved with *interception proxies*, became a popular alternative.

8.3 Transparent Proxy Deployment

Transparent deployment of a proxy relies on some network element (a switch or a router) to intercept all traffic from Web clients to Web servers and divert it to a proxy server instead of its actual destination. For this reason, proxies deployed in

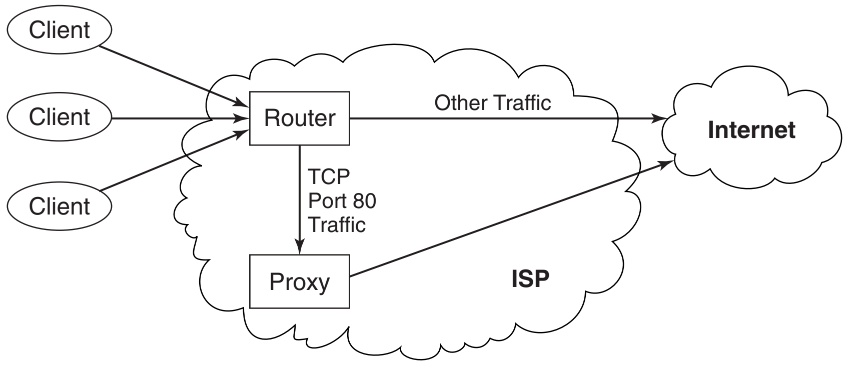


Figure 8.4 Transparent proxy deployment in an ISP

this manner are called interception proxies.[2] The network elements that intercept and divert packets to interception proxies are called *intercepting elements* or simply *intercepters*.

The network element identifies packets to be intercepted by examining IP headers of all incoming packets. Packets transported by TCP and adressed to port 80 are intercepted. Since HTTP communication occurs over TCP transport and an overwhelming majority of Web servers use the default port 80 [Woodruff et al. 1996; Douglis et al. 1997a], this method captures most of the HTTP request traffic flowing through the network element.

Figure 8.4 shows an example of interception proxy deployment in an ISP network. In this example, an ISP router serves as the intercepter and sends all TCP port 80 packets to a proxy instead of their intended destinations. Deployment in an enterprise network is similar: a gateway router connecting the enterprise network to its ISP can divert Web request traffic to an enterprise proxy.

When returning an object to the client, the interception proxy *impersonates* the originally intended destination of the request (that is, the origin server of the requested object), by putting into the IP headers of the response packets the IP address of the original destination rather than its own IP address. The client therefore never realizes that it communicates with a proxy and not the origin server. Such forced traffic interception and impersonation of the origin server is often referred to as *connection hijacking*, or *connection interception*. Connection interception has been expanded to include NNTP (the Network News Transfer Protocol, which carries newsgroups) and FTP traffic, in addition to HTTP communication.

Because clients are unaware of the interception proxy, they act as if they accessed the origin servers directly. In particular, unlike with explicit proxies, clients themselves

[2] Interception proxies are also referred to as *transparent proxies*. However, the HTTP 1.1 description uses the term *transparency* to mean semantic transparency (that is, referring to a proxy whose presence does not affect Web behavior semantically) rather than deployment transparency.

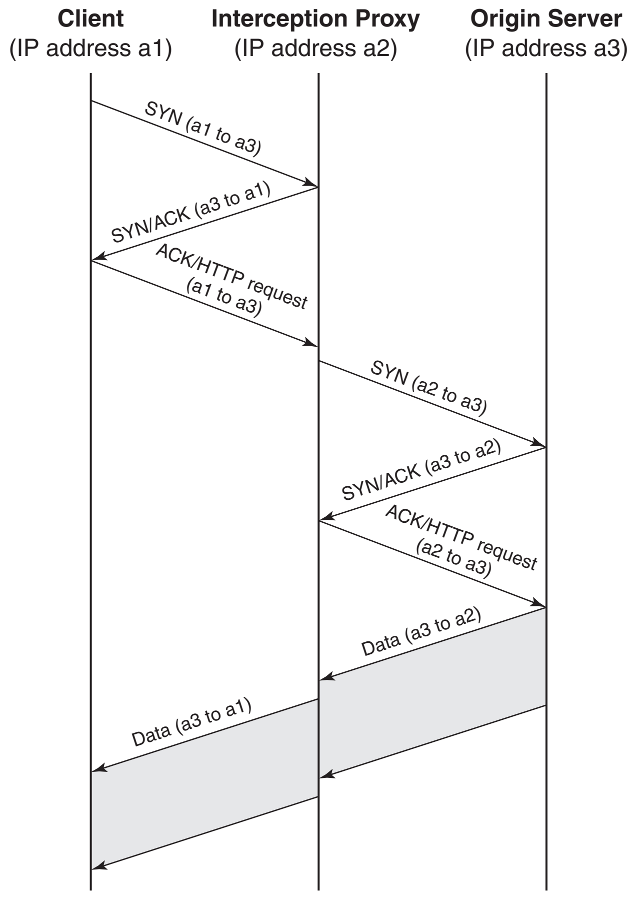


Figure 8.5 Packet flow in interception caching (Information in parentheses indicate source and destination IP addresses of the packets.)

resolve origin servers' domain names into their IP addresses and send HTTP requests containing only the path portion of URLs. However, when the interception proxy needs the domain names of origin servers (as described in Section 8.3.7), the proxy can extract this name from the `host` header of client requests.

Figure 8.5 illustrates packet flows for an HTTP interaction with an interception proxy.[3] The proxy impersonates the origin server and establishes the TCP connection with the client (packets SYN and SYN/ACK). The client then sends the packet containing the HTTP request. Assuming that the proxy does not have the requested object, it sends its own request to the origin server, using its own IP address. After the object starts to arrive, the proxy sends it back to the client.

[3] Although most TCP implementations send data only after fully completing the handshake, we assume for simplicity that the HTTP request is sent together with the final acknowledgment of the handshake. This assumption does not affect in any way the issues discussed here.

Transparent proxy deployment offers several major advantages to an ISP. First and foremost, the administrative obstacle of configuring clients is removed. Managing the proxy becomes a purely internal affair. The ISP can bring it down at will by simply switching off connection interception at the appropriate switches or routers (of course, the interception will stop only for new connections, and the proxy will be brought down only after it completes serving existing connections). The proxy DNS name and IP address are not exposed to clients and can be easily changed. Another advantage is that interception proxies behave better than explicit ones under failures and overload. The intercepting element constantly monitors the health of the proxy (see Section 8.3.2 for details on how monitoring is done). Once the intercepting element detects a failure, it simply stops diverting the traffic to the proxy; at this point, clients retain their Internet connectivity, perhaps at a diminished performance. Network elements, based on hardware and firmware, are generally more reliable pieces of equipment than software-based proxies. Thus, the ability to quickly shortcut a failed proxy improves the overall system behavior under failures.[4]

At the same time, transparent deployment has several pitfalls. Some of these are serious enough for this deployment method to remain controversial despite its wide use. The main controversy surrounding transparent proxy deployment and interception proxies stems precisely from the fact that clients are unaware of the proxy existence. Clients address their datagrams to origin servers and assume that the responses come from the origin servers. Thus, interception proxies violate the *end-to-end principle* of the Internet, which says that an application function "can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system" [Saltzer et al. 1984]. HTTP 1.1 improves the situation slightly by requiring proxies to identify themselves to origin servers and allowing clients to find out whether any proxies stand between them and particular origin servers (see Section 5.9). While not changing the fact that hosts to which clients address packets may be different from the hosts with which clients end up communicating, these HTTP features at least allow one to diagnose a problem that may arise from this fact.

The most important limitation of interception proxies is a direct corollary of the violation of the end-to-end principle. We call this limitation a *multipath problem*.

8.3.1 Multipath Problem

The main limitation of interception proxies is that they can only work properly if *all* packets from a given client to a given destination flow through the same intercepting element. This limits the kinds of clients for whom interception proxies can be used and

[4] With explicit proxies, clients can also learn about failures of their proxies and stop using them. However, because clients and explicit proxies are more loosely coupled (and, in particular, not always connected by a fast LAN as the interception proxy and its intercepting router or switch are), detecting proxy failures and recoveries is slower and less reliable in this case.

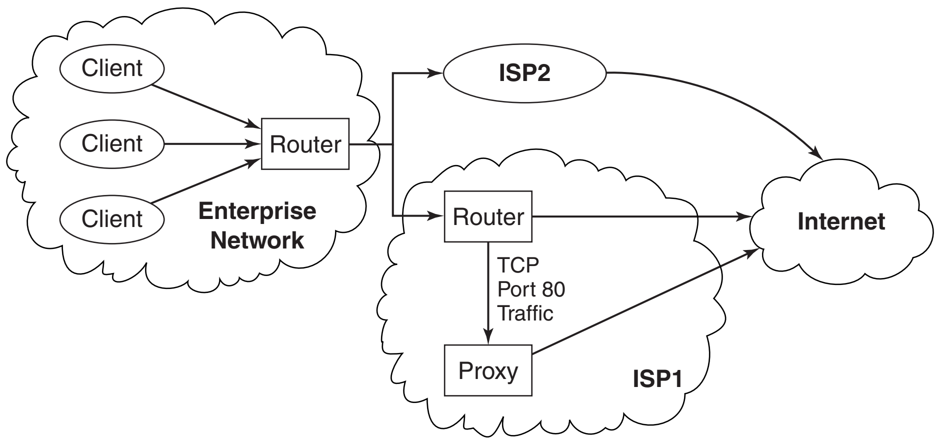


Figure 8.6 An incorrect use of the interception proxy

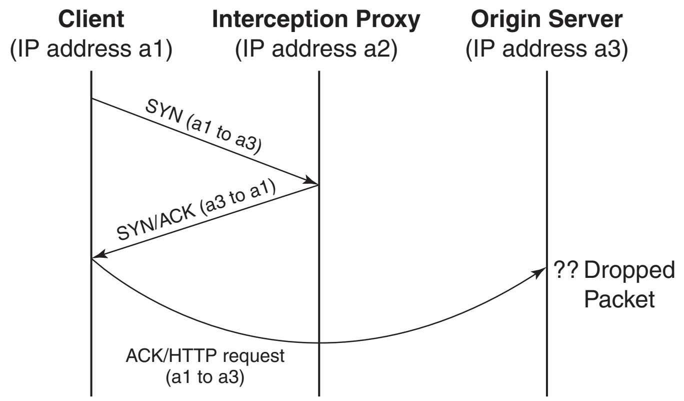

Client (IP address a1)
Interception Proxy (IP address a2)
Origin Server (IP address a3)
SYN (a1 to a3)
SYN/ACK (a3 to a1)
?? Dropped Packet
ACK/HTTP request (a1 to a3)

Figure 8.7 A packet flow for a broken TCP connection

the number of places in the network where the proxies can be deployed. Figure 8.6 illustrates a network setup that may cause a problem for an ISP proxy. It shows a client that buys its Internet connectivity from two different ISPs, as many enterprise customers do.

Consider the packet flow shown in Figure 8.7. The client starts an HTTP interaction by sending the SYN packet to the origin server. This packet is sent using ISP1 and is intercepted by the proxy, which responds with SYN/ACK, impersonating the origin server. The client now sends the HTTP request over the established connection. The request chooses ISP2 and is delivered directly to the origin server. Because the latter has not established a connection with this client, it discards this packet. The effect is a broken HTTP interaction; the client perceives it as lost connectivity to the Internet. Thus, interception caching may disrupt client connectivity even though the entire network functions properly. One may argue that the routing instability illustrated in this example occurs infrequently. However, most ISPs will find it unacceptable to introduce even a few percent additional connection failures.

At an ISP, interception proxy deployment is typically safe when it is used for customers who have this ISP as their only provider of Internet connectivity *and* utilize a single primary connection to this ISP. Having backup connections to this or other ISPs is also possible as long as these connections are used only when the primary connection fails (in other words, the primary and backup connections are never used concurrently). The intercepting element for these customers can be installed at any point along the primary connection, up to the point where the packet flow can first be divided among multiple paths.

A similar problem occurs with interception proxies in enterprise networks when they have multiple connections to the Internet, as shown in Figure 8.2b and 8.2c. It follows from this discussion that an enterprise network can safely deploy interception caching if it uses a single primary gateway to the Internet, or if it cleanly divides all Internet destinations among all its gateways, so that all packets to a given destination are always routed through the same gateway. These gateways could then be used as intercepting elements.

8.3.2 Interception Mechanisms

Let us take a closer look at the mechanisms behind transparent proxy deployment. The packets from the client carry the client IP address as the source IP address and the origin server IP address as the destination IP address. Once intercepted by the intercepting element, these packets must be delivered to the proxy. Delivering packets to an unintended destination can be accomplished at the data link layer (Layer 2 solution) or IP layer (Layer 3 solution).

Layer 2 solution: The intercepting element delivers packets to the proxy by sending them to the proxy's data-link-layer address (MAC address, see Section 1.1). No information in the IP packets themselves is changed. This simple scheme is very efficient but implies a restriction, in that the intercepting element and the proxy must be directly connected by the same data link network. This is because the packets still carry their original destination IP addresses, so if there is a router on their path from the intercepting element to the proxy, the router will faithfully send them back to their original destinations. Nortel's ACEDirector switch is an example of an intercepting element using the L2 diversion mechanism. This mechanism also requires a change to the proxy protocol stack: it must be changed so that packets would be accepted and passed up to the application regardless of their destination IP addresses. Most modern proxies already implement this functionality.

Layer 3 solution: The intercepting element wraps the packet into another IP datagram, the technique called *IP-in-IP encapsulation* [Hanks et al. 1994]. The

IP headers in the outer wrapper have the proxy IP address as the destination address. The packets are therefore delivered to the proxy using normal IP protocols. This option does not impose restrictions on the network connection between the intercepting element and the proxy. They may even be connected via a wide area network and have any number of routers in between. Also, no changes to the proxy protocol stack are required: all changes can concentrate at the application layer. Cisco routers follow this approach.

It may seem that the intercepting element could also deliver packets to the proxy by simply replacing the destination IP address of intercepted packets with that of the proxy. However, a major problem with this approach, which is called *network address translation* (*NAT*) is that the packets lose the information about their original destination. When the proxy needs to obtain the object from its origin server, how will it know where to send the request? It could use the HTTP `host` header of the client request, which contains the identity of the Web site, but that would require a DNS lookup by the proxy, since the site identity in the `host` header is usually represented by its domain name. This lookup would add extra latency to the misses because the client has already performed this DNS lookup before sending the request. As we show in Section 14.1.2, NAT as a mechanism for packet forwarding is more viable in the server surrogate setting, although IP-in-IP encapsulation remains a more general approach even there.

Several other basic questions must be answered when considering interception solutions.

1. How does an intercepting element detect and bypass a failed proxy?
2. How can an intercepting element distribute load among multiple proxies?
3. Can multiple intercepting elements divert packets to the same proxy?
4. Can an intercepting element be efficiently configured to selectively intercept packets based on some configured rules? For instance, can it be told not to intercept packets originating from or destined for specific IP addresses?

We have already touched upon the first question. Network elements, being implemented in hardware and firmware, are usually more reliable than proxy caches and can be deployed in *fail-over pairs*, that is, as a primary unit and a similar backup unit that takes over when the primary fails. The ability to quickly shortcut a failed proxy is one of the main advantages of interception proxies.

The second question arises because network elements typically have substantially higher performance than proxies. A typical backbone link with a throughput of 2.4Gbps can in principle tranfer up to 26,000 Web objects per second (assuming an average object size of 10KBytes). On the other hand, the best proxy in the benchmarking study by Rousskov et al. 2000] had a throughput of 2,400 objects per second.

Therefore, to balance the system, multiple proxies must be able to share the load sent to them by a single network element.

The third question addresses the multipath problem (Section 8.3.1). Normally, ISPs would not use interception proxies when packets from a client to an origin server can take multiple paths. However, if the intercepting elements on all these paths diverted packets to the same proxy, then that proxy would receive all packets from the client regardless of which path they took, and interception caching could still be used. In other words, the answer to this question affects how flexibly one can choose where in the network to intercept the packets.

Consider, for example, Figure 8.1 again. Assume that each AR alternates between the two local BRs when sending packets further in the network. If multiple intercepters could use the same proxy, the ISP could place the intercepting elements on the lines between ARs and BRs. Otherwise, intercepting on these lines would not be an option, since the proxy servicing each line would not be guaranteed to see all packets for any TCP connection, due to the multipath problem. As another example, if a corporate customer connects at both ARs in Figure 8.1, request interception normally would not be done at ARs for clients belonging to this customer because of the multipath problem. However, if both ARs divert intercepted packets to the same proxy, using ARs as intercepting elements becomes an option.

Another motivation for using the same proxy is more efficient cache sharing. Assume that ARs in Figure 8.1 act as intercepting elements. If each AR uses its own proxy, then a client connected to one AR cannot use an object cached at the other AR's proxy without proxy cooperation (see Chapter 9). If both ARs share a proxy, then clients connected to either AR can share the content cached by the proxy directly.

Note that there is no contradiction between the second and third questions. Multiple intercepters can distribute load among the same *set* of proxies in such a way that packets from the same client going to a given destination are always sent to the same proxy.

The fourth question is very important. As we discussed earlier, interception caching is only appropriate for customers who are not affected by the multipath problem. Further, we will see in Section 8.4.1 that interception caching is not appropriate for requests to some Web sites. One needs the flexibility to describe, by specifying filtering rules, which connections should or should not be intercepted. Most if not all intercepting elements provide this functionality but differ in the expressiveness of the rules and the performance overhead. In addition to the intercepter, the proxy could also identify the packets that should not have been diverted to it. The proxy could return these exception packets back to the Internet unchanged.[5] These packets would then be forwarded to their intended destination with their original source IP addresses (that is, the client IP addresses), and the destinations would respond directly to the clients.

[5] If IP-in-IP encapsulation is used to deliver the packets to the proxy, the proxy would obviously remove the wrapper.

The only danger in making the proxy handle exception packets is that the proxy's connection to the intercepter is often the only network connection of the proxy. Thus, all proxy communication, including the returned exception packets, flows through the switch or router that acts as the intercepter. Consequently, the intercepter might faithfully divert the exception packets right back to the proxy (after all, these are TCP port 80 packets from the original source IP)! Fortunately, most intercepters can be configured not to intercept packets that arrive on certain network connections. One only need take special care to configure the intercepter so that it never intercepts packets arriving on the connection from the proxy.

Four types of intercepting elements are used for transparent proxy deployment: Layer 4 switches, routers, Layer 7 switches, and intercepting links. Layer 4 switches, intercepting links, and, in most cases, routers use only the information in IP and TCP packet headers, whereas Layer 7 switches also consider the packets' application data. Let us now consider these types of intercepting elements in turn.

8.3.3 Layer 4 Switch as an Intercepter

An L4 switch is a network switch capable of reading and interpreting the IP and TCP headers of a datagram (hence the name, since it operates at the transport layer in the ISO protocol stack). There are a variety of applications for L4 switches, and different switches target different applications [Conover 1999]. Nortel, Foundry, Cisco, and Radware are examples of companies offering L4 switches for interception Web proxies. These switches look at the IP headers to determine packets sent by TCP, and then at TCP headers to detect packets destined to port 80. The switch then sends these packets to a proxy instead of their original destination.

Figure 8.8 illustrates using an L4 switch in a backbone node.[6] The L4 switch is placed between clients and the access router they connect to, so that all traffic between these clients and the access router flows through the switch. The switch then examines the packets arriving from the clients, intercepts those packets that carry Web traffic, and diverts them to an interception proxy or proxies.

L4 switches can distribute load among several proxies by a variety of methods. One method involves partitioning the IP address space among the proxies and diverting a packet to the proxy responsible for the packet's destination IP address. For example, in the case of three proxies, one possible partitioning would assign IP addresses from 0.0.0.0 to 85.255.255.255 to the first proxy, IP addresses from 86.0.0.0 to 170.255.255.255 to the second proxy, and IP addresses from 171.0.0.0 to 255.255.255.255 to the third proxy. A function that maps all possible values of a data item (in our case IP addresses)

[6] Note that this and other similar figures present a simplified view of the backbone node. In fact, clients do not connect to an access router directly, but rather they first terminate at some sort of an aggregation point, which then connects to the router via a LAN. It is on this LAN that the L4 switch is deployed.

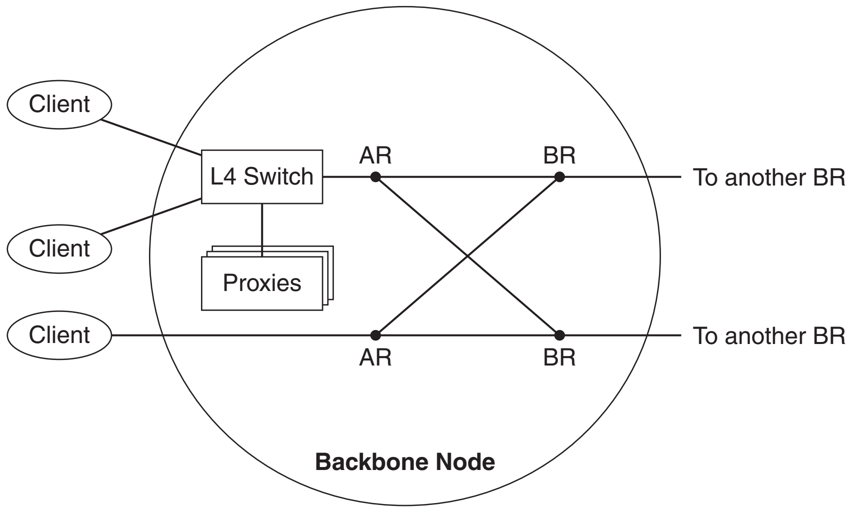


Figure 8.8 An interception proxy deployment with an L4 switch

to a set of numeric values (in our case the proxy number) is called a *hash function*, and applying a hash function is called *hashing*. In reality, switches use more complex hash functions than the one described here.

This load distribution in effect implements hash-based intercache cooperation (see Section 9.3.3) at the switch level: all objects hosted at a given IP address will be cached at one proxy only (no duplicates), and all clients that request these objects will be directed to the proxy responsible for them. Switch-level cooperation is much more efficient than proxy-level because the same request does not have to be processed by multiple proxies. This cooperation breaks down, however, when a Web site is mirrored on multiple servers and the site's DNS server balances load by assigning IP addresses of different mirrors to different requests, because the switch is oblivious to mirroring and treats each replica with a different IP address as if it were a different site. In particular, a request for an object may result in fetching the object from the origin server even if a different replica of the same object (with the same URL) is cached at another proxy connected to the switch.

In addition to distributing load based on the destination IP address, L4 switches can also be configured to distribute load among proxies by hashing the source IP address of packets instead. Other available mechanisms can select an arbitrary proxy for any given connection using a variety of algorithms. However, these mechanisms are more useful for distributing load among multiple replicas of an origin server on a LAN, which we discuss in Section 14.1. In the context of proxy load balancing, destination IP hashing has an important advantage over all these methods: it reduces duplication of cached objects and avoids situations where a request experiences a cache miss at one proxy while another has the requested object. We assume destination IP hashing for proxy load balancing in the rest of this chapter.

L4 switches can share the same set of proxies. For example, in Figure 8.8, one could deploy two switches on the links from one of the ARs to the two BRs. These two switches can share the same set of proxies, and the configuration will work even though neither switch sees all the packets for a given TCP connection. For proxy sharing to make sense, it is essential that all the switches use the same hash function in distributing requests among the shared proxies. Without this requirement, different switches could divert packets with the *same* source and destination IP addresses to *different* proxies. In particular, packets belonging to the same TCP connection could end up at different proxies, resulting in a broken connection.

L4 switches typically monitor the health of their proxies using standard HTTP. They place a small dummy Web page at each proxy and periodically fetch it. Failure to fetch this page from a proxy indicates a failure of the proxy. This elegant trick allows L4 switches to monitor proxy health without any special protocols, making these switches compatible with any proxy. The downside is that, since fetching pages uses TCP, failure to fetch the dummy page is detected through TCP retries and timeouts, and can take several seconds, with a corresponding service interruption. As we describe in Section 8.3.4, intercepting routers and links implement a quicker failure detection using special protocols designed for this purpose. Nothing prevents L4 switches from implementing such quicker failure detection as well; the only problem is compatibility with various proxy products which would have to implement the special protocol. This has become less of an issue as many proxy products now implement two competing intercepter-proxy protocols that include failure detection: Cisco's *Web Cache Communication Protocol* (*WCCP*) [WCCP n.d.] and *Network Element Control Protocol* (*NECP*) developed by an industrial consortium [NECP 2001]. We come back to these protocols in the next subsection.

8.3.4 Router as an Intercepter

Packets can also be intercepted by a router, which can look at the relevant IP and TCP headers of a packet just as well as an L4 switch does. One difference from switches is that routers (at least intercepting routers by Cisco) use IP-in-IP encapsulation to deliver packets to proxies while switches typically use the data link approach. Like L4 switches, several intercepting routers can share the same set of proxies and balance the load among multiple proxies based on hashing the destination IP addresses. We should note that unlike switches that are often configured to use other methods of load distribution, especially among origin server replicas, routers are hardly ever configured to use other methods. The reason is that routers are highly optimized for examining destination IP addresses in hardware, and having to consider other information can slow them down significantly.

Intercepting at a router is attractive because it does not require any additional network elements. Routers are also natural traffic aggregation points that are more likely to see all packets from a client. In the architecture of Figure 8.1, intercepting could be done at ARs. While customers often connect at more than one backbone node and

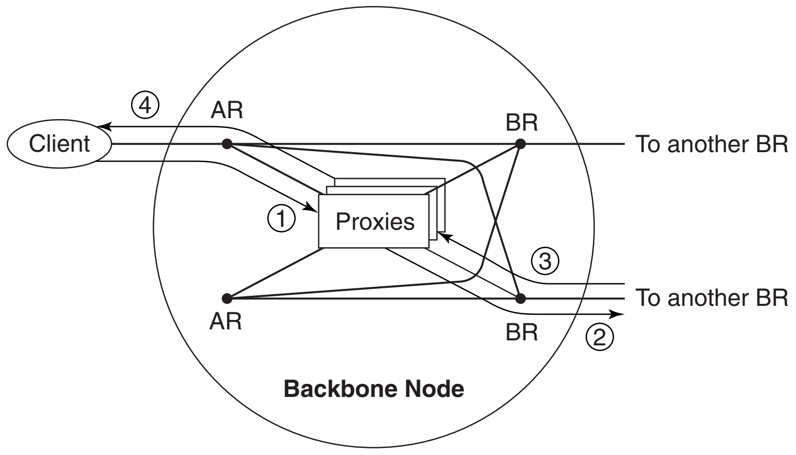


Figure 8.9 A router-based deployment of interception proxies

more than one AR, they typically can be configured to use a single primary AR, with other connections acting as backups. Thus, in the absence of failures, the same client sends all its packets via the same AR and no connection disruption—of the type shown in Figure 8.7—may occur. We discuss handling exceptions later in this subsection.

Figure 8.9 shows a backbone node using a router-based interception proxy deployment. Arrows indicate information flow on a cache miss. First, the AR intercepts request packets and diverts them to one of the proxies (the proxy is chosen by hashing the packet destination IP address). The proxy detects a miss and fetches the object from the origin server using its connection to BRs (steps 2 and 3). The proxy then sends back the object to the client via the AR (step 4). It is important that the proxy communicate with the Internet via BRs rather than use its link to the AR again. If the proxy sends these packets to the AR, then the AR will have to handle all packets related to missed requests *twice:* once when delivering them from the Internet to the proxy, and the second time when delivering them from the proxy to the client. If we assume 70 percent of all Internet traffic to be Web traffic (see Section 6.7) and a rather conservative miss ratio of 50 percent (see Chapter 7), this could easily cause a 35 percent load increase on the router. In practice, proxies are often connected only to ARs and use them for all communication, but unless there is a port shortage at BRs, such a configuration is suboptimal.

Another reason for connecting the proxy to a BR for communication with the Internet is efficient handling of exceptions in intercepting connections. There are always going to be cases (clients or Web sites) that should be excluded from interception. A Cisco router allows elaborate configuration rules to specify such exceptions. The trouble is, the rule processing occurs mostly in software, on a so-called slow processing path. The result is significantly lower router throughput and higher message delays. Moreover, all packets, whether or not they fall under the exception, must be tested against these rules, so the performance degradation applies to all Web traffic.

As previously mentioned, one could offload some of this rule processing to the proxy. In this case, the router would more or less blindly divert all Web traffic to the proxy, while the proxy would identify the exception packets and return them back to the Internet. This works especially well when the proxy is connected to the BRs for its communication with the Internet, as shown in Figure 8.9, since the returned packets bypass the intercepting router and do not increase its load.

Almost all models of Cisco routers are now able to intercept connections. Unlike L4 switches, they employ a special protocol, WCCP, for control communication between the router and the proxies. Cisco developed WCCP as a proprietary protocol but later opened it for use by others.

With WCCP, proxies use periodic "I am alive" messages to let the router know they are operational. While HTTP could also be used for failure detection (as we saw in Section 8.3.3), explicit messages allow a finer-grained failure detection. WCCP also gives proxies some control over how the intercepter distributes requests among them. Essentially, the proxies elect a master proxy that can communicate to the router the (re)assignment of IP address space partitions to proxies (see Section 8.3.3 for details on partitioning IP address space among proxies). The router then sends a packet to the proxy assigned to the packet's destination IP address. This type of feedback from proxies to the intercepting element cannot be accomplished over HTTP. Thus, an intercepting element that only uses HTTP for communication with the proxies can only perform "blind" request distribution, without any feedback from the proxy.

NECP, developed by an industrial consortium as an open (that is, nonproprietary) protocol, is an alternative to WCCP for intercepter-proxy communication. In some ways it is even more flexible than WCCP. It allows the proxy to choose between data link and IP layer methods for delivering packets to proxies. It also allows a proxy to dynamically install simple filtering rules at the intercepter, specifying that packets with certain source or destination IP addresses should not be intercepted.

8.3.5 Layer 7 Switch as an Intercepter

A Layer 7 (L7) switch operates at the application layer in the ISO stack. It intercepts requests for opening TCP connections to Web servers (that is, TCP SYN packets directed to port 80, the default port for Web servers) but, unlike an L4 switch, it does not forward these packets immediately to a proxy. Instead, the L7 switch itself performs TCP handshake with clients, impersonating their intended destinations. It then intercepts and interprets HTTP requests (or those of other application-level protocols) and only then forwards client packets to a proxy or the Internet. Because of its ability to interpret requests, L7 switches are otherwise known as *content-aware* switches. Arrowpoint Communications (now part of Cisco) was among the first vendors to bring these switches to the market. Several L4 switch vendors have added content awareness to their products as well.

Content awareness opens several possibilities. In the context of forward proxies, the main advantage is that the switch can be configured to send requests for different types of content to different proxies. For instance, requests for images and HTML files can be sent to one proxy, requests for video and audio files to another, and requests for live streaming data to yet another proxy. Thus, different proxies can be tuned to different content types. A proxy serving video files can be tuned differently, or even have an entirely different architecture from a proxy serving relatively small HTML pages. An L4 switch without content awareness is oblivious to what object is requested; it therefore cannot take content type into account when distributing requests among proxies.

Another advantage of L7 switches has to do with providing differentiated quality of service to different Web sites. Since this feature is mostly relevant to hosting service providers and other server-centric platforms, we defer further details of this feature until Section 14.3.3 in Part III of the book.

An alleged advantage of L7 switches is that they can avoid diverting packets when requests are for uncacheable content [Johnson 1999]. Since the proxy does not have to process these requests, its load is reduced; at the same time, the processing path for these requests (and, hence, their response time) becomes shorter. This might look like a great advantage of L7 switches except that it interacts in an unfortunate way with persistent connections.

Consider an uncacheable page with many embedded images. When an L7 switch receives a request for this page, it sends the request directly to the origin server, bypassing the proxy. But then the client may decide to reuse this TCP connection for embedded objects. Since the connection has been established between the client and the origin server and not between the client and the proxy, *all* requests over this connection must go to the origin server. Indeed, the proxy will discard any packets that belong to this connection since it has no information about this connection. In other words, once the switch chooses where to send a request, it must send all other requests over the same TCP connection to the same host, regardless of their cacheability. This may severely degrade the hit rate because it is common for a dynamically generated page to contain multiple embedded images on the same server, and these images account for much of the hit rate.

Arguably, a switch can fix this problem by maintaining its own persistent TCP connection with the client and merging responses from the origin server and proxies onto that connection. Figure 8.10 provides a simplified illustration of the approach. In this example, a client requests an uncacheable object X and then a cacheable object Y over the same connection. The switch maintains the connection to the client on behalf of the origin server; at the same time, the switch uses separate connections to the origin server, to get X, and to the proxy, to get Y. It then forwards X and Y to the client over the persistent connection between the client and the switch. In other words, the data that the switch receives over the persistent connection from the client it splits between the two connections, one to the origin server and one to the proxy; the switch then merges the data received over these two connections back into the single connection to the client.

No vendor, to our knowledge, has implemented this approach at the time of this writing. In fact, it remains to be seen if the approach is worthwhile. It triples the

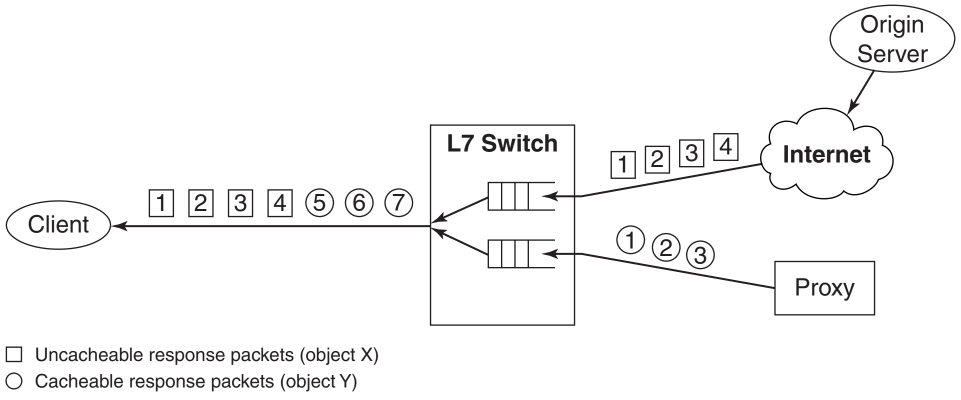

Origin Server
Internet
L7 Switch
Client
Proxy
1 2 3 4
1 2 3 4 5 6 7
1 2 3
☐ Uncacheable response packets (object X)
○ Cacheable response packets (object Y)

Figure 8.10 TCP connection merging by an L7 switch

number of connections the switch must handle when downloading a Web document that includes a dynamic container page and static embedded objects. Indeed, whereas the switch had to deal with one connection from the client to the proxy before, now the switch must handle one connection between the client and the switch itself, one between the switch and the origin server, and one between the switch and the proxy. Moreover, without this approach, once the switch establishes the TCP connection with the client and chooses the proxy for the HTTP request, the switch passes all packets between the proxy and the client without any buffering. Now the switch must maintain full TCP state for each of the three connections. In some cases, this approach forces the switch to buffer even more data than TCP would normally. In our example, if the client pipelines the request for Y after the request for X, then the responses from the switch must return in the same order. In the worst case, if object Y arrives before X, the switch must buffer the entire object Y to send it after sending X, as stipulated by the response ordering in the pipeline.

Thus, TCP connection merging will undoubtedly increase the performance cost of content awareness, and content awareness already comes at a hefty performance cost. Depending on the required amount of processing at the application layer, it can easily decrease throughput of the switch by 40 percent or more. Imposing additional penalty for TCP connection merging may not be worthwhile. Without connection merging, selective interception for HTTP 1.1 clients seems a bad idea.

8.3.6 Intercepting Link

One can also use a network link to intercept packets. Cobalt Networks, now part of Sun Microsystems, and InfoLibria offer such a device, which we call an *intercepting link*. The intercepting link is a box consisting of a tightly coupled pair of modules: a simple switch and a proxy (Figure 8.11). The switch contains only two exposed network interfaces and one internal interface to the proxy. Logically, a network link

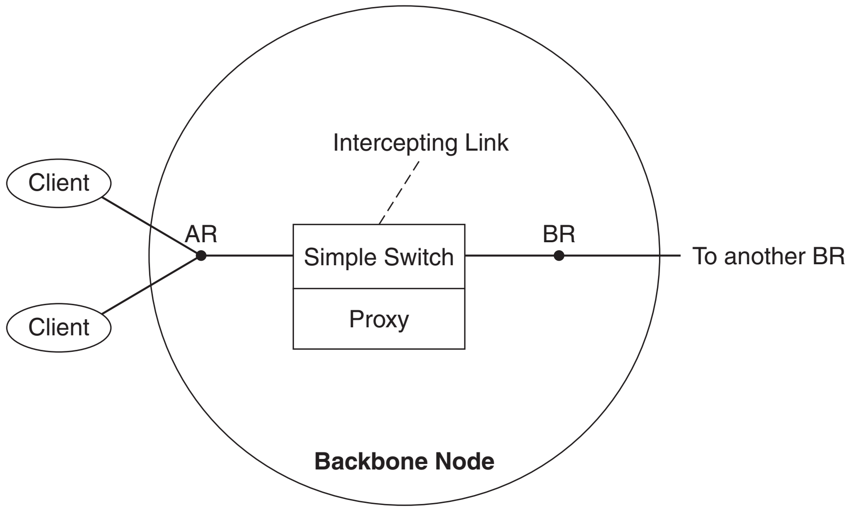


Figure 8.11 An interception proxy deployment with an intercepting link

is "cut" and the device interposed by connecting the two "loose" ends of the link to the two switch interfaces. The switch monitors all traffic on the link and diverts all Web request packets (the TCP port 80 packets) to its internal proxy. For example, Figure 8.11 shows the deployment of an intercepting link in a backbone node with only one pair of access and backbone routers. In this figure, the intercepting link is inserted into the link between the routers.

The main advantage of the intercepting link is its simplicity. The switch is a very simple device, easy to make reliable and efficient. This simplicity also makes the device easy to configure and administer (vendors claim these are truly plug-and-play devices). The intercepting link also avoids the overload of existing routers and switches for handling missed objects twice, once when they flow from the origin server into the proxy and once when they go from the proxy to the client (see Section 8.3.4). Finally, tight coupling of the switch and the proxy facilitates fast detection of proxy failures and efficient shortcutting of the failed cache [InfoLibria 2001].

The main limitation of this approach is that it may be difficult to find a link where an intercepting link could be interposed. A candidate link must see all traffic from a given session. Large ISPs, however, usually ensure that there are always redundant paths between any points in their networks. If traffic is distributed across redundant paths, then no single link can be guaranteed to see all packets. If redundant links are used only as backups, then finding a link appropriate for interception becomes easier. Figure 8.11 shows an appropriate link because the system has only one access router and one backbone router.

A backbone node more typical for a large ISP, shown in Figure 8.1, does not contain an appropriate link if each access router splits its traffic between the two backbone routers. In this case, intercepting links could only be used on the links between ARs and

BRs if ARs were configured to split traffic at the granularity of TCP sessions, so that all packets from a given session would go on the same link. While this mode of operation is available in modern routers, it places an extra load on routers and therefore is unlikely to find acceptance from network administrators. Another disadvantage is that the tight coupling of the switch and the proxy means that multiple switches cannot feed the same proxy cache and, conversely, the same intercepting link cannot feed multiple proxy caches.

Overall, it appears that the most likely market for intercepting links is small low-cost ISPs as well as enterprise networks. Their network administrators will find the ease of administration most appealing. There will also more likely be a convenient point in these networks to deploy an intercepting link.

8.3.7 Performance Pitfalls

We already discussed the main limitation of interception proxies: the violation of the end-to-end principle of the Internet and the multipath problem that stems from it. This limitation prevents some clients from using a proxy and requires careful placement of intercepting elements in the network. This subsection considers more subtle pitfalls of interception proxies that do not cause service disruption but that may degrade the performance of interception proxies when they are compared to their explicit counterparts.

Replicated Web Sites

When a client is configured to use an explicit proxy, it sends the proxy an entire URL including the host name portion, for example, http://www.firm-x.com/index.html. With interception proxies, the client assumes it speaks to the origin server; it therefore does not send the host name portion in the request line of the HTTP request. The request line would contain only the "/index.html" portion of the URL. How can the interception proxy distinguish in its cache similarly named objects from different sites, such as http://www.firm-x.com/index.html and http://www.firm-y.com/index.html? A simple way to extract the DNS name of the Web site from the HTTP `host` header of the request and use this name as the host ID. Studies have shown that over 98 percent of HTTP 1.0 requests and virtually all HTTP 1.1 requests contain this header [Feldmann 1999], so this mechanism largely solves the problem but can reduce bit rate for accesses to replicated Web sites.

Consider a Web site that performs DNS-level request distribution among replicated servers. We discuss this mechanism in detail in Chapters 14 and 15. In short, an object can be replicated on several hosts, and different requests can be directed to different hosts by the DNS system of the object's site. When the intercepting element divides load among multiple proxies, requests to different server replicas may well arrive at different proxies. This is because the intercepter chooses a proxy for a given request based on the destination IP address in the request (see Section 8.3.3), and different server replicas have different IP addresses. When one proxy has an object in its cache and a request for this object arrives at another proxy that does not have the object, this request will be a miss. Only cooperation between the proxies (described in Chapter 9) can avoid the reduction in hit rate. The degree of this hit-rate reduction in practice has not been adequately studied. Note that L7 intercepting switches can balance the load among proxies based on the requested URLs rather than destination IP address. They would not be prone to the hit-rate reduction described here, because all requests for a given object go to the same proxy.

Limited TCP Connection Reuse

Interception proxies have less potential than explicit proxies in reducing latency for users. As discussed in Section 7.2.3, a large portion of latency reduction comes from connection caching, that is, reuse of TCP connections for requests from multiple origins by multiple clients. An interception proxy can cache its TCP connections to origin servers just as an explicit proxy can. However, caching connections to clients becomes impossible when clients are not aware of the proxy [Danzig 1998a].

The situation is illustrated in Figure 8.12. Assume the client sends a request to Web site firm-x.com, and that the client requests a persistent connection. The proxy intercepts the request and agrees to maintain the persistent connection with the client. Recall that the proxy impersonates firm-x.com to the client, so the client thinks it has a TCP connection with firm-x.com. Assume that the client now wants to send a request to firm-y.com. From the client's perspective, it has not spoken to firm-y.com, so it needs to establish the TCP connection first. The proxy must accept this new connection on behalf of firm-y.com. The result is that the client has not reused the existing TCP connection to the proxy, even though it is available and idle, whereas if the client communicated with the proxy explicitly, it could send both requests over the same connection.

At this writing, the impact of this phenomenon on performance has not yet been investigated. Note that the interception proxy does not make matters worse compared to the no-proxy case: when a client uses a persistent connection to fetch multiple objects from the same server, the client also uses the persistent connection in the presence

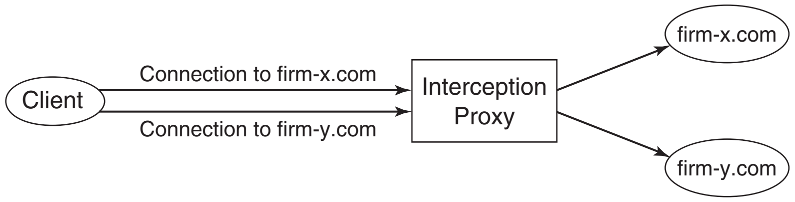


Figure 8.12 Duplication of TCP connections between a client and an interception proxy

rabinovich-60087 book November 29, 2001 9:25

of the interception proxy. However, the interception proxy reduces *additional* opportunities of using persistent connections that can arise when an explicit proxy is used.

8.4 Security and Access Control Issues

Every time a new component is added to the Internet, it is vital to consider the impact this component has on the security and reliability of the Internet as a whole. In the case of proxies, this impact can be profound; it affects how Web servers implement access control—that is, limit the clients that can use their services—as well as a variety of security issues.

8.4.1 Proxies and Web Server Access Control

Many Web sites provide services only to specific clients, for example, those who agree to pay for the service. To enforce access restrictions, Web servers must implement client access control. Some Web servers implement access control at the IP layer. In this mechanism, a server maintains a list of IP addresses of authorized clients and accepts requests only from those clients. Unfortunately, a proxy may break IP-based access control. If a proxy intercepts requests and issues its own requests on a client's behalf, these requests carry the proxy IP address as their source IP address. The Web site will not recognize this IP address and will refuse to service these requests.

An actual example of this problem involved three companies, CyberCache, Gleim, and Digex. CyberCache is a company that signs up business customers to execute their monetary transactions with their clients (we refer to them as end-customers). Gleim was CyberCache's customer, and Digex was the ISP providing connectivity to Gleim. Once Digex deployed an interception proxy, CyberCache stopped honoring Gleim's requests, which were now conveyed to CyberCache by Digex's proxy. Gleim was unable to process orders from its end-customers and was considering suing Digex for losses it suffered from Digex's interception proxies.

Of course, an explicit proxy would cause the same problem; the difference is that the business customer would have to configure its client software to use the proxy. Presumably it would not do this or would reconfigure its software so as not to use the explicit proxy once the problem transpires. With an interception proxy, the client is not aware of the proxy and cannot unilaterally bypass the proxy even if it learns about its existence.

One could say that a proxy should impersonate the client when sending its request to the origin server; then situations such as CyberCache's would not arise. Unfortunately, asymmetric routing (see Section 2.3) invalidates this solution. Considering Figure 8.6 again, we note that asymmetric routing could cause the bottom route to be the primary route for packets from a client to a server while the top route would be the primary route for responses from the server. Then, if the interception proxy at the bottom ISP impersonated the client, the server packets would bypass the proxy and arrive at the client. The proxy would get no response, assume that the connection to the server is broken, and return an error to the client.

Thus far, the only current technical remedy to the IP-based authentication problem is for the proxy to identify this problem from the error messages of the origin servers. Once an origin server is suspected of performing IP-based access control, the proxy should stop caching this server and simply forward the client packets to the origin server unchanged. In a transparent deployment using the NECP protocol, the proxy can also inform the intercepter to stop intercepting packets destined for this server.

A nontechnical remedy is for the Web sites to avoid this method of access control. There are better methods for authenticating clients and restricting access to information that could be used instead. These methods include HTTP authentication, cookie headers, digital signatures, and encryption.

8.4.2 Proxies and Security

Proxy deployment leads to security concerns that can be classified in four major areas. The first is secure systems management. Proxies are generally high-performance, well-connected computers running complex software. This makes them prime candidates for hackers to break into and to use for various attacks. On the other hand, proxies do not score worse than other well-connected computers such as popular Web servers. Therefore, the problem of secure systems management is not really specific to proxies. Still, experience has shown that this important issue is frequently neglected, and a reminder in the context of this book seems advisable.

The second area of concern is the protection against denial-of-service attacks. In a denial-of-service attack, the attacker overloads either the proxy, the network connection to the proxy, or other necessary components like DNS servers, to prevent legitimate users from receiving service. There are multiple possible methods an attacker can use to achieve this goal. For example, the attacker could send IP fragments to the proxy, overloading the proxy and thereby preventing legitimate clients from receiving service. In contrast to a single Web site, such an attack on a proxy blocks not only a single site, but all sites for all clients using the proxy. This makes it extremely important that proxies provide a fail-over mechanism in case they become unavailable.

The third area of concern is the security of a client browsing the Web. The Secure Socket Layer (SSL) protocol is generally used to secure traffic between a client and a Web server. The use of SSL protects the privacy and integrity of the data transmitted. Since SSL aims at ensuring end-to-end privacy, it by definition precludes the use of proxies, which stand in between the intended communicating parties. The SSL issue is therefore a concern for proxy caching because SSL narrows the applicability of the latter: ubiquitous use of SSL would push proxies into oblivion.

The last area of concern is integrity of the content. Since caches do serve data for a Web server, it is important to ensure not only that the content is up to date (as described in Chapter 10), but also that the content cannot be substituted or altered by an attacker. In the context of proxies, there are two potential problems. The first is the retrieval of the content from the origin server. The attacker could pose as the origin server to the proxy by, for example, subverting the routing infrastructure. Requests to the origin server would then be fulfilled by the attacker, which could replace the content of the Web site arbitrarily. The second possible problem is that an attacker could target the proxy itself. An attacker breaking into the proxy can add, delete, and replace cached documents arbitrarily.

In terms of integrity, explicit proxies have one additional area of concern. An explicit proxy has to perform a DNS resolution to find the origin server. This DNS resolution could be intercepted by an attacker that could trick the explicit proxy into retrieving the content from the wrong server. Interception proxies do not face the DNS problem, since they already know the IP address of the origin server and do not have to resolve a DNS name to reach it.

8.5 Summary

A key issue in proxy deployment is delivering client requests to the proxy. The main alternatives include nontransparent deployment, where clients explicitly send requests to the proxy, and transparent deployment, which leaves clients unaware at the IP layer of the proxy's existence.

With nontransparent deployment, the client knows at the IP layer that it speaks to a proxy and not the origin server; therefore, the client must somehow be told to communicate with the proxy. This can be achieved through explicit client configuration, an autoconfiguration file using a configured URL, or autodiscovery of the configuration file. Proxies deployed in a nontransparant manner are called explicit proxies.

Transparent proxy deployment relies on a network element that intercepts requests from clients to origin servers and diverts these requests to the proxy. Proxies deployed using this method are called interception proxies. Request interception can occur at the transport layer (L4) or application layer (L7), and can be accomplished by a switch (L4 or L7 switch), a router, or a special device interposed in a network link (an intercepting link). L4 elements interpret only IP and TCP headers of packets and use destination IP addresses to choose a proxy for the request. Some routers, intercepting links, and L4 switches operate at this layer. L7 elements, represented by L7 switches, can interpret the contents of the request and use this content in choosing a proxy for the request or for implementing fine-grained quality of service policies. However, such content awareness comes at a cost and significantly reduces the throughput of the network element. L4 elements are simpler and leave more processing to the proxy itself but have less sophisticated policies. Recently, intercepting elements that historically operated at the L4 layer have been adding optional content-aware features, while L7 elements have allowed operation in the L4 mode. With this convergence, the choice between L4 and L7 interception is becoming a choice between operation modes rather than products, and the product choice is increasingly based on issues such as price and performance.

Explicit proxies require client configuration and rely on cooperation from the clients. Interception proxies have their own limitations, such as a need for careful placement in the network to avoid a possibility of connection disruptions, disruption of IP-based access control, and more limited reuse of TCP connections. These pitfalls stem from the fact that interception proxies violate the end-to-end principle of the Internet.

For an environment such as an enterprise network where both proxies and clients belong to the same administrative domain, deciding between explicit and interception deployment is a matter of the tradeoff between upholding the end-to-end Internet principle versus administrative convenience and proxy enforcement (forcing clients to go through a proxy may be motivated by the desire to monitor or control users' Web surfing). In environments where clients are outside the control of the network administrator deploying the proxy, interception proxies may be the only feasible alternative.