CHAPTER II

LITERATURE REVIEW & PROXY SERVER

2.1 **Introduction**

This chapter is described technical parts which are the feature of HTTP and TCP packets that used for World Wide Web, Proxy server deployment, and the simulation test of The MODEL-DRIVEN SIMULATION OF WORLD-WIDE-WEB CACHE POLICIES..

2.2 **HTTP: Hypertext Transfer Protocol**

Hypertext Transfer Protocol also called HTTP, is the basic protocol for the World Wide Web (WWW), which is supplied refer to as the Web. In this chapter we examine this protocol and the operation of Web server.

Statistic from NSGnet backbone (Table 2.1) shows the incredible growth in HTTP usage since January 1994.

Table 2.1 Packet count percentage for various protocols on NSFnet backbone

Monthly	HTTP	NNTP	FTP	Telnet	SMTP	DNS	Packets x 10e9
1994 Jan	1.5%	8.8%	21.4%	15.4%	7.4%	5.8%	55
1994 Apr	2.8	9.0	20.0	13.2	8.4	5.0	71
1994 Jul	4.5	10.6	19.8	13.9	7.5	5.3	74
1994 Oct	7.0	9.8	19.7	12.6	8.1	5.4	100
1995 Jan	13.1	10.0	18.8	10.4	7.4	5.4	87
1995 Apr	21.4	8.1	14.0	7.5	6.4	5.4	59

The packet count is calculated in these percentages [9]. As the HTTP percentage increases, both FTP and Telnet percentages decrease. It is noted that the total number of packets increased through 1994, and then started to decrease in 1995. The packet percentage is still valid, and shows the growth in HTTP traffic.

A simplified organization of the Web is shown in the Figure 2.1

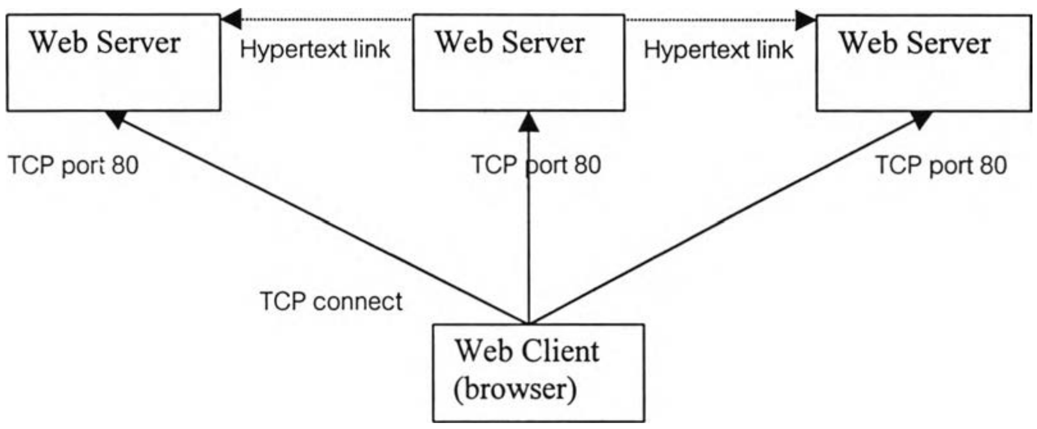

Figure 2.1 Organization of Web client –server

The Web client, usually is called a browser, communicates with the Web server using one or more TCP connections. The well-known packet for Web server is TCP port 80. The protocol used by client and server to communicate over TCP connection is called HTTP, the Hypertext Transfer Protocol, which is described in this chapter. It also shows how a given Web server, point to other Web servers with hypertext links. These links are not restricted to pointing only to other Web servers.

Although HTTP has been in use since 1990, the first available documentation appeared in 1993 appropriately describes HTTP version 1.0, but this Internet Draft expired long ago. As this newer documentation is available, though still as an Internet Draft [4].

One type of document returned by a Web server to a client is called an HTTP, documents, followed by a more detailed examination of the protocol. Then look at how a popular browser (Netscape) uses the protocol, some statistics regarding HTTP's use of TCP, and some of the performance problems with HTTP.

2.3 Introduction to HTTP and HTML

HTTP is a simple protocol. The client establishes a TCP connection to the server [8], issues a request, and reads back the server's response. The server denotes the end of its response be closing the connection. The first returned by the server normally contains pointer to other files that can reside on the other server.

As with many of the Internet protocols, an easy way to see what's going on is to run a Telnet client and communicate with the appropriate server. This is possible with HTTP because the client sends lines containing ASCII commands to the server's is ASCII with extensions [6].

2.3.1 HTTP Protocol

The example in the previous section, with the client issuing the command GET /, is an HTTP version 0.9 command. Most of server support this version but the current version with 1.0 the client specifies the version as part of the request line, for example

Message Types: Requests and Responses.

There are two HTTP/1.0 message types: requests and responses. The format of an HTTP/1.0 request is:

Request-line

Headers (0 or more)

<blank line>

body (only for POST request)

The format of the request-line is;

Request request-URL HTTP version

Three requests are supported [4];

1. The GET request, which returns whatever information is identified by the request-URL
2. The HEAD request is similar to the GET request, but only the server's header information is returned, not the actual contents (the body) of the specified document. The request is often used to test hypertext link for validity, accessibility, and recent modification.

3. The POST request is used for posting electronic mail, news, or sending forms that can be filled in by an interactive user. This is the only request that sends a body with the request. A valid Content header field is required to specify the length of the body.

In a sample 500,000 client requests on a busy Web server, 99.68% were GET, 0.25% were HEAD, and 0.7% were POST. ON server that accepted pizza order, however, we would expect a much higher percentage of POST requests.

The format of an HTTP/1.0 response is;

Status-line

Header (0 or more)

<blank line>

body

2.3.2 Header fields

With HTTP both the request and response can contain a variable number of header fields. A blank line separates the header fields from the body. A header field consists of a field name, followed by a colon, a single space, and the field value. Field names are case insensitive. Headers can be divided into three categories; those that apply to requests, those that apply to responses, and those that describe the body. Some headers apply to both requests and responses. Those that describe the body can appear in POST request and response.

2.3.3 Response codes

The first line of the server's response is called the status line. It begins with the HTTP version, followed by a 3-digit numberic response code, followed by a human readable response. The meanings of the numberic 3-digit response codes are shown in Table. 2.2. The first of the meanings of the three digits divides the code into one of five general categories.

Using a 3-digit response code of this type is not an arbitrary choice. We'll see that NNTP also uses these type of response codes (Table 2.2), as do other Internet application such as FTP and SMTP.

Table 2.2 The HTTP 3-digit numberic response code

Response	Description
200	OK, response succeeded
201	OK, new resource created (POST command).
202	Request accepted but processing not completed
204	OK, but no content to return
301	Requested resource has been assigned a new permanent URL
302	Requested resource resides temporarily under a different URL
304	Document has not been modified
400	Bad request.
401	Unauthorized, request requires user authentication
403	Forbidden for unspecified reason
404	Not found
500	Internal server error
501	Not implemented
502	Bad gateway; invalid response from gateway or upstream server
503	Service temporarily unavailable.

2.3.4 TCP Client-Server

Our next example of a client-server transaction application uses TCP. Figure 2.2 shows the client program [5].

2.3.5 Create TCP socket and connect to server

A TCP socket is created by socket and then an Internet socket address structure is filled in with the IP address and port number of the server. The call to connect causes TCP's three-way handshake to occur, establishing a connection between the client and server.

2.3.6 Send request and half-close the connection

The client's request is sent to the server by write. The client then closes one-half of connection, the direction of data flow from the client to the server, by client is done sending data: it passes an end-of-file notification from the client to the server. A TCP connection—only one direction of data flow is closed. This is called TCP's half-close.

2.3.7 Ready reply

The reply is read by our function read_stream, shown in Figure 2.2. Since TCP is a byte-stream protocol, without any form of record markers, the reply from the server's TCP can be returned in one or more TCP segments. This can be returned to the client process in one or more reads. Furthermore we know that when the server has sent the complete reply, the server process closes the connection, causing its TCP to send a FIN segment to the client, which is returned to the client process by read returning an end-of-file. To handle these details, the function read stream calls read as many times as necessary, until either the input buffer is full, or an end-of-file is returned by read.

```
#include      *cliserv.h
int
main(int argc, char *argv{})
{                                       /* simple TCP client */
        struct sockaddr_in serv;
        char    request [REQUEST], reply[REPLY];
        int     sockfd, n;
        if (argc != 2 )
        err_quit (*usage: tcpcli <IP address of server>*);
        if ((sockfd = socket (PF_INET, SOCK_STREAM, 0 )) < 0 )
        err_sys (*socket error*);
        memset (&serv, 0 , sizeof(serv));
        serv.sin_family = AF_INET;
        serv.sin_addr.s_addr = inet_addr(argv [1]);
        serv.sin_port = htons (TCP_SERV_PORT);
        if (connect (sockfd, (SA) & serv, sizeof(serv) < 0 )
        err_sys (*connect error*);
        /* form request[]....*/
        if (wirte(sockfd, request, REQUEST) != REQUEST)
        err_sys ("write error*);
        if (shudown (sockfd, 1) < 0)
        err_sys(*shutdown error*);
        if ((n = read_stream(sockfd, reply, REPLY)) < 0)
        err_sys (*read error*);
        /* process "n" bytes of reply[] ...*/
        exit (0);
        }
```

Figure 2.2 TCP transaction client

```
#include      "cliserv.h"
int
read_stream (int fd, char *ptr, int maxbytes)
{
        int     nleft, nread;
        nleft = maxbytes;
        while (nleft > 0) {
        if ((nread = read(fd, ptr, nleft)) < 0)
        return (nread) ;
        else if (nread = = 0)
        break;
        nleft - = nread;
        ptr + = nread;
        }
        return (maxbytes – nleft );
```

Figure 2.3 TCP transaction client

These are other ways to delineate records when a stream protocol such as TCP is used. Many Internet applications (FTP, SMTP, HTTP, and NNTP) terminate each record with a carriage return and linefeed. Others (DNS,RPC) precede each record with a fixed-size record length. In our example we use TCP's and-of-file flag (the FIN) since we send only one request from the client to the server, and one reply back. FTP also uses this technique with its data connection, to tell the other end when the end-of-file is encountered.

2.3.8 Create listening TCP socket

A TCP socket is created and the server's well-known port is bound to the socket. As with the UDP server, the TCP server binds the wildcard as its local IP address. The call to listen makes the socket a listening socket on which incoming connections will be accepted, and the second argument of SOMAXCONN specifies the maximum number of pending connections the kernel will queue for the socket.

```
#include      "clisev.h"
int
main()
{                         /* simple TCP server */
     struct sockaddr_in serv, cli;
     char request[REQUEST], reply[REPLY]
     int listenfd, sockfd, n, clilen
     if ((listenfd = socket (PF_INET, SOCK_STREAM, 0)) < 0 )
     err_sys ("socket error")
     memset (&serv, 0, sizeof (serv));
     serv.sin_family = AF_INET;
     serv.sin_addr.s_addr = htonl (INADDR_ANY);
     serv.sin_port = htons (TCP_SERV_PORT);
     if (bind(listenfd, (SA) & serv, sizeof (serv)) < 0)
     err_sys("bind error");
     if (listen(listenfd, SOMAXCONN) < 0)
     err_sys (#listen error*);
     for (;;) {
     clilen = sizeof (cli);
     if ((sockfd = accept(listenfd, (SA) & clilen)) < 0)
     err_sys (*accept error*);
     if ((n = read_stream(sockfd, request, REQUEST )) < 0)
     err_sys ("read error");
     /* process "n" byte of request[] and create reply[]...*/
     if (write(sockfd, reply, REPLY) ! = REPLY)
     err_sys (*write error*);
     close(scokfd);
       }
     }
```

Figure 2.4 shows the TCP server

2.3.9 Accept a connection and process request

The server blocks in the call to accept until a connection is established by the client's connect. The new socket descriptor returned by accept, sockfd, refers to the connection to the client. The client's request is read by read_stream and the reply is returned by write.

With TCP the client's measured transaction time is at least 2 x RTT + SPT than it might appear form Figure 2.5 [3]. The additional RTT in this example is from the establishment of the TCP connection: the first two segments that we show in Figure 2.5.

If TCP could combine the establishment of the connection with the client's data and the client's FIN (the first four segments from the client in the figure), and then combine the server's reply with the server's FIN, we would be back to transaction time of RTT + SPT.

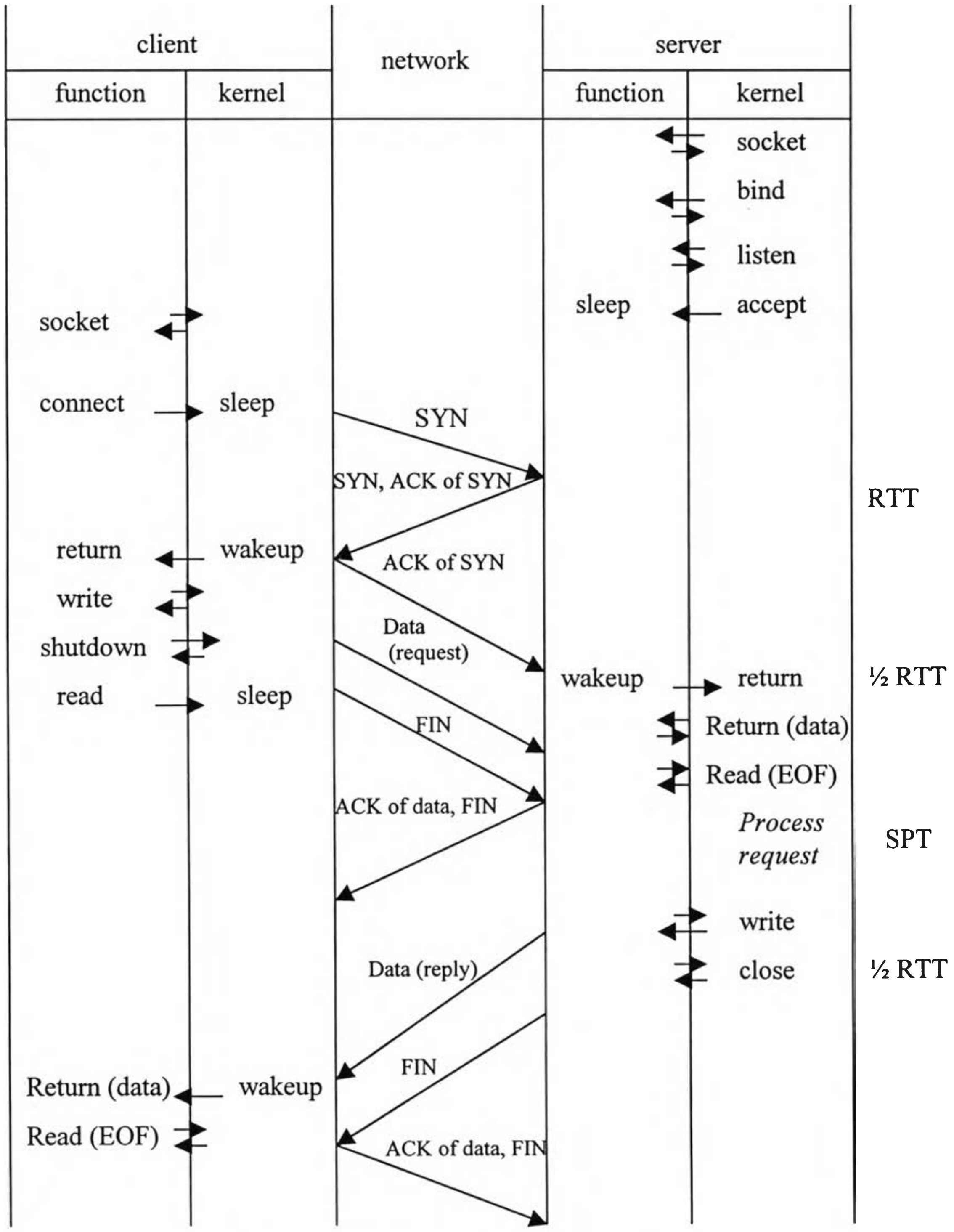

Figure 2.5 The transaction time of RTT + SPT

2.4 TCP's TIME_WAIT State

TCP requires that the endpoint that sends the first FIN, which in our example is the client, must remain in the TIME_WAIT state for twice the maximum segment lifetime (MSL) once the connection is completely closed by both ends. The recommended value for the MSL is 120 seconds, implying a TIME_WAIT delay of 4 minutes. While the connection is in the TIME_WAIT state, that same connection (i.e., the same four values for the client IP address, client port, server IP address, and server port) cannot be opened again.

In our example the client sends the first FIN, termed the active close, so the TIME_WAIT delay occurs on the client host. During this delay certain state information is maintained by TCP for this connection to handle segments correctly that may have been delayed in the network and arrive after the connection is closed. Also, if the final ACK is lost, the server will retransmit its FIN, causing the client to retransmit the final ACK.

Other applications, notably HTTP, which is used with the World Wide Web, have the client send a special command including it is done sending its request (instead of half-closing the connection as we do in our client), and then the server sends its reply, followed by the server's FIN. The client then sends its FIN. The difference here is that the TIME_WAIT delay now occurs on the server host instead of the client host. On a busy server that is contacted by many clients, the required state information can account for lots of memory tied up on the server. Therefore, which and of the connection ends up in the TIME_WAIT state needs to be considered when designing a transactional client-server.

2.4.1 Reducing the Number of Segments with TCP

TCP can reduce the number of segments in the transaction shown in Figure 2.6 by combining data with the control segments. Notice that the first segment now contains the SYN, data, and FIN, not just the SYN as in Figure 2.6. Similarly the server's reply is combined with the server's FIN. Although this sequence of packets is legal under the rules of TCP, the author is not aware of a method for an application to cause TCP to generate this sequence of segments using the sockets API and knows of no implementations that actually generate this sequence of segments.

What is interesting to note is that even though we have reduced the number of segments from nine to five, the client's observed transaction time is still 2 x RTT + SPT because the rules of TCP forbid the server's TCP from delivering the data to the server process until the three-way handshake is complete. The reason for this restriction is that the server must be certain that the client's SYN is "new", that is, not an old STN from a previous connection that got delayed in the network. This is accomplished as follows: the server ACKs the client's SYN, sends its own SYN, and then waits for the client to

ACK the server's SYN. When the three-way handshake is complete, each end knows that the other's SYN is new. Because the server is unable to start processing the client does not decrease the client's measured transaction, shown in Figure 2.6.

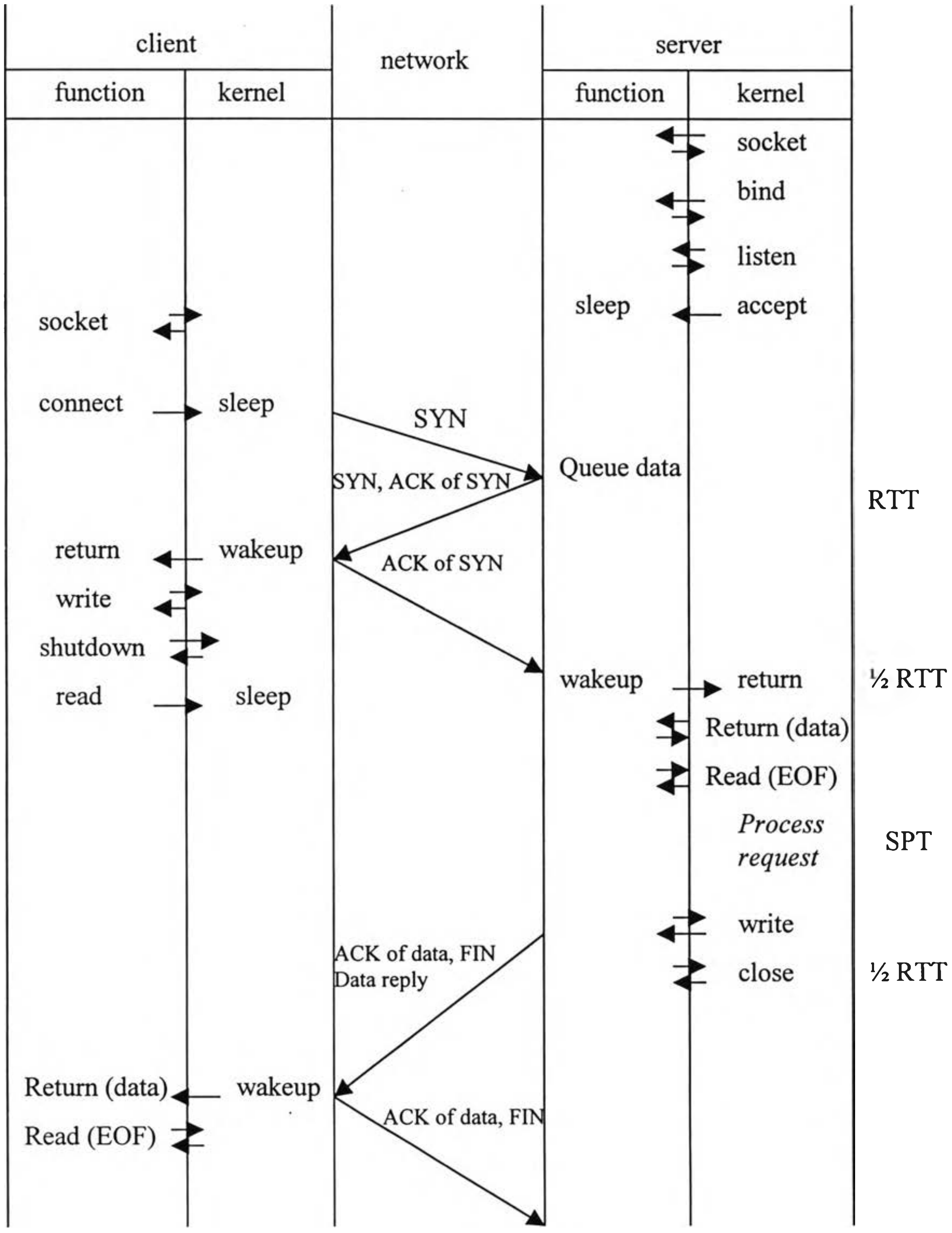

Figure 2.6 The transaction time of Client

2.5 INTERNET Gateway- Proxy

Historically, web proxies were developed to serve as an intermediary for clients as they requested content from remote servers on the Internet. A web proxy was designed to act both as a server to clients and as a client to remote server machines. The proxy enhanced network security by preventing network packets from passing directly between the user's network and the Internet. It effectively screened the network and prevented external audiences from having visibility to network information. At the same time, internal clients maintained the perception of having a direct connection to the Internet [7].

The web proxy also served as a gateway for organizations to control and monitor their users' access to the Internet. Traditionally deployed at the Internet gateway as part of the organization's firewall solution, web proxies would often reside just inside a firewall, or inside a DMZ ("demilitarized zone").

The web proxy's location at the edge of the network was a natural place to cache content as well. By caching content close to users, subsequent requests for the information could be returned directly from the cache rather than fetching it from a remote server. As Internet access has become increasingly important to the business processes of many organizations, the amount of traffic passing through the Internet gateway has increased dramatically and content availability has become critical. The caching web proxy has emerged as an effective solution for reducing network congestion and ensuring content availability.

More recently, intranets have become prevalent and organizations are now deploying caching web proxies at their branch offices, retail outlets, and other locations to reduce traffic at bottlenecks in their internal networks. As new web-based technologies, such as push-based services, emerge and offer the potential for dramatic increases in network traffic, caching web proxies are expected to become even more important.

2.6 Design Goals

As one of the original providers of commercially available web proxies, administrators and users by providing the following benefits:

Scalable and flexible caching. Proxy Server's efficient caching model distributes data where users need it, so requests to remote content servers and network traffic are reduced. Proxy routing makes it possible for organizations to deploy Proxy Server at branch offices and network bottlenecks to benefit from caching on intranets. Caching on-demand intelligently caches documents based on user requests. Batch updates also enable caching on-command, so administrators can download documents or sites on a scheduled basis. Now Proxy Server enhances the scalability and reliability of caching by supporting proxy arrays. This distributed caching mechanism enables multiple proxies to operate as a single logical cache for load-balancing and failover. Support for dynamic proxy routing allows Proxy Server to query other caches for document availability.

Fine-grained filtering. Networks are only as strong as their weakest link, which is often the gateway. Proxy Server enhances network security by providing a control point for Internet traffic and by logging all transactions. Proxy Server's fine-grained controls limit access to specific documents or sites based on individual users, groups, IP addresses, host names, or wildcard expressions. Proxy Server also provides filtering of objectionable URLs, filtering of content including viruses and HTML tags, and filtering of content types. Proxy Server facilitates user access through the firewall. In addition to being able to tunnel protocols supported by the web proxy.

Ease of management. Proxy Server makes it easy for administrators to manage intelligent networks of proxy servers. Native Lightweight Directory Access Protocol (LDAP) support is now available to centralize user name and password management via an integrated Netscape Directory Server [7]. Clustered management capabilities enable administrators to configure and maintain multiple proxies. Netscape Communicator's Automatic Proxy Configuration (APC) feature permits modifications to the proxy infrastructure without touching client software on each desktop. Proxy Server also supports Simple Network Management Protocol (SNMP) versions 1 and 2 for monitoring server status.

2.7 Scalable and Flexible Caching

Caching is one of the core functions of a proxy server. By monitoring network traffic and intelligently caching documents, a proxy server can conserve network bandwidth and dramatically reduce response times for users.

2.7.1 Flexibility

In the default caching model for proxy servers, a web client makes a request that is routed to the proxy server. The proxy server then executes the following steps:

1.It checks to see whether the user is valid.

2.It checks to see whether the request is allowed based on the access controls and filtering rules defined by the administrator.

3.It fetches the document from the remote content server.

4.It returns the document to the user, while filtering content based on administrator-specified criteria.

5.It writes the document to the cache.

6.It logs the transaction.

This process allows the document to be returned from the cache the next time it is requested, rather than having to be fetched again from the remote server, minimizing network traffic and response times for users. This process is also referred to as caching on-demand.

Which protocols does the proxy server cache? Note that this is not necessarily the same as the ability to "proxy" a protocol, which does not always include caching.

Which controls are provided to enable the administrator to specify how frequently the proxy should perform an up-to-date check for the document at the remote web server? In general, more frequent up-to-date checks emphasize document freshness at the expense of performance. The proxy server should provide parameters that maximize efficiency in this process.

Does the proxy server cache secure documents (that is, documents that have access controls associated with them on the remote web server) and Hypertext Transfer Protocol (HTTP) queries?

How does the proxy server handle requests for content that resides on local web servers? In many cases, it may not make sense to cache documents that reside on local hosts.

Does the proxy server provide a tally of hits, or requests for content, to the origin server where the documents were published, even for requests that were returned from the cache? This is important to enable content publishers to track the total number of hits their documents are receiving for billing and other purposes.

Does the administrator have the ability to schedule batch updates (caching on-command) to the cache? This includes: The ability to preload documents or sites into the cache in anticipation of user demand. The ability to automatically refresh documents that already reside in the cache.

2.7.2 Scalability

Scalability increases the efficiency of caching. The larger the effective size of the available cache, the more likely it is that a requested document will reside in the cache.

Hierarchical Caching and Proxy Chaining

As Intranets become pervasive, organizations increasingly need to create multitiered proxy networks with proxies cascaded (as shown in Figure 2.7) to enhance performance and decrease traffic on internal networks. For example, in addition to deploying proxy servers at the Internet gateway, companies should be able to deploy them at major subnets, wide-area network (WAN) connections, and remote offices to create a hierarchical cache network. In this scenario, the organization decreases network traffic across the Internet gateway as well as on internal network bottlenecks. Specific content can then be cached locally, where it is needed.

2.7.3 Distributed Caching

A proxy server must be able to scale as the amount of web-based content it is handling increases. One option is to increase the hardware that the proxy is running on by adding hard disk space or RAM. Another option is to share the load among multiple proxies that are effectively acting as a single logical entity. In this scenario, client requests are deterministically routed to specific proxy caches within an array, based on the document that has been requested. This is an efficient means of load-balancing and enables, for example, three 2GB proxy caches to effectively act like a 6GB cache.

Organizations cannot afford to deny their employees access to network resources if the proxy is unavailable. Requests to a particular proxy should be able to failover to another proxy in the array in the event that the primary proxy becomes unavailable. This fault-tolerance enhances the reliability of network access.

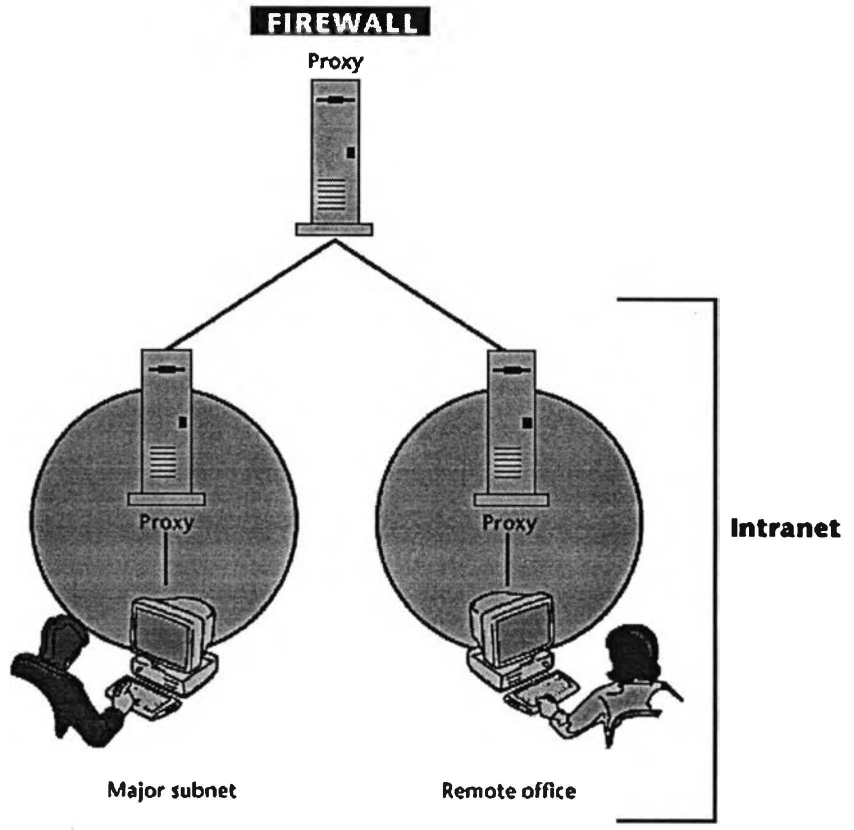

Figure 2.7 Intranet Proxy Network

2.7.4 Dynamic Proxy Routing

In cases where a proxy server does not have a requested document cached, it is often more efficient to fetch the document from another proxy, rather than requesting it from the remote web server on the Internet. If the proxy server can query neighbor or parent caches to determine whether they have the requested document cached, and then obtain the document from the neighbor or parent, it will prevent the request from being routed to the Internet where connections are sometimes unpredictable.

Dynamic proxy routing for automatic content discovery complements distributed caching. In cases where neighbor or parent proxies are outside the administrator's direct control and cannot be included in an array, dynamic querying is an effective means of sharing content.

2.8 Proxy Server Deployment

This guide is intended to assist customers who have decided to deploy Netscape Proxy Server on their enterprise intranet. It assumes that familiar with the product and therefore does not cover its features in depth.

The Netscape Proxy Server Administrator's Guide provided with the software is the best resource for detailed information on proxy server configuration. For more detailed information on product features and functionality, refer to the Netscape Proxy Server data sheet, FAQ, or evaluation guide.

This guide concentrates on the information it need to plan and deploy the proxy server in the organization. It covers the deployment process sequentially, from beginning to end, and is designed to answer the questions it may have at each stage.

The guide is organized as follows:

1 Deciding Which Services It Want to Provide

2 Determining Where to Deploy the Servers

3 Deciding Which Architecture to Use

4 Determining How Many Servers It Will Need

5 Deciding What Type of Hardware to Use

6 Configuring the Servers

7 Tuning the Servers

8 Monitoring the Servers

9 Planning for Enterprise Growth

Although this guide provides much of the information it will need to implement the proxy solution, it does not attempt to cover every possible scenario. The most common setups and configurations are addressed. Customers planning to deploy more unusual proxy server architecture may need to seek additional resources.

2.8.1 Deciding which services IT want to provide

A proxy server can bring various capabilities to the Intranet. Understanding these capabilities will assist it in developing the deployment strategy. Netscape Proxy Server is designed to provide the following core capabilities:

1 Proxying.

The server provides access for internal clients, through a firewall, to the Internet. This service is often provided as part of a larger intranet security strategy and is known as "forward proxying." Forward proxying allows the clients to go outside the firewall without compromising the integrity of the private network. A server can also provide access for external clients, through a firewall, to internal content. This service is often used for secure Web publishing and is known as "reverse proxying."

2 Caching.

The server can cache web content locally, conserving bandwidth at network bottlenecks by storing frequently requested content locally. This content can be downloaded and regularly checked for changes in order to return up-to-date documents to all requests.

3 Filtering and Access Control.

The server provides fine-grain access control to web content from the intranet. Network administrators can use filters to block access to any Internet URL or to alter the actual content stream. Using an access control list, filters can be applied to specific addresses, groups of addresses, individual users or groups of users.

4 Logging.

The server records all errors and accesses for reporting purposes. Logs provide useful information to network administrators and group managers. Network administrators often analyze log files to monitor server usage and performance. Group managers find log-based reports useful in tracking Internet usage among their employees.

It may find that the organization has a particular need for one or more of these services. Perhaps it will fully exploit all of them. In any case, the deployment strategy it chooses and the ultimate proxy configuration should facilitate the services it will use most often.

2.8.2 Determining where to deploy the server

The most common place to deploy a proxy server is at a network bottleneck. Bottlenecks are often created by slow connections at network gateways. Managing bandwidth at these locations is imperative as the business grows and network traffic continues to increase. The Internet gateway and the branch office connection are two

1. INTERNET GATEWAY - FORWARD PROXY

Placing one or more proxy servers at the Internet gateway is the most common deployment scenario for the enterprise. In this location, Netscape Proxy Server provides gateway services at the application level with a web proxy as well as at the circuit level through SOCKS. The benefit of this type of deployment is enhanced Internet access. Web content caching reduces response times, facilitates bandwidth conservation, and helps reduce the overall communications expense. In addition, content filtering and access control allow it to manage the material on the intranet.

A variety of architectures can be used to deploy proxy servers at the Internet gateway. These implementations of Netscape Proxy Server will be discussed in the section on architecture alternatives below.

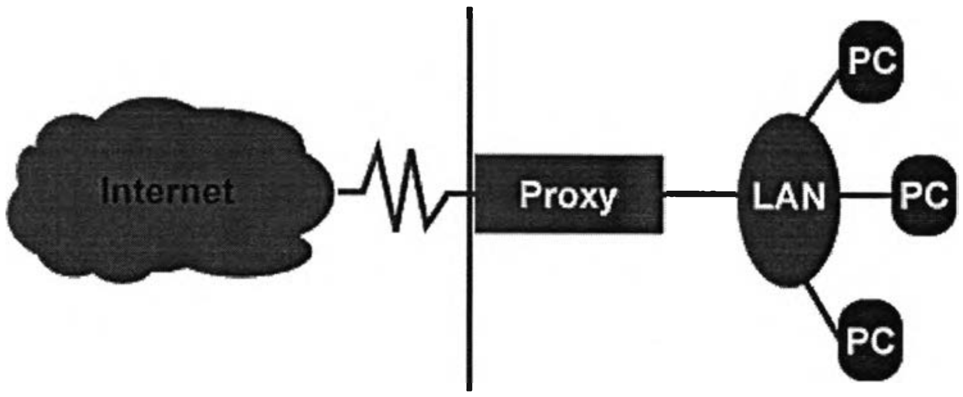

Figure 2.8 Forward Proxy Server Deployed at the Internet Gateway, Inside the Firewall

2. BRANCH OFFICE - FORWARD PROXY

Corporations are deploying proxy servers in increasing numbers on their intranets, both in remote locations and on major subnetworks. Proxy servers deployed at major subnetwork connections can drastically reduce the traffic on the corporate backbone. At remote offices, which are often connected via slow links to the corporate network, proxy servers can provide a quick mechanism for replicating content, providing better company integration, and increasing network performance - all of which can be achieved without large capital and communications expense. Outside the United States, proxy servers offer even more savings potential because of the great expense of communications bandwidth overseas.

Many organizations are seeing the value of deploying proxy servers throughout their Intranet. Types of deployments that use multiple servers can take advantage of the proxy routing capabilities of Netscape Proxy Server. Proxy routing allows it to chain proxies together to create a hierarchical caching system that can better serve the various organizations within the enterprise. Proxy chaining allows multiple Netscape Proxy Servers to cache content locally, setting up a hierarchy of servers for client access. The result is a managed network of proxy servers that is completely transparent to the user. In a typical implementation, smaller, local proxies might be situated near end user communities, with larger proxies near the firewall and external connections. For most installations, two levels of hierarchy is optimum, but it may benefit from adding more levels, depending on the size of the organization and where the bottlenecks occur on the network.

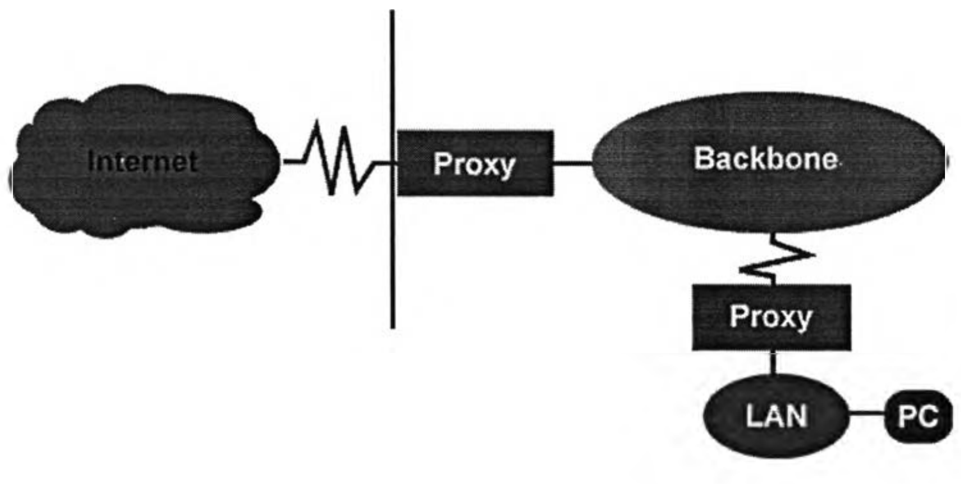

Figure 2.9 Forward Proxy Servers Deployed at the Internet Gateway and at a Remote Location

3. INTERNET GATEWAY - REVERSE PROXY

Reverse proxying is a special deployment case in which a proxy server is placed outside the firewall to represent a content server to external clients. This type of deployment allows it to expose selected content without exposing the web servers that host it or other elements of the private network.

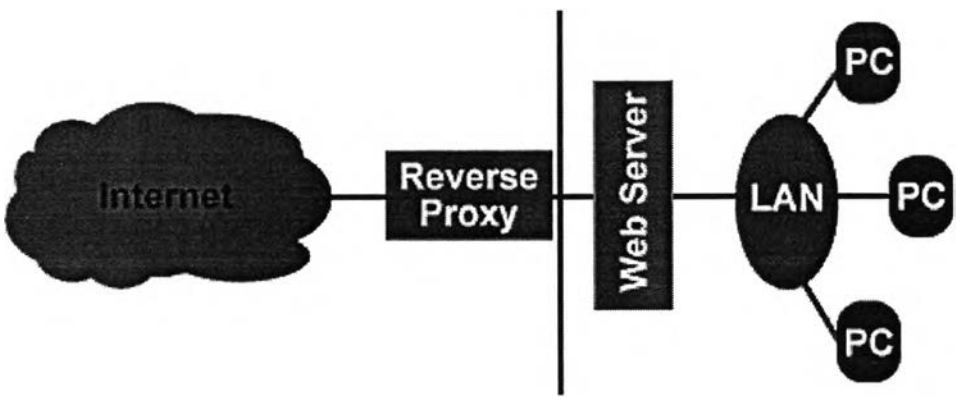

Figure 2.10 Reverse Proxy Server Deployed at the Internet Gateway, Outside the Firewall

In reverse proxy mode, the proxy server functions more like a web server with respect to the clients it services. Unlike internal clients, external clients are not preconfigured to access the proxy server. Instead, the site URL routes the client to the proxy as if it were a web server. Replicated content is delivered from the proxy cache to the external client without exposing the origin server or the private network residing safely behind the firewall. Multiple reverse proxy servers can be used to balance the load on an over-taxed web server in much the same way.

Reverse proxy servers are commonly used for secure web publishing. Having a proxy server accepting and filling outside requests allows it to keep the web server behind the firewall. It can then use the web server as a protected web site, staging documents for testing before they are published externally. When it are ready, it can publish selected content to the reverse proxy server's cache.

2.8.3 Deciding which architecture to be used

The Netscape Proxy Server can be deployed independently or in conjunction with a firewall. Or it can be used with a firewall at the Internet gateway but independently at a branch office. The Proxy Server by itself does not constitute a firewall and does not eliminate the need for one.

Because proxy servers are often deployed as part of a firewall solution, this section discusses the various firewall architectures it may use at the site. The special case of reverse proxy servers and their implementation are also addressed.

The proxy server can be deployed in conjunction with almost any firewall architecture. However, the firewall architecture it choose may affect the way it implement the proxy server. Three common firewall architectures are described below, but there are many variations. All architectures use some combination of proxy servers, firewall software, and hardware routers.

1 DUAL-HOMED HOST ARCHITECTURE

A dual-homed host is a computer that has two network interfaces, one connected to an internal LAN and the other to the Internet. As a firewall architecture, the dual-homed host usually incorporates a firewall software package. This firewall basically acts as a software router providing secure connectivity through packet filtering. The proxy deployed in conjunction with a packaged firewall on a dual-homed host provides a complete firewall solution. In addition to caching, Netscape Proxy Server brings fine-grain filtering and virus scanning to the solution.

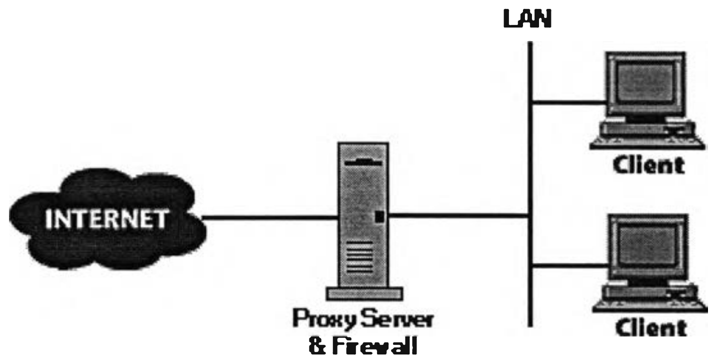

Figure 2.11 Proxy Server Implemented With a Dual-Homed Host Firewall

One drawback to this solution is that a security breach on the single host machine could jeopardize the whole network. For this reason, many security experts recommend firewall solutions made up of multiple redundant components. Still, à dual-homed host solution might appeal to small offices on a budget or organizations that do not require redundant security measures.

2 SCREENED HOSTS

A screened host consists of a router deployed in front of a server that is hosted on a private network. This router can be a traditional hardware router or a firewall software application providing packet-filtering capabilities and restricting inbound access to the internal network.

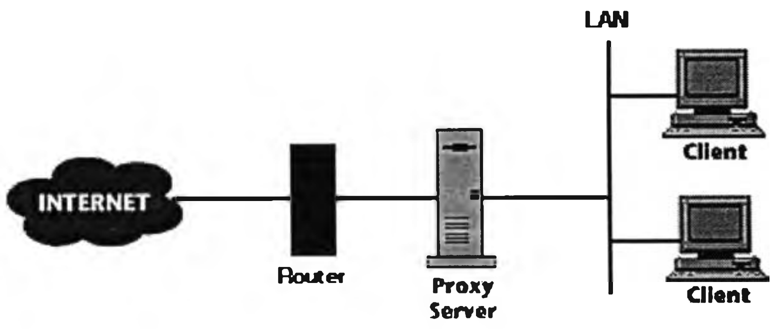

Figure 2.12 Proxy Server Implemented Behind a Screening Router

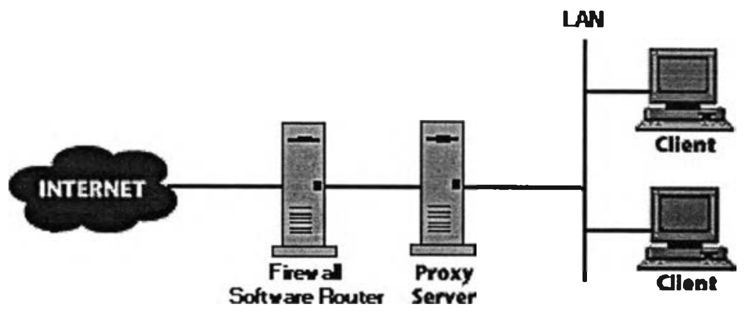

Figure 2.13 Proxy Server Implemented Behind a Screening Firewall

Proxying allows network traffic to gain Internet access through the router. A screening router could also support multiple hosts such as multiple proxy servers or web servers.

One drawback to the screened host deployment is a loss of security should the router fail. This scenario has encouraged the use of multiple routers and the screened subnet architecture. The screened host architecture is appropriate for small to medium-size intranets that require a simple, yet effective security solution.

3 SCREENED SUBNETWORK

A screened subnetwork consists of multiple routers sandwiching a nonsecure network that is outside or part of the firewall solution. This subnetwork is commonly referred to as a DMZ (demilitarized zone). In this scenario, the proxy is deployed in the DMZ and is allowed access to both internal and external networks through the routers. Both internal and external traffic can enter the DMZ but neither can pass through without the assistance of the proxy server and the packet filtering routers.

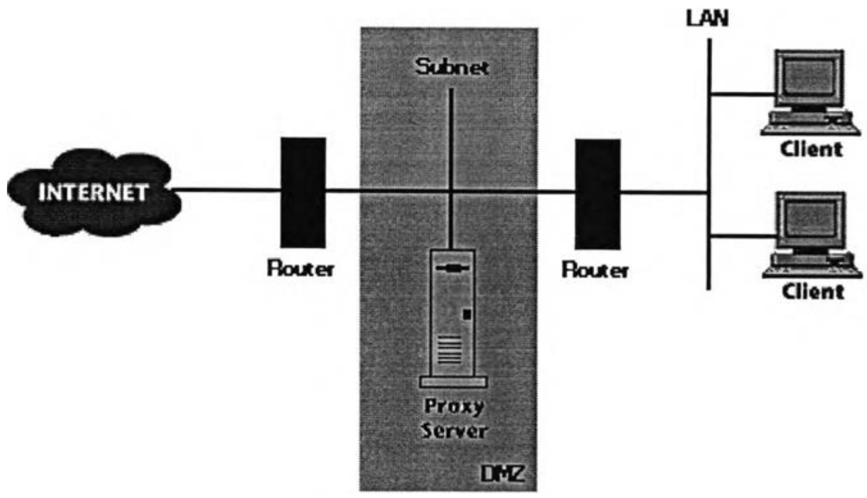

Figure 2.14 Proxy Server Implemented in a DMZ Between Two Screening Routers

The screened subnetwork is a popular architecture choice for larger organizations with heavily trafficked gateways. For these customers, security is critical and therefore redundancy is imperative. The protected subnetwork also provides an ideal location for other servers that must interface with the secure network and the Internet.

4 REVERSE PROXY

Independent of the firewall architecture, it may want to implement some type of reverse proxy. Reverse proxies are generally deployed in one of two configurations: alone, as a server stand-in; or in groups, for load balancing.

1. Server Stand-in.

In the server stand-in mode, the proxy receives requests for a web server that is protected behind the firewall. Server stand-in facilitates secure web publishing because it allows content on the web server to reside inside the firewall for protection.

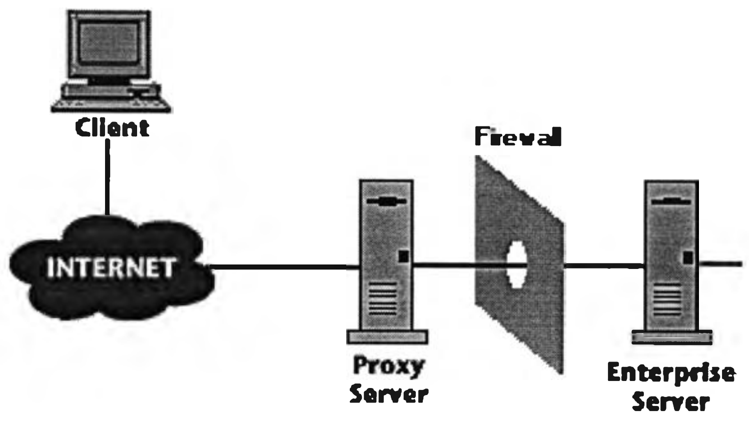

Figure 2.15 Proxy Server Implemented in Reverse Mode as a Stand-in for a Web Server

Server stand-in prevents direct, unmonitored access of internal resources from outside the enterprise. In its stand-in role, the proxy server acts like a virtual server mirror. The proxy is positioned similarly to a web server, and is usually placed in the DMZ or on an external subnetwork. As a server mirror, the proxy server provides replication only. The contents of the secure server will be replicated or mirrored in the proxy server cache.

2. Load Balancing.

Multiple reverse proxy servers can be use to balance the load on an overtaxed Web server. In this configuration, DNS round-robin is used to route incoming requests to one of a bank of servers. Load balancing helps the host machine handle high-volume requests while reducing the impact on overall performance.

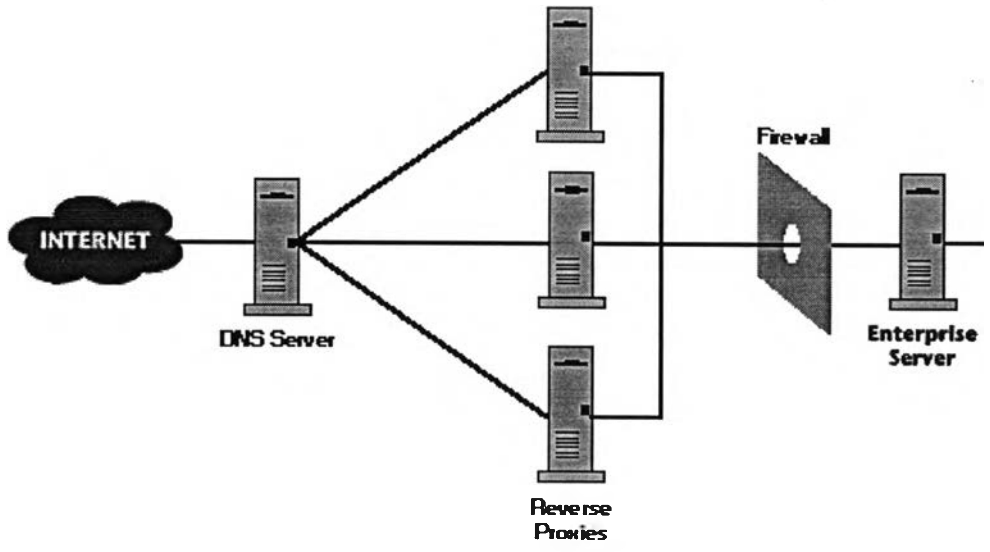

Figure 2.16 Multiple Proxy Servers Implemented in Reverse Mode to Balance the Load on a Web Server

2.8.4 Determining how many servers it will need

It may have already assessed where the bottlenecks occur on the network. The number of bottlenecks is a good starting point for the number of proxy servers it will want to deploy. Below is a diagram of one possible enterprise implementation. Proxy servers have been deployed at the network bottlenecks.

For each bottleneck location it will need to decide whether to deploy a single proxy or multiple proxy servers. This decision will depend on the user load and requirements for redundancy. While a single proxy may be easier to maintain and manage, multiple proxies provide greater reliability and maximize the use of the available bandwidth.

Assuming these are standard bottlenecks, we recommend deploying a Netscape Proxy Server at any site where ten or more people connect to the intranet through a WAN. In this situation, a proxy server easily pays for itself with reduced network traffic and increased user performance. And, if the WAN slows down or stops, the regional site has a local copy of the latest content.

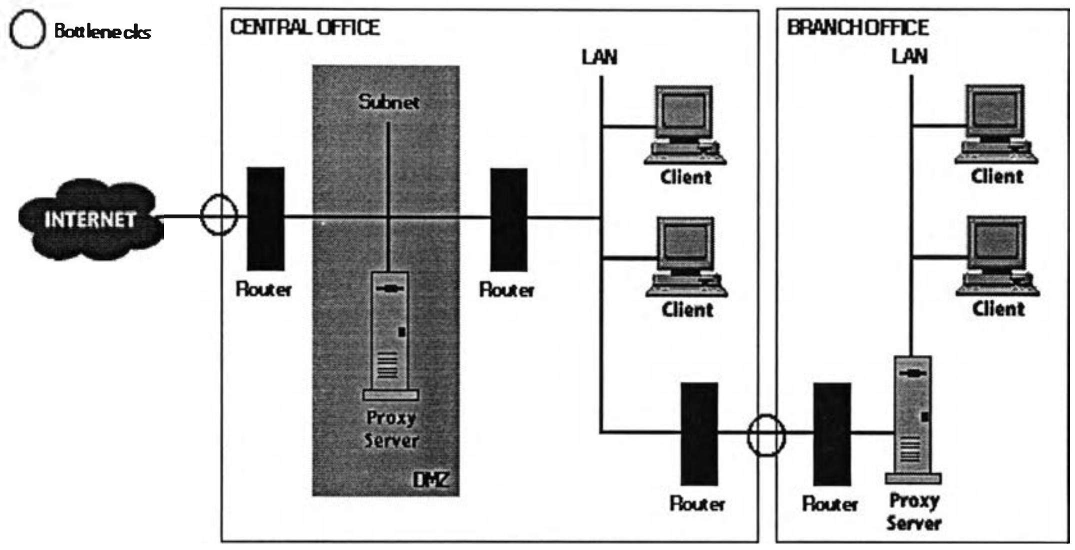

Figure 2.17 Proxy Servers Implemented at Common Network Bottlenecks

1 LOAD BALANCING AND FAIL-OVER

If proxy availability is critical to the enterprise, it may want to consider deploying multiple proxies to provide fail-over capability. In the diagram below, three proxy servers are deployed at the Internet gateway to balance the load of internal client requests. In this situation, Proxies 1 and 2 might share the load under normal conditions and Proxy 3 might be kept in reserve should one of the other proxies go offline.

In addition to load balancing between the client and the proxy server, it can also balance the load between proxy servers in a hierarchical chain. Load balancing between Proxies 1, 2, or 3 and Proxy A would be accomplished through an NSAPI plug-in disucssed in the configuration section of this guide.

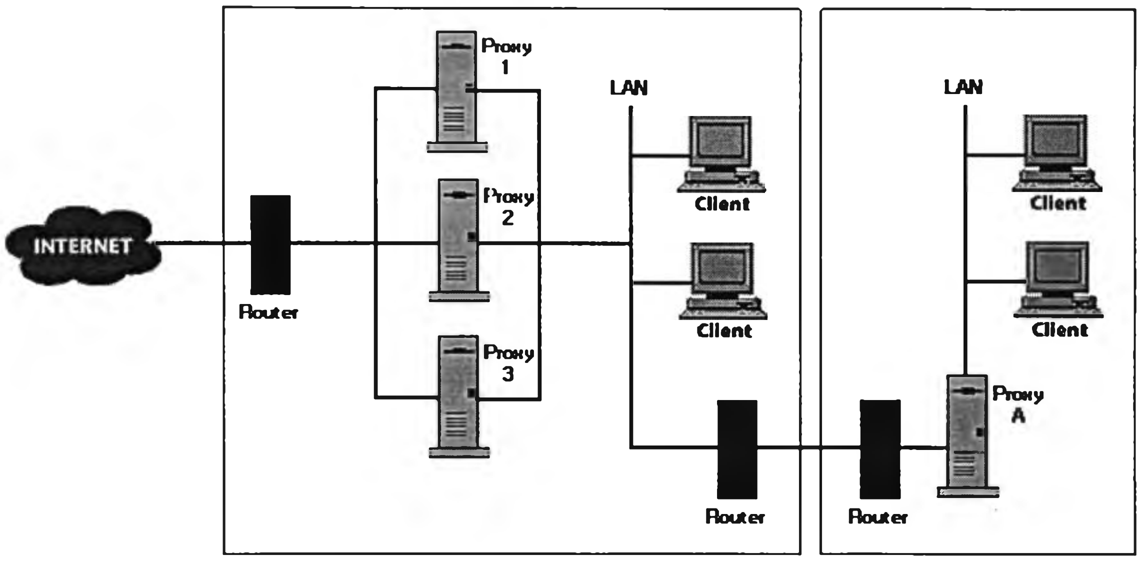

Figure 2.18 Chained Proxy Servers Providing Load Balancing and Fail-Over Capabilities

be

2.8.5 Deciding what type of hardware to use d

While planning the number of proxy servers to deploy, it need to anticipate growth and consider how the proxy services will scale. The number and type of users it support and the bandwidth available to them will greatly affect the ability to meet the growing demand.

1 CAPACITY PLANNING

Knowing the total number of users it need to support is important; however, scaling of the proxy server is really dependent on the number of users active at any given time. Each concurrent user is represented by an individual HTTP transaction. In most installations, the concurrent user count is much smaller than the total user base.

It also need to consider the type of use the proxy server will see. A full-time web surfer can generate thousands of requests per day. On the other hand, someone who uses the web less frequently may only generate a few hundred requests in the same period. It need to consider the types of users in the organization and how they will use the proxy services.

I 18633092

Users request different types of content as well. The average size of a web object is somewhere between 10K and 20K; however, the organization may have a much higher average if the majority of the downloads contain rich graphics. The increase in network traffic associated with larger objects must be accounted for when determining capacity requirements.

2 ESTIMATING LOAD

Whatever type of users in the organization has and whatever type of content they access, the proxy server will need to handle the load. The highest load may occur at a few peak times. For most businesses, the peak times for web traffic are between 10:30 and 11:30 in the morning and between 1:30 and 4:00 in the afternoon. The proxy server must be sized to accommodate the peak load, but it need not calculate the peak load to do so. It just need to estimate the average load for a given day. Use the following steps to estimate the load and the bandwidth it will need to accommodate it:

1. Record the number of requests for a normal business day, N, and divide this number by the length of that business day in seconds, T. The result, Atotal, is the average number of accesses per second.

$$\text{Atotal} = \text{N/T} \tag{2.1}$$

2. Experience has shown that it can roughly size the peak capacity by looking at the total accesses per second, Atotal, and multiplying by 2. The resulting number, Cpeak, will be the capacity required to handle the accesses at peak times expressed in requests per second.

$$\text{Cpeak} = 2 \text{ x Atotal} \tag{2.2}$$

3. Once it know the number of accesses it must accommodate, it can estimate the necessary bandwidth for the corresponding data that will be transferred. Assuming a typical web object size, such as 15K, it can determine the bandwidth required between the proxy and clients, Bclient.

$$\text{Bclient} = \text{Cpeak x } 15 \tag{2.3}$$

4. Requests that are serviced by the cache require less bandwidth than those that must go to the origin server. To account for this difference, it must include a factor that assumes a cache hit rate appropriate for the network and user base. Typical cache

hit rates are between 30 and 60 percent. A factor, F, of 7 or 4 assumes a 30 or 60 percent hit rate, respectively. By assuming a factor to adjust the data transfer rate, it can estimate the amount of bandwidth required between the proxy and origin servers, Bserver.

$$\text{Bserver} = \text{Bclient} \times F \tag{2.4}$$

5. Once it knows the bandwidth required on either side of the proxy server, it know the total bandwidth required, Btotal.

$$\text{Btotal} = \text{Bserver} + \text{Bclient} \tag{2.5}$$

or

$$\text{Btotal} = \text{Cpeak} \times 15\,(1 + F) \tag{2.6}$$

6. Next it should examine the uplink bandwidth utilization, since this is the ultimate limiting factor. The available uplink bandwidth will depend on the type of Internet connection it have. Various network analysis tools will allow it to see how much bandwidth it are using. From this information it can determine the percentage of available bandwidth and take this into account in the capacity planning.

Reverse proxy loads may be more difficult to predict than those for the traditional forward proxy server. Since it cannot know with certainty the number of external users, it must rely on an average usage profile. It can get a good idea of the potential reverse proxy load by analyzing the load on the web server currently hosting the content it plan to cache. Gathering this type of information is discussed in the monitoring section of this guide.

3 HARDWARE SIZING

An estimate of the required capacity will help it size the hardware. After deployment, it will be able to tune the proxy server to optimize performance, but in the meantime, it must determine an appropriate hardware configuration. Netscape Proxy Server will provide the best performance when run on a dedicated machine. If at all possible, consider that implementation.

Ideally, hardware sizing is based on the number of incoming connections and the average transaction time of those connections; however, most deployments start with an entry-level or typical hardware setup, such as those outlined below [7].

Table 2.3 The Hardware sizing

Variables	**Entry-level Proxy Server**	**Typical Proxy Server**
Users	Up to 1500	1500 to 3000
Operating System	Entry to mid-level Unix or NT server	High-end NT or Unix Server
CPU	120MHz or greater	1 or 2 Pentium Pro processors, UltraSPARC, or R5000
RAM	Minimum 32MB, 64MB to 128MB for Heavy traffic	128MB to 256MB
Server Hard Disk	Minimum 15MB; 100MB recommended	200MB
Cache	2GB to 4GB	5GB to 9GB

The speed of the CPU it choose is important, but not as important as AM and disk size. The CPU is normally not a bottleneck for server-grade machines; however, proxy performance does scale with more or faster CPUs on lower end hardware.

In the Table 2.4, suggests a minimum amount of RAM for the proxy server, but it will generally need more RAM as the user base expands. The following table suggests RAM sizes based on the number of users accessing the proxy server. Large deployments should also consider a logging file system or nonvolatile RAM to allow the server to perform asynchronous writes to the cache and greatly improve performance in high-traffic environments.

For a Unix system, each process uses about 200K of RAM for listening and 300K to 500K for working. Estimate approximately 700MB in total per process or concurrent user. (As mentioned before, the number of concurrent users will be much less than the total number of users.) It is critical that it have enough actual RAM to hold all the processes in memory when they are active.

Table 2.4 A minimum amount of RAM for the Proxy Server

Users	RAM (MB)
0 to 300	32
300 to 500	64
500 to 1000	96
More than 1000	128

Typical cache sizes, may range from 1MB to 20MB per user. An estimate of 10MB is a good place to start. After deployment, continue to monitor the cache performance, watching for increases in the cache hit ratio. It can do this using tools such as sitemon on Unix. It should keep increasing the cache size as long as the cache hits ratio continues to increase.

There is a tradeoff in selecting the type of disk for the cache. Consider spreading the cache across multiple disks whenever possible. One 10GB disk will store as much content as ten 1GB disks. However, the 10GB disk, while less expensive to purchase and maintain, will not perform as well as the 1GB disks. In general, multiple disks will perform better than a single disk, and multiple disk controllers will always be faster than a single controller.

2.8.6 Configuring the servers

Once it has used the Netscape Proxy Server, it will need to install the software and begin to configure the services. In this section it will find some general comments on configuring Proxy Server and some tips to make the process run smoothly.

1 AUTOMATIC CLIENT CONFIGURATION

To manage the proxy deployment efficiently, it should enable automatic proxy configuration in Navigator clients on the Intranet [7]. Client configuration is administered by a Proxy Automatic Configuration (PAC) file, which is downloaded from the server at restart. The PAC file allows it to specify which proxy server, if any, Navigator uses when accessing various URLs. This allows it to do load balancing across multiple proxy servers and to alter the proxy architecture without modifying end user settings. When it add additional proxy servers, it can even specify that they share the same URL. When one proxy server is down, the backup will respond.

2 CACHING

Proper cache setup is critical to the performance of Proxy Server. The most important rule to remember when laying out the proxy cache is to distribute the load. Caches should be set up with approximately 1GB per partition and should be spread across multiple disks and multiple disk controllers. This type of arrangement will provide faster file creation and retrieval than is possible with a single, larger cache.

The Cache Batch Update feature in Proxy Server allows it to proactively download content from a specified web site or perform scheduled up-to-date checks on documents already in the cache. This gives it the ability to cache content in large quantities at times when traffic on the server is low. Use batch updates to download the most commonly accessed sites at the end of each business day for quick access the following morning. It can use the log files to help determine which sites are frequently accessed. Refer to the Administrator's Guide for in-depth instructions on creating batch update configurations.

3 SOCKS

While the web proxy server provides caching and filtering capabilities suitable for web protocols, SOCKS provides a tunneling mechanism for protocols that cannot be proxied or for which is there is no benefit from proxying. SOCKS is firewall software that establishes a connection from inside a firewall to the outside when a direct connection would otherwise be prevented by the security measures. SOCKS is a circuit-level proxy and is indifferent to the protocols it serves at the application level. For this reason, an application-level proxy server is often configured to use SOCKS for protocols it does not support. Refer to the Administrator's Guide for instructions on configuring SOCKS.

4 TEMPLATES

Proxy Server can use templates to assign unique procedures to specific URLs. It can make the server behave differently depending on the URL the client tries to retrieve. It can also configure different cache refresh settings based on the file type.

The template is just an object that is created in the proxy server's object configuration file, obj.conf. Templates allow it to customize how Proxy Server interacts with clients by allowing it to do such things as name a set of directives for later reference, simplify complex configurations, or associate named objects with URI patterns.

Experience has shown that many proxy services do not require templates, especially in the early stages of deployment. However, templates become very helpful when it is necessary to edit multiple objects on a regular basis. Check out a sample object configuration file. The template "josh" at the end of this file is called by the "default" object in order to enable certain cache settings.

Instead of making the change several times in each object, the administrator can set up a template that is called by multiple objects and edit the template only once. The payoff increases as the obj.conf file becomes more complex and contains objects with a high degree of commonality.

5 FILTERING

Netscape Proxy Server allows it to filter URLs as well as content. URL filters are applied to requested URLs to determine whether they meet a predetermined set of criteria. URLs can either be allowed or denied based on this criteria. URL filters can also be used in conjunction with access control lists (ACLs) to filter the content that is requested by each client. Filtering with ACLs gives it the ability to restrict or allow access to selected users and groups in the organization. Several URL filters provided by third parties are supported by Netscape Proxy Server.

Content filters allow it to actually scan and modify the content stream. Virus scanning and HTML tag filtering are achieved through content filtering. Keep in mind that content filtering occurs out of process and can therefore significantly affect performance when applied to large volumes of data. For example, virus scanning places a significant additional CPU load on the server hardware, since the proxy server must compare all incoming data against the known virus patterns.

Refer to our list of Netscape Proxy Server Partners for more information on extending the filtering capabilities of the proxy server. To create filters and ACLs using the administration server, refer to the Administrator's Guide.

6 SERVER PLUG-IN FUNCTIONS

It can create plug-in functions to extend the capabilities of the proxy server by using the Netscape Server Plug-in Application Programming nterface, NSAPI. The server plug-in API is a set of functions and header files that will help it create functions to use with the directives in the server configuration files.

Using the NSAPI directive classes, it can override server functionality, add to it, or create the own custom functions. For example, it could create functions that use a custom database for access control or create custom log files with special entries.

2.8.7 Tuning the servers

As it operates the proxy server and the organization continues to grow, it will probably need to tune the proxy server to get the optimum performance for the particular implementation. Below are a number of tuning tips to help it. Netscape Proxy Server also provides online forms that can help it tune many of the settings that affect the server's performance.

1 TIME-OUTS

There are two time-out settings that significantly affect the performance of the proxy server. These time-outs are the proxy time-out ("timeout") and the time-out after interrupt ("timeout-2"), which is particular to Unix proxy servers. Here are some tips to help it use these time-outs correctly:

timeout. This time-out is the proxy time-out and tells the server how long to wait before aborting an idle connection. A long time-out commits a valuable proxy process to a potentially dead client, whereas a time-out that is too short will abort CGI scripts that take a long time to produce their results, such as a database query gateway. For these reasons, we suggest a time-out of 2 to 5 minutes, with an absolute maximum of one hour. To determine the best proxy time-out for the server, consider whether the proxy will be handling many database queries or CGI scripts or whether it will be handling a small amount of requests. In the latter case, it may opt for a higher proxy time-out value because it are less process constrained.

timeout-2. This time-out is the time-out after interrupt and is used only on Unix platforms. When a client has aborted a transaction while the proxy is writing a cache file, this time-out allows the proxy to continue writing the cache file; timeout-2 is the idle time-out for a connection in this state. The highest recommended value for this time-out is 5 minutes.

2 UP-TO-DATE CHECKS

The proxy server performs cache up-to-date checks to determine whether requested content in the cache is still valid or needs to be refreshed. It can tune proxy server performance by controlling the number of up-to-date checks. Under the

Caching tab in the administration server, specify that documents are not always checked. Choose a reasonable lifetime, somewhere between 8 and 24 hours, that balances caching with the need for fresh data. The value it chooses translation to the longest time the proxy server will wait before performing an up-to-date check. The proxy will only check on documents that have not been checked within the specified lifetime.

3 LAST-MODIFIED FACTOR

A last-modified header is returned from the server with the time the document was last modified. The document is updated based on its freshness and the last-modified (LM) factor. The LM factor is a floating point number that is multiplied by the age of the document since its last modification. The effect here is that recently changed documents are checked more often than old documents. A recommended value for the LM Factor is between 0.1 and 0.2.

4 DNS LOOKUPS

Domain Name Service (DNS) is the system used to associate standard IP addresses with host names. The proxy server can use forward DNS lookups to resolve an origin server name to an IP address and reverse DNS lookups to resolve client station addresses to names. Excessive DNS lookups can affect the performance of the proxy server and should be avoided. In addition, the load on the DNS servers and their location on the network can also affect performance. Here are some things it can do to avoid a performance hit:

Enable DNS Caching. On the NT platform, DNS caching and negative caching are always enabled.

Log Only Client IP Addresses. The proxy server has the ability to log client host names; however, it will see better performance if it can get along without it. Set the log preferences to log client IP addresses only.

Disable Reverse DNS. If it will not be logging client host names, it can disable reverse DNS. Avoid ACLs With Client Host Names. Use client IP addresses instead, if possible.

5 NUMBER OF PROCESSES

On the Unix platform, the administrator must specify the number of processes that will be preforked on the server. Preforking enhances the performance of the proxy server because the number of processes limits the concurrent requests the proxy can handle. The following table can provide a starting point in determining the number of processes as shown in Table 2.5

Table 2.5 The number of Process Determining

Users	Processes	Memory (MB)	Swap (MB)
0 to 300	32	10	64
300 to 500	64	20	128
500 to 1000	96	30	192
More than 1000	128	40	256

While it can estimate the number of processes it need, the proxy server should be properly scaled to meet the load in the peak periods to minimize delays. Typically, if it need to tune the server, the current process allocation is insufficient. The number of processes in use on sitemon will typically register 100 percent, but this does not show it the number of clients it want - those that are queued by the operating system.

It can estimate the number of clients in waiting by using netstat. Read the system's manual page for netstat, and determine the correct command line options to list all TCP sessions. From this snapshot, count all connections to port 8080, or the designated proxy port, that are in TCP states between SYN_RECVD and CLOSE_WAIT, including ESTABLISHED. (The system may vary slightly. The count it now have should be a snapshot of accepted connections, those that have been established, plus any that have been queued by the system.

6 HTTP KEEP-ALIVE

The proxy server supports HTTP keep-alive packets in order to provide improved performance on some systems. However, the majority of customers will see better performance with keep-alive connections turned off; this is the default setting. Experience has shown that the benefit gained from keeping a connection open for a single client does not justify the penalty placed on subsequent requests from other clients. An open connection effectively ties up a process even if it is idle. Unless it have a small user base requesting primarily noncacheable content, the net effect on the proxy server of using keep-alives will be a performance decrease.

2.9 Proxy Advantages

This section discusses how Netscape Proxy Server satisfies the requirements discussed in the Evaluation Criteria section.

SCALABLE AND FLEXIBLE CACHING

A) FLEXIBILITY

Netscape Proxy Server provides administrators with a tremendous amount of control to ensure efficient caching. This makes Proxy Server useful for reducing network traffic and user response times in a wide variety of network configurations and for common content types.

1 Caching Common Content

Proxy Server caches the web content types that make up the vast majority of traffic on the Internet, including:

- Hypertext Transfer Protocol (HTTP)
- File Transfer Protocol (FTP)
- Gopher
- "Pushed" content. Because Netscape Netcaster is based on HTTP, Proxy Server can reduce network traffic that is produced by this content "push" service.

2 Caching Controls

Caching on-demand is the ability to cache frequently accessed documents based on user requests in order to conserve network bandwidth and reduce network response times for users. This is the default caching mode for Proxy Server.

Proxy Server provides administrators with the ability to optimize caching for their needs. For example:

- **Cache refresh policy ("time-to-live")**. This feature enables the administrator to control the length of time the Proxy Server waits before performing an up-to-date check on requested documents. If it store information that rarely changes, this setting might be set to a higher number. If the data changes freqently, it'll want documents to be checked more frequently.

- **Cache expiration policy**. This feature lets Proxy Server determine how often an HTTP document should be checked to see if it is up-to-date. One option is to use expiration information that is included by the document publisher, but because this information is rarely provided, Proxy Server can use the time an HTTP document was last changed (along with a last-modified factor to estimate how long the document is likely to remain unchanged.

- **Protected documents**. Proxy Server caches documents that have access controls associated with them on the remote web server, but continues to check access rights on the remote server for subsequent requests for the document. For example, when a user initially requests a document, if Proxy Server detects that there are access controls associated with the document on the remote server, it will require the user to authenticate to the remote server before fetching the document. Once the document has been fetched, Proxy Server will cache the document and return it to the user. For subsequent requests for the document, Proxy Server will continue to require users to authenticate to the remote server, while returning the document from the cache. Organizations can thereby gain the performance benefit of having documents cached locally, while preserving the security of their access controls.

- **Caching queries**. Proxy Server can cache queries for HTTP documents, and administrators can limit the size of queries to be cached. Longer queries are more specific, and therefore less likely to be repeated, so it may not be useful to cache them.

- **Cache file size limits**. This feature enables administrators to specify upper and lower file-size limits.

- **Hit counting**. Proxy Server can count how many times a given document was accessed from the proxy cache between up-to-date checks, and then send that count back to the remote server the next time the document is refreshed. This way, the remote server gets a more accurate count of how many times a document was accessed.

- **Cache local hosts**. In many cases, it may not make sense to cache documents from web servers located on the same local network as the Proxy Server, because it will generally be more efficient to request the documents directly from the web servers. Proxy Server gives administrators the flexibility to control caching of documents from local hosts.

Caching on-command gives administrators the ability to specify batch updates to the cache. Batch updates in Proxy Server include both of the following capabilities:

- **Preloading documents or sites into the cache**. Administrators specify a URL that Proxy Server will recursively pull into the cache.

- **Automatic refresh of documents that already reside in the cache**. Administrators can select one or more documents from the cache to be updated.

In addition, batch updates can be scheduled to occur at regular intervals. So if an organization knows that content will be periodically updated and would like to make the updates available to employees on the intranet, the administrator can schedule batch updates to the Proxy Server cache to occur during off-peak hours to ensure that the content is fresh when users request it and avoid tying up the network during normal business hours.

Batch updates enable the administrator to intelligently pre-select content to be cached. Proxy Server's extensive transaction logs can be used by the administrator to ensure that content which is accessed by users on a recurring basis is actually worthwhile before configuring the batch update to occur. In addition, organizations do not need to wait for usage patterns to develop over time before proactive caching can occur. If the administrator knows that a new document of interest to many users will be published, batch updates can be configured in advance.

B) SCALABILITY

Massive scalability increases the likelihood that documents will be returned from the cache and ensures high cache hit rates.

1 Hierarchical Caching

Using proxy routing (or chaining), it is possible to create multitiered networks of Proxy Servers to enhance performance and reduce congestion across internal networks. Proxy Server can be deployed at any network bottleneck to provide the benefits of caching as the organization grows. Administrators can specify that traffic be routed to another Proxy Server or to a SOCKS server at the Internet gateway.

2 Client IP address forwarding.

Normally Proxy Server doesn't send the client's IP address to remote servers when requesting documents. Instead, Proxy Server acts as the client and sends its IP address to the remote server. This behavior enhances the security of the network by screening internal addresses.

However, there are times when the administrator might want to pass on the client's IP address if the Proxy Server is one of a chain of internal proxies. Proxy Server provides optional client IP address forwarding for these situations.

2.10 Web cache Performance

The metrics is commonly used when evaluating Web caching policies. These include the following [6]

a) Hit rate – The hit rate is a ratio of documents obtained through using the cache size mechanism versus the total documents requested. High hit rate is a better performance measurement.

b) Bandwidth Utilization – A reduction in the amount of bandwidth that is consumed shows the cache.

c) Response time/access time – The response time is the time that take user to get a document.

There are various parameters such as user access patterns, cache removal policy, cache size and document size that can significant affect cache performance. The performance of Web caching is not satisfactory in network. Typically, averaging hit rates is below 55%.

The paper of title "MODEL-DRIVEN SIMULATION OF WORLD-WIDE-WEB CACHE POLICIES" was studied by Ying Shi, Edward Watson and Ye-sho Chen [6] performs cache algorithms and utilizes actual empirical data to develop cache management strategies as the following result;

The study believes that there are similar between word usage in text and Web access on the Internet. Simon's Model described that technique for estimating α in a paper. The parameter α is the probability of a new entry. Thus the value of α is approximately equal to R/T, where R is the total number of different documents and T is the total number of utilization. The ability to approximate the value of α from R/T is an intuitive relationship. It is know that α is the probability of the entry of a new document. Therefore, the total number of document accesses should be the product of T and α. Simon's model is a robust dynamic model of Web user access. It integrates frequency and recency into a simple mathematical model.

It estimates α (R =2457, T=329385, and α = R/T = 0.0072594) and Columbia data (R=509, T= 156926, and α = 0.003244). To estimate γ calculate areas of Pareto curves for different combinations of α and γ. Then a linear regression equation is obtained as Area = $-0.64622 * \alpha + 0.341996 * \gamma + 0.107573$ [6]

The results for Cache size

Broadly speaking, larger caches reflect to getting higher hit rates. This common knowledge is reflected in the graphs. Moreover it is very important to understand that some cache size threshold, cache performance improves marginal adding more cache size. Also, the threshold varies with different removal policies and different α and γ values. The threshold is more obvious as γ gets smaller [6]

The determination for a break point can save cache size at a slight decreasing hit rate. This is very important for cache resources of Proxy cache.

Broadly speaking, larger caches get higher hit rates. This common knowledge is reflected in the graph [6]. However, it is very important to understand that beyond some cache size threshold, cache performance improves only marginally by adding more cache space. This phenomenon can be clearly observed from graph. Also, the threshold varies with different removal policies and different α and γ values. The threshold is more obvious as γ gets smaller. The determination of the break point can save cache space in spite of a slight decrease in hit rate. This is very important for scarce cache resources in a proxy cache [6].

2.10.1 CONCLUSIONS

Through the simulation of Simon's model and the experimental simulation of Web cache policies, our conclusions can be summarized. Simon's model is a robust dynamic model of Web user access patterns. It integrates frequency and recency into a simple mathematical model [6]. Model-driven simulation of Web cache policies has advantages over the trace-driven simulations most often used in research studies. Model-driven simulation allows a much broader range of access patterns for experimental purposes and allows us to link user access patterns with the performance of the Web cache policies.