2002

This is to certify that the

dissertation entitled

High Performance, Scalable Web Server Systems

presented by

Wenting Tang

has been accepted towards fulfillment
of the requirements for

Doctor of ~~Philosophy~~ degree in ~~Computer~~ Science & Eng.

Major professor

Date September 30, 2001

High Performance, Scalable Web Server Systems

By

Wenting Tang

A Dissertation

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

Department of Computer Science and Engineering

2001

Abstract

High Performance, Scalable Web Server Systems

By

Wenting Tang

The web has experienced phenomenal growth during the last couple of years. During the commercialization process of the Internet, the web has become a "killer app" and millions of browsers now access comparatively fewer web sites compared to the number of clients. As more applications are built around the web architecture, web sites have to deal with potentially unlimited number of users. Peak rates for a web service might be as high as 10 times of the average. Therefore, the performance and scalability issues of such big sites will be an important issue.

In this dissertation, we address problems related to the development of *high performance, scalable web server systems.* We make contributions in two aspects of web server performance: The first related to the performance of a *single web server* and the second related to the performance of *multiple replicated web sites.* For a single web server, *Browser Initiated Pushing* (BIP) is proposed to improve performance based on the observation that today's typical web page has one or more embedded images.

Measurement shows that BIP is an important technology to improve a single web server's throughput.

We propose two approaches and develope a framework to address the scalability of replicated web sites.

A non-dispatcher approach, *Static Allocation and Client Redirection* (*SSCR*) shares load between replicated web sites and is mainly targeted for global scale web sites.

Smart Server Selection (S3) addresses the server selection problem when multiple replicated sites exist. In S3, a client DNS server is extended to prioritize a pool of IP addresses based on the routing metric information collected from routers and other information it collects (geographical location of servers and clients). An efficient scheme to collect routing-metric information from routers is proposed.

A framework to support *Content-Aware* request distribution in STREAMS-based TCP/IP implementation is developed and prototyped. Content-Aware request distribution provides the ability to support partial replication, flexible web site arrangements, Web Quality of Service, and security. Our framework is based on the TCP handoff mechanism. The TCP handoff mechanism is designed as STREAMS module(s) in the protocol stack. Three different designs are reported according to workload characteristics. The differentiated web service in the STREAMS-based TCP/IP implementation is discussed.

ACKNOWLEDGMENTS

First, I would like to take this opportunity to thank my academic advisor, professor Matt W. Mutka. During my PhD study at Michigan State University, professor Mutka showed his care and his support both in academia and everyday life. He spent enomorous hours to discuss problems with me. He always provided the insight on the problems and encourages me when I was frustrated. I also would like to thank Dr. Mutka for his great support of my internship at HP Labs. I am grateful that professor Mutka corrected my grammar and spelling errors in several papers and this thesis so patiently and nicely.

I would like to thank my committee members, Dr. Esfahanian, Dr. Ni and Dr. Stapleton, for their helpful suggestions and insights in the research problems and all they did for me during my period of study.

I am indebted a lot to my colleagues in HP Labs, Ludmila Cherkasova, Magnus Karlsson, Todd and others, for their support and encouragement.

I want to express my great gratitude to my wife, Yujuan Cheng. Yujuan sacrified all her available time to do housework and showed her unselfish support for my study. She has contributed a great deal of her time to raise my two healthy kids so I may focus on my research.

Thanks also to my parents and my grand parents who supported me during my academic years financially and spiritually. Without their support, I could not finish my studies.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The World Wide Web (WWW) has experienced phenomenal growth, both in terms of the number of users and the amount of the information available. It has been reported that, currently, the world wide web has become the killer application of the Internet and 75% of traffic on the Internet backbone is Web (HTTP) traffic [1]. Currently, the web has evolved into the major platform for information delivery and retrieval, distance learning, application service. More and more people depend on the WWW to access information, purchase products, etc. Companies and organizations depend on the WWW for fulfillment of daily activities to save costs and maintain a lead over the competition.

As more users access the world wide web, we expect that more companies and organizations will move their daily activities onto the web. This trend further triggers more people to access the web to obtain information and to conduct business activities. While the information available through the web is enormous and millions of users access the Internet, many issues should be addressed to meet the user's expectations:

delivering the *right information at the right time to the right people at the desirable place.* Much effort has been put into improving web users' web browsing experience. Such efforts include but are not limited to:

1. performance improvements, which deliver a document as quickly as possible to a web user when it is requested. This information includes static web pages, dynamic web pages, audio and video streams.

2. web search engines and document ranking, which crawl through web pages to provide the web documents a user is looking for and to rank the returned documents in the order of relevance.

3. web information personalization, which customizes and organizes the information according to web user profiles, to provide a better web tour experience.

4. wireless access, enabling web users to access the WWW through wireless devices.

5. security, which provides solutions to make sure that information is only available to the right people at the proper time.

In this dissertation, we focus on the performance of web systems, which deliver the document as quick as possible to a web user. Technically speaking, a web delivery system can be divided into three subsystems: *web client systems*, *intermediate web systems* and *web server systems*, primarily according to the management boundaries as shown in Figure 1.1. The web server system refers to the subsystem that is related to web services and maintained by the service provider side. The intermediate web system refers to the services and equipment specifically for web traffic and primarily

maintained and managed by the ISP (Internet Service Provider). The web client system refers to the system where the web related services are managed by client side. Such a classification facilitates the discussion of the technologies.

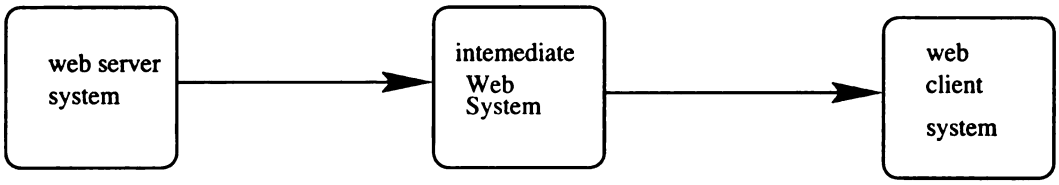

Figure 1.1: Subsystems of web system

Generally, a web server system and an intermediate web system tend to be bottlenecks because many users access the system simultaneously. Client web systems seldom become the bottleneck for the following two reasons:

1. the desktop computer is powerful. Due to the advances in microprocessors and computer systems, system performance of desktops doubles every 18 months.

2. the desktop has much less communication and computation compared to web servers and intermediate networks in terms of number of users.

There is a hot debate whether the network system or the end system is the bottleneck. However, network speed has moved to Gigabit speeds in the local area network, and switching technology greatly improves the bandwidth available between two nodes. While the Internet backbone moves to optical networking, we expect that more bandwidth will become available. Under this assumption, we put our focus on the technology which can improve the performance of the web server systems.

The web server system is the system that delivers the content to the clients. The core of the system is *web server* software, which runs on the machine named *web server*. A web server accepts HTTP requests and serves documents to web clients. As more activities are performed through the web server system, a web server must process many requests in a short time period. The capacity of the web server system must be increased to process and deliver these requests in a timely manner as the number of users accessing the web site increases. For those popular and big sites, it is increasingly difficult to deliver the contents to the end users.

Two trends further exacerbate the problem of the web server system:

1. Web document content become more complex. In order to make the web page more illustrative and informative, an organization's web page includes several embedded contents, typically images. E-commerce web sites generally include clips (images) for products. Such an image-rich web page leads to several more HTTP requests on the web server, compared to a text-only version.

2. People would like to access the information no matter When, Who, Where (WWW). This leads to the popularity of web portals such as Yahoo. The Yahoo web site receives about 4.5 billion *page* visits each month and has multiple replicated web sites on each continent. Inside the organization, in order to maintain the information integrity and availability and to facilitate web content management, web content tends to be served and managed centrally.

In this dissertation, we identify four problems that are important for the development of *high performance and scalable web server systems* to meet web users' expectations. Solutions have been developed for these problems.

1.1 Identified Problems

1.1.1 Efficient Delivery of image-rich web pages

Reduction of the download latency of web documents is very important to web users and is especially important to web content providers. Most web users will not spend much time browsing a web site that is slow to download. Experience indicates that people browsing the web are quick to click "stop" to terminate a slow download and access another similar site. Reducing download latency is important to encourage a web user to continue to browse at the current server.

A new category of applications has emerged that use the web browser as the graphical user interface (GUI) to the application service so that the user does not need to purchase the stand-alone application. One such application is TurboTax, provided by the Intuit company. Essentially the web page appears as the GUI for the traditional tax preparation application. In order to make these GUI pages illustrative, intuitive, and very similar to the traditional application GUI, greater numbers of small images are used as menus, buttons, and logos. A page may easily have more than 30 embedded images. As E-commerce become pervasive on the Internet, e-catalogs also include

large numbers of images. A typical entry in an e-catalog includes a small image that shows the product and illustrates the description of the product.

Under current implementations, a browser sends a separate request for each embedded image. By embedding larger numbers of images in a page, the situation becomes a potential performance bottleneck. The reasons are:

1. End-users experience longer download latency. Since the images are menus and buttons, the end users might not do anything until all the images are downloaded. Epediting image downloading becomes more important.

2. The embedded image requests place greater pressure on server resources. In order to service each request, the web server has a fixed overhead. Network level overhead includes receiving the HTTP request and sending a reply, even if the messages are small. Depending on the implementation of the web server, embedded image requests introduce different amounts of overhead. For example, a thread-based web server implementation creates a thread to parse the URI, service the request and destroy the thread. As the number of embedded images increase, the overhead introduced by each HTML page increases.

An efficient mechanism is desirable to deliver such a web page as quickly as possible to reduce the web server's load and reduce the download latency.

1.1.2 Load distribution for replicated web servers

As the number of accesses to a site increases, a site might not be able to serve all the requests by a single machine. Since most web requests are short-lived and may

be independently processed, replicated web servers provide cost-effective way to scale up such a large web server system. In a web cluster that all the web servers are connected by very high-speed networks (called a *Web Farm*), each web server serves all the documents available on the web site. Since each web server in the farm is only responsible for a fraction of the requests to the web site, reasonable response times may be achieved. In order to provide more capacity of such a system, a web farm may be replicated on the global scale, taking advantage of the network proximity. That is, the web server farm strategically replicates pages so that the web client may access the same content on the nearest servers to reduce network latency and the load on the central server. Load distribution and server selection are two important problems to be addressed in such a system in order to ensure better performance. One fundamental issue is how the load is shared between multiple replicated sites effectively and efficiently in order to provide reasonable performance. An approach that considers the network proximity and server load needs to be developed.

1.1.3 Server Selection for Global Replicated Web Server Systems

The increase in traffic at popular web sites has been tremendous. As we mentioned before, Yahoo has about 4.5 billions page requests per month. If the image requests are included, the number would be much higher. Since a single computer cannot meet the requirement of such a high request rate, replicated servers are widely used. For example, *Yahoo* (*www.yahoo.com*) and *Altavista* (*www.altavista.com*) have replicated

server groups in each continent. With multiple replicated servers running simultaneously, the number of requests serviced by each server is reduced and a reasonable response time may be achieved. The replication of servers increases fault-tolerance, which is essential to these popular sites. By putting replicated servers at appropriate places, replicated servers also increase network proximity and thus significantly reduce access latency perceived by users.

One of the fundamental problems for a global-scale replicated service is how the best server is selected based on the user's preference and location.

1.1.4 Differentiated / predictive Web Quality of Services

As the web becomes the platform to deliver information and conduct business, requirements from different users to access the information might be different. For example, some online stock trading users are willing to pay more money to receive real time quotes of stock prices while other users would rather check the stock price on a best effort basis for a lower price. In the first case, an online user would like to receive some predictable response time for a web request. This is commonly referred as WebQoS. As the Internet progresses toward Internet QoS (Quality of Service), end-to-end WebQoS service, which combines network QoS and end system QoS, becomes possible. Since replicated web servers are widely used to address capacity needs for serving a large number of users, how can WebQoS be efficiently supported on a web cluster?

1.2 Our Contribution

We address problems and provide solutions to the forementioned problems. Our contribution includes the following four parts:

1. Intelligent Browser Initiated Server Pushing (BIP)

   BIP addresses the first problem to improve the performance of web servers which serve web pages with much embedded content. In the BIP approach, the HTML page and embedded images will be pushed to the client in one round trip time. Three approaches are proposed to addresses the image-sharing problem between pages in the BIP context, and the performance of these approaches is evaluated.

2. Smart Server Selection (S3)

   S3 addresses the problem of server selection when multiple global-replicated web sites are available. We propose a mechanism to prioritize IP addresses by network metrics and also propose an efficient way to collect routing metrics from Internet routers.

3. Load Distribution with Static Scheduling and Client Redirection (SSCR)

   SSCR is a proposed way to address the wide area server load distribution problem. The number of accesses from each network are predicted and accesses are statically assigned to the preferred server; temporary overload is addressed by client redirection.

4. A Framework for Content-Aware Request Distribution and Processing

Content-Aware request distribution is an important technology to improve cluster throughput and enable QoS support in the web clusters. In this work, we designed modular approach to support the content-aware request distribution and content-aware request processing. Modular design enables such technologies to be deployed much quicker in commercial operating systems, most of them have STREAMS-based TCP/IP implementation.

1.3 Structure of the Content

The remainder of this document is structured as follows. In Chapter 2, related work and background information is presented. In Chapter 3, browser initiated server pushing (BIP) is proposed and evaluated in terms of web server performance. Chapter 4 describes S3, an approach to address the server selection problem when multiple replicated web server sites exist on the global scale. In Chapter 5, SSCR, an approach proposed to deal with the load distribution issues for replicated web sites, is presented. Content-Aware request distribution greatly improves cluster scalability in terms of throughput. Three TCP handoff designs which are targeted for different workloads and architectures are presented in chapter 6. Rare-TCP Handoff design has been prototyped and some performance evaluation is also reported. Web QoS and differentiated service is discussed in chapter 7. Kernel and user level mechanisms and designs in FreeBSD and STREAMS environment to support web differentiated service are presented. End-to-end QoS and its relationship with an emerging technol-

ogy, Content Delivery Networks are also discussed. A summary is given in chapter 8.

Possible future work is outlined in chapter 9.

Chapter 2

Background and Related Work

Much work has been done in order to improve web system performance. As we mentioned in the Chapter 1, the web delivery mechanism may be divided logically into three parts: the web server system, the intermediate web system, and the web client system. Figure 2.1 shows a more detailed picture of a possible path a web request follows. All the proxies shown in the figure are optional.

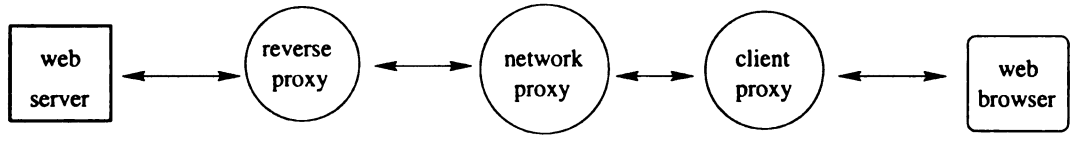

Figure 2.1: Components along the path of a HTTP request

In this chapter, we first discuss the related work on latency reduction techniques, especially web proxies. Next we discuss some of the related work on selection algorithms for replicated web servers. After that, related work and background on load

distribution of the replicated web servers are presented. Lastly, we briefly present some of the existing work on differentiated web services.

2.1 Latency Reduction

Latency reduction techniques currently proposed include proxying, prefetching, and web server pushing.

2.1.1 Web Server Pushing

In this strategy, the web server pushes documents to a place along the network path near the client. When a client requests those documents, the network latency will be reduced. Bandwidth may be saved because the client will no longer need to access the original server.

Web hosting and *web co-location* are two easy ways to achieve this goal. Web hosting refers to an organization or a person that provides web content on its ISP and the ISP manages the web servers and hosts the content to serve web clients. Web co-location refers to an organization that moves its own web servers and equipment to its ISP. In these two cases, web traffic from web users will go to the web servers directly without entering the organization's network and the link between the organization and its ISP. Generally, the link connecting the organization to the ISP has relatively limited bandwidth. Such an approach saves the amount of WAN traffic, and also reduces the download latency.

Some companies, such as Akamai[2], Inktomi[3] and Digital Island[4], recently started to build *Content Delivery Network* (CDN), a web pushing infrastructure. Companies may push their web documents to these CDNs, and the CDNs are responsible for shipping the documents around the Internet to meet the client access patterns. Since a CDN ships the documents to a place much nearer to the client, the client access latency is reduced and the load on original web server is reduced as well.

Some work has been done in this area. The WebOS [5] project at Berkeley proposes a "rent-a-server" idea by which a server can be temporarily rented when needed to serve client requests. Research conducted at Boston University [6] uses traces to investigate where to push the documents from the server's point of view. RaDaR[7] describes a scalable architecture for web hosting service providers when the documents may be pushed to caches near the clients so that bandwidth may be saved.

2.1.2 Client Prefetching

In a client prefetching strategy, the client (either proxy, or browser) prefetches the document before the user makes the request. The main idea of prefetching is that it tries to overlap the download time with the user thinking time. The proxy may prefetch some popular files, and the browser may prefetch some of the documents according to user's access history. Client prefetching does not reduce the load on the original server.

Mogul[8] evaluates the performance of client prefetching. Several researchers from Boston University[9] discuss the effect of prefetching on network traffic patterns and

propose an algorithm to smooth it. Prefetching also depends on accurate prediction of the user's next move. It has been reported that the accuracy of prediction of next access from a client can be as high as 85%[10].

2.1.3 Proxy

Web access patterns demonstrate temporal locality and spatial locality. Temporal locality refers to the fact that the same user tends to revisit the same page with certain period of time. Spatial locality refers to the fact that different users behind the same proxy tend to access the same document.

A proxy is a cache device that takes advantage of the temporal locality and spatial locality of web accesses. It caches the most frequent accessed files in the cache. A proxy sits between the client and web server as shown in Figure 2.1. The client first establishes the TCP connection with the proxy, then the proxy checks the URL requested. If the requested URL is in the cache, the document is served from cache. Otherwise, the proxy will establish a connection to the original web server, get the document and relay the response to the client It also might cache the document according to a *caching policy.* It has been reported that the proxy hit rate is about 50%[11].

Benefits

Since the proxy may supply the document locally, the response time will be much shorter. A client proxy also reduces the amount of WAN traffic to its ISP. In addition to the performance improvement, a proxy offers other benefits. These benefits include:

1. *Security.* Since a proxy is the only point directly connected to the Internet for an organization, only authorized users may access the external world from an intranet. Also, only authorized users may access the intranet from outside.

2. *Privacy.* Since only the proxy's IP address is publicized, the internal network information and IP address will remain anonymous.

3. *IP address saving.* The network behind the proxy may use private IP addresses and accesses the Internet through a proxy.

Different management domains use proxies for their own purposes. A proxy may be classified into three categories according to the management domains: client proxy, network proxy, and reverse proxy (server side proxy). A client proxy sits in the client network domain and is managed by the client site. A reverse proxy is the proxy that sits at the front of the web servers at the server domain. Network proxies are managed by ISPs. Among of the purposes of the network proxy are:

- To reduce network traffic inside the ISP and traffic to other ISPs.

- Ease of management. While it is possible for each web browser to set up a client proxy in a client domain, it is inconvenient for ISPs, which serve thousands of users. In the network proxy case, the client does not know of the presence of the proxy at all. When the web traffic comes into the ISP network, an edge router will automatically forward the web traffic to the nearby proxy (cache engine). The network proxy supplies the document to the client if it is in the cache, and

relays the requests to the original server otherwise. Because it is transparent to the clients, sometimes it is referred as *transparent proxies.*

Cache Replacement Policy

When a proxy is full, and a new document must be cached, the proxy must decide which document is to be evicted from the current cache in order to cache the new document. This is referred as a *Cache Replacement Policy.* Work reported in [12, 13, 11] are some of recent results regard to this topic. It has been reported that LRU is not the best cache replacement policy and many algorithms are proposed and evaluated.

Cache Document Consistency

When a proxy caches a document and later serves the document locally, it is possible that the proxy returns a stale document to the client. Currently a Time-To-Live (TTL) scheme is used to ensure consistency. The document may be cached by any proxy at most for the period of TTL. It has been shown that a stronger consistency model based on update is possible and the overhead is comparable to the current scheme[14]. An approach that proposes a piggyback scheme to carry document invalidation information within the current response has also been reported[15].

Scalability: Cooperative Proxies

Since the proxy is the only gateway for the web traffic, it must relay web requests from the client to the web server and relay responses from the web server to the client. For a

large client site, a single proxy server can easily become a bottleneck. Proxy clusters, where a number of proxies are interconnected by high speed networks, provide the necessary scalability and availability. In such a configuration, it is desirable that the contents of different caches in the cluster be *shared* or different caches in the cache cluster cache a *disjoint* set of documents. The first case is referred as *signature-based* cooperative proxies. The second case is referred as *partition-based cooperative proxies.*

1. *Signature-based cooperative proxies.* In this configuration, each web proxy may independently cache any documents. However, when a proxy does not find a requested document in its cache, it will try to get the document from neighboring proxies before sending a request to the original server. In such a configuration, a mechanism is needed to exchange what is in each proxy's cache. In [16], a cache sharing protocol based on multicast is proposed. When a proxy can not find a document in its cache, the proxy sends a multicast message and sends the request to the proxy which has the document. The Internet Cache Procotol (ICP) is based on the multicast approach. It has been reported [17] that multicast packet exchange overhead is too large and a new protocol is proposed. In the new protocol, each proxy keeps the information about which document is cached in other cooperative proxies, called a signature. The proxy will send updates if enough changes have been made in its cache contents.

2. *Partition-based cooperative proxies.* In this configuration, each proxy only cache a subset of documents, and at any given time, a document will only be cached by one proxy. Since there is no replication in cached documents, the total

available cache size to cache unique document will be large and the cache hit ratio will be improved. CARP[18] and Consistent Hash[19] are two examples in this direction. In these approaches, when a client accesses a document, the client must know which proxy must be accessed. A mechanism must be introduced to address this problem.

2.2 Server Selection

The problem server selection tries to address is that when multiple replicated servers exist on the global scale, which server is the best in term of expected performance? A number of methods for the server selection problem have been proposed. They can be divided into two categories depending on where the server selection is performed: the client side or the server side. In client side approaches, client applications/network services generally collect information from networks and replicated servers, and make a selection. In server side approaches, server side network services gather information and make a decision.

There are many client side approaches proposed. They differ in where and how the information is collected. SmartClient [20], the probabilistic model proposed in [21] and automatic selection [22] provide application-level implementations to collect the metrics and make server selection. The SONAR service [23] is a special server that prioritizes a list of IP addresses for a client according to the information it collects. Cisco DistributedDirector [24] is a server side approach. The advantage of approaches such as DistributedDirector is that these do not need to make changes at the client

network; all work is done by the service provider. The disadvantage is that the service provider must purchase and maintain special devices. Also, such an approach generally suffers a linear increment in cost as the number of replicated servers increase. In the long run, such a problem (finding nearby replicated servers) should be addressed at the client side, which will make support of global replicated services much easier for the content providers. Some researchers address different aspects of the problem: what kind of metrics are independent and what is the correlation of existing metrics. Metrics include network hops, AS hops, network latency, and application level latency. Examples include [25, 21]. Some network companies [26, 27, 28] propose approaches based on a pure network metric. A pure network metric server selection may be divided into two categoies: all the servers reside in the same AS, or servers are located in different AS.

All servers located in the same AS

If all the servers are located in the same AS, the solution provided by these companies are similar: a device is put in front of each server cluster distributed in wide area. In addition to balancing load for each local cluster, this device also has some routing functionalities. In such a configuration, all the web server clusters have the same virtual IP address. All the devices in front of clusters will advertise a route to this virtual IP address as a host route. To the client network router, it sees multiple routes to the same IP address (virtual IP address), each route with possibly different network metrics, and the router will automatically select the shortest path or some other metric. In this way, each client network automatically selects the nearest server.

All servers located in different AS

The above approaches do not apply for server clusters in different AS. The problem with different ASes is that the devices in front of each cluster cannot advertise the route to the same IP address any more, because different AS should not originate routes to the same IP address or network. In this case, some kind of variation of DNS server is used. Cisco Distributor is one of them.

2.3 Load Sharing for Replicated Web Servers

Load sharing is very important for high throughput of the clusters. A lot of research has been conducted both in academia and in industries. Basically all of the approaches fall in one of two categories: *front-end distribution* and *DNS-based distribution.* In the front-end distribution, a special device, named *front-end* or *Load Balancer* is introduced and is put at front of the web farms. The load balancer accepts all the inbound web traffic, and decides which web server (backend node) is going to serve the request. In DNS-based approaches, the server side DNS server will map the web site name to a list of IP addresses. The decision is made when the client DNS server queries the server side DNS server for name to IP address mapping. When a client DNS request arrives, the server DNS can decide the IP addresses and TTL value to return.

2.3.1 DNS based load distribution vs front-end load distribution

Both DNS based request distribution and front-end based request distribution have their advantages and disadvantages. The front end accepts all incoming traffic, so it has the following advantages:

1. better load balance.
2. better security.
3. easy management.

It has the following disadvantages:

1. possible bottleneck.
2. single point of failure.

A DNS based policy addresses the problems of the front-end based approaches, namely bottleneck and single point of failure. Since these approaches expose each web server's IP address to the clients, it has some disadvantages compared to front-end based approaches. For example, it is not easy to deal with server failure. In addition, some times it is not desirable to expose the web server's IP address directly to the clients.

2.3.2 Front-end request distribution

Bryhni *et al.* provides a good classification of front-end approaches. The front end approaches can be classified according to *decision mechanisms* and *delivery mechanisms.* Decision mechanisms are classified according to the protocol layer of the ISO

reference model in which the decision is made regarding which servers are to serve the request. The delivery mechanisms are classified according to the layer in which the server is uniquely identified in the cluster. Packets are delivered to the server from the frontend accordingly. With this classification, front-end approaches are classified into the following categories:

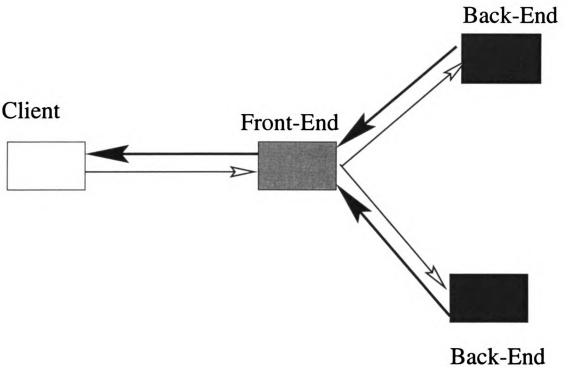

Figure 2.2: L4/L3: Frontend forwarding incoming and outgoing traffic

1. *NAT* (Network Address Translation) (L4/L3). In these approaches, each web server is identified by a unique IP address. The frontend has a *virtual* IP address. When the client establishes a TCP connection, it sends the connection request to the virtual IP address. The front-end makes the decision based on Layer 4 information, for example, IP address and port information. After the decision

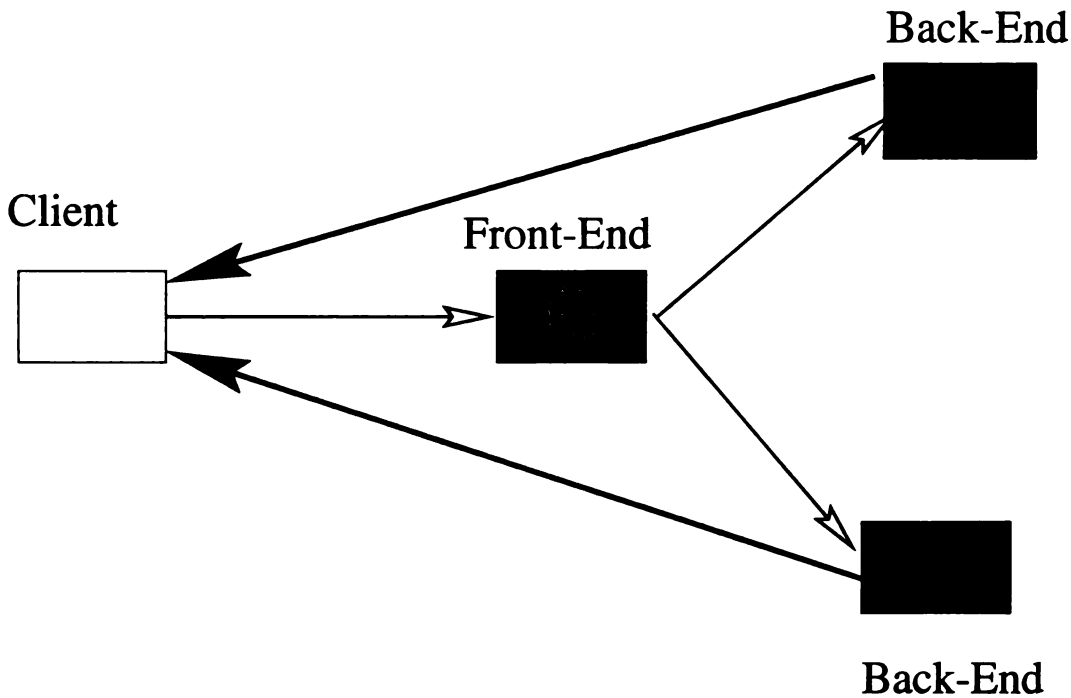

Figure 2.3: L4/L2: Response go directly from server to the client

is made, the flow (connection) is delivered to the correct server. The packets' IP address of that flow (connection) is changed to the selected server's IP address (L3) by the front-end and is forwarded accordingly. The outgoing packets' IP address of the flow is changed back by the front-end to the virtual IP address (Figure 2.2). Because these approaches introduce the IP address translation from virtual IP address to selected server's IP address back and forth, it is commonly referred as the Network Address Translation approach.

2. *NAT-Backend* (L4/L3). The above approach has considerable overhead because the front-end must update the IP address and checksum of each packet. In NAT-backend approaches, modification of the outgoing packets' IP address and

checksum is performed by the backend node, instead of the front end. Since the number of outgoing packets is larger than the incoming packets due to asymmetric web traffic, the load on the front-end should be reduced. Also, if there is another path to the client, the server may send traffic to the client directly without passing through the front-end. This improves the scalability of the front-end. Magic Router [29], Cisco Localdirector [30] and IBM Network Dispatcher [31] belong to this category.

3. *L4/L2.* In this approach, each web server in the cluster is aliased to the virtual IP address of the cluster. Each web server has a unique MAC address. When the client sends a connection request to the front-end, the front-end makes the decision and forwards the incoming packets to the selected server using its MAC address. The selected server responds to the client directly without passing through the front end. The IP address and checksum update is avoided. Therefore, the load on the frontend and backend are both reduced. The shortcoming of this approach is that since each web server is identified by MAC address, it has to be on the same LAN. The ONE-IP approach [32] belongs to this category(Figure 2.3).

4. *L7/L3.* Recently, the request distribution moved from L4 to L7 in order to address some of the problems of the L4 approach, for example, the inability to read the user's cookies. L7 load distribution also provides a number of benefits, such as partial replication, ease of management, imporved security, and overall throughput (efficient memory usage). Two mechanisms are employed to

implement L7 load distribution: *TCP handoff* and *TCP splicing.* The main complexity is that in order to check the URL to make the decision, the front-end must establish the connection with the client first. However, when the front-end establishes a connection with the selected server to serve the request, complicated manipulation is needed to maintain the necessary transparency and high throughput. Approaches from Resonate, Ipivot, Alteon and Arrowpoint[33, 28, 27, 34] provide solutions based on L7 load distribution. Academic efforts include [35, 36, 37].

2.3.3 DNS based distribution

DNS based approaches may be further divided into two categories:

1. *pure DNS round-robin.*

2. *DNS round-robin combined with redirection.*

Pure DNS round-robin based approaches schedule the Internet domain and influences the load on each web server by the way IP address and the TTL value of the name resolution is assigned. Approaches proposed in [38, 39] are such efforts. The DNS round-robin based approach is widely used because of its simplicity. It has been reported that DNS round-robin causes some imbalance in the server load and does not always provide satisfactory performance.

The algorithms of the second category come from two sources: DNS based policy supporters that tries to provide better performance and frontend approaches that try

to avoid the central point. The approach proposed in [40] belongs to the first source, while the approach proposed in [41] belongs to the second source.

2.4 Differentiated/predictive web QoS

Differentiated web QoS has received some attention in recent years. In the last two years, IETF groups and network vendors put much effort into providing differentiated network QoS on the Internet, and many standards and drafts has been proposed. For some applications, network delay is most troublesome. For example, VoIP and streaming media require a preferred treatment of such a flow. However, for web traffic, network delay is only one part of latency the web client experiences. The latency also depends on how fast the end system can deliver the information to the network. It is preferable to provide end-to-end QoS. When a client needs QoS, not only does the network provide preferable treatment to the traffic, but also the end system provides preferable treatment for the traffic as well.

There are two categories of differentiated Web QoS. One is *user-based*, the other is *request based*. Different sites can choose one of them or combine them together. User-based policy provides differentiated web services based on user identity. For a particular user, no matter which URL he requests, he always gets the predefined service. Web sites which have different groups of subscribed users might consider this option.

Request based policies provide differentiation according to which document the user is requesting. Some URLs have higher priority than others. For some online shopping

sites, for example, the checkout URL has the highest priority in order to fulfill the purchase transactions. Another application for request-based policy is *shared web hosting*, where a single computer hosts multiple virtual web sites and different sites would like to have different levels of service. Much research has been done in this area. In [42], the authors evaluate the possibility of supporting a request-based policy on top of a commodity operating system. WebQoS [43] is a commercial system available to support WebQoS. It supports both user-based and request-based differentiation policies. Researchers from HP Labs [44] discuss what the end user perceives as WebQoS and what that means for web site developers.

Chapter 3

Intelligent Browser Initiated Server Pushing (BIP)

As we mentioned in Chapter 1, web pages tend to include large number of embedded images for various reasons. These image requests create a burden on the web server resource and cause longer delays. Experience indicates that people browsing the web tend to click "stop" to terminate a slow download and access another similar site. Reducing download latency is one factor to encourage a web user to continue browsing at the current server. Presently, three popular techniques are used to decrease web download latency: *Web Proxies*, *web server pushing*,and *client prefetch*, as we described in Chapter 2.

All the existing approaches have their own drawbacks. The proxy approach is not helpful if the document is not in the cache. Furthermore, it is often unsatisfactory due to dynamic documents and other factors (administrative overhead, one-point bottleneck, etc). The web server pushing approach is not popular because infrastruc-

tures for receiving large amounts of documents and placing them closer to clients are just emerging [2, 3, 4]. These commerical infrastructures are not sufficiently flexible for most of the web sites due to inconvenience and high-cost. Techniques to decide whether it is appropriate to exploit these technologies are not available yet. Also it is not clear if these approaches can be applied in the intranet environment, where latency is mainly caused by the overloaded server and network latency plays only a limited role. For the client prefetching approach, it is very difficult to predict a user's next access.

Browser Initiated Pushing (BIP) addresses these problems. It tries to minimize the download latency upon a cache miss using a proxy or client prefetch. It may be used for dynamically generated HTML pages, such as active pages and CGI pages.[1] It also provides benefit to users even if no proxy is installed. It reduces the number of HTTP requests received by web server, which may help to reduce the web server's load.

Presently, most HTML pages include embedded content. Since an image is the most obvious embedded content, we focus our discussion and evaluation on images. For simplicity, we refer to those HTML pages with embedded images as I-HTMLs, and refer to those I-HTML pages with all embedded images on the same web server with the I-HTML page as L-HTMLs. Those I-HTMLs that have at least one embedded image that is not on the same server as the I-HTML page are referred to as R-HTMLs. Under existing approaches, two or more RTTs are required for such an I-HTML, depending on the browser implementation. BIP tries to expedite the download time

[1]Embedded links of dynamic HTML web pages may be generated as needed.

of such an I-HTML. Under BIP, such an I-HTML may be downloaded using one request. One benefit is that BIP reduces the download latency of L-HTML to one RTT.

Another benefit of BIP is that it improves server resource utilization. There are two reasons for this. First, it reduces the number of HTTP requests received by the web server by about 20% in our simulations. This will free network and processor resources to satisfy other requests. Second, it may reduce the connection-hold time by at least one RTT. This leads to the improvement of server connection management under HTTP 1.1.

BIP may be used to reduce the download latency without prefetching or proxies. It may also be used by proxies or prefetching schemes to expedite the download of I-HTMLs and may serve as a quick recovery mechanism upon a cache miss. Based upon a workload found in our department and college, potentially more than 60% of total successful HTML requests may benefit from BIP.

The rest of the chapter is organized as follows. We describe the motivation, mechanism, and benefits of BIP in Section 3.1. The inefficiencies of BIP and some enhanced approaches are presented in Section 3.2. In Section 3.3, we describe our simulation model and performance results. In Section 3.4, we show our benchmarking results. Implementation and possible limitations are given in Section 3.5. Finally, we summarize results and discuss possible future work in Section 3.6.

3.1 Motivation of BIP

As we mentioned before, BIP works only for I-htmls, which have embedded images. Before we describe the BIP mechanism, we need to answer the following two questions:

1. What percentage of html pages accessed are I-htmls ?

2. In those I-htmls, how many I-htmls can be downloaded in 1 RTT? That is what percentage of I-htmls are L-htmls?

In order to answer these two questions, we use two workloads. One is the CSE workload, which is about one-day's trace file from our department's web server. The other is the EGR workload, which is about one-day's trace file from our college web server.

In table 3.1, some statistics of the two workloads is presented. For embedded content, only images are considered. In our study, only static htmls are processed.

In the CSE workload, among the total of 13,433 successful html accesses, there are 4,828 non-I-htmls(36%) and 8,605 I-htmls (64%). Among these I-htmls, 7,949 (92%) I-htmls are L-htmls, which can be downloaded by 1 RTT. [2]. In the EGR workload, in the total of 13,910 html accesses, there are 4,366 non-I-htmls (31%) and 9,544 I-html pages (69%). Among these I-htmls, 9284(97%) I-htmls are L-htmls. In summary, we can see that there are 64% to 69% of all the requested html pages are I-htmls and more than 90% of those I-htmls can be downloaded in 1 RTT.

[2]BIP can not reduce latency of R-htmls as much as that of L-htmls because a browser has to make at least one new request to another web server. However, it does reduce the number of requests and downloads those local images in 1 RTT.

Another characteristics is that on average, there are about 5 embedded images in each I-html(weighted), obtained by total accesses pushed divided by total I-htmls. We will analyze the performance of BIP based on these characteristics.

Table 3.1: Characteristics of the Workload

Trace	*CSE Workload*	*EGR Workload*
Duration (in days)	1	1
Total number of accesses	116,636	90,095
total successful html accesses	13,433	13,910
total image accesses	89,352	64,883
total misslinks html	239	235
total non-I-htmls	4,828	4,366
total I-htmls	8,605	9,544
total L-htmls	7,949	9,284
total accesses pushed	42,767	44,170

3.1.1 How BIP works

In the BIP approach, the web server maintains the link structure of all the I-htmls it serves. For the static I-htmls, such information can be retrieved by an html editor or a special tool that runs periodically to update the link structure of all the updated I-htmls. For all the dynamic I-htmls, this can be parsed on the fly.

When a browser sends a request, it explicitly tells the web server that it allows embedded images to be pushed. After the server receives the request, it retrieves the link structure of the requested I-html and sends these images in addition to the I-html page. During this process, repeated images will be suppressed and pushed only once for each request. In the server response, the information of which embedded image(s) will be pushed for this request is provided. When the browser receives this response,

it parses the I-html as before. The browser will make requests to those embedded images which are not pushed.

The main advantage of browser initiated embedded content pushing instead of server automatic pushing is flexibility. For example, when a user turns off the automatic image download option in his browser, the browser does not initiate the push for this request. However, server side pushing does not have such flexibility. BIP also makes the browser have full control on the server's pushing behavior, which will prevent the ambitious server from flooding the browser with too many images. Furthermore, the browser can decide to turn off BIP due to too much overhead or it can selectively turn on BIP for some sites while turning it off for others.

3.1.2 Benefits of BIP

In this section, we analyze the performance benefits of BIP: *latency reduction* and *server resource utilization improvement.*

Latency Reduction

In this section, we use the number we obtained from our workload statistics to compare the download latency of an L-html page under three different configurations: HTTP 1.0, HTTP 1.1, BIP. The sample html is a L-html A, with 5 embedded images.

1. HTTP 1.0 (Without persistent connections)

Under HTTP 1.0 specification, the browser will establish a new connection for each request. If we assume at most 4 parallel connections can be established by the browser to the server at the same time, then for A, a browser will first ask for A. After receiving A, the browser will parse A and will find it must download 5 additional embedded images. It establishes 5 connections with 4 parallel connections each time. The browser will need 2 RTTs to download those embedded images. So in total the browser needs 3 RTTs to display html A with all the embedded images.

2. HTTP 1.1 (Persistent connections)

   In this senario, a browser first gets A and by parsing A, the browser will need to download 5 embedded images. The browser reuses this existing connection and sends 5 consecutive HTTP requests. Due to the pipeline of HTTP 1.1, the browser will download those embedded images in 1 RTT approximately. In total the browser needs 2 RTTs to display A with all the embedded images.

3. BIP with HTTP 1.0 / 1.1

   In this senario, a browser sends a HTTP request with the server push option. The server will supply all the embedded images along with A. After the browser parses A, it will find all the embedded images needed are pushed by the server, so it takes just 1 RTT to display A.

Server Resource Utilization Improvement

BIP improves the server resource utilization in two ways. First, it reduces the number of HTTP requests. Simulations show that it can reduce the number of HTTP requests by about 20%-26% in two workloads. Reducing the number of HTTP requests relieves the pressure on the CPU and the number of messages processed by the web server. So it improves the web server throughput.

Second, it helps to manage the connections under HTTP 1.1. The general pattern of web accesses is that a user makes a request, takes some time to read the page, makes another request and reads that page for a while, etc. Without BIP, the connection management is difficult because the requests for embedded images interfere with the requests asking for html pages. To further complicate this problem, the cache effect of the browser makes it difficult to predict the request arrival time for the next request to fetch those embedded images. These two factors make it more difficult for web server to predict the next access for an established connection. With BIP, these two problems are addressed. By web server pushing, it is insured that all the embedded images are pushed. Also, by filtering these embedded image requests, it makes predicting the next arrival time easier and more accurate. The web server connection management predictor only needs to predict the next html request arrival, so the predictor accuracy is improved. In [45], with the embedded images in the log, the connection hold time is reduced only by 25%. However, if the embedded images are filtered in the log, the connection hold time is reduced by 50%. Combining a highly precise html request predictor with BIP, we can expect a much better connection management scheme.

3.2 Enhanced BIP

BIP works fine if there are no shared embedded images between I-HTMLs or there is no client cache. However I-HTMLs from the same person or the same group generally share some embedded images. In this case, downloading the first I-HTML will cause the embedded images to be cached at the client cache. Downloading the second I-HTML does not need to download these embedded images again. This is a problem in our simple BIP scheme because the server does not know what is in the client cache. Simulations show that without considering the client's cache, the web server will push much more images than needed. This will waste system bandwidth (network bandwidth, web server throughput, and client throughput).

The main problem we must address is that we need to convey the contents of the client cache to the web server so that the web server does not push embedded images already within the client cache. The main obstacle is that too much system bandwidth would be consumed if a browser transmitted information about the contents of the client cache to the server directly, either by the full URL or the MD5 digest.

3.2.1 Three Approaches to the Client Cache Problem

1. BIP-Ref

   BIP-Ref relies on the HTTP referrer field[46]. When a browser sends a request, it contains the last URL this browser visited, which is the referrer field (the referrer URL is not necessary a URL from this web server). The referrer field has been implemented in Netscape Navigator and Microsoft Internet Explorer

browsers. The web server may exploit this information, and speculate on what might be in the client cache. When the web server receives a request from the client, the server will not push the embedded images of the referred HTML document.

2. BIP-Hist

   BIP-Hist maintains the browser's access history information, so that when a request comes, the web server will check the last N accessed pages and will not push embedded images that have been embedded in these pages. Because maintenance of access history information is quite expensive, the depth of the access history should be kept as small as possible.

3. BIP-Hash

   In BIP-Hash, the browser uses a Bloom filter to transmit information about its cache contents to the server in the request. Before the server pushes an embedded image, it will first check the Bloom filter to see if the embedded image is in the browser cache. If it does not, then the server will push the image. Otherwise, the server will not push the image to the browser.

BIP-Ref and BIP-Hist are rather straight-forward techniques. BIP-Hash merits further discussion of the Bloom filter mechanisms.

3.2.2 Bloom filters

A Bloom filter is a method for representing a set $V = \{\nu_1, \nu_2, \ldots, \nu_n\}$ of n elements (also called keys) to support membership queries. It was invented by Burton Bloom in 1970 [47] and was proposed recently for use in the proxy context by Cao and Fan [17] as a mechanism for efficiently identifying pages in cooperative proxies.

The idea of Bloom filter is to allocate a vector of m bits initialized to 0 (see Figure 3.1). The Bloom filter has k hash functions $h_1, h_2, \ldots, h_k$, each with a range {1,...,m }. For each element $\nu \in V$, the bits at positions $h_1(\nu), h_2(\nu), \ldots, h_k(\nu)$ are set to 1 (a bit may be set to 1 multiple times). Given an element μ, we check the bits $h_1(\mu), h_2(\mu), \ldots, h_k(\mu)$. If all of these bits are set to 1, then we assume μ is in the set. However, there is a possibility that we are wrong, which is called a "false positive".

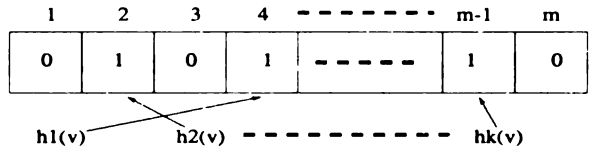

Figure 3.1: Bloom Filter

The "false positive" probability is given by the expression $(1 - (1 - 1/m)^{kn})^k \approx (1 - e^{\frac{-kn}{m}})^k$.

A nice feature of the Bloom filter is that there is a clear tradeoff between the probability of "false positive" and the length of vector m. For details about the Bloom filter, its tradeoff between m and the false positive probability, and possible hash functions, see [17].

Applying a Bloom filter in BIP-Hash saves overhead and makes it possible for a client to successfully transmit what is in the cache to the server with little extra overhead. If a browser transmits the full URL, each URL might need 100 bytes. If the browser transmits the MD5 signature, each embedded image in the cache needs 16 bytes. If we assume 99% Bloom filter accuracy, each embedded image in the cache needs 1.3 bytes.

3.2.3 Bloom filters and BIP-Hash

In the BIP-Hash approach, the Bloom filter is maintained by the browser on a per-site basis. The full URL is treated as the key in the Bloom filter. When a page is downloaded from a particular site, the browser updates the Bloom filter of that site to include those embedded images. Similarly, when an embedded image is removed from the cache, the Bloom filter of that site is updated accordingly. When a browser establishes a connection with the web server, it sends the Bloom filter with the HTTP request. Each subsequent HTTP request over the same connection will send only an update to the Bloom filter and the server will cache the old Bloom filter and update the Bloom filter on a per-connection basis.

Two nice features of Bloom filter make BIP-Hash a very promising approach. First, it will never push an embedded image existing in the client cache. This is because if the embedded image is in the client cache, all k bits of the filter are set and the web server will definitely know that the image is cached. This nice feature saves system bandwidth, except for the Bloom filter overhead itself.

A second nice feature is that BIP-Hash has a clear tradeoff between overhead and the number of L-HTMLs reducing latency. If the browser reduces the length m of the Bloom filter, due to a "false positive," some embedded images that should be pushed are actually not pushed. Therefore, the browser has to make requests for these embedded images and these I-HTMLs cannot save the download latency. On the other hand, if the browser increases the length m of the Bloom filter, the "false positive" rate decreases, and more I-HTMLs save download latency. However, the number of bytes of the filter that are sent over the network will increase. In practice, the browser and the web server may negotiate the value of m.

3.2.4 BIP-Hash and a Proxy

The above description of BIP-Hash works nicely for browsers directly connected to Internet. As we mentioned, BIPs may be used by a proxy as a recovery mechanism upon a cache miss. Under such a scenario, the proxy becomes a client from the server's point of view. BIP, BIP-Ref and BIP-Hist work fine. BIP-Hash depends on the number of cached embedded images from a particular web server. Since the relationship between the length of the Bloom filter and the number of embedded images cached is linear, caching of embedded images by a proxy means a larger Bloom filter is needed.

For browsers that are behind a proxy, the browser will send a request with a Bloom filter as normal. When the proxy receives this request from the browser, it will attach

its own Bloom filter to the request header and forward this request to the web server. The web server will check both Bloom filters before pushing an embedded image.

3.3 Performance Evaluation

We evaluate the performance of BIP, BIP-Ref, BIP-Hist, and BIP-Hash in this section. We simulate the performance of BIP, BIP-Ref and BIP-Hist by trace-driven simulation and show the performance of BIP-Hash by analysis. We also analyze memory requirement for maintaining the link structure information.

3.3.1 Performance Metrics

BIP performs well if we only consider the RTT for I-HTMLs because it tries to push all the embedded images in an I-HTML to the browser. However, it introduces overhead. Other approaches have similarly tried to reduce overhead while providing the benefits of BIP. For evaluating performance we determine the following measures:

1. What is the percentage of HTMLs that will have latency saved (L-HTMLs/total HTML accesses)? This is "HTML saving latency" in following tables.

2. How much overhead does the approach introduce? There are two ways to measure the overhead. One is to measure the overhead as the number of pushed image misses (pushed image miss means that an embedded image will be pushed by the web server but it is not requested by the web browser in the trace file) divided by the total number of images pushed. We call this overhead Overhead-L. Another way to measure overhead is the number of pushed image misses divided

by the total number of image accesses in the trace file. We refer to this overhead as Overhead-G.

3.3.2 Simulation Setup

We evaluated the performance of these approaches by using trace-driven simulation. First, we obtained the trace file from the web server for one day. Then we downloaded these HTMLs accessed by the browsers and recorded in the logs from the server. We could not download all pages because some HTMLs are password-protected. Next, we analyzed each HTML to generate its embedded link structure information. We also analyzed the embedded image to see if it refers a local file or if it refers to a remote resource.

For our simulation, only images were considered as embedded documents. We also assumed that each host ran only one instance of a web browser. Because we use the actual trace file from the web server instead of the browser, we need to determine which embedded images are actually accessed by the browsers. Some browsers may not download the images. Others may access another page without waiting until all the images are retrieved. The browsers also have "warm" caches. However we only simulated "cold" browser caches. We first obtained the number of embedded pushes of all HTMLs accessed (this is maximum number of embedded images that may be pushed), then we kept track of all the embedded images accessed by each host and put those images accessed into the host's infinite client cache. Web servers will only push those embedded images not in the client cache. In this way, we obtained the

minimum number of accesses of images from all the HTML accesses if we assume that all the browsers turn on the automatic image download option and have infinite caches. The difference of these two numbers is those accesses that were never made by the browsers.

Next, the simulator examined the log file. For each successful HTML access requested, the simulator checked the embedded link structure information and did the corresponding action.

3.3.3 Performance of BIP, BIP-Ref, and BIP-Hist

We used the two web access logs described in table 3.1 to evaluate the performance of BIP, BIP-Ref and BIP-Hist. The result is presented in Table 3.2. BIP-Hist-1 means the server keeps the access history at depth 1. BIP-Hist means the server keeps access history with infinite depth and infinite client cache. A hit means that an image pushed to a browser by the web server was actually requested by that browser.

Table 3.2: Performance Result of BIP,BIP-Ref, BIP-Hist-1,BIP-Hist

Trace	*CSE Workload*				*EGR Workload*			
	BIP	BIP-Ref	BIP-Hist-1	BIP-Hist	BIP	BIP-Ref	BIP-Hist-1	BIP-Hist
Total images pushed	33,330	30,014	26,163	23,704	34,669	31,353	28,047	25,486
Total hits	22,612	22,409	22,024	21,387	24,866	24,535	23,983	23,527
Total misses	10,718	7,605	4,039	2,317	9,803	6,818	4,064	1,959
Hit ratio	68%	75%	84%	90%	72%	78%	86%	92%
Htmls saving latency	59%	58%	56%	55%	67%	65%	63%	61%
Overhead-L	32%	25%	16%	10%	28%	22%	14%	8%
Overhead-G	11%	8%	6%	2%	17%	12%	8%	4%

Table 3.2 indicates that BIP introduces significant overhead. From the table, we see that about 30% of total pushed images are not used. Overhead-G is up to 17%.

Table 3.2 shows that the referrer field helps with web server pushing. Compared to BIP, Overhead-L of BIP-Ref is reduced by 6% and 7%, and Overhead-G is reduced by 3% and 5%, and the number of HTML saving latency is reduced a small amount. Table 3.2 also shows the performance of BIP-Hist-1. Overhead-G is reduced by 2% and 4%, compared to BIP-Ref. At the same time, the number of HTML saving latency reduces by 2%. This is a slight improvement over BIP-Ref.

The result of BIP-Hist indicates that with the server maintaining the history of the client, BIP-Hist reduces overhead-G without too much reduction of HTMLs saving latency. However, the overhead for a web server to maintain the access history for each browser is large. It is our observation that maintaining a limited (as small as 1) access history is enough to obtain benefits.

3.3.4 BIP-Hash Results

The performance of BIP-Hash depends on what is in client caches and what kind of hash functions are used. The simulation of BIP-Hash would involve the cache replacement policy and cache size. The following provides an analysis of BIP-Hash. Let us assume there are n unique embedded images in a browser cache from a particular site, and the "false positive" rate of the Bloom filter for this site is p. Then, there are $p * n$ images not in the cache and the server will assume they are in the cache. If we assume that all embedded images in the cache are embedded in an HTML document with equal probability, and that a particular page has k unique embedded

images, then the probability that the page will have an embedded image that is not in the browser cache but that the server will assume it is in the cache is given by:

$$1 - \frac{C((1-p) * n.\, k)}{C(n,k)}$$

When n is quite large and p is quite small,

$$\frac{C((1-p)n, k)}{C(n,k)} \approx (1-p)^k$$

If we set $k = 5$ and we would like to have 95% of I-HTMLs latency-saved, then

$$(1-p)^5 >= 0.95 \Longrightarrow p <= 0.01$$

In order to satisfy the condition, $m/n >= 10$ and $k >= 5$. If we suppose $n = 500$ then $m = 5000$ bits = 625 bytes. This does not require much bandwidth for high-bandwidth networks, especially when this Bloom filter is sent only when the connection is established (HTTP 1.1). Consider the overhead without any type of BIP employed: the browser must make several requests. Each request might require several hundreds of bytes of request header. These multiple small messages require more CPU and network bandwidth than one large message as in the BIP-Hash approach. We further compare the performance of BIP-Hash and BIP-Hist-1 by using the CSE workload. If we assume the above configuration, the total number of the HTMLs saving latency is 56% (59% x 95%), which is the same as the BIP-Hist-1. We further compare the overhead of the two approaches. In BIP-Hist-1, 4,039 pushed images are wasted. If we assume that one embedded image on average is about 2K bytes, then there are total 8,078K bytes bandwidth wasted. If we consider BIP-Hash and

we assume each HTML request carries one Bloom filter, then since there are total 13,433 HTML requests in CSE workload, each Bloom filter will have 601 bytes, which means the client cache will cache about 462 embedded images on average. In practice, we expect the bandwidth overhead of BIP-Hash should be much smaller than BIP-Hist-1. So under a typical configuration, we may expect BIP-Hash to have much less overhead than BIP-Hist-1.

3.3.5 Memory requirements for maintaining the link structure information

In order to reduce the memory requirements for BIP, the web server may maintain two data structures. One is a mapping array, which maps each embedded image (URL) to a unique sequential number. The other is an embedded link structure list. In this list, for each HTML the web server maintains a list of those sequential numbers corresponding to each embedded image in the HTML page.

We assume the size of the mapping array is m and each URL takes p bytes. We further assume the total number of HTML pages is n and on average each HTML page has q embedded links. If we assume a sequential number can be represented by a 4-byte integer, then the total amount of memory required by the two data structures may be expressed as: $mp + 4nq$. If we assume $m = 20000, p = 100, n = 4000, q = 5$, then the total amount of memory required is about 2 megabytes [3]. For a typical web

[3]The CSE workload contains about 4,000 unique URLs.

site, the memory requirement for maintaining the link structure information is not a problem.

3.4 Benchmarking BIP approaches

The benefits of BIP, as we discussed in Section 3.1.2, can be summarized as follows: *Latency reduction*, *HTTP requests reduction*, *TCP connection keep-alive time reduction.*

The three benefits help each other. For example, BIP may avoid some overhead introduced by embedded image requests and therefore throughput may be improved. For the same reason, BIP improves the web server overload behavior and latency may be further reduced. Even though we analyzed the benefits in Section 3.1.2, it is difficult to analyze the interactions among them. Measurements provide a direct method to evaluate the impact of BIP on web server performance.

3.4.1 Benchmarking Tool

The tool we use to benchmark the performance of the Apache Web Server is called *httperf* [48], which was developed by Hewlett-Packard Laboratories. *Httperf* is a comprehensive tool with many features. *Httperf* may provide sustained load and is easily extensible. It reports network I/O throughput, reply rates, session rates, session time and connection time. Features related to the measurements conducted in this paper are:

1. *Support for persistent connections and request pipelines.* Multiple requests may be sent over one persistent connection in a manner controlled either by the trace file or at a fixed rate.

2. *Tracefile support.* A trace file may specify which URLs should be accessed and in which order.

3. *Session support.* Sessions may be defined by a tracefile to simulate session behavior. For example, a session may simulate the behavior of a browser accessing the web page with embedded images. It may also support CGI requests.

3.4.2 Experiment Setup

The measurements we conducted were done mainly within the HSNP (High Speed Network and Performance) laboratory of the Computer Science and Engineering Department at Michigan State University. In our experimental environment, a network of SUN Ultra 1 machines are connected by switched 100BaseT Ethernet. The raw bandwidth between machines is 100 Mbps. This is an easily controlled environment in which the network and the machines are lightly used.

The benchmark tool we described in Section 3.4.1 is running on these SUN Ultra 1 machines. One machine is used as the web server, which runs the most popular web server - Apache 1.3.9. Other machines are used as clients to generate a sustained load to the web server. Each Ultra 1 workstation has 64 Megabytes of memory and 1.8 Gigabytes of local hard drive. The operating system running on each workstation is Solaris 2.7.

For our measurements, we use a single large HTML file to simulate image pushing. For example, if document A has a size of 1.5 KBytes, and 5 embedded images of 2 KBytes each, we simulate this by a single file of 11.5 KBytes.[4] Only one sample static page with a number of embedded images is accessed repeatedly, except for the case of our CGI performance measurement. For the CGI measurement, the request is sent and the sample static page is generated by a PERL script. In BIP, the CGI script will generate the HTML page with the designated size. Trace files are used to generate sessions. For Non-BIP cases, each session is comprised of a retrieval of the HTML page, followed by the retrievals of the embedded images in the HTML page. For BIP, after receiving the large HTML file, the connection is closed.

No trace file workload is introduced in our measurements because web traffic, access patterns, and other characteristics change. It is very difficult to collect a representative trace file workload. Furthermore, such a measurement result would soon become obsolete. It is our intention to show the potential benefits of BIP instead of BIP's performance under a specific workload.

Image pushing generally has some overhead [49]. We simulate pushing overhead by increasing the size of the single file by a percentage of BIP overhead. In our example, we select 10% BIP overhead [49].

[4] We do not measure the link structure maintenance overhead in order to avoid an implementation limitation of Apache. Because the current version of Apache is implemented based on a process-pool, link maintenance has to be provided by a separate process and all Apache processes access the shared link structure by shared memory. This overhead might not exist on a thread-based implementation, which Apache 2.0 plans to provide. Using a single large file to simulate pushing without considering the opening and closing of embedded image files is to avoid excluding the possible optimization of web server implementations for file access. For example, some proxies place frequently accessed documents in memory all the time. Such an optimization does not introduce disk access for frequently accessed documents. Therefore, we essentially measure the upper bound performance of BIP.

The system we measured has three parts: clients, networks, and the web server. In order to make sure that we are measuring the web server performance, network and client machines are tested to ensure that the two components are not the bottlenecks. Network bandwidth is measured by *netperf*.[5]. With a 99% +/-2.5% confidence interval, the measured bandwidth for one TCP connection is 90.44 Mbps. When we use *httperf* on the Apache Web Server, the largest effective network bandwidth (payload) consumed by one machine is only 40 Mbps. Since one machine is sufficient to generate a sustained load on the web server, even if we consider the protocol overhead, the network is not the bottleneck in our measurement.

Next, we use two machines to conduct the measurements to ensure that the client machine is not the bottleneck. Both machines have the same configuration. We determined that the throughput of the web server does not increase compared to the throughput when only one client machine is used. Therefore, one machine is enough to generate load to the web server.

We also conducted our measurements across the Internet in order to obtain a better understanding of BIP. We conducted a limited number of measurements since such an environment is difficult to control. For such measurements, the web server runs in the Chemistry Department at the University of New Mexico. The machine hosting the web server is a Intel Celeron 466 with 128 Megabytes of memory and 8 GB disk space. The operating system running at the web server is Linux Redhat 6.2 and the web server software is Apache 1.3.9. The local area network is 10Mbps Ethernet.

[5]http://www.netperf.org/

The network round trip latency is about 71 milliseconds. The effective bandwidth measured by *httperf* is 4.6 Mbps.

3.4.3 Measurement Results

This section provides measurement results of the web server's performance under various metrics. It also provides analysis of these results. In all the figures, BIP-NV refers BIP without pushing overhead. BIP-V refers BIP with 10% pushing overhead. Non-BIP is referred as NBIP.

Table 3.3 provides the measurement parameters and their default values.

Table 3.3: Simulation Parameters and their Default Values

Parameter Description	*Default value*
Number of embedded images	5
Size of the sample HTML page	1.5 Kbytes
Size of embedded images	2 KBytes
Network Bandwidth	100 Mbps
Pushing Overhead	0
Static Page	Yes
Persistent Connections	Yes

Default Configuration

In this section, we measured the web server performance of BIP-NV, BIP-V and NBIP under the default parameter configuration. Figures 3.2- 3.4 show the comparison of latency, throughput and concurrent connections, respectively. In Figure 3.2, when the session request rate is below 60 requests/sec, all approaches have reasonable performance. BIP's latency is about one-half of NBIP. When the session request rate

goes above 60 requests/sec, the web server starts to become overloaded under NBIP and the performance of NBIP degrades very quickly. Nevertheless, BIP continues to provide very good performance. When the session request rate reaches 160 requests/sec, where BIP starts to become overloaded, the latency of NBIP is 2 seconds. The latency of BIP is 0.05 second. BIP has the maximum of 160 session requests/sec, while NBIP only handles 60 session requests/sec. This indicates that, potentially, BIP may improve the throughput by 150%. When both NBIP and BIP operate at their maximum throughput (60 requests/sec and 160 requests/sec, respectively), the latencies are 85 ms for NBIP and 51 ms BIP. This means that BIP may improve the throughput 150% without sacrificing the latency.

The number of concurrent connections is the average number of open connections at any given time. This reflects connection resource consumption. As we can see, when the server is not overloaded, the number of concurrent connections is quite low. Once the server reaches the knee point, the number of concurrent connections increases dramatically. In our measurements, all approaches finished their session over one persistent connection. At a request rate of 80 requests/sec, even the latency of NBIP increases significantly, and *httperf* still takes the same amount of time to connect to the web server as it does at lower rates. This indicates that connection establishment is not the bottleneck for this default configuration. The increase in the number of concurrent connections is because the web server has the power to accept all the connections but cannot process the requests soon enough, such that the connections stay in an established state longer. The benefit of BIP for connection management is not demonstrated here.

In all three figures, BIP-V has similar performance as BIP-NV, and is much better than NBIP. Therefore, we will not consider BIP-V further. The three figures show that BIP is very effective under a LAN environment and BIP boosts web server performance dramatically for static pages with embedded images.

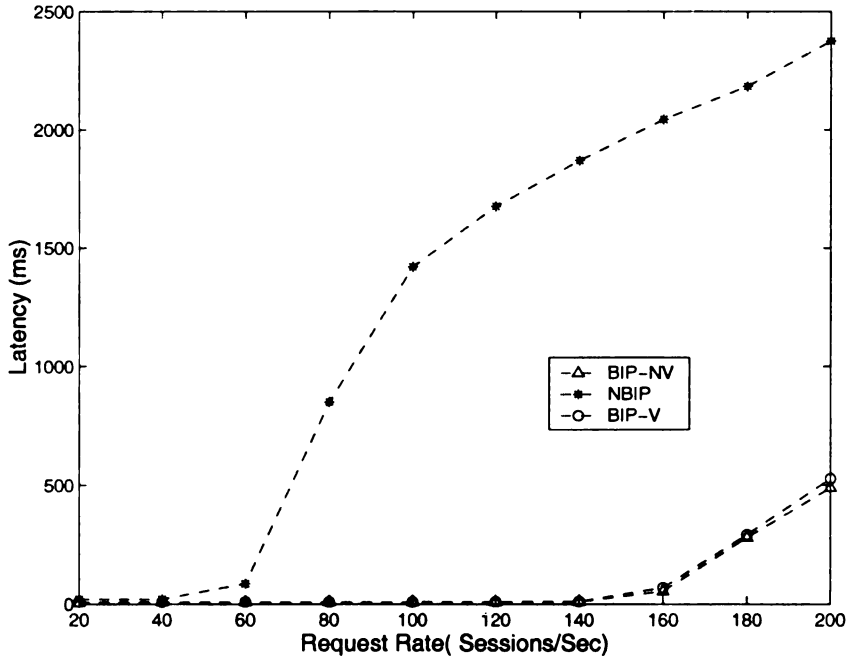

Figure 3.2: Latency Comparison of BIP-NV, BIP-V and NBIP (Default)

Varying the Size of Embedded Images

In this section, we show measurements of how size of the embedded images affects the performance of BIP. We assume in our experiments that a sample page has 5 embedded images. We further assume that all embedded images have the same size. We set image sizes to 0.5 KBytes, 1 KBytes, 2 KBytes, and 4 KBytes respectively, to measure the performance of both approaches.

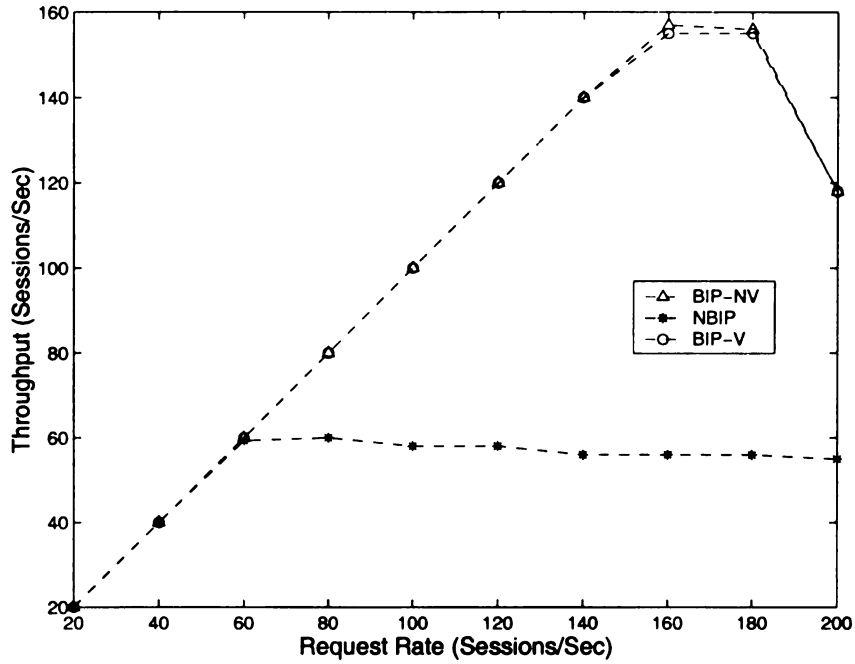

Figure 3.3: Throughput Comparison of BIP-NV, BIP-V and NBIP (Default)

Figure 3.5 and Figure 3.6 show the latency and throughput of both approaches using different image sizes. In Figure 3.5, the upper group of curves shows the latency of NBIP. The bottom group of curves shows the latency of BIP. The curves in each group from bottom to top are corresponding to image sizes of 0.5 KBytes, 1 KBytes, 2 KBytes, 4 KBytes. In Figure 3.6, the curve sequence is in reverse order since as the sizes of the images increase, the throughput decreases.

We may clearly see from the two figures that the size of the embedded image is not as important as it first appears. All NBIP approaches have similar throughput with limited difference on the latency. For BIPs, increasing size of the embedded image gradually degrades the throughput, from 180 requests/sec for 0.5 KBytes to 140 requests/sec for 4 KBytes. Surprisingly, we did not see any overlap between the

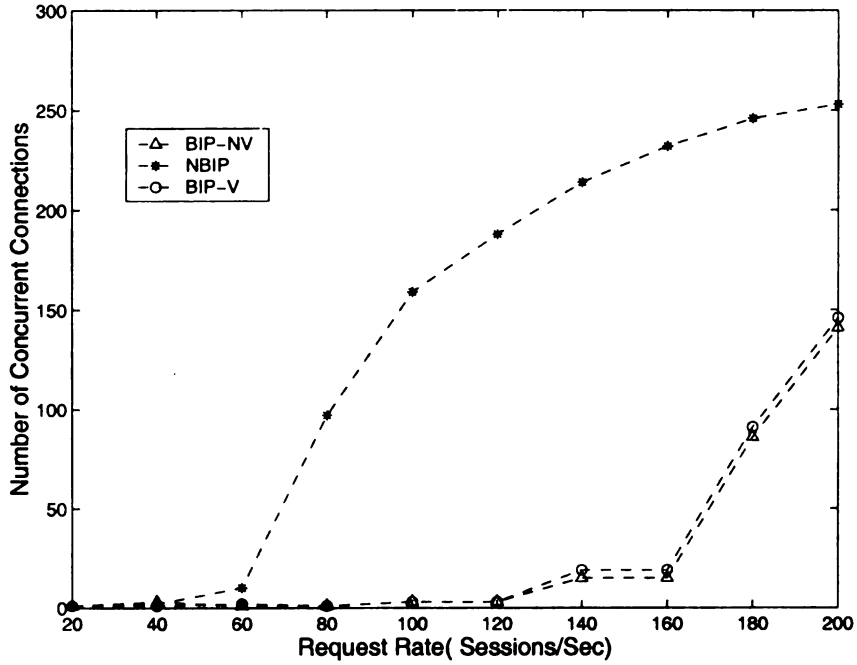

Figure 3.4: Concurrent Connections Comparison of BIP-NV, BIP-V and NBIP (Default)

two groups of curves using our measurements. Therefore, when the size of embedded image changes, BIP continues to offer good performance.

Varying the Number of Embedded Images

In this section, we show measurements of how the number of embedded images in each HTML page affects the performance of the web server. In our experiments, we use a sample page with a number of 5, 10, 20, and 40 embedded images, respectively. The latencies shown in Figure 3.7 are the latencies when the maximum throughput under each configuration is achieved. The throughputs shown in Figure 3.8 are the maximum throughputs under each configuration. If we examine Figure 3.8, we see that as the number of the embedded images increases, the throughput suffers tremendously.

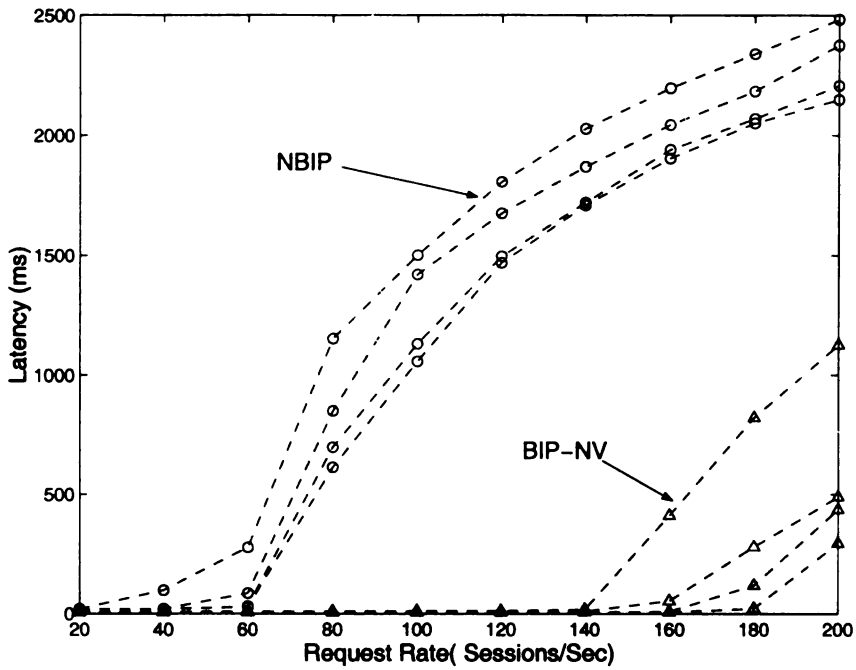

Figure 3.5: Latency Comparison of BIP-NV and NBIP (Size)

While the throughput of NBIP is 60 session requests/sec if a page has 5 embedded images, it drops to less than 11 session requests/sec when 40 images are embedded. At the same time, BIP throughput drops from 180 to 140. The latency increases while the gap between the two approaches becomes larger.

NBIP needs several connections to finish a session under a larger number of embedded images in a page. If a page has 20 embedded images, on average, the browser needs 2 connections to serve the session. When the page has 40 embedded images, the browser needs 3 connections to serve the session. Since the size of the image increases, the only overhead is the transmission time, which is low in our measurements. When the number of the embedded images increases, a separate request must be sent for

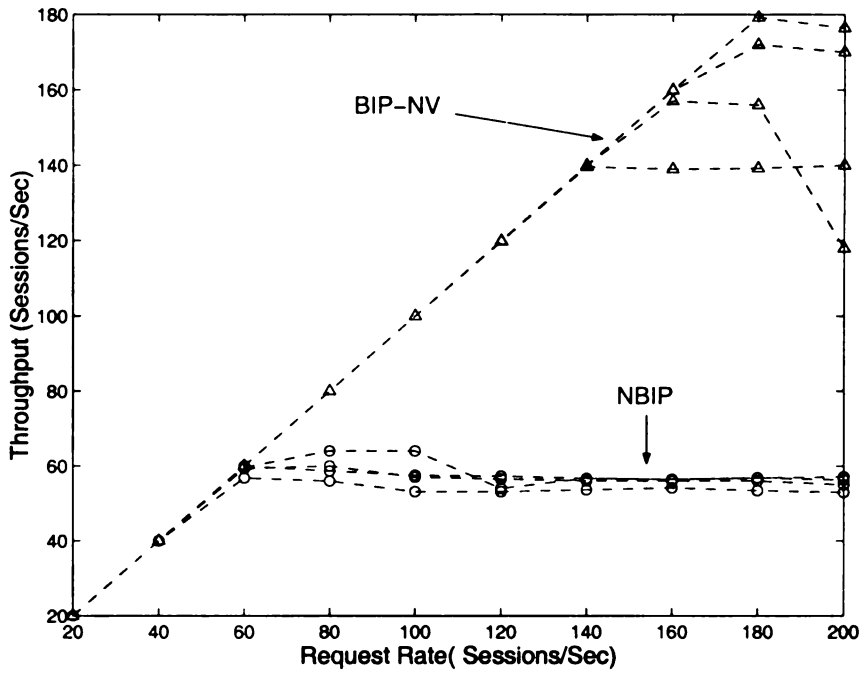

Figure 3.6: Throughput Comparison of BIP-NV and NBIP (Size)

each embedded image and the overhead increases dramatically. This is the situation where BIP may work very well.

BIP over Wide Area

We describe in this section some measurements we conducted over a wide area as detailed in Section 3.4.2. The default configuration is used. Figures 3.9, 3.10 and 3.11 show the latency, throughput and number of concurrent connections for both approaches over the wide area. From our measurements, the throughput for BIP is about 50% more than NBIP, and latency is about 500 ms less than NBIP. Since it takes much longer for the web server to process transfers, the benefits of throughput

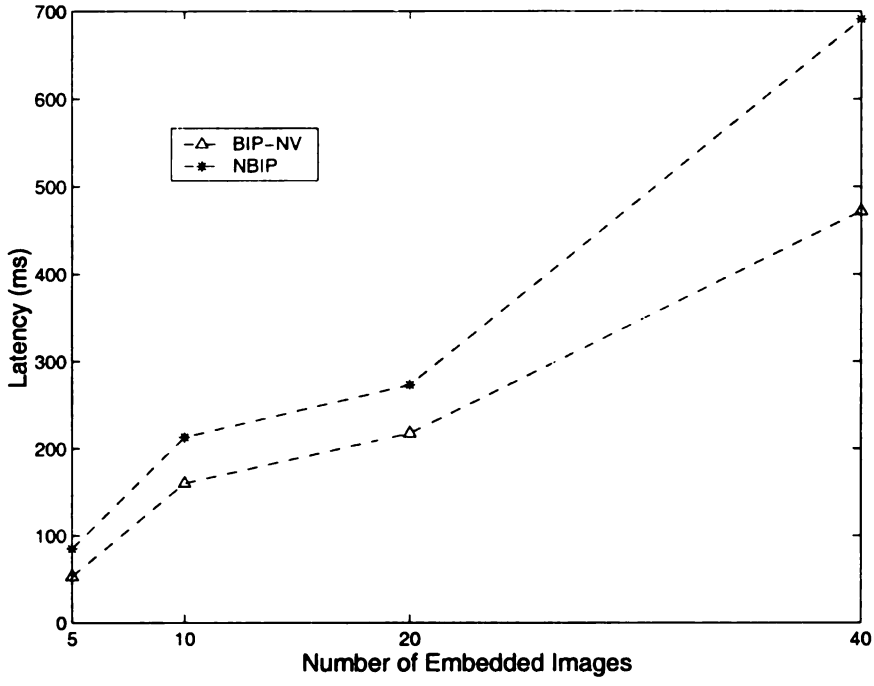

Figure 3.7: Latency Comparison of BIP-NV and NBIP (Number)

drops because the overhead introduced by processing embedded images compared to the time to serve the session is much less than that in the LAN case.

CGI

In this section, we show measurements of the performance where CGI scripts (PERL) are executed. Figures 3.12 and 3.13 show the latency and throughput of both approaches. Our Apache Web Server does not include a built-in PERL module. Therefore, each time that a PERL script is invoked, a process is created. Due to this prohibitive overhead, the throughput drops dramatically. Since BIP tries to decrease the overhead of embedded image processing, BIP does not provide much benefit in this case, because the overhead to process the embedded image is much less com-

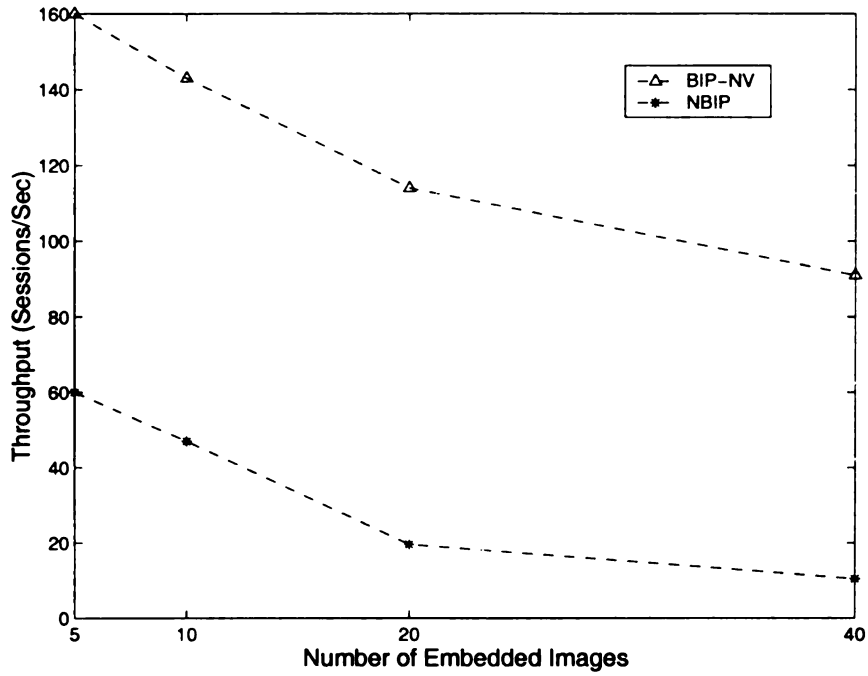

Figure 3.8: Throughput Comparison of BIP-NV and NBIP (Number)

pared to the case where a static page is served. For example, in the CGI case, it takes 400 ms to receive the first response while in the static page case, the first byte is received after about 8 ms. BIP has only 20% throughput improvement. Nevertheless, BIP provides benefits of reducing latency because it still takes time for the server to process the embedded image requests. If the PERL module is built within the Apache binary, the overhead of processing a PERL script is much smaller. Therefore, better throughput improvement would be expected.

Summary

From the above measurements, we see that in any case, BIP reduces latency in comparison to NBIP. The improvement of throughput of BIP depends on the overhead

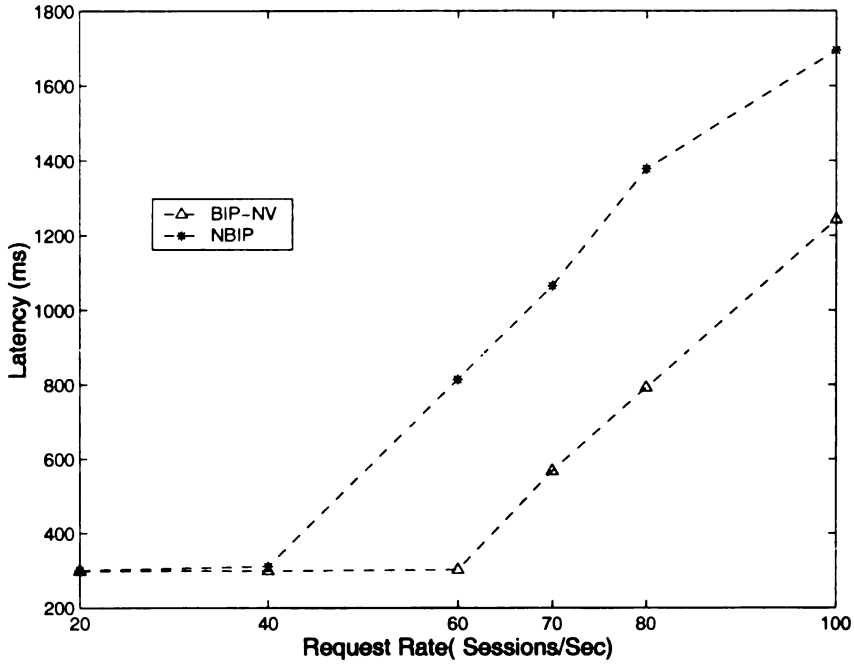

Figure 3.9: Latency Comparison of BIP-NV and NBIP (Wide Area)

introduced to process the embedded images compared to the time required to serve the web requests in both BIP and NBIP.

We summarize our results:

1. BIP is very effective for boosting web server performance to serve static pages.

2. The size of the embedded image is not as important as would be intuitively expected within a LAN environment.

3. BIP works very well when the number of embedded images in the HTML page is large.

4. BIP improves the latency and throughput reasonably well when the *relative overhead*(the overload compared to the total time to service the request) intro-

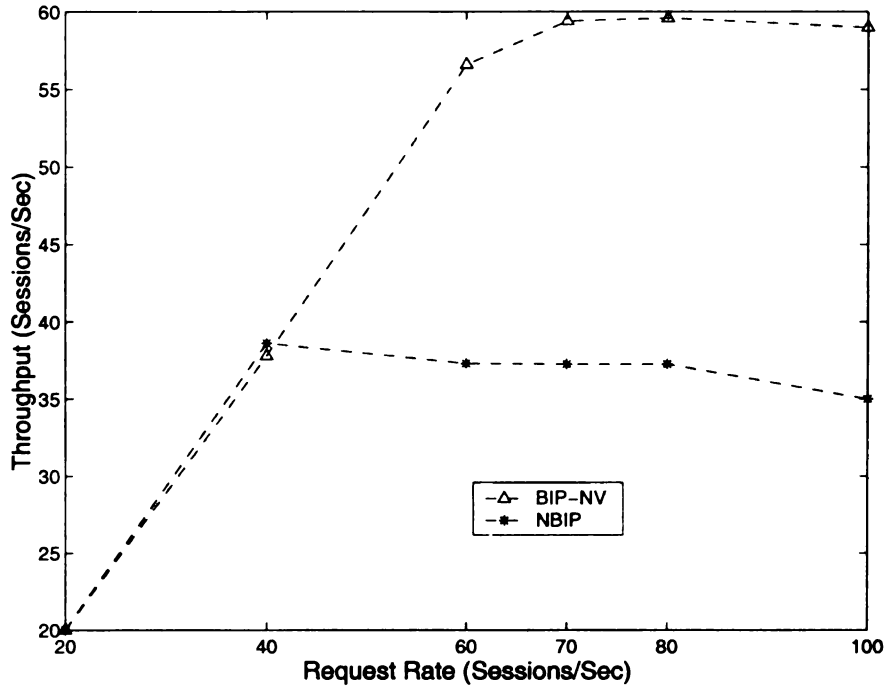

Figure 3.10: Throughput Comparison of BIP-NV and NBIP (Wide Area)

duced by processing embedded images is much less compared to the relative overhead of serving static web pages over a LAN. This conclusion is valid for the WAN and CGI case.

3.5 Implementation and Possible Limitations of BIPs

It is important to consider how our mechanism might fit with the current HTTP 1.1 specification. Within the HTTP 1.1 standard, the HTTP request header may have optional fields which may be interpreted by the web server and by the proxy. The addition of a "push request" optional field and a "push response" optional field may be easily fitted into the HTTP 1.1 standard. For deployment, BIP approaches may be implemented by the web browser and supported by web server. Because it

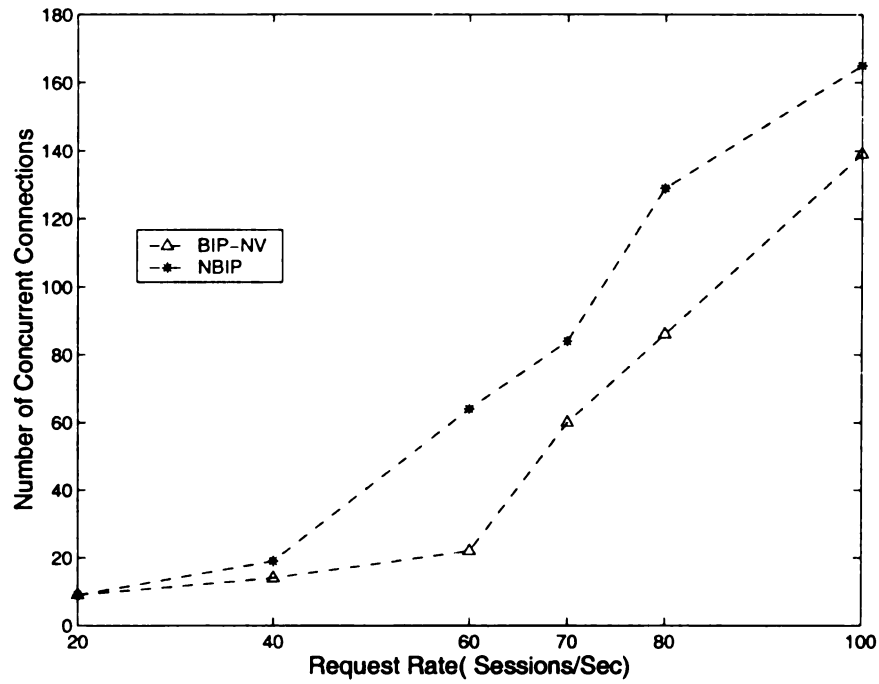

Figure 3.11: Concurrent Connections Comparison of BIP-NV and NBIP (Wide Area)

is implemented in the browser and web server, no special hardware or software is needed or administrated. Since web users tend to upgrade their browsers often, quick deployment could be expected.

Possible limitations include:

1. BIP has not been extended to download embedded images in one I-HTML in parallel using different connections originated from the same browser. It is not clear whether this is a desired behavior because one of benefits of HTTP 1.1 is to pipeline the requests and to provide a type of fairness among clients by limiting the connections to the server.

2. BIP does not work well in low-bandwidth links, such as modem users. This is because in such an environment the dominant latency is the transmission delay.

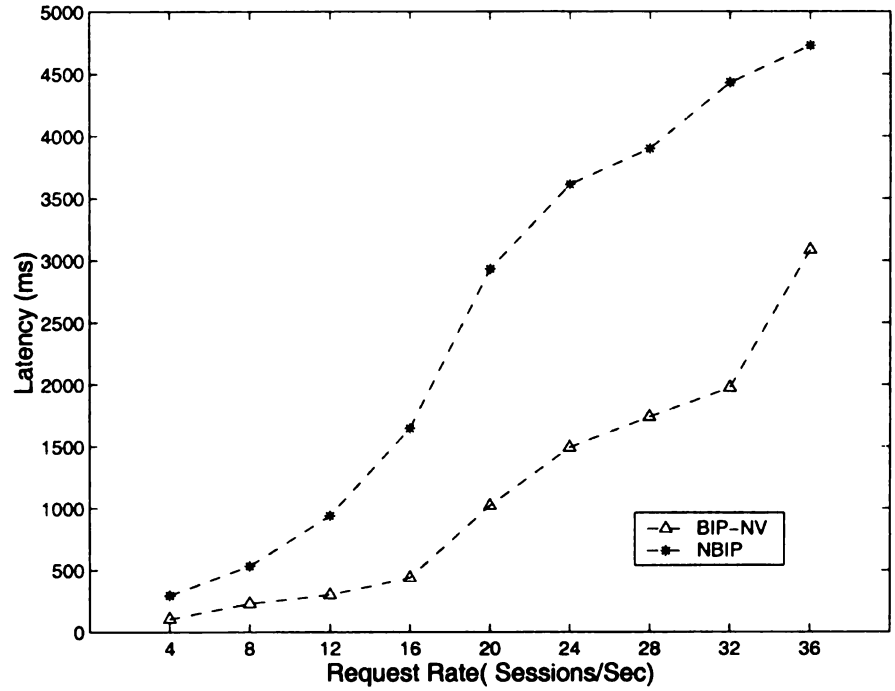

Figure 3.12: Latency Comparison of BIP-NV and NBIP (CGI)

3.6 Summary

In this chapter we proposed a browser initiated approach (BIP) to web document pushing. BIP reduces the download latency of those HTML pages having embedded contents and improves server resource utilization. Simulations show that BIP can download up to 56% of all the HTML pages in one RTT and saves up to 20%-26% of total web accesses. BIP-Hash is best for modest caches with a limited number of proxy levels. Furthermore, BIP-Hist with history depth 1 and BIP-Ref offer reasonable performance with acceptable overhead. Benchmarking shows that in all cases, BIP improves the throughput and reduces latency. BIP impressively boosts the throughput of the web server to serve static web pages with embedded images. BIP improves the throughput of a web server reasonably over a wide area and improves

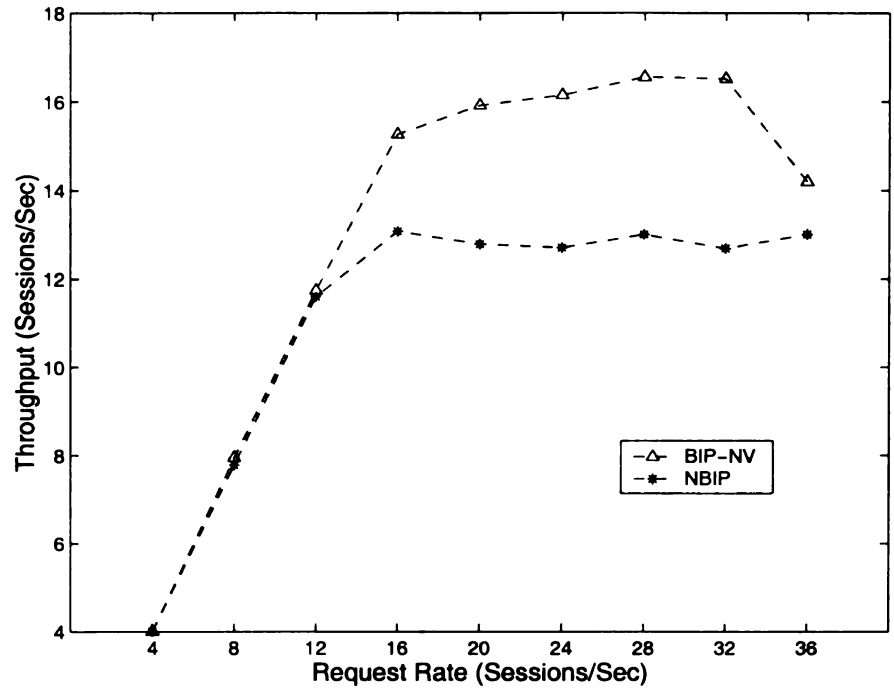

Figure 3.13: Throughput Comparison of BIP-NV and NBIP (CGI)

throughput of web servers that serve CGI requests. The size of the embedded image is not as important as the number of embedded images in an HTML page. BIP offers reasonable latency reduction in all configurations. Our work provides an upper bound for the benefits of BIP. Future work includes implementing an Apache Web Server module to determine a lower bound of the benefits of BIP.

Chapter 4

S3 - Smart Server Selection

Replicated servers are widely used to meet the increasing traffic demand of some busy web sites such as *Yahoo* (*www.yahoo.com*) and *Altavista* (*www.altavista.com*). These web sites have replicated server groups in each continent. With multiple replicated servers running simultaneously, the number of requests serviced by each server is reduced and a reasonable response time may be achieved. The replication of servers increases fault-tolerance, which is essential to these popular sites. By putting replicated servers at appropriate places, replicated servers also increase network proximity and thus significantly reduce access latency perceived by users.

A fundamental issue for a global-scale replicated service is how the best server is selected based on the user's preference. Most replicated server sites now have a menu at the home page of the site. Users may manually select one server according to their knowledge of geographical proximity. However, geographic proximity does not necessarily reflect the network proximity and thus it cannot guarantee the best server selection. It is desirable to automate the server selection so that it is transparent to

users. In this chapter we propose an approach, called *Smart Server Selection* (S3)[50], to perform the server selection based on multiple network metrics efficiently collected from routers. The rest of the chapter is organized as follows. In section 4.1, we argue that network metrics should be used in server selection. Based on this assumption, Section 4.3 presents an overview of our approaches and extensions to current routers and the DNS. Performance evaluations are presented in Section 4.4. Deployment issues are discussed in Section 4.5. A brief discussion is given in Section 4.6. Section 4.7 provides conclusions.

4.1 Selection Metric

The very first question we need to answer is the selection metric. Some of the approaches consider the server's load and some of the approaches do not. Some researchers argue that the purpose of the replication of the web site is to provide better web user navigation experience. The most important issue is the user-peceived latency. When server selection is performed, the server's load should be considered. Other approaches only take network metrics into consideration.

Research has shown very dynamic server load behavior. In Figure 4.1, the number of accesses per minute computed from a trace file of the web server of the department of computer science and engineering at Michigan State University is shown. As we can see, even during peak hours, the web server load exhibits dramatic changes in several minutes. Accurately estimating a server's load on time is a challenge. Studies on server selection traditionally have tried to address server selection and load sharing

together. In our approach, we separate the server selection problem from the load sharing problem. The server selection is proposed as one of the network services. In our approach, only network metric information is considered. The client is always directed to the nearest site according to network metrics. It is the service provider's responsibility to provide load sharing and resource scheduling to address the resource requirement.

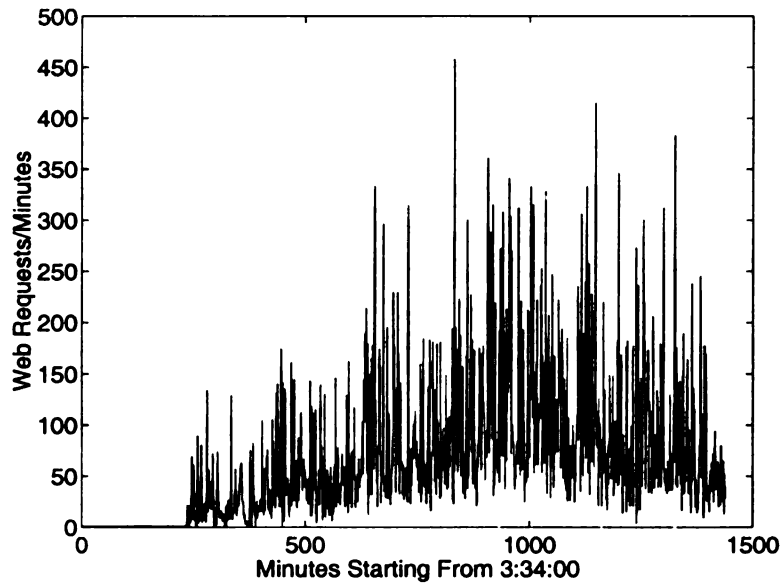

Figure 4.1: One Day's Access History From CSE workload

The principles of S3 are:

1. The server selection criteria should select the server based on network metrics.

2. Temporary server overload should be addressed by efficient server side load balancing approaches such as [40, 41].

3. Persistent severe overload should be addressed by capacity planning and monitoring on the server system, given that today's hardware is inexpensive.

We argue that the principles of S3 make better use of network resources. Load balancing may redirect a user to a server half way around the world simply because that server has lighter usage. By doing this, valuable network resources are wasted. We believe overloaded servers should be avoided by adding new servers, moving the locations of the servers, or choosing server locations that better matches users' access pattern.

4.2 Introduction of the S3 approach

A promising method to solve the server selection problem is to use DNS servers at clients side. All replicated servers may be aliased to the same DNS name. When a client accesses the service, the client DNS may select from a pool of IP addresses corresponding to the replicated servers and return the IP address of the server best matching the user's preference, such as shortest path or shortest latency.

In order to select a server based on shortest path or latency, the DNS server must know metrics regarding routing information leading to different servers. However, such information is only available from the routers. Routers know the metrics of the routes corresponding to each IP address but they have no idea which IP addresses are providing same services. This is only known by DNS.

To solve the server selection problem, a mechanism is needed to enable DNS servers and routers to communicate with each other. Router extensions are proposed in this paper to support a route metric query for IP addresses. Extensions to current DNS

are proposed to allow it to collect and cache routing metrics and select the best server. We named the mechanism *Smart Server Selection (S3).*

S3 takes advantage of DNS's role in the activity of accessing the replicated service. Due to the popularity of the replicated services, it is very likely that multiple hosts that share the common DNS access the same service within a short period of time. When that happens, the routing metric information to the service cached in the DNS may be shared by multiple DNS queries, and therefore reduce the number of routing metric queries and the DNS server response time.

Our simulation results show that S3 provides substantial performance improvement over the DNS round robin approach, which is the default method of server selection, both in terms of the number of hops used and total latency experienced. The overhead analysis shows that the overhead introduced by our routing metric collection scheme is negligible.

4.3 S3 - Smart Server Selection

We begin by giving an overview of S3.

S3 operates as follows:

1. The server side DNS maps a DNS name to all replicated servers' IP addresses.

2. When a client DNS asks for the name-to-IP-address mapping for the replicated services (DNS name), the server DNS sends all available IP addresses to the client DNS.

3. When a host sends a domain name query to the client DNS, the client DNS examines whether multiple addresses for this service are available. If multiple addresses are available and the metric is not cached, the client DNS sends a query to collect necessary routing metrics.

4. The DNS selects a server according to the route metrics it collected from the routers and other information available to DNS servers (such as the geographical distance between the client and the replicated servers) and returns the selected IP address to the host.

4.3.1 Possible Approaches to Collect Routing Metrics

The DNS must query a router that has full knowledge of Internet routes, which is usually a BGP (Border Gateway Protocol) router [51]. We propose two different approaches. The approach to use depends on the conditions of the local network.

Query the gateway router

In this first approach, we assume that the client DNS knows at least one gateway router when it is configured. Armed with such information, the DNS sends queries directly to this gateway router. The gateway router searches its routing table and supplies the necessary information to the DNS. The DNS must find the gateway router through other means, and is probably configured manually. A daemon runs on both the DNS and the gateway router, which allows them to communicate with each other. The daemon on the DNS enables the sending of routing information queries

and the daemon on the gateway router accepts such queries and sends responses back to the DNS.

The main advantage of this approach is the ease of deployment. Only the gateway router needs to be upgraded to support the routing metric query. All other routers need not be upgraded.

A technical issue arises if one replicated server is in the same Autonomous System (AS) and may be reached without passing through the gateway router. In this case, the gateway router may not know the routing metric from the DNS to the replicated server. This is the case where OSPF (Open Shortest Path First) [52] is used as the Interior Gateway Protocol (IGP) and the gateway router is in a different *area* than the DNS. The gateway router provides routing metrics based on the routes from itself to the replicated servers, which may be different from what is perceived by the DNS and the actual clients. Fortunately, since the difference is only for routes inside the same AS, the inaccuracy is not likely to cause noticeable performance degradation.

Query the direct-attached router

In this second approach, the DNS may send a query to the router to which it is directly attached (default router) to obtain the routing metric information. This approach relieves the requirement of DNS knowledge of the gateway router.

We propose extending ICMP (Internet Control Message Protocol) [53] to support such a mechanism instead of developing a brand new protocol because ICMP is implemented on every router and such a mechanism is a natural extension of ICMP.

The ICMP packet header has a "Packet Type" field. We extend ICMP to make use of two currently obsolete packet types: type 15 ("information request") and 16 ("information reply"). We specify that routing queries from the DNS use ICMP packet type 15 and responses from the router use type 16. The scenario of collecting routing information is as follows:

1. In a query packet, the DNS sets the packet type to 15 and specifies in the packet body the IP addresses and the routing metrics for which it is looking (hops, latency or others). It then sends the packet to its default router.

2. Upon receiving such a query, a router looks up its own routing table. If it can provide all the required information, it replies to the DNS. Otherwise, it fills the information available in the ICMP packet and forwards the packet along the default path to its default router, if there is one.

3. The upper level router tries to find the missing information. It will not overwrite or repeat the information that is supplied by lower level routers.

4. This process continues until a router supplies the last piece of information. This router creates and sends a reply packet (ICMP type 16) to the DNS.

The query packet eventually reaches an EGP router if some routing information about destinations outside the AS is requested. If there is no default path on a router and the information is still not complete, then it means some IP addresses are not reachable from that network. This router may safely send a reply to the DNS.

Discussion of the two approaches

The second approach has obvious and important advantages over the first approach. First, it relieves the requirement of DNS's knowledge of the "working" router address. It is completely transparent to the network and DNS administrators, and it works independently of the routing protocol used. Second, it results in highly accurate routing metrics because ICMP packets go through the same path as the data packets to those IP addresses. Third, because it queries IGP routers, the problem of the first approach is addressed. Last, because ICMP extension provides a way to collect route metrics for multiple IP addresses, this service can be exploited by other servers or hosts too.

Compared to the first approach, the second one involves all the routers along the default path up to a router where all routing metric can be supplied. All these routers need to be upgraded to support the new ICMP functions. This may result in longer time to deployment, and more investment.

These two approaches are complementary to each other. The first approach may be deployed quickly, while the second one should be used when enough new ICMP function enabled routers are available.

4.3.2 DNS Extensions

DNS servers will be extended to handle routing metric queries. The DNS cache entries will also be extended to store routing metrics. Under the current approach, the selection criteria based on the available metrics is configured in a central place:

the DNS server. Technically, it is easy to let the host or application specify the selection criteria (by applying different weights to each criteria). However, this will lead to the extension of the DNS protocol. At this point, it is not clear if such an extension is necessary.

4.4 Performance Evaluation

In this section, we present the results of several simulations conducted to evaluate S3's performance.

4.4.1 Performance Metric

In our simulations, the following two metrics are used to evaluate the performance.

1. *Total hops.* This is the total number of hops all the packets traverse.

2. *Total network latency in milliseconds.* This is the total network latency experienced by all the packets.

4.4.2 Simulation Setup

In our simulations we measured web accessing performance from a network to a site that provides the service globally. The client network modeled in the simulations is the public computer laboratory connected by a 100 Mbps Fast Ethernet in the Department of Computer Science and Engineering at Michigan State University. Yahoo (www.yahoo.com) is chosen to be the site that provides the global replicated

services. The web site of Yahoo is replicated at over 20 places, and each place has at least two different IP addresses. To be more realistic, only servers providing the English language are selected. The 5 places chosen were: yahoo.com and ca.yahoo.com (North America), sg.yahoo.com (Asia), uk.yahoo.com (Europe), and yahoo.com.au (Australia). Each group is on a different continent. The hops and latency from each place were measured by traceroute [54], and the collected data is shown in Table 4.1.

Table 4.1: Route metrics for Yahoo replicated servers

Location	*Hops*	*Latency(ms)*
United States	15	66
Canada	15	69
United Kingdom	18	154
Australia	20	425
Singapore	15	340

Our client network is modeled as a cluster of N_H hosts. In our simulation, this number is set to 100, which is roughly the number of machines in the public instructional laboratories in our department. Each host generates requests to replicated servers independently according to a Poisson process with arrival rate of λ_H. Different arrival rates are used to model different usage levels of the labs in our department. Usually the labs are heavily used during daytime and lightly used during the night time. Only the number of requests generated by the hosts is considered without simulating the network condition of labs.

A change in the route metric occurs as another Poisson process with rate of λ_R. In our simulation, this parameter is fixed to approximate the real situation between our local network and the five targeted Yahoo sites.

Based on the data we observed, the hop number change is modeled as a weighted discrete uniform distribution. The latency change is modeled as a weighted linear combination of Gamma distributions α, λ and a uniform distribution, which happens as a Poisson process with rate of λ_L. For some sites, the Gamma distribution has been reported as the model of latency change[1]. But for many other routes, the latency change is so vigorous that it cannot fit into a Gamma distribution. To simulate such a condition, a uniform distribution is introduced to the model. The parameters λ and α are found by manually fitting the data we collected. The client DNS caches domain-name-to-IP-address mapping during the simulation period. It refreshes the route metric information at a constant rate of TTL_{DR}. The resolver at each host refreshes the knowledge of the best replicated server at a constant rate of TTL_H.

4.4.3 Simulation Results

Figure 4.2 shows the hops traversed by all the packets during different length of simulations and using different host request arrival rates. The route selection criterion is shortest path first. The figure shows that the total number of hops increases linearly as the simulation time increases. In all cases, the number of hops experienced by the S3 method is 12% less than that generated by the RR method. We have 3 different sites in these experiments with the same mean of the path length (15). If the difference between different routes becomes larger, the savings due to S3 would be greater.

[1]Based on work done by the ACS lab, Department of Computer Science and Engineering, Michigan State University.

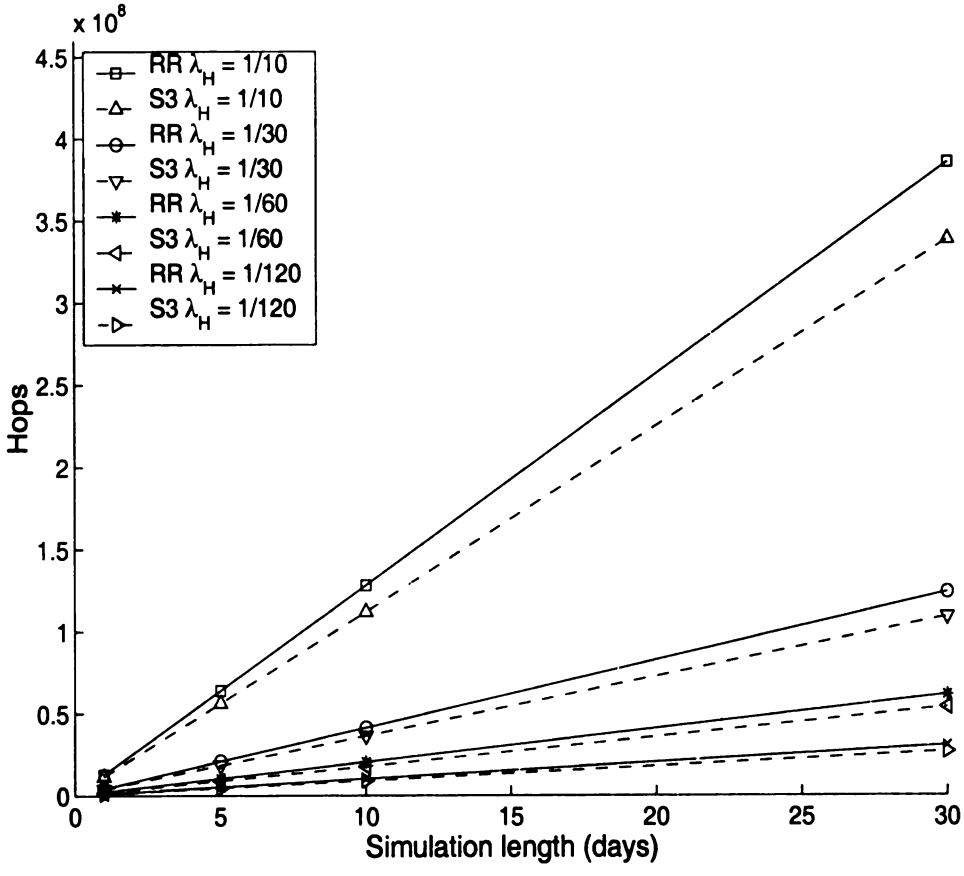

Figure 4.2: Hops vs Simulation Length

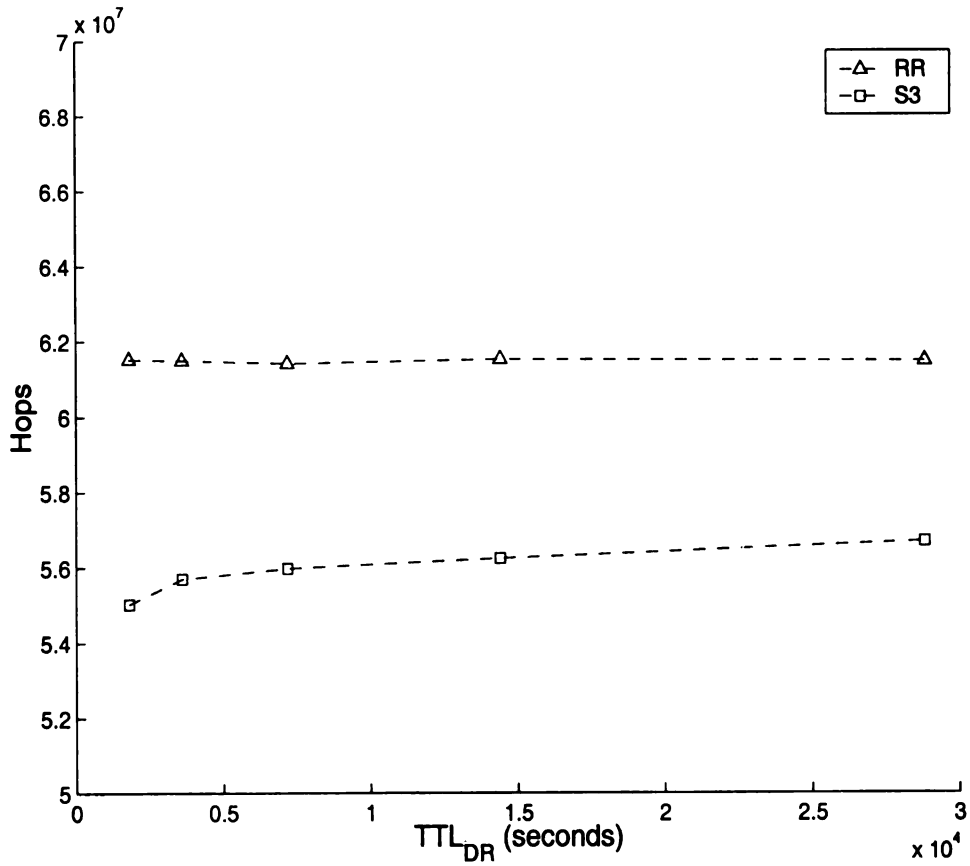

Figure 4.3: Hops vs TTL_{DR} ($\lambda_H = 1/60$)

Figure 4.3 shows the hops versus TTL_{DR} (the frequency DNS refreshes the routing metric from routers). It should be noted that the experiments were performed under different TTL_H. Each dot in the graph represents the results of 6 experiments. The experiments use the same DNS method, host request rate and TTL_{DR} value, but with different TTL_H values at 300, 600, 900, 1800, 2700 and 3600 seconds. Since each set of the 6 numbers is similar (less than 1% difference), the average is used to represent them.

For the round robin method, there is no concept of server selection based on the routing metric. It is not affected by the value of TTL_{DR}. For S3, the hops increases when TTL_{DR} becomes larger. This suggests that in order to save hops, the DNS needs to refresh the routing metric information more frequently. The slope of the line has a major change when TTL_{DR} changes from 3600 seconds to 1800 seconds, which marks the critical point where DNS changes from the status of being able to update routing information promptly to the status that it can no longer do so.

Figure 4.4 shows hops versus TTL_H. Similar to Figure 4.3, this is an aggregate representation of a set of experiments. Each dot in this figure corresponds to different TTL_{DR} values at 1800, 3600, 7200, 14400, and 28800 seconds. The graph shows that smaller TTL_H results in larger hop saving.

A large TTL_H means that the host machine refreshes a domain-name-to-IP-address mapping for a long period. In practice, it may also be a result of longer session. In a long session, once a host establishes a connection with a replicated server (such as TCP), all packets of this session should be forwarded to this particular server no matter how the route metric changes at the DNS.

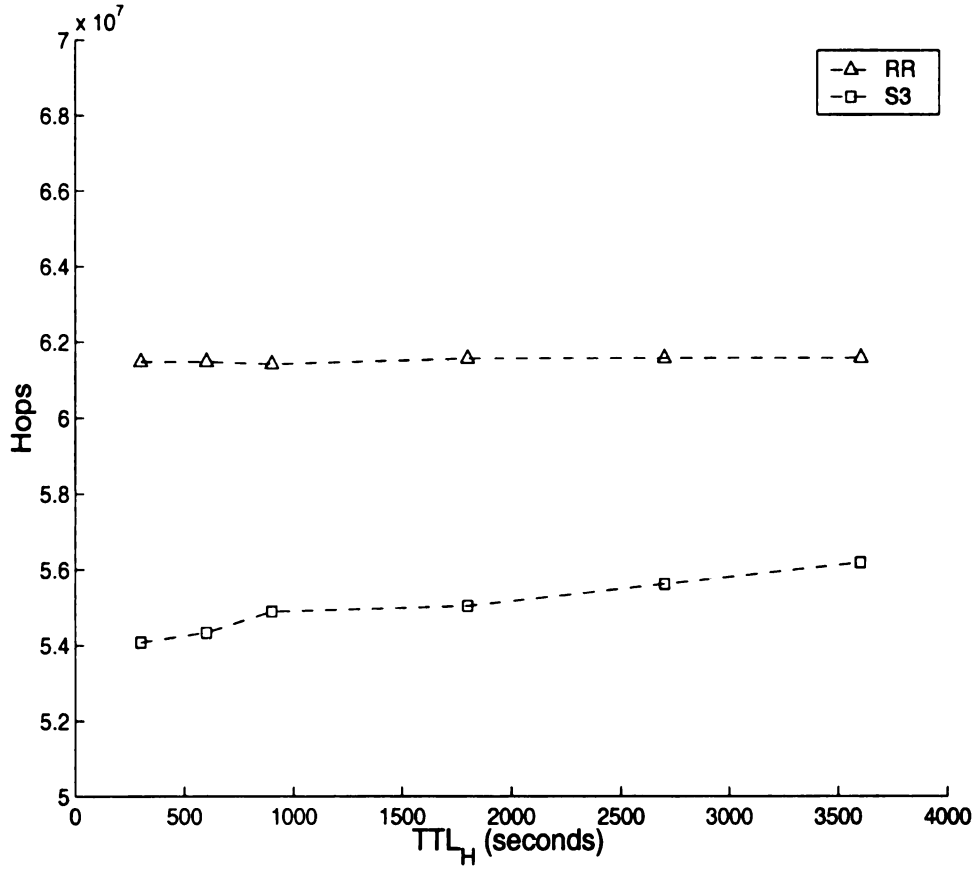

Figure 4.4: Hops vs TTL_H ($\lambda_H = 1/60$)

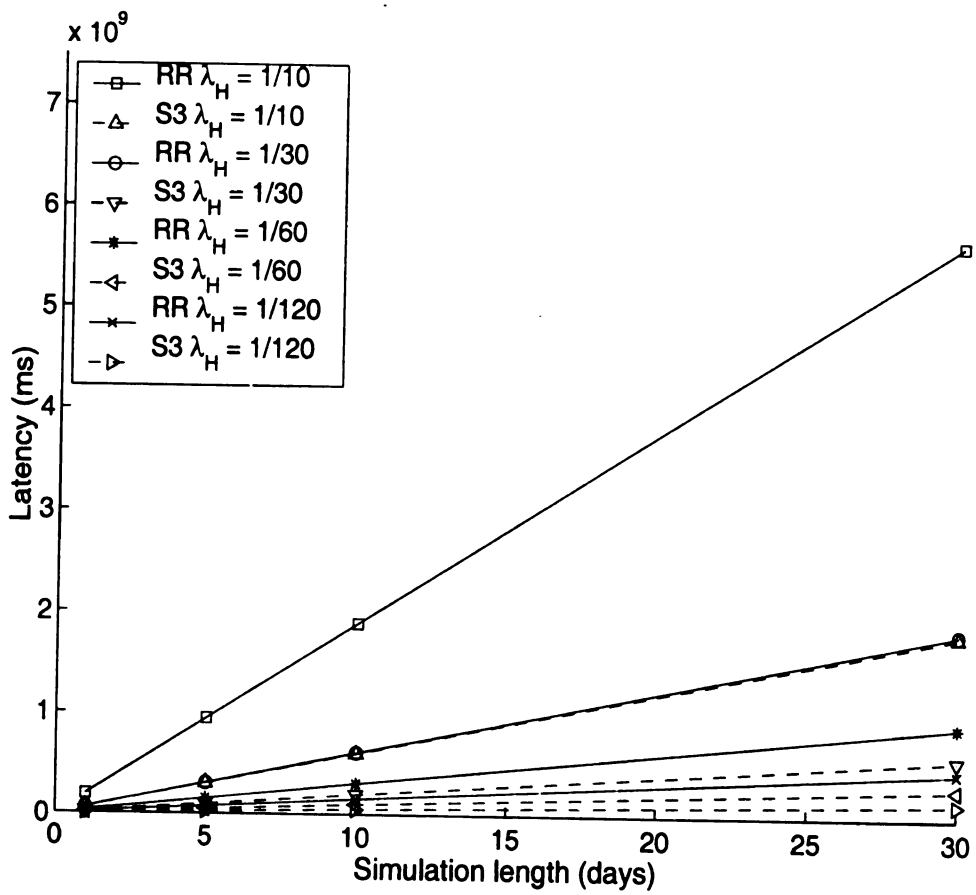

Figure 4.5: Latency vs Simulation Length

Figure 4.5 shows the simulation results of finding the best latency routes. Overall, they are very similar to the results based on hops, but the latency changes may occur frequently due to network congestion, and the number of hops between two networks is relatively stable. This requires the DNS to collect the route metric more frequently. For stable routes, the latency changes according to a Gamma distribution. For unstable routes, the distribution is difficult to find and we approximate it using uniform distributions. We further mix the two types of distributions to reflect different situations.

$$W * Gamma(n, \lambda) + \quad (1 - W) * Uniform(a, b),$$

$$where\ 0 \leq W \leq 1$$

The weight W represents how the two distributions are mixed together.

Figures 4.6-4.9 are the result of running S3 for different latency change distributions. In Figure 4.6, W is 1, so the distribution is Gamma. In Figure 4.9, W is 0, corresponding to a uniform distribution. Figures 4.7 and 4.8 use W of 0.6 and 0.3, respectively.

Let us temporarily call the latency change distribution D. If TTL_H is reasonably small, such as 30 seconds, we see that the closer D is to Gamma, the more insensitive the latency is to the TTL_{DR} change. The closer D is to uniform, the more sensitive the latency is to the TTL_{DR} change. In other words, TTL_{DR} changes cause large latency variations if D is similar to uniform; but can only cause small latency variation

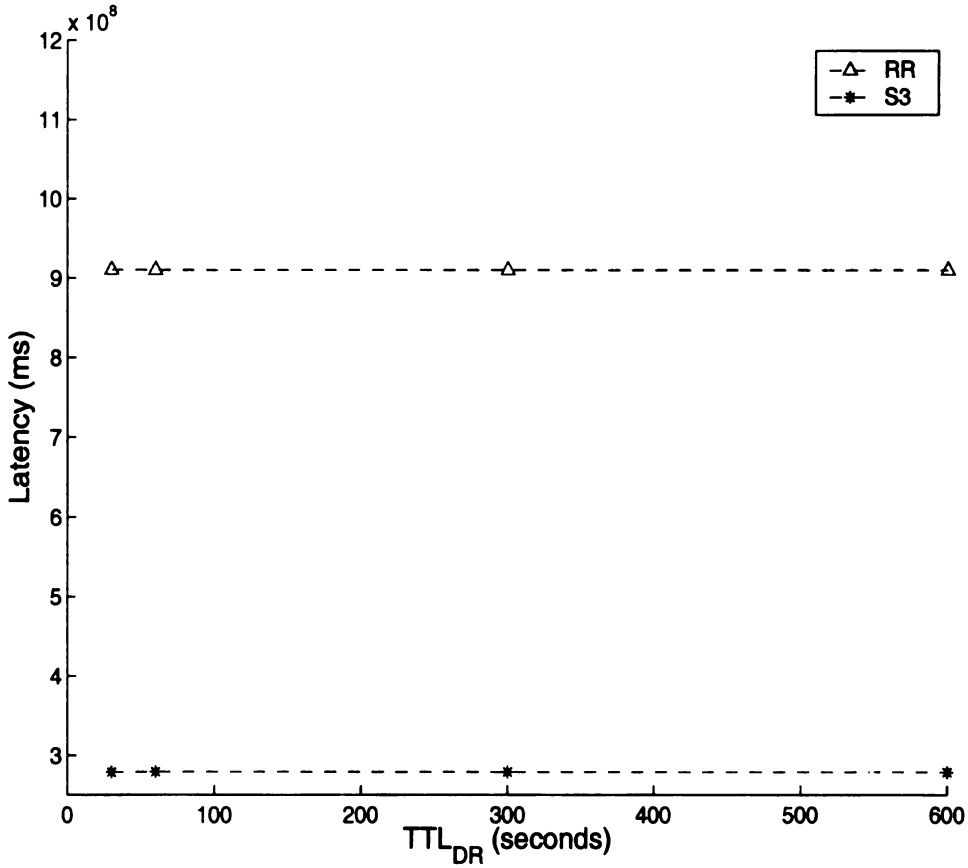

Figure 4.6: Latency vs TTL_{DR} (W= 1,Gamma,$\lambda_H = 1/60$)

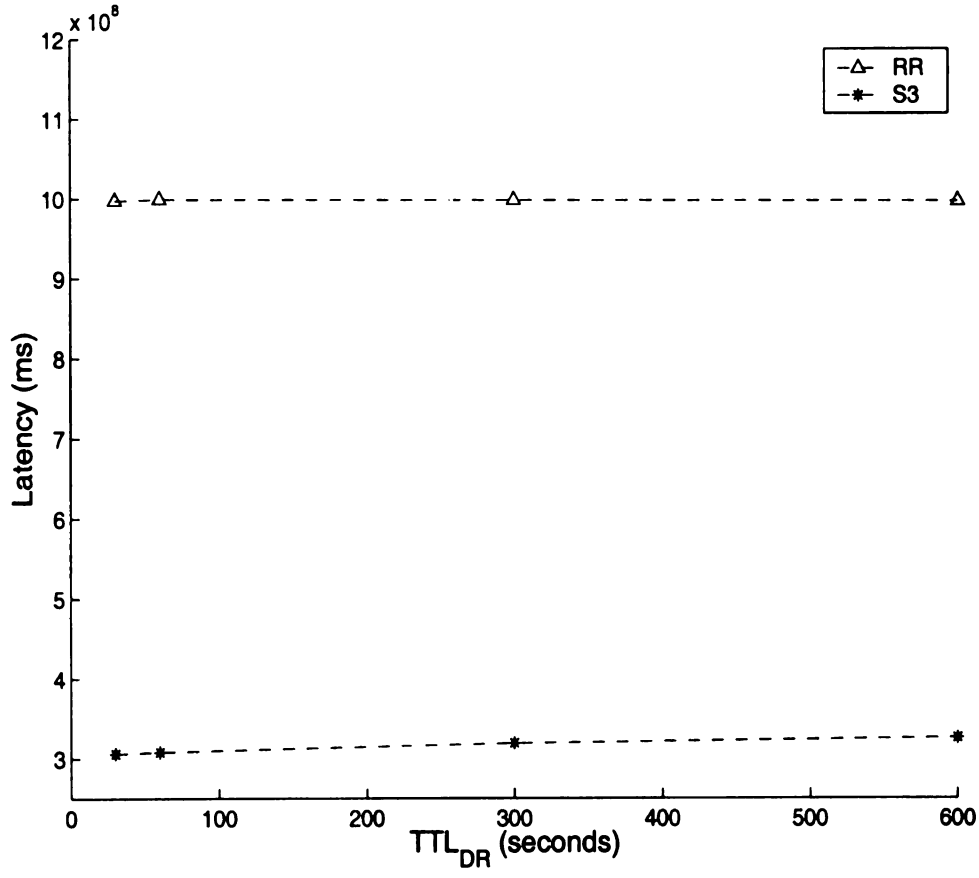

Figure 4.7: Latency vs TTL_{DR} (W= 0.6,mixed,$\lambda_H = 1/60$)

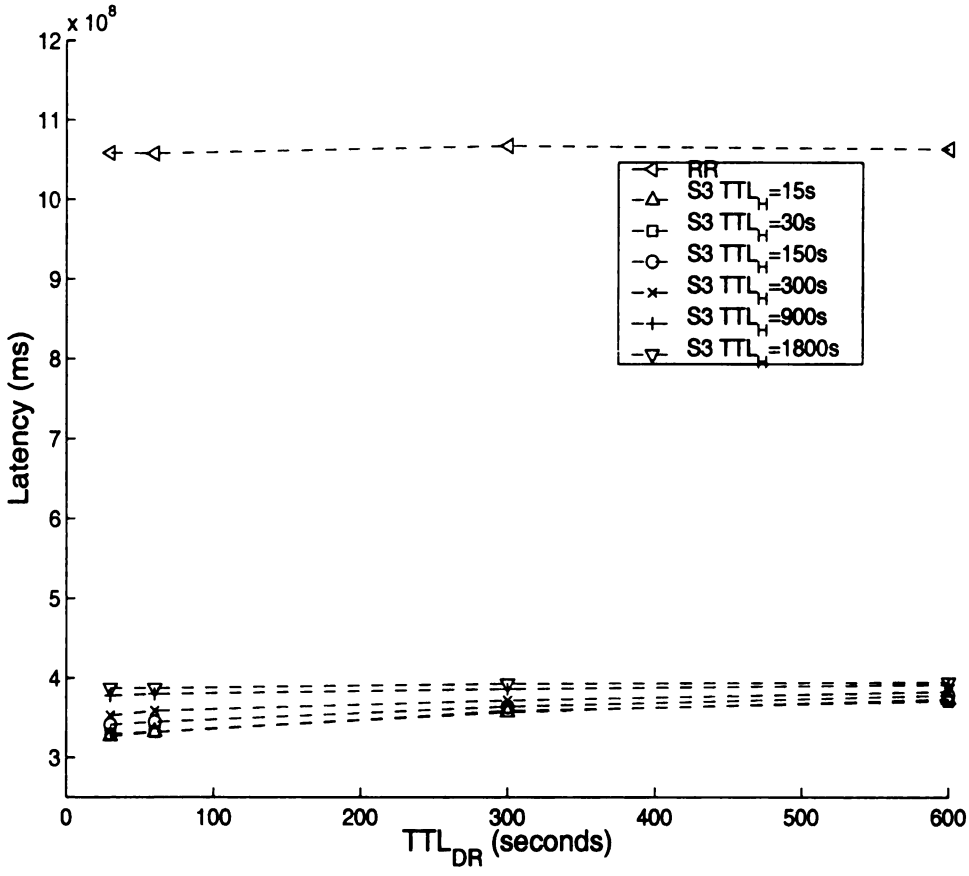

Figure 4.8: Latency vs TTL_{DR} (W= 0.3,mixed,$\lambda_H = 1/60$)

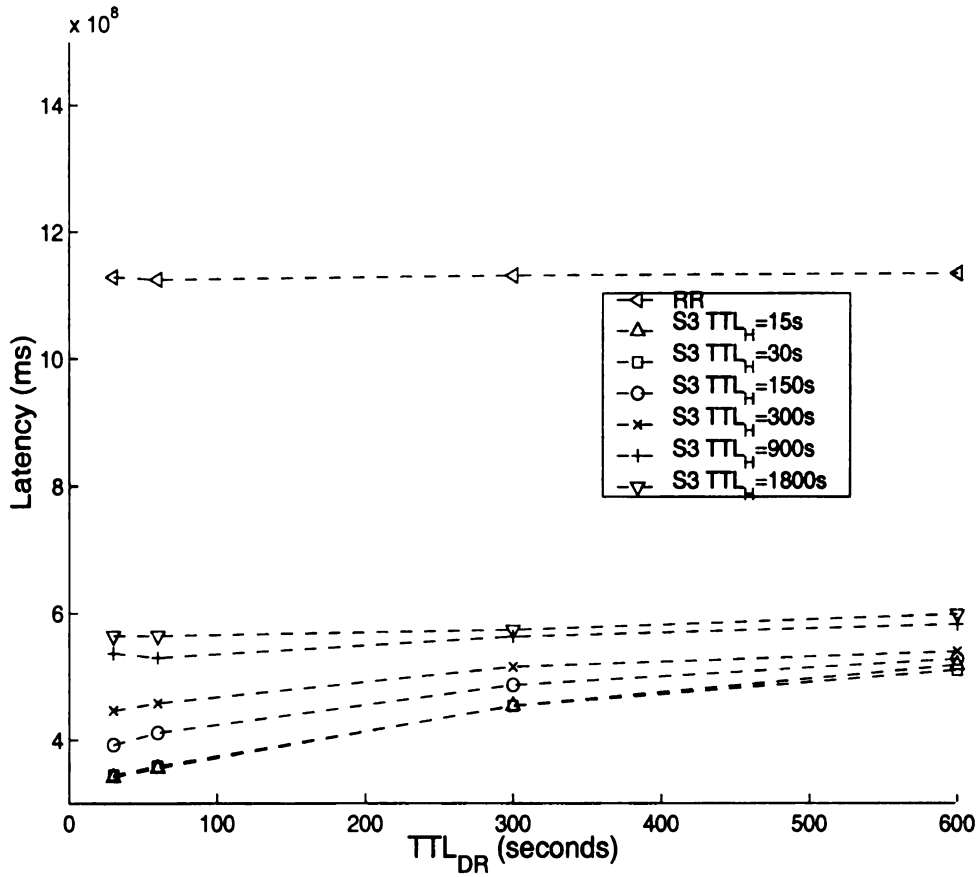

Figure 4.9: Latency vs TTL_{DR} (W= 0,Uniform,$\lambda_H = 1/60$)

if D is similar to Gamma. We see a large slope in Figure 4.9 and an almost flat slope in Figure 4.6. Figure 4.7 and Figure 4.8 have slopes in between the two extreme cases. In Figures 4.8 and 4.9, the performance of S3 under different TTL_H is also provided[2]. From Figure 4.9, we see that when the TTL_{DR} value is small, the DNS is able to capture the route metric quickly, and TTL_H should be set to a small value to exploit the information on the DNS. When the TTL_{DR} is large, the benefit of a small TTL_H decreases. For example, when $TTL_{DR} = 30$, the latency of S3 degrades about 65% when TTL_H increases from 30 to 1800. However, when $TTL_{DR} = 600$, the latency of S3 degrades about only 20% when TTL_H increases from 30 to 1800.

The results suggest that DNS should not treat all routes and route metrics in the same way. It should observe the pattern of the changes. A vigorously changing route deserves a high frequency of routing metric refreshing, while a relatively static, close to Gamma distribution route only needs to be refreshed with a larger interval. The TTL_H should be set according to the value of current TTL_{DR}.

4.4.4 Overhead Analysis

The overhead of the S3 method mainly occurs at the exchange of routing collection packets. However, compared to the total packets forwarded by a router, the traffic overhead introduced by S3 is negligible. The smallest TTL_{DR} in our simulation is 10 seconds, which is significantly smaller than what is used in a real route (current router information exchange happens every 30 seconds). Since the ICMP query and

[2]Figure 4.6 and Figure 4.7 do not include the performance of S3 under different TTL_H because the performance is very similar.

response each will need one IP packet, there will be only 2 IP packets exchanged every 10 seconds.

4.5 Deployment Issues

The deployment of S3 depends on the approach being selected. The attributes of the network under deployment are another key factor. If approach 1 described in Section 4.3 is used, the deployment process is rather simple. This involves only the client DNS server and the router having full knowledge about Internet routes, usually a BGP router. The DNS server needs to be manually configured to know the router it needs to query. Also, on both the DNS and the router, S3 extension of functions needs to be set up, which enables them to communicate with each other.

If approach 2 described in Section 4.3 is used, S3 requires support from routers on the default path from the client DNS to a full knowledge router. That path consists of the links from each router to its default higher level gateway router. The routers on that path have to understand the ICMP extensions S3 introduced. We can classify the client networks into two classes. One class includes large networks, where a full-knowledge BGP router is available. In such a network, all routers needed may be upgraded for S3 in one step since the managers have full control. The other class includes small networks. These networks are connected to ISPs and do not maintain a full-knowledge BGP router. In this case, the ISP of the small network has to upgrade their routers to support S3 first, then the small network can be upgraded afterwards.

4.6 Discussion

This section discusses other aspects of the S3 approach.

1. Fast metric collection and low level resource consumption.

   We discussed in the previous section that as the number of servers increases, other server selection methods suffer either a linear increment in latency of metric collecting or a linear increment in resource consumption. With S3, the latency of metric collection is independent of the number of replicated servers and may always be done by sending a query to the router. Therefore, it is fast and resource conservative.

2. High level of fault tolerance.

   Traditional DNS does not know the status of each replicated server. It may select the IP address at random or round robin. The server it chooses may be unreachable because the DNS has no knowledge of the routing information. With S3, a server that cannot be reached from a router's point of view will not be selected by the DNS.

4.7 Summary

In this chapter, a new mechanism to support replicated server selection, S3 (*smart server selection*), is presented. Extensions to current DNS and routers are proposed. With S3, users may choose the best server among the replicated servers that meet their preferences. The selection metric may be hops, latency, monetary cost, or a

combination of these. Simulation results show that S3 significantly reduces network resources usage and the latency perceived by the user.

Chapter 5

Static Scheduling and Client Redirection

In this chapter, we describe an approach, to address the load sharing problem for global replicated web services. It is complementary to the approach proposed in previous chapter.

5.1 Introduction

Global replicated services provide an economic way to provide high throughput and availability required by popular web sites such as Yahoo! and Altavista. In the last chapter we proposed an approach to select servers based on network metrics. In this chapter, we address the complementary problem: how to share the load between multiple replicated sites effectively and efficiently in order to provide reasonable performance.

Two categories of load sharing approaches, front-end dispatcher approaches such as Network Dispatcher [31] and DNS round-robin approaches such as those algorithms developed by Philip Yu [38], cannot be applied directly under global replicated services because they do not consider the network proximity. An approach that considers the network proximity and server load needs to be developed.

As the web evolves into the primary information and application platform, the web access patterns becomes more regular and predictable. This is especially true for membership-based applications, for example, mail.yahoo.com, my yahoo!, E*trade, etc. If such a site is replicated, the number of accesses from a specific user and where those accesses originated are usually more predictable. It is possible for these web sites to replicate only portal pages (interfaces) without the user-specific data. User-specific data can be migrated to a place where network proximity to the user is improved. However, we still want to keep the load on each web server balanced.

Based on these considerations, in this chapter we propose SSCR (Static Scheduling and Client Redirection)[55]. Our approach has advantages over existing approaches in the following ways:

1. It is applicable to both the NOW architecture and web servers dispersed on wide-area networks.

2. It introduces marginal overhead.

3. It scales well in comparison with other competitive approaches because load sharing is determined in a distributed manner without central control.

4. It avoids those drawbacks caused by aggressive DNS approaches which have short TTLs [40].

The rest of the chapter is organized as follows. The details of SSCR are given in section 5.2. In section 5.3, we describe our simulation model and performance results. Finally we summarize the conclusions and future work in section 5.4.

5.2 Our Proposed Approach: Static Scheduling and Client Redirection (SSCR)

The basic idea of SSCR is:

- A static scheduler at the web server site does static scheduling upon each T_s interval.

- Each client network is assigned a preferred web server. This is considered as a permanent assignment for a period of T_s. After T_s, each client network is rescheduled by the static scheduler and might be assigned to another web server. Generally T_s is quite large; for example, T_s can be one day.

- The client networks always try to access their preferred servers. If the preferred web server is overloaded, then the web server redirects some clients to other servers for a period of T_t in a distributed manner. This new assignment is temporary. T_t is quite small (2 minutes) relative to T_s. After T_t period, these redirected clients continue to access their preferred servers.

SSCR has the following steps:

1. Future load Prediction
2. Static Scheduling
3. Load Distribution
4. Client Redirection.

5.2.1 Future Load Prediction

A web server's access log currently provides the following information:

- hostname/IP address
- request date and time
- URI
- return code
- response length

To predict the future load (the number of accesses in the next day), the access history is examined and the number of accesses from each network (preferably C class networks) on each day is calculated. Different weights are given to information derived from logs of different days. Several values of weights have been examined. We have found that giving the most recent information the heaviest weight seems to perform appropriately. Therefore, SSCR puts weights of 0.6, 0.3, and 0.1 (these weights produce the best performance in our simulation) on the number of accesses from the

last three days except weekends, giving information from the latest day the largest weight. By having several days' history information, the variation can be reduced and prediction is more accurate.

Static Scheduling

Here we briefly discuss static scheduling for a NOW configuration that is not applicable for the wide-area configuration, which is discussed in section 5.2.4. Our method for static scheduling is quite simple. After the static scheduler obtains the future load prediction, it sorts all networks according to the number of accesses in descending order. Then the static scheduler schedules all the networks in this order. When the static scheduler schedules a network, it checks all available servers' assigned load and picks the server with the least load.

5.2.2 Load Distribution

After the static scheduler generates the assignments (client network, assigned server), it sends the assignments to all the servers to enforce this access policy. Each web server has a different IP address but shares one host URL (e.g., **www.mysite.com**). All web servers have all the documents served by this web site. Load distribution has been implemented at two places and static scheduling is enforced by the web servers themselves. When a client first accesses a web site, it does not have any knowledge about the web server group. It relies on the client side DNS server to find an IP address to access. The current implementation of DNS supports round-robin selection. First, the server side DNS server returns a list of available IP addresses to

the client DNS in a round robin fashion to DNS queries. Second, the client DNS uses the list to return to each individual host in the domain an IP address in the round-robin manner. When a web server receives an HTTP request from a client, it checks whether the client is assigned. If this client has not been assigned, the web server uses the client's network and look up the assignment table for the static assignment and returns the assigned server (IP address) to the client. From now on the client accesses the assigned server. This assignment is considered a permanent assignment and expires upon the next static scheduling point. For performance considerations, permanent assignments might be used across multiple sessions so that each browser does not ask the DNS several times during each T_s period. If a client is from a network that is not in the assignment table, the web server assigns the client in the round robin fashion by itself. In this case, the scheduling unit is not a network, but an individual host. It is possible that two clients from the same network access different servers. Usually these assigned hosts have a small amount of traffic so that the round-robin assignment works well.

It seems natural to enforce static scheduling based on domains instead of networks and static scheduling should be enforced by the server side DNS. We use the network as the scheduling unit for the following reasons:

1. When the server DNS receives a request, it does not know which DNS originated the request except when the client DNS operates in the "recursive mode". Expecting all DNS servers to operate in the recursive mode may have performance implications, and more importantly, this change cannot be made quickly.

2. A network offers a smaller scheduling granularity than a large domain, which might not be handled by a single web server. Since each small network in this domain may be assigned to different web servers, better performance is possible. Also, it is natural to base the measurement of the network proximity on network address instead of domain names because a domain might have several networks at several places, which have different network proximities to the web servers.

5.2.3 Client Redirection

Under static scheduling, the server's load is nearly balanced in the long run (T_s period). However, due to inherently bursty accesses, a server may be temporarily overloaded. In order to address this problem, SSCR relies on the HTTP redirect mechanism [46] to temporarily redirect accesses to other servers.

When a web server reaches its predetermined threshold, it sends an alarm message to the web server group by multicast to indicate it is overloaded. This prevents remote servers from redirecting clients to the overloaded server. The web server group does not need to exchange the load information periodically.

At the same time, the overloaded web server triggers the redirection mechanism, and redirects the clients that currently have accesses to the web server to other web servers not overloaded in a round-robin manner. This is done by returning a different IP address to each client. The redirected client accesses the temporary redirected server (temporary assignment) for period T_t. After that period, it continues to access the original server again.

It is very interesting to know if the document should be piggy-backed to the client in addition to the redirection, which would save the latency and a connection to the redirected server under HTTP 1.0. Since there is a fixed overhead associated with each request processing, only a small amount of extra time may be needed to serve this request. Also, if one server is overloaded, other servers might also have high load. Piggy-backing can prevent a client request from being bounced back several times before being served, even if it is a small HTML page. Therefore, a server may estimate the processing time. If the processing time is not too long, the server may serve it. Otherwise, the server will return a redirection code without serving it.

5.2.4 Wide Area Extensions

SSCR can be easily extended to wide area networks, in which the web servers are dispersed over a wide area connected by relatively slow links. In this case, accesses to the "nearest" server are preferred. Our definition of "nearest" server is that the number of hops between the client network and the server is minimal. Depending on the objectives, other definitions are possible.

Using our definition, after the static scheduler predicts the future load, the static scheduler tries to schedule these networks to their nearest server, while maintaining appropriate load distribution. For those networks that do not appear in the assignment table, two approaches might be used depending on the effectiveness of static prediction. The effectiveness of static prediction is determined by the web traffic characteristics. If there is less random traffic, round robin is sufficient. When there is

much more random traffic and the static scheduler is not effective, a more elaborate approach can be applied.

In such an approach, a central database of network distances from each web server to known client networks is collected and maintained by the network distance server (one part of the static scheduler). Each web server has an agent, which can find the network distance from itself to a client network when a request for network distance is received. When a web server receives a request from a client whose network is not assigned and there is no entry in the assignment table, it will ask the static scheduler for the appropriate server. The static scheduler returns the appropriate server if this client network has been assigned. Otherwise, the static scheduler utilizes the network distance information and load information on each web server to select a web server and return the decision to the web server that makes the request. The web server will add this assignment in its own assignment table.

The static scheduler either maintains or accesses an available knowledge database that contains the geographical information of each individual network.

Maintaining a knowledge database of the network distances from the client networks to the web servers is possible. *Traceroute* and *ping* [54] may be used off-line to collect network distance information. More efficient online approaches are proposed in [24, 50].

5.2.5 Fault-Tolerance and Scalability

DNS approaches raise concerns because if one server is down or needs to be maintained, those domains mapped to that server cannot access the services anymore and there is no simple method to invalidate these DNS mappings. This may be addressed by the browser implementation. When a browser calls the *gethostbyname*() system call, the DNS will return a list of IP addresses corresponding to a particular host name. The browser may try the second IP address available if the first IP address is not reachable. The problem of "server is not available" can be prevented.

5.2.6 Advantages of SSCR

The advantages of the SSCR can be summarized as follows:

1. It works for a globally replicated service.

2. It relieves requirements for small TTL values proposed in DNS-based approaches. SSCR saves a lot of DNS requests made by client DNS and improves the DNS system performance.

3. It avoids the front-end dispatcher. Because a front-end dispatcher accepts all the incoming traffic and makes a decision, the front-end dispatcher may become a bottleneck to the whole system. Such a dispatcher generally has to be backed up in case of primary dispatcher failure.

4. It can implement more advanced resource scheduling. Since the static scheduler controls how the web server resources are allocated, it is possible to implement

more advanced resource scheduling instead of load sharing. For example, with web server's priority queues, it is very easy for the static scheduler to support differentiated web services based on IP addresses and make coresponding resource allocations.

5.2.7 Implementation of Redirection

Redirection can be implemented in the following three ways. The first way is to rely on HTTP redirection. Current HTTP standards only support per URI-based redirection. In order to redirect web accesses to non-preferred servers, the preferred server has to do redirection on a per URI basis. This is undesirable.

A second way is to rely on connection redirection [41]. When the preferred server is overloaded, it redirects the TCP connection request (the SYN packet) to another web server.

The last choice is URI rewriting. In this approach, when a preferred server is overloaded, it changes all the URIs in the page it serves to point to another non-overloaded web server. The next time the user follows the link to access a document, it accesses the non-overloaded server. URI rewriting serves the purposes of both permanent assignment and temporary assignment.

5.3 Performance Evaluation

We compare the performance of our proposal with existing DNS approaches using trace driven simulation. Below, we discuss the performance metrics that we are interested in, the web trace file, and the performance results.

5.3.1 Performance metrics

Our performance comparison of approaches includes the following criteria:

1. *Percentage of intervals that at least one server is overloaded.* We count the number of intervals that at least one server is overloaded at any given moment during this interval. The server is overloaded when the number of accesses between two load evaluation points exceeds the server processing capacity. This is the same criteria as that used in [38].

2. *Number of redirections.* Since redirection causes a new connection to open on a web server, the number of redirections should be minimized.

3. *Number of DNS requests.* The number of DNS requests reflects the load on the DNS system. Keeping the number of DNS requests small is essential to the health of the DNS system.

5.3.2 Traces used in our study

We use three traces of web accesses that we obtained from various sources. The following table provides some information about the traces:

Table 5.1: Web Access Traces Used for Trace-Driven Simulation

Trace	*CLARK*	*CSE*	*EGR*
Duration (in days)	14	28	28
Total number of accesses	3,328,632	2,116,072	3,942,989
Average accesses /day	237,759	75,574	140,821

The CSE workload is a 4-week trace file from our Computer Science and Engineering Department's web server. The EGR workload is a 4-week trace file from our engineering college's web server. The CLARK workload is from a commercial Internet Service Provider in the metropolitan Baltimore-Washington DC area, which we downloaded the Internet.[1]

5.3.3 Simulation Setup

We use the traces of web access to run our simulations since we do not think models of web access are well understood to drive simulation studies. For our work at this point, we only simulate the NOW configuration of a web server. The network of each host is found in the following way: if the host is named, we strip off the first part of the name. If the host is an IP address, then we strip off the last byte. Piggy-backing is used in our simulation by default. Also, the latency of redirection is not considered. That means that redirection is seen at once by the client. Once the decision is made by a web server, the next request from that client will go to the newly assigned server. After the assignment is generated as described before, the simulator reads the trace file and obtains the number of accesses during each overload evaluation interval.

[1]Workload downloaded from `http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html`.

After the end of each interval, the simulator checks the number of accesses on each server. If any server's load is beyond its capacity, the number of overload intervals is incremented by 1. For each individual server, if its load is beyond the server capacity, it will carry those excessive requests to the next interval. A capacity Cs where 97% of time the web server is not overloaded in the trace is determined. Each server's capacity Cb is computed as Cs/n where n is the number of servers. The relative capacity is the ratio where the actual server's capacity compared to its Cb.

5.3.4 Simulation Results

Table 5.2: Load distribution policies studied.

Approach	*Description*
PP	Perfect Policy. In each interval, the number of accesses is averaged to each server. This is the best performance possible.
RR2	Two-tier Round-Robin.
RR2_alarm	RR2 with asynchronous alarm message.
ACR_RR_alarm	Round Robin with Client Redirection and alarm message.
SSCR	Static Scheduling with Client Redirection.

We simulated the five approaches listed in Table 5.2 using the three traces. *RR2*, *RR2_alarm*, *ACR_RR_alarm* (DNS-based approaches) are proposed in [40]. Figure 5.1 shows the simulation results for CSE workload. The x-axis is the server capacity relative to the base-line capacity. The y-axis is the percentage of periods that is overloaded.

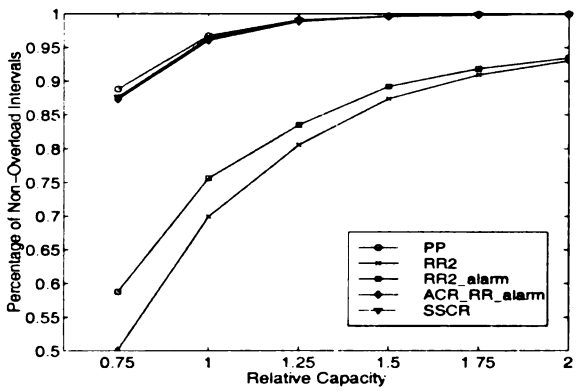

Figure 5.1: Performance of Different Approaches under CSE workloads

Under our load model and simulation configuration, we have the following observations:

1. Web traffic has some burstiness. It is very hard to prevent overload from happening by either static assignment or DNS mapping with reasonable short TTLs. This is verified by our performance simulations which are not shown in the figure.

2. The alarm message (overload notification) is very important. This can be verified by the fact that *RR2_alarm* is much better than *RR2*. For example, when there were 4 servers, em RR2_alarm is about 12% better than *RR2* using the EGR workload under the default configuration. This is confirmed by [38].

3. Redirection is an effective method and is sufficiently robust to deal with the distribution of web traffic. In the performance figure, we can clearly see that both SSCR and ACR_RR_alarm achieve performance similar to the perfect policy by redirection. Redirection improves the performance by more than 20% over

RR2_alarm in the default configuration. Also, it improves performance across all the server capacity configurations, which makes it a very effective way to prevent overload. SSCR is slightly better than ACR_RR_alarm.

4. In our simulation, especially when the server has limited capacity, piggy-backing is important. The reason is that when the capacity of the web server is limited, in order to maintain the appropriate load, the redirection traffic is much more than that of the web server which has larger capacity. Redirection traffic may put more burden on a capacity-starved web system and makes things worse. In our default configuration, without piggy-backing, the overloaded percentage decreases more than 15%.

5. Quick redirection (which is redirection made by a non-preferable server) is necessary if only an alarm message is used. The reason might be because if an alarm message is available, when a web server decides to redirect a client, it only knows that a specific web server is not overloaded, but it does not know what is the load on that server (it might be overloaded 1 second later). It may be the case that a web server will redirect clients to a server nearly overloaded. Quickly redirecting a client to a new server is needed even if this client has been redirected before.

We simulated the number of DNS requests introduced by DNS-based approaches and SSCR. If we assume TTL is one day in SSCR, we found out that RR based approaches generate 2.5 times of DNS requests than SSCR, and the DNS traffic of

RR based approaches is about 1/7 - 1/10 of total web requests made by all the clients. Compared to RR based approaches, SSCR introduces much less DNS traffic.

We also ran the simulations on the redirection traffic. Under the default configuration, SSCR has about 5.4% of redirections compared to the total number of web requests. The redirected accesses are about 20% of total web accesses. This number gives some sense of how the approach is performed in the wide area case. ACR_RR_alarm has about 6.3% of redirections compared to the total number of web requests.

5.4 Summary

In this chapter, we investigated the possibility of using web servers to enforce static scheduling and to redirect web traffic. Our performance simulations show that SSCR provide nearly optimal performance with little overhead. Redirection is very effective in dealing with the load distribution of web traffic and the alarm message is important in load distribution. Piggybacking is important when a server has limited capacity and quick redirection might be needed when only an alarm message is available as the indication of load on each web server. There are three future directions in our research. One is to investigate the trace files further to do better load prediction and static scheduling. Another direction is to evaluate the performance under the assumption that HTTP 1.1 is being used. The last direction is that we will simulate the performance of SSCR on a wide-area configuration, if possible.

Chapter 6

A Framework for Content-Aware Request Distribution

The replicated web server cluster is the most popular configuration used to meet the growing traffic demands imposed by the World Wide Web. However, for clusters to be able to achieve scalable performance as the cluster size increases, it is important to employ the mechanisms and policies for a "balanced" request distribution. As the web sites become the platform to conduct the business, it is important to protect the web server from overload and to provide service differentiation when different client requests compete for limited server resources. Mechanisms for intelligent request distribution and request differentiation help to achieve scalable and predictable cluster performance and functionality, which are essential for today's Internet web sites. This chpater discusses content-aware request distribution. The next chapter covers content-aware request processing to provide differentiated web service.

6.1 Introduction

Traditional request distribution tries to distribute the requests among the nodes in a cluster based on the parameters, such as IP addresses and port numbers, and some load information. Since the request distribution has the ability to check the packet header up to Layer 4 in the OSI network reference model (in this case, TCP) when it makes the distribution decision, it commonly referred to as Layer 4 request distribution.

Content-aware request distribution takes into account the content (URL name, URL type, or cookies, etc) when making a decision to which server the request is to be routed. Content-aware request distribution mechanisms enable smart, specially tailored routing inside the cluster and provide many benefits. Some of the benefits are:

1. It allows the content of a web site to be only partially replicated. Dedicated nodes can be set up to deliver different types of documents.

2. It provides support for differentiated Web Quality of Service (WebQoS).

3. It can significantly improve the cluster throughput. Previous work on content-aware request distribution [35, 36, 56] has shown that policies distributing the requests based on cache affinity lead to significant performance improvements compared to the strategies taking into account only load information.

Compared to traditional Layer 4 request distribution, the complexity of content-aware request distribution lies in the fact that HTTP is based on the connection-oriented

TCP protocol. In order to serve the client request (URL), a TCP connection has to first be established between a client and a server node. If the node cannot or should not serve the request, some mechanism has to be introduced to forward the request for processing to a capable node in the cluster. *TCP splicing* and *TCP handoff* are two mechanisms proposed to support content-aware request distribution.

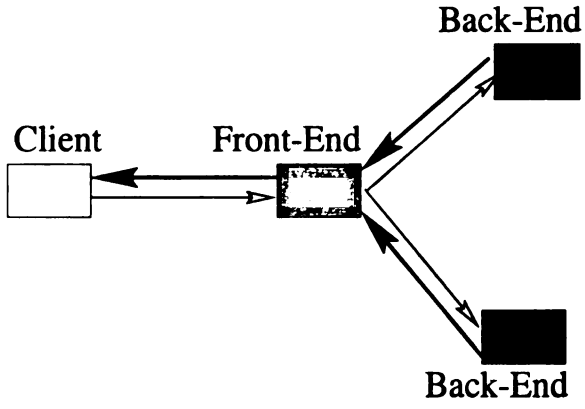

Figure 6.1: Traffic flow with TCP splicing mechanism

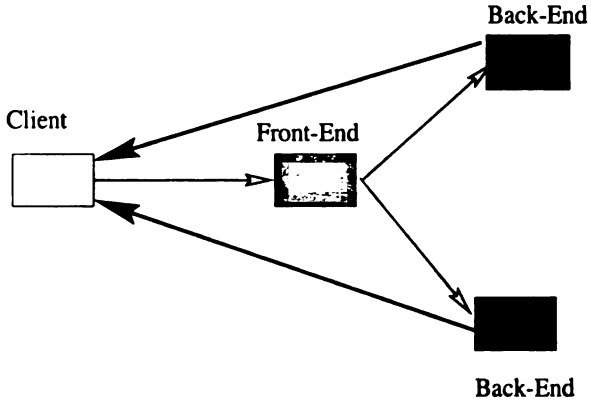

Figure 6.2: Traffic flow with TCP handoff mechanism

6.1.1 TCP Splicing

TCP splicing [57] is an optimization of the front-end relaying approach, with the traffic flow represented in Figure 6.1. In this cluster architecture, the front-end only dispatches the requests to the back-end node and it does not serve any requests at all. The response is forwarded to the client by the request distributor. A straightforward approach is to setup a content-aware proxy, which distributes the requests based on the URL. However,because each request and response has to be forwarded at the user level of the front-end, the throughput of the front-end is very limited. TCP splicing optimizes the front-end relaying approach by forwarding the data at lower levels in the protocol stack. For example, data may be forwarded at the socket layer, on top of IP layer or at IP layer. TCP sequence numbers and IP addresses are updated accordingly so that the whole process is transparent to the clients. By forwarding the packets at the lower level in the protocol stack, the performance of the front-end is greatly improved.

6.1.2 TCP handoff

The TCP handoff mechanism was introduced in [35] to enable the forwarding of back-end responses directly to clients without passing through the front-end, with traffic flow represented in Figure 6.2. After the front-end establishes the connection with the client and the request distribution decision is made, the established connection is handed off to the selected back-end node to serve the request. The TCP state related to the established connection is migrated from the front-end to the selected back-

end node. The main benefit of the TCP handoff mechanism compared against TCP splicing is that the back-end node can send the response directly to the client. The front-end is not involved in the response data forwarding. It has been shown in [36] that TCP handoff provides better performance and scalability than TCP splicing.

6.1.3 Content Aware Request Distribution in the STREAMS environment

STREAMS-based TCP/IP implementations, which is available in leading commercial operating systems, offer a framework to implement the TCP handoff mechanism as plug-in modules in the TCP/IP stack, and to achieve the flexibility and portability without much of a performance penalty.

In this chapter, we use three different applications to discuss specifics of content-aware request routing and related architectural design issues:

- a multi-language web site;
- partition-based cooperative web proxies;
- a simple e-commerce site.

Using these applications, we distinguish three most typical usage patterns of the TCP handoff mechanism. The usage pattern is defined by the fraction of the requests being handed off:

- *rare*-TCP handoff – when only a small fraction of the requests are handed off for processing to a different cluster node;

- *frequent*-TCP handoff – when most of the requests are forwarded for processing to a different node using the TCP handoff mechanism;

- *always*-TCP handoff – when the requests are always handed off for processing to a different cluster node.

This difference in the usage patterns leads to different trade-off decisions in the modular implementation of TCP handoff mechanisms. We discuss these trade-offs and propose a library of STREAMS modules implementing the TCP handoff functionality which addresses different cluster architectures and optimizes the TCP handoff mechanism for specific usage patterns.

The proposed approaches and a library of STREAMS modules[58, 59] have the following advantages:

- *portability*: the new modules are relatively independent of the implementation internals. New STREAMS modules are designed to satisfy the following requirements: all the interactions between new modules and the original TCP/IP modules are message-based, no direct function calls are made; and new modules do not change any data structures or field values maintained by the original TCP/IP modules. This enables maximum portability, so that the new modules may be ported to other STREAMS-based TCP/IP implementation very quickly.

- *flexibility*: the new modules may be dynamically loaded and unloaded as DLKM (Dynamically Loadable Kernel Module) modules without service interruption.

- *transparency*: no application modification is necessary to take advantage of new solutions. This is a valuable feature for applications where no source code is available.

- *efficiency*: the new modules are only peeking into the messages, with minimum functionality replication of the original TCP/IP modules.

The rest of the chapter is organized as follows. Section 6.2 gives a brief introduction to STREAMS and STREAMS-based TCP/IP implementations. Section 6.3 outlines three different web applications using content-aware request distribution and discusses the corresponding supporting architectures. Sections 6.4, 6.5, 6.6, argue that the efficient TCP handoff implementation in a STREAMS environment should take into account the TCP handoff usage patterns to minimize the overhead introduced by the TCP handoff mechanism.

6.2 STREAMS-Based TCP/IP Implementation

STREAMS is a modular framework for developing communication services. Each stream generally has a *stream head*, a *driver* and multiple optional *modules* between the stream head and the driver (see Figure 6.3). A stream is a full-duplex processing and data transfer path between a STREAMS driver and a process in user space. Modules exchange the information by *messages*. Messages can flow in two directions: *downstream* or *upstream*. Each module has a pair of *queues*: a *write queue* and a *read queue*. When a message passes through a queue, STREAMS modules for this queue

are called to process the message. The modules may drop a message, pass a message, change the message header, and/or generate a new message.

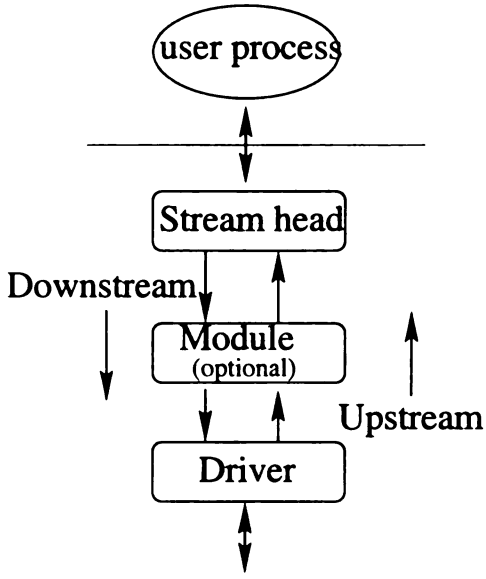

Figure 6.3: STREAMS

The stream head is responsible for interacting with user processes. It accepts the process request, translates it into appropriate messages, and sends the messages downstream. It is also responsible for signaling to the process when new data arrives or some unexpected event happens.

The STREAMS modules for a STREAMS-based TCP/IP implementation are shown in Figure 6.4. The Transport Provider Interface (TPI) specification [60] defines the message interface between TCP and the module on top of it. The Data Link Provider Interface (DLPI) specification [61] defines the message interface between the driver and the IP module. These specifications define the message format, valid sequences of messages, and semantics of messages exchanged between the neighboring modules.

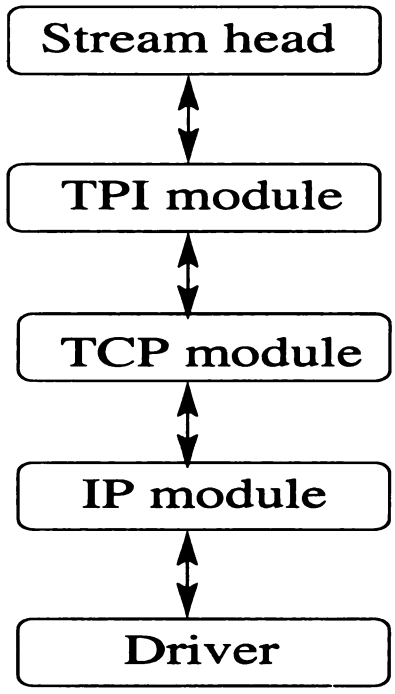

Figure 6.4: STREAMS-Based TCP/IP Implementation

When the TCP module receives a SYN request for establishing the HTTP connection on the listen stream, the TCP module sends a T_CONN_IND message upstream. Under the TPI specification, TCP should not proceed until it gets the response from the application layer. However, in order to be compatible with BSD implementation-based applications, the TCP module continues the connection establishment procedure with the client. When the application decides to accept the connection, it sends the T_CONN_RES downstream on the listen stream. It also creates another stream to accept this new connection, and TCP module attaches a TCP connection state to this new stream. The data exchange continues on the accepted stream until either end closes the connection.

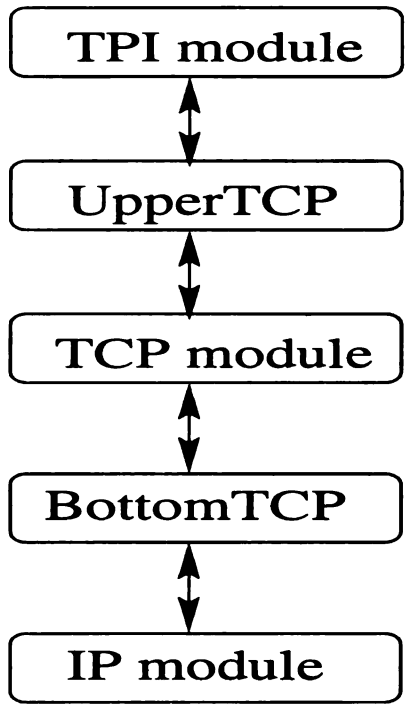

Figure 6.5: New Plug-in Modules for rare-TCP Handoff in STREAMS-Based TCP/IP Implementation

6.3 Content-Aware Request Distribution: Cluster Design and Application Specific Issues

TCP handoff is a mechanism which enables the intelligent routing of web requests between cooperative nodes either locally or in the wide area. In this section, we use three different applications to discuss content-aware request routing and request processing and related architectural design issues.

In our discussion of different cluster designs which can be used to implement content-aware distribution strategies, we adopt terminology proposed in [36]. There are three main components comprising a cluster configuration with content-aware request distribution: the *dispatcher* which implements the request distribution strategy, it de-

cides which web server will process a given request; the *distributor* which interfaces with the client and implements the TCP handoff that distributes the client requests to a specific web server; and the *web server* which processes HTTP requests.

1. Multi-language web site design.

Big sites have different language versions to service different client communities. For example, Yahoo has a site representation in different languages to serve different language groups. A client may access the same content in different languages. Another example is the big commercial companies which have on-line manuals in different languages to serve different communities. Due to the volume of traffic to these big sites, generally these sites are replicated at different places. It is neither economical nor necessary to replicate all different language versions of the same document in all the places at the same time. Typical practice is that servers at a particular place will only partially replicate a group of languages commonly used in the local community. It would be desirable that a client will be automatically directed to the right server to get the desired language document.

One simple and typical solution is to put a link on each page pointing to different versions of the same document in different languages, and let the user to select the right version manually. However, this method is not convenient for web page developers and very hard to manage in big sites.

The alternative solution is to apply a TCP handoff mechanism to automatically handoff the connection to the right server when the content is not present on the original server, and the selected server will respond to the client directly.

Each server in a cluster keeps a mapping (defined in a dispatcher module) to manipulate the URL according to some rules established in advance by the site administrators. In particular, these rules assign the specific language versions to be served by different servers in a cluster. This is a more flexible and convenient way to manage a multi-language web site.

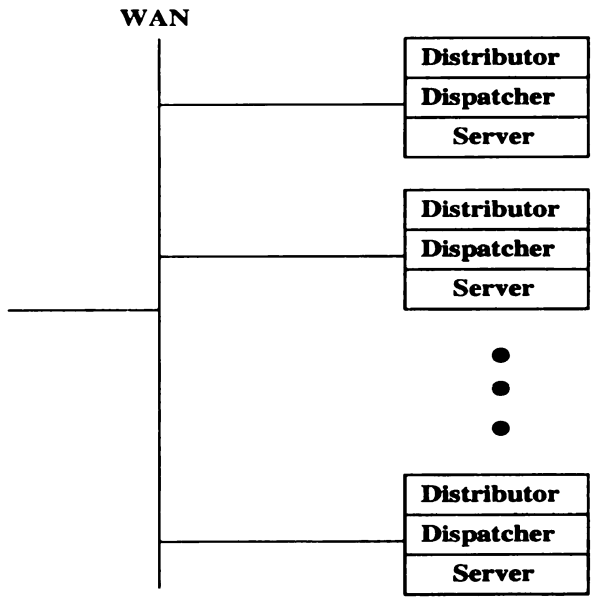

Figure 6.6: Web server cluster configurations with content-aware request distribution to support a multi-language web site design.

Figure 6.6 shows a cluster architecture to support a multi-language web site. Web servers are connected by a wide area network, and thus, handoff has to be implemented over a wide area network. In this architecture, the distributor component is co-located with the server and dispatcher components.

For simplicity, we assume that the clients directly contact the distributor, for instance via Round-Robin DNS. In this case, the typical client request is processed in the following ways:

1. The client web browser uses the TCP/IP protocol to connect to the chosen distributor;

2. The distributor component accepts the connection and parses the request;

3. The distributor contacts the dispatcher for the assignment of the request to a server;

4. The distributor hands off the connection using the TCP handoff protocol to the server chosen by the dispatcher;

5. The server takes over the connection using the handoff protocol;

6. The server application at the server node accepts the created connection;

7. The server sends the response directly to the client.

The specifics of this cluster architecture is that each node in a cluster has the same functionality: it combines the functions of distributor and web server. In other words, each node acts as a front-end and back-end node in providing TCP handoff functionality. For each web server, we expect that most of the HTTP requests are processed by the node accepting the connections (we refer to such requests as local), and hence TCP handoff happens relatively infrequently. We use the term *rare-TCP handoff* to specify this usage pattern. Under such a usage pattern, a goal for the rare-TCP handoff design and implementation is a minimization of the overhead imposed by the TCP handoff mechanism on local requests.

2. Partition-based cooperative web proxies design.

As a web proxy is a typical place where an entire intranet accesses the Internet, it is very easy for a single proxy to become a bottleneck. In order to provide a scalable proxy service, cooperative proxy clusters are commonly used. One kind of cooperative proxy is partition-based proxies[19]. In partition-based cooperative proxies, each proxy caches a *disjoint* subset of the documents. Partition-based web proxy clusters increase the number of cached documents and improve the cache hit ratios.

However, the same partition function has to be applied by the browser to contact the correct proxy for a particular URL. Implementing the partition function in a browser-transparent way is a difficult task in partition-based proxy clusters.

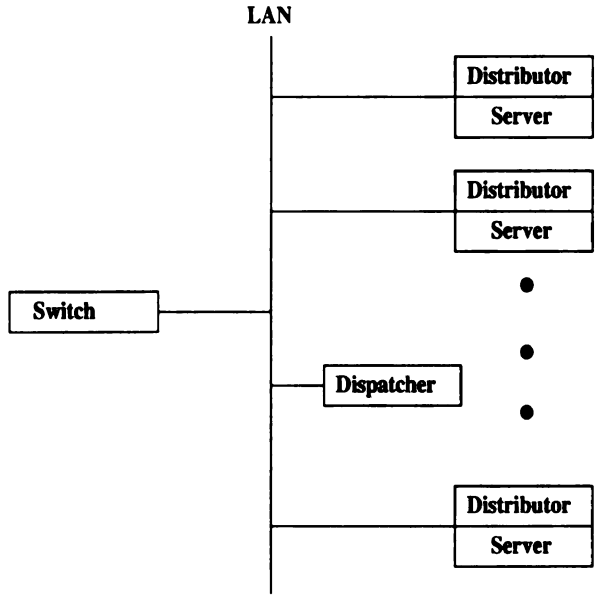

Figure 6.7: Partition-based cooperative web proxies design with content-aware request distribution

TCP handoff can be used to implement the partition function in a client transparent manner. Figure 6.7 shows a cluster architecture to support a partition-based proxy

cluster. In this architecture, the distributor component is co-located with the server component. The dispatcher component can be centralized as shown in Figure 6.7 or decentralized as shown in Figure 6.6 (typically, the decision is influenced by the choice of the partition function aiming to cache different files on a different servers). Round-Robin DNS or a front-end Layer 4 switch can be used to direct the traffic to the proxies in the cluster. When an HTTP request comes in, the proxy consults with a dispatcher module to determine which server is to serve the request. If the request is to be served by another proxy in the cluster, the original proxy hands the connection off to the proxy designated to process this request. The serving proxy will send the response back to the client directly.

This cluster architecture is similar to the architecture considered above for multi-language web site design. The difference is in the usage pattern of the TCP handoff. Let N be the number of nodes in the partition-based proxy cluster. Statistically, each node in the cluster will be serving only $1/N$ of the requests locally, while forwarding $(N-1)/N$ of the requests to the different nodes in the cluster using the TCP handoff mechanism. We use the term *frequent-TCP handoff* to specify this usage pattern. Under such a usage pattern, an efficient frequent-TCP handoff design and implementation should minimize the TCP handoff overhead for remote request processing.

3. E-commerce site design.

The HTTP protocol is stateless, i.e. each request is processed by the web server independently from the previous or subsequent requests. In an e-commerce environment, the concept of *session* (i.e. a sequence of requests issued by the client) plays an es-

sential role [62, 63]. For a session, it is important to maintain state information from the previous interactions between a client and a server. Such state might contain the content of the shopping cart or a list of results from the search request. Thus, when the user is ready to make a purchase, or is asking for the next 10 entries from the search list, the state information from the previous request must be retrieved. For efficient request processing and *session integrity*, it is desirable to send the client request to the same server. One popular scheme for handling the state on the web is cookies. Content-aware request routing provides a convenient mechanism to support session integrity (the other common term for this is a "sticky" connection).

Figure 6.8 shows a cluster architecture to support a simple e-commerce site. In this architecture, the front-end node has co-located distributor and dispatcher modules to support session integrity, i.e. based on the cookie in the HTTP header, it sends subsequent requests belonging to the same session to the initially chosen back-end server.

The specifics of this cluster architecture is that front-end and back-end nodes now have different functionality: front-ends combine the functions of distributor and dispatcher, while the back-ends perform as web servers. The front-end node checks the cookie and decides which back-end server is to process the request. The distributor module always hands off the connection to the appropriate back-end server. The front-end node never processes the request. We use the term *always-TCP handoff* to specify this usage pattern. Under such a usage pattern, the design and implementation of the always-TCP handoff is very different from the previously discussed cases of rare- and

frequent-TCP handoff. The crucial difference is that front-end and back-end nodes play very different roles in this architecture.

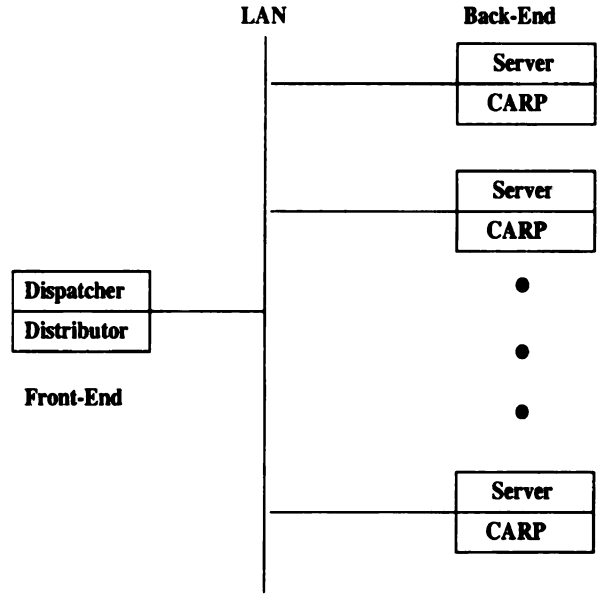

Figure 6.8: Web server cluster configurations to support session integrity and differentiated services for e-commerce site design

Web server QoS is very important for business web sites. When a web site is overloaded, it is desirable that important requests get preferable service [62], or some form of the admission control mechanism is employed [63]. Content-aware request processing (CARP) proposes an interesting framework to implement differentiated services in a web server. For example, different request scheduling and processing policies could be deployed based on client or request priority. Session-based admission control, introduced in [63], can be easily implemented using CARP on the back-end web server nodes.

Thus, content-aware front-end nodes in the cluster configuration shown in Figure 6.8 provide the session integrity mechanism, while back-end web server nodes can de-

ploy differentiated services such as request classification, session management, request queuing, admission control, and/or request scheduling.

This concludes our discussion of three different applications employing content-aware request processing and TCP handoff for different purposes. We illustrated the specifics of the TCP handoff usage patterns in these applications. Efficient implementation of the correspondent TCP handoff mechanism should take into account these usage patterns. The TCP handoff modules may be developed at different places in a TCP/IP stack to implement the content-aware request distribution, according to the architecture and workload characteristics.

Section 6.4 will present a detailed design of rare-TCP handoff. Using this detailed description, we discuss what should be done differently for efficient implementation of frequent-TCP handoff and always-TCP handoff.

6.4 Rare-TCP Handoff Design

In the cluster architecture shown in Figure 6.6, each node performs both front-end and back-end functionality: the distributor is co-located with the web server. We use the following denotations: the distributor-node accepting the original client connection request is referred to as FE (Front-End). In the case, where the request must be processed by a different node, this node receiving the TCP handoff request is referred to as BE (Back-End).

Two new modules are introduced to implement the functionality of rare-TCP handoff for multiple language web sites as shown in Figure 6.5. According to the relative

position in the existing TCP/IP stack, we refer to the module directly on top of the TCP module in the stack as UTCP (UpperTCP), and the module directly under the TCP module as BTCP (BottomTCP).

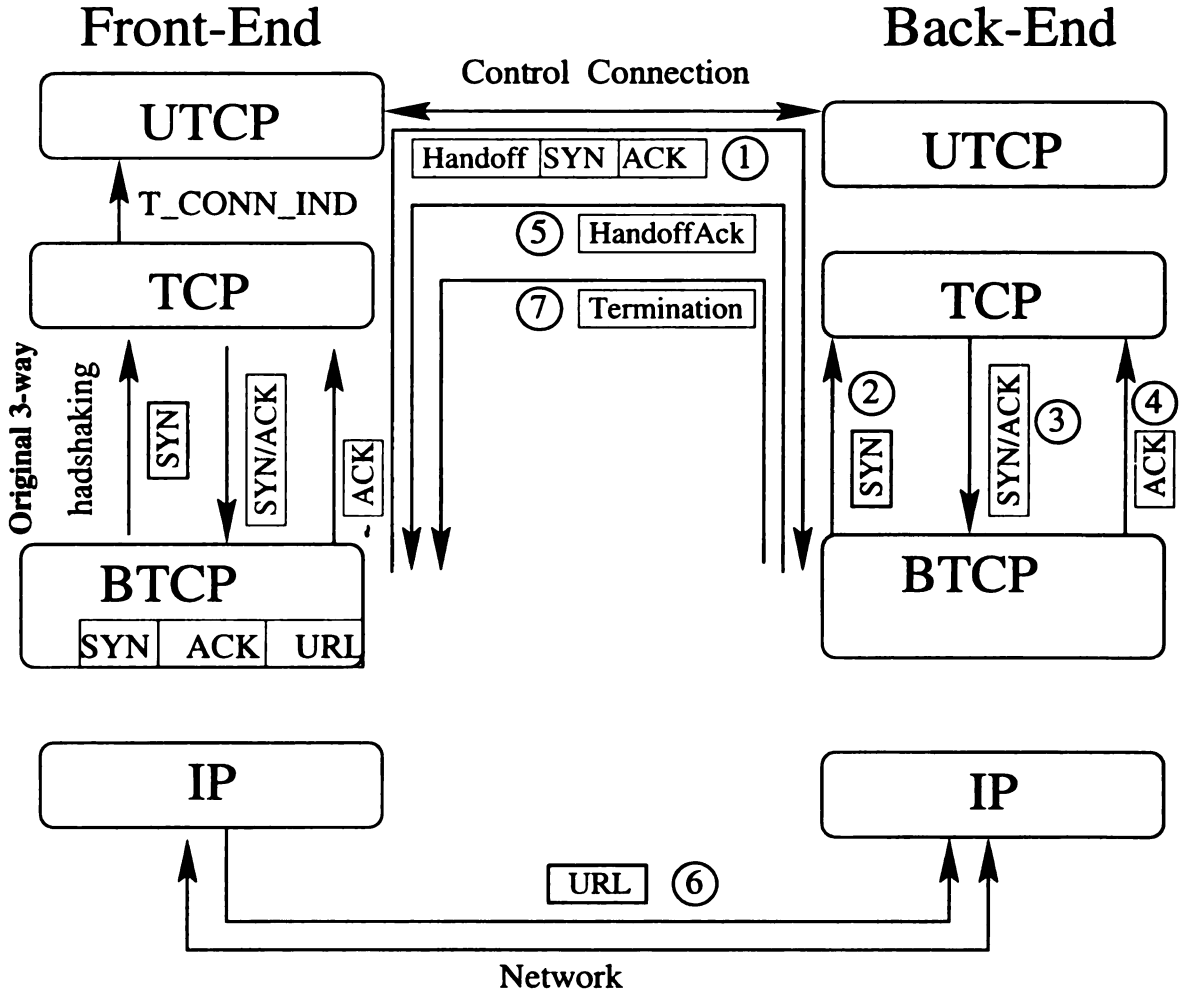

Figure 6.9: Remote Request Processing Flow During rare-TCP Handoff

These two modules provide a wrapper around the current TCP module. In order to explain the proposed modular TCP handoff design and its implementation details, we consider a typical client request processing. There are two basic cases:

remote request processing, i.e. when the front-end node accepting the request must

handoff the request to a different back-end node assigned to process this request; and *local request* processing, i.e. when the front-end node accepting the request is the node which is assigned to process this request.

First, we consider *remote request* processing. There are six logical steps to perform the TCP handoff of an HTTP request in rare-TCP handoff:

1. Finish 3-way TCP handshaking (connection establishment) and get the requested URL.

2. Make the routing decision: decide which back-end node is assigned to process the request.

3. Initiate the TCP handoff process with the assigned BE node.

4. Migrate the TCP state from the FE to the BE node.

5. Forward the data packets.

6. Terminate the forwarding mode and release the related resources on the FE after the connection is closed.

Now, we describe in detail how these steps are implemented by the newly added UTCP and BTCP modules and original TCP/IP modules in the operating system.

1. *3-way TCP handshake.* Before the requested URL is sent to make a routing decision, the connection has to be established between the client and the server. The proposed design depends on the original TCP/IP modules in the current operating system to finish the 3-way handshake. In this stage, $BTCP_{FE}$ allocates

a connection structure corresponding to each connection request upon receiving a TCP SYN packet from the client. After that, $BTCP_{FE}$ sends the SYN packet upstream. Upon receiving a downstream TCP SYN/ACK packet from the TCP_{FE} module, $BTCP_{FE}$ records the initial sequence number associated with the connection and sends the packet downstream. After $BTCP_{FE}$ receives an ACK packet from the client, it sends the packet upstream to TCP_{FE}. During this process, the $BTCP_{FE}$ emulates the TCP state transitions and changes its state accordingly.

In addition to monitoring the 3-way TCP handshaking, $BTCP_{FE}$ keeps a copy of the incoming packets for connection establishment (SYN packet, ACK to SYN/ACK packet sent by the client) and URL (Figure 6.9), for *TCP state migration* purposes, which are discussed later.

Also, because the TCP handoff should be transparent to server applications, the connection must not be exposed to the user level application before the routing decision is made. $UTCP_{FE}$ intercepts the T_CONN_IND message sent by TCP_{FE}. TCP_{FE} continues the 3-way handshake without waiting for explicit messages from the modules on top of TCP.

2. *URL parsing.* $BTCP_{FE}$ parses the first data packet from the client, retrieves the URL and makes the distribution decision.

3. *TCP handoff initiation.* A special communication channel is needed to initiate the TCP handoff between FE and BE. A *Control Connection* is used for this purpose between two $UTCP_{FE}$ and $UTCP_{BE}$ as shown in Figure 6.9. This

control connection is a pre-established persistent connection set up during the cluster initialization. Each node is connected to all other nodes in the cluster. The TCP handoff request is sent over the control connection to initiate the handoff process. Any communication between $BTCP_{FE}$ and $BTCP_{BE}$ modules goes through the control connection by sending the message to the $UTCP$ module first (see Figure 6.9). After $BTCP_{FE}$ decides to handoff the connection, it sends a handoff request to the $BTCP_{BE}$ (Figure 6.9, step 1). The SYN and ACK packets from the client and the TCP initial sequence number returned by TCP_{FE} are included in the message. $BTCP_{BE}$ uses the information in the handoff request to migrate the associated TCP state (steps 2-4 in Figure 6.9, which are discussed next). If $BTCP_{BE}$ successfully migrates the state, an acknowledgement is returned (Figure 6.9, step 5) $BTCP_{FE}$ frees the half-open TCP connection upon receiving the acknowledgement by sending a RST packet upstream to TCP_{FE} and enters forwarding mode. $UTCP_{FE}$ discards the corresponding T_CONN_IND message when the T_DISCON_IND is received from the TCP_{FE}.

4. *TCP state migration.* In the STREAMS environment it is not easy to get the current state of a connection at TCP_{FE}, to transfer it and to replicate this state at TCP_{BE}. First, it is difficult to obtain the state out of the black box of the TCP module. Even if this could be done, it is difficult to replicate the state at BE. TPI does not support schemes by which a new half-open TCP connection with predefined state may be opened. In the proposed design, the

half-open TCP connection is created by replaying the packets to the TCP_{BE} by the $BTCP_{BE}$. In this case, the $BTCP_{BE}$ acts as a client(Figure 6.9). $BTCP_{BE}$ uses the packets from $BTCP_{FE}$, updates the destination IP address of SYN packet to BE and sends it upstream (Figure 6.9, step 2). TCP_{BE} responds with SYN-ACK(Figure 6.9, step 3). $BTCP_{BE}$ records the initial sequence number of BE, discards SYN-ACK, updates the ACK packet header properly, and sends it upstream (Figure 6.9, step 4).

5. *Packet forwarding.* After the handoff is processed successfully, $BTCP_{FE}$ enters forwarding mode. It forwards all the pending data in $BTCP_{FE}$, which includes the first data packet (containing the requested URL) (Figure 6.9, step 6) It continues to forward any packets on this connection until the forwarding session is closed.

   During the packet forwarding step, $BTCP_{FE}$ updates (corrects) the following fields in the packet: 1) the destination IP address (to BE's IP address); 2) the sequence number of the TCP packet; 3) the TCP checksum.

   For packets that are sent directly from BE to the client, the $BTCP_{BE}$ module updates (corrects): 1) the source IP address (to FE's IP address); 2) the sequence number; 3) TCP checksum. After that, $BTCP_{BE}$ sends the packet downstream.

6. *Handoff connection termination.* Connection termination frees state at BE and FE. The data structures at BE are closed by the STREAMS mechanism. $BTCP_{BE}$ monitors the status of the handed off connection and notifies the

$BTCP_{FE}$ upon the close of the handed off connection in TCP_{BE} (Figure 6.9, step 7). $BTCP_{FE}$ releases the resources related to the forwarding mechanism after receiving such a notification.

Local request processing is performed in the following way. After the $BTCP_{FE}$ finds out that the request should be served locally, the $BTCP_{FE}$ notifies $UTCP_{FE}$ to release the correct T_CONN_IND message to upper STREAMS modules, and sends the data packet (containing the requested URL) to the original TCP module (TCP_{FE}). $BTCP_{FE}$ discards all the packets kept for this connection and frees the data structures associated with this connection. After this, $BTCP_{FE}$ and $UTCP_{FE}$ send packets upstream as quickly as possible without any extra processing overhead.

6.5 Frequent-TCP Handoff Design

Partition-based web proxy clusters demonstrate a different usage pattern of TCP handoff: HTTP requests are more likely to be handed off compared to rare-TCP handoff, as we pointed out in the section 6.3. This difference leads to a different TCP handoff design. In this design, the overhead of remote request should be minimized. The flow of the remote request processing is illustrated in Figure 6.10.

The additional modules are introduced at the same positions in the protocol stack as in the previous design, and are referred as B_FTCP and U_FTCP module, to indicate these modules have different functionalities.

1. *Connection setup.* Under rare-TCP handoff design, the connection-related resources in the TCP module are released by the RST message when the handoff

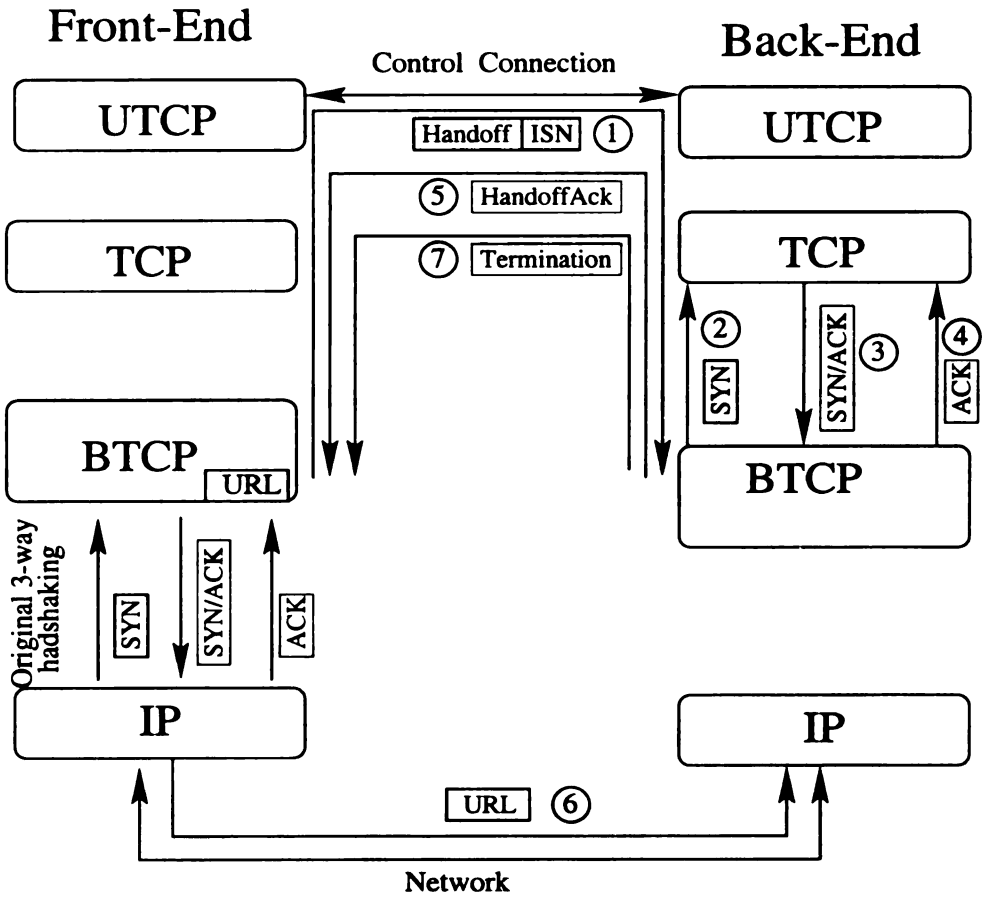

Figure 6.10: Remote Request Processing for Frequent-TCP Handoff

is successful. In the case of frequent TCP handoff, it is inefficient to establish a TCP connection with the TCP module at the front-end node and then free the connection most of the time. Connection setup (the original 3-way handshaking) is reimplemented by the B_FTCP_{FE} module to trigger the client to send the URL. The B_FTCP_{FE} also has better control on TCP options. After B_FTCP_{FE} receives the URL and makes the decision, B_FTCP_{FE} may initiate the handoff connection through the control connection as before (Figure 6.10, step 1). However, no packet is shipped along the persistent connection in the handoff request at this time. B_FTCP_{FE} may gather necessary information (for exam-

ple, Initial Sequence Number (ISN), etc.) from the connection establishment packets, and then B_FTCP_{BE} may construct these packets from the information provided in the handoff request, and replay the packets locally at the back-end node (Figure 6.10, step 2-4). An acknowledgement is returned by the BE to the FE through the control connection to indicate if the handoff is successful (Figure 6.10, step 5). If the request is processed locally at the front-end, the kept connection establishment packets are replayed to the local TCP module to create the necessary state.

2. *Packet forwarding.* Packets may be forwarded to the selected server on top of the IP layer, in the IP layer, or under the IP layer, depending on the cluster configuration and the ratio between the local traffic and forwarding traffic. While the B_FTCP module may forward the packets on top of IP layer, similar functionality can be achieved by inserting a module on top of the device driver. When B_FTCP is implemented on top of the device driver, and all the back-end nodes are located on the same LAN (as in the described partition-based proxy application), it is possible for a cluster to have a virtual IP address, each back-end node is uniquely identified by MAC address, and the packet is forwarded by filling in the right MAC address. This avoids Network Address Translation (NAT) on the front-end and NAT on the back-end node for outgoing traffic. Upon receiving the DLPI message from the device driver, B_FTCP_{FE} changes the DLPI message format and destination MAC address and sends the message downstream.

When the forwarding packets must traverse a router or a WAN, the packet's destination may be changed to the selected server's IP address and a proprietary protocol may be developed to carry the packet's original IP address to the selected server so that the response packet's source IP address may be updated accordingly. B_FTCP_{FE} updates the packet's IP address to the selected server's IP address, and sends the packet upstream. The IP_{FE} forwards the packet according to its routing tables to the back-end node. B_FTCP_{BE} has to manipulate the TCP header anyway and updates the initial sequence number and TCP checksum.

The original TCP connection establishment can be done in two different modules: either in the operating system TCP module or the B_FTCP module. When the TCP module is used to establish the connection with the client, the initial sequence number is correct so local request may be sent to the client directly without any packet header manipulation. If the B_FTCP module is used, because of the local handoff, the initial sequence number used by the TCP module and BTCP module might be different, and therefore B_FTCP has to update the initial sequence number and TCP checksum for every outgoing packet for local requests. In this design, we improve the remote request processing at a price of an additional small penalty for local request processing.

3. *Handoff connection termination.* In order to successfully and robustly free the front-end forward session, either the back-end or the front-end node has to observe the two-way TCP control traffic. In the rare-TCP handoff design, the

$BTCP_{BE}$ sees the two-way traffic and knows the status of the TCP connection and sends the notification to the front-end upon termination of the connection.

In the frequent-TCP handoff design, connection termination depends on where the modules are inserted. If the modules are on top of IP level, the rare-TCP handoff termination approach is more elegant. If the functionality of the B_FTCP is implemented by a module inserted on top of the device driver, the termination approach described in the next section for always-TCP handoff is better[1].

6.6 Always-TCP Handoff Design

In always-TCP handoff, there are two kind of nodes, the dedicated front-end node and the back-end web servers. The purpose of the front-end node is to trigger the client to send the URL, and then handoff the connection to the selected server. The request flow of always-TCP handoff is shown in Figure 6.11. In this configuration, only one module is introduced, B_ATCP module, at both the front-end and the back-end nodes. The difference between always-TCP handoff and the previously described rare- and frequent-TCP handoff is as follows.

1. *Connection setup.* The B_ATCP_{FE} implements the connection setup function as in B_FTCP_{FE} module. Packets are exchanged to establish the connection with the client and to get the URL. State migration is done by replaying the

[1]The module on top of the device driver may not see two-way traffic in case multiple network interface cards exist, and incoming traffic and outgoing traffic go through different interfaces.

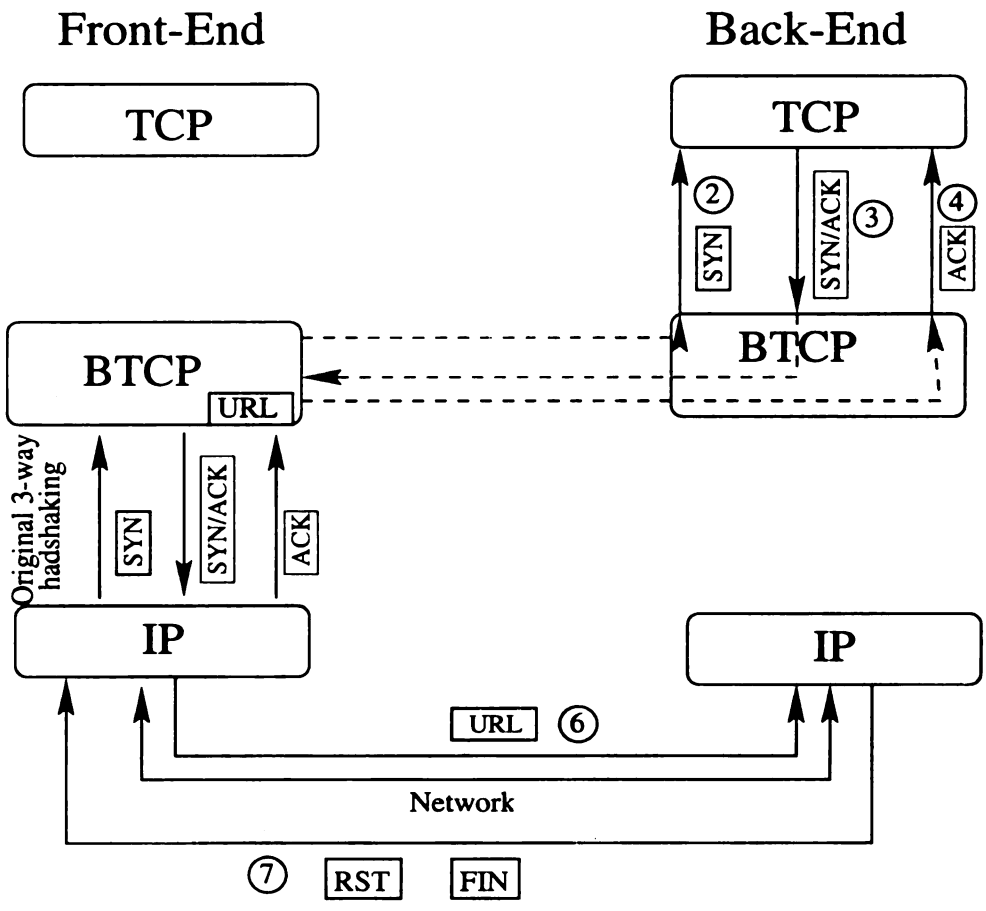

Figure 6.11: Request Processing for always-TCP Handoff

packets between the front-end and the back-end node. Since all web traffic is handoff traffic, a TCP SYN packet arrived at web server listening port indicates a handoff request is initiated (Figure 6.11, step 1) and B_ATCP_{BE} sends the SYN packet upstream (Figure 6.11, step 2). A TCP persistent connection may not be needed because there is no need to tell the back-end node that a particular flow is the handoff connection.

The SYN-ACK packet from TCP_{BE} is intercepted by B_ATCP_{BE} to the front-end by changing IP address (Figure 6.11, step 3). The B_ATCP_{BE} receives the ACK and records the initial sequence number returned by the B_ATCP_{FE}

(Figure 6.11, step 4). No acknowledgement is needed. the URL is forwarded as before in step 6.

Packet forwarding should be done as quickly as possible. In this configuration, it might be better to forward the packet on top of the device driver. Also virtual IP address should be used to avoid network address translation at the front-end because it is easy for the front-end to become the bottleneck for the whole cluster.

2. *Handoff connection termination.* The handoff connection is closed in the following fashion. The B_ATCP_{BE} intercepts the TCP control packets (packets with flags on, for example, RST, FIN) and sends it to the B_ATCP_{FE} (step 7). The B_ATCP_{FE} records the connection progress and relays the packets to the client. Data traffic goes directly to the client. The front-end sees two ways traffic and may keep track of handoff connection status and closes the connection in timely manner.

6.7 Performance evaluation

The rare-TCP handoff design is prototyped on HP-UX platform. The HP-UX operating system supports *Dynamically Loadable Kernel Modules* (DLKM). As we described before, two new modules are developed: BTCP and UTCP.

In this section, we report the performance and overhead introduced by modular rare-TCP handoff design.

6.7.1 Measurement testbed

A small testbed is set up to evaluate the modular TCP handoff mechanism. A 100 Mbps Ethernet switch is used to connect four HP-UX workstations. Two machines (HP J6000 workstations) are used as web servers. These two HP J6000 workstations each have the following configurations: dual 553M HZ CPU, 1G bytes of memory, 120 MHZ central bus, 512KB/1MB cache and more than 16 Gigabytes of disk space. The tool *httperf* is used to benchmark the web server performance and latency. Two client machines are used to drive the measurement.

6.7.2 Performance Measurement

This section shows the TCP handoff measurement. The four configurations we measured are described in table 6.1.

We measured the latency and throughput with different document sizes: 120 bytes, 1200 bytes, 6000 bytes and 12000 bytes.

Table 6.1: Configuration measured

Configuration	Description
No modules	Standard original operating system protocol stack.
Dummy modules	Two dummy modules which forward the packets without any further processing.
Handoff-local	Local handoff. After the decision is made, the packets are forwarded without further processing.
Handoff-remote	Remote handoff. Requests will be handed off to another server.

Table 6.2 shows the TCP handoff latency performance.

As we may see, the two dummy modules introduce a negligible delay in all the configurations on all file sizes. TCP handoff introduces a little extra delay. For example, if the document size is 120 bytes, the latency is increased from $0.9ms$ to $1.1ms$. The latency does not depend on the file size significantly. This is because after the decision is made, the only delay is the forwarding delay. In the case of handoff, the delay increases proportionally with the document size. This is because as the document size increases, the number of forwarded packets (ACK packets) increase.

Table 6.2: Latency of TCP handoff (ms)

Configuration	120	1200	6000	12000
No modules	0.8	1.2	1.4	2.8
Dummy modules	0.9	1.2	1.4	2.8
Handoff-local	1.1	1.3	1.5	2.9
Handoff-remote	1.6	2.1	2.8	4.4

Table 6.3 shows the TCP handoff throughput performance. As we may see, there is overhead associated with the insertion of two dummy modules. This is the overhead of the STREAMS mechanism. In our approach, the two dummy modules degraded the performance by 5.6% if the document size is 120 bytes. This overhead is per packet overhead. Because this overhead is proportional to the size of the document, the throughput degradation increases as document size becomes larger. In the case of local handoff and a document size of 120 bytes, the throughput degraded about 9%. When the document size reaches 6000 bytes, the network becomes saturated (about 78Mbps web server throughput) and becomes the bottleneck. We were not able to measure the overhead under such configurations. In the case of TCP handoff, the performance degraded about 16% if the document size is 120 bytes. The network

is saturated again when the document size is 6000 bytes; the throughput is 770 requests/second.

Table 6.3: Throughput of TCP handoff (req/s)

Configuration	120	1200	6000	12000
No modules	1430	1385	850	800
Dummy modules	1350	1300	800	800
Handoff-local	1300	1260	800	800
Handoff-remote	1200	1130	770	770

6.8 Summary

Research on request distribution and request differentiation receives much attention from both industry and academia. Providing scalable and predictable service is essential for future Internet web sites. Content-aware request processing enables intelligent routing and request processing inside a web cluster to support the quality of service requirements for different types of content and to improve overall cluster performance. A STREAMS-based TCP/IP implementation, which is available in leading commercial operating systems, offers a framework to implement the TCP handoff mechanism as plug-in modules in the TCP/IP stack.

In this chapter, we use three different applications to discuss specifics of content-aware request routing and related architectural design issues. The difference in the usage patterns leads to different trade-off decisions in the modular implementation of TCP handoff mechanism. We discuss these trade-offs and propose a library of STREAMS modules implementing the TCP handoff functionality which addresses

different cluster architectures and optimizes the TCP handoff mechanism for specific usage patterns.

The library of STREAMS modules proposed in this chapter offers a set of attractive benefits: *portability*, *flexibility*, *transparency*, and *efficiency* to support scalable web server cluster design and smart, specially tailored request routing inside the cluster. More importantly, these modules allow much easier deployment of the solution in commercial systems.

Chapter 7

Web Quality of Service and Differentiated Service

Providing a positive user navigation experience is very important for web site operators and is especially important for business web sites. It has been reported that eight seconds is the practical critical point for a web site. A healthy web site should deliver the content of a web page within 8 seconds.

As we discussed in chapter 1, the web request serving process involves a web server system, a network system and a client system. As we pointed out in chapter 1, the web server system and the network system are most likely to be the bottlenecks.

In this chapter, we first discuss some of the approaches available to address web differentiated service in the web server system and outline the design and implementation of such approaches in both a FreeBSD-like TCP/IP stack and a STREAMS-based TCP/IP stack. We then briefly discuss network QoS and differentiated service. After

that, we discuss end-to-end QoS. At last, we discuss an interesting approach: content delivery networks and its relationship to end-to-end web QoS.

7.1 Differentiated Web Service at Web server systems

Differentiated web service at web server systems refers to the situation where certain requests receives preferred treament in terms of resource allocation than do other web requests when the site capacity may not be able to satisfy all the requests with acceptable performance. Differentiated service assumes that access to a web site may be divided into several classes according to some criteria. In chapter 2, we outline two such criteria: one criteria is to classify the users into several classes, which we refer to as client-oriented differentiated service. In this case, users in different classes might receive different levels of service when the resource at the site is limited, no matter which URL the users access. The other criteria is to classify the URLs into several classes, and accesses to different URLs receive different levels of service. We refer to this as service-oriented differentiated service. In practice, the two criteria may be combined. Other criteria may be specified when such a requirement arises. Before we discuss the mechanisms to provide various levels of differentiated service, we take a look at the main steps involved to serve a web request. Web request processing may be logically divided into four sequential stages:

1. TCP connection setup with the web site. Web access is based on the HTTP protocol, which uses the TCP protocol. In this stage, the client initiates the TCP connection to the web server site.

2. Request scheduling. During this stage, the web server site schedules this request according to the predetermined policy.

3. Web request processing by the web server. During this stage, the request is received by a web server and processed.

4. Response transmission. During this stage, the response data is transmitted through the network to the client.

7.1.1 Mechanisms

In this section, we discuss some of the possible mechanisms and policies to provide differentiated service at each stage according to the four stages we outlined in the previous section.

- TCP connection setup. During this stage, the client initiates the TCP connection setup by sending a SYN packet. The web server site returns a SYN/ACK packet, then the client sends the ACK packet possibly with the URL as well. From a web site's point of view, the web site may want to reject unwanted traffic as early as possible. A web site may decide to reject the SYN traffic based on layer 4 information, such as IP address and/or port number according to the predefined policy. However, it is not clear if silently rejecting the SYN packet

is beneficial for the server's load. It has been reported the client will try to reconnect to the server again soon.

When the web site is overloaded, it has also been noticed that the web server is slower to respond to a SYN packet. The reason is that when the web server has lot of data to transfer, it receives a lot of ACK packets from clients to acknowledge the data packets sent by the web server. If SYN packets are processed in the order they are received, connection establishment will be significantly delayed. To make the situation worse, when the client waits for connection establishment for too long, it starts to resend SYN packets. This further creates load on the server system. One way to overcome this problem is to give connection establishment related packets a higher priority.

Before discussing multiple-priority traffic support in the protocol stack, the process of the packet reception in the protocol stack will be discussed. Here the FreeBSD TCP/IP implementation is used as an example. First, the incoming packet processing is described.

When a network interface card (NIC) receives a frame, the handler for this device is invoked. The interrupt handler checks the packet and puts the packet into an appropriate queue. In case of an IP packet, the packet is put into the IP queue. A software interrupt handler (IP packet handler) is invoked to process the IP packets from the IP queue once the system interrupt priority drops below the software interrupt's priority.

The IP packet handler (*ip_input*() function) gets an IP packet from the IP queue and processes the IP header, then it calls the TCP protocol handler function (*tcp_input*()) to process the TCP layer information. The TCP layer puts the data in the socket buffer. The IP packet handler gets the next packet on the IP queue and starts the same process again until either the queue is empty or this process is interrupted by higher priority interrupts.

The application issues a *receive*() or *read*() system call to receive this data from the socket buffer. After system call *read* copies the data from the socket buffer to the buffer supplied by the application, the data is removed from the socket buffer.

Multiple queues with different priorities are supported by replacing the IP queue with multiple IP queues with different priorities. When the IP packet handler starts to process the IP packet, depending on the queue policy, the IP packet handler gets a packet from a particular queue, and processes it according to the process described before.

Since TCP connection setup related packets have higher priority, the connection setup progresses much more quickly than before. This can solve the previously mentioned problem.

- Request scheduling. After the connection is established, the client sends the URL. The request scheduler has the ability to check the URL (more general, the HTTP header) and schedules the request based on the content requested by

the user. The request scheduler schedules the order the web requests are seen by the web server software.

The request scheduling is supported by ordering the ready TCP connections in the listen queue. In the FreeBSD-based TCP/IP implementation, the listen queue is the queue of the sockets. Each socket represents either a complete connection (connections that have finished 3-way handshaking with the clients and are ready to be accepted by the application) or an incomplete connection (connections that have been initiated by the clients, but for which 3-way handshaking has not yet been completed yet). In FreeBSD, two separate queues are maintained by the kernel: so->q is the queue for complete connections, and so->q0 is the queue for incomplete connections. The application may only accept connections on the complete queue. When the kernel receives the SYN packet, the kernel creates an incomplete socket corresponding to this connection if the particular listen queue is not full and puts it on the incomplete queue so->q0. After the 3-way handshake finishes, the socket is moved from so->q0 to s0->q in preparation for application acceptance. By default, the apache web server sets the listen queue length to 511 connections.

The completed connections will be accepted in the order on the listen queue. Web differentiated service may be supported by ordering the connections in such a way that the high priority connections will be placed in front of the low priority connections.

Different scheduling policies might be used here, for example, FIFO (first in, first out), fair queue, weighted fair queue, etc. More advanced policies may be supported, too. For example, one of the policies proposed serves the shortest requests first. Smaller web objects are served earlier than longer web objects, and static pages served earlier than dynamic pages.

The previous discussion only applies for HTTP 1.0 connections, because the scheduling happens at the granularity of a connection, instead of a request. The decision for scheduling is made based on the first HTTP header or URL. For example, WebQoS only supports connection scheduling. Support for request scheduling over persistent connections (HTTP 1.1) is not as easy as the connection scheduling.

- Request processing. After the web server software accepts the connection, it starts to process this request. At this stage, depending on the web server implementation, the process or thread to process the high priority request may be given a high priority.

  For a typical E-business application, there are 4 tiers: web client (browsers), web servers, application servers, and database servers. The browsers are used by clients to access the web sites. The web server provides the interface to the server side system. The application server is used to fulfill the processing, and the database server provides the data to be manipulated. In this chain, differentiated service at the web server contributes only partially to this process. Ideally, the priority could be maintained across all the activities in the tiers at

the server side. Howerver, this is much more complex and involves all the components along the processing path.

- Response transmission. During this stage, the kernel might implement several policies regarding the packet transmission. As we may imagine, the kernel may transmit the response data of high priority requests earlier than that of low priority requests. One such policy is shortest request first. That is, the response data from the shortest job receives highest priority.

Some of the kernel mechanisms described above may also be implemeneted at the application level. Request scheduling may be supported either at user level or at the kernel level. At user level, the request differentiation is typically implemented by the distribution process, which accepts all the incoming HTTP requests and classifies the requests into one of several classes (queues). The working processes get the requests from the queues and process the requests. The distribution process may be implemented by the web server software itself, or may be supported by a separate process that feeds the requests to the web server software transparently.

Kernel based request differentiation has the following advantages:

No extra bottlenecks. User-level implementation introduces the distribution process, which controls all the incoming traffic, and classifies them into different priority queues. That introduces another central control point, which might become a bottleneck. The number of distributing processes that are sufficient to process the incoming requests efficiently is highly workload dependent. Kernel-level implementation does

not introduce the additional central control points except the ones existing in the kernel already.

Less resource consumption. When the server reaches the overloaded point, the admission control has to take place. It is strongly desirable that if the server decides to reject the request (or a new session), a simple rejection message is returned so that the user will not try to submit the same request again and again. For such requests, processing them as quickly and efficiently as possible is critical. User-level implementation has to accept the connection, get the request, and return a message. Kernel-level implementation performs similar actions much more efficiently: it does not introduce context switches and socket buffers.

Efficient measurements. Kernel-level implementation may get accurate information quickly and accurately compared to user-level implementation. These measurements may be important to the decision process, such as the number of connections openned at a given moment, activities on each open connection, average response time, the network bandwidth and roundtrip times of each connection, etc. The kernel implementation may take advantage of these measurements and make more intelligent decisions.

7.1.2 Content-Aware Request Processing and Differentiation

In this section, we outline a modular design of the web differentiation in a STREAMS-based TCP/IP implementation. In order to provide web differentiated service, two

new modules, UTCP (Upper TCP) and BTCP (Bottom TCP), are introduced. We discuss the request classification, request scheduling, admission control, and transmission design to support web differentiated service.

- *Request Classification.* The classification identifies a request and assigns it to a class. Information within the request or provided by the application is applied against the classification policies to determine the request class.

  The client sets up the TCP connection with the TCP module in the operating system. The UTCP maintains the necessary number of priority queues and a partially-finished connection queue. BTCP creates a simple structure for each connection and sends the connection establishment packets upstream. UTCP holds the corresponding T_CONN_IND message into the partially-finished connection queue. After TCP module finishes three-way handshaking with the client, the client sends the URL to the server. BTCP retrieves the URL after receiving the packet and classifies the request according to the policy specified by the administrator. BTCP sends the classification of the URL to the UTCP, and UTCP places the corresponding T_CONN_IND message from the partially finished connection queue to one of the supported class queues.

  This design may support persistent connections. Since for persistent connections, the connection has been already established and is exposed to the application, UTCP intercepts the subsequent requests, and classifies them into one of the classes, and places the message (T_DATA_IND) in one of the queues.

- *Admission Control.* The admission control mechanism prevents server overload. The classes of requests, the admission control policies and the system or service load measurements may be used to determine whether to accept a request (or a new session).

  The admission control mechanism can be deployed using the UTCP module. First, BTCP checks the URL (or cookie). If this request can not be served at this time, then the BTCP notifies the UTCP module. The UTCP module then sends the message to the client and releases the connection by sending T_DISCON_REQ.

  For subsequent requests on a persistent connection, UTCP checks the URL (cookie) from the T_DATA_IND message and makes the decision. If such a request can not be served at this time, UTCP sends the customized message to the TCP module and releases the connection. The returned message may be a redirection to other servers, or a customized message stating that the server is busy at this moment.

- *Request Scheduling.* Request scheduling is used to provide performance isolation or differentiation depending on the scheduling and classification policies. UTCP module may support a set of request scheduling strategies, for example, FIFO, fair-sharing, weighted fair-sharing, strict priority queue, etc.

- *Transmission Bandwidth Management.* STREAMS already supports flow control. Flow control may be used to implement the rate based transmission policy by default.

7.2 Network Quality of Service

In the last section, the web differentiated service mechanism is discussed. In this section, we briefly discuss network QoS.

Traditionally, Internet only provides *best effort* service. Packets are delivered by the underlying network as fast as possible. Packets might be delayed, or dropped. Upper level protocols are responsible for recovering from the packet loss. In the last three years, many efforts have been contributed to provide Quality of Service over the Internet. Network QoS may be classified into two categories: Integrated service and differentiated service.

- Integrated Service. In an integrated service environment, the client reserves the resources for a flow according to its usage along the path. RSVP may be used as a resource reservation protocol. All the routers along the path have to support resource reservation. Per-flow status has to be kept by routers in order to deliver desired Quality of Service. The per-flow status greatly impairs the scalability of network core routers. Given that today's "killer app", the web, is built on top of TCP and the connections are short-lived, it would be interesting to see whether such an approach could be justified in overhead.

- Differentiated Services. In differentiated service, the priority of the traffic is supported at each router. No hard QoS is guaranteed. The client traffic will be classified and marked in the IP header at the network egde before it enters the network. The backbone routers are only responsible to forward the packet as quickly as possible according to the traffic priority, and higher priority traffic is

forwarded first. No per flow status is kept. Backbone router design is greatly simplified.

Differentiated service seems easier to be supported by ISP and might be incrementally deployed. It may also be applied for web applications.

7.3 End-to-End Web QoS

From the user's point of view, the ultimate measurement is the response time, the time between initiation of the web request and arrivial of the response data. In a typical web request, many components are involved. The client needs to access the DNS system first. Then the network system is involved. The web server system is a big part.

Providing End-to-End QoS for web applications is a big challenge. In order to provide some performance guarantee, different components involved have to provide some level of guarantee, too. DNS and network are operated by different operating domains and are difficult to modify in order to provide guaranteed services without extensive changes in the infrastructure. Providing QoS at the web system side is not an easy task as well. As we described before, today's typical web application involves four tiers: clients, web servers, application servers, and database servers. The client uses a web browser to access the web site. The web server works as a gateway to the web applications' logic. The application server executes the application logic and generates the response data to web servers. The database server provides necessary data to the application servers. Providing QoS means that the QoS requirement

has to be propogated through all the components, including web servers, application servers and database servers. Moreover, processing time for every component has to be bounded. This is not an easy task.

Providing a differentiated service of the web server system is not as challenging as the guaranteed service. Also, it is much easier to integrate the network differentiated service with the web system differentiated service. WebQoS supports integration of network differentiated service and web server system differentiated service.

7.4 Content Delivery Networks

As we mentioned in chapter 2, web hosting and content delivery networks are two complementary ways to improve the web site performance and user-perceived latency for web documents. In web hosting, the content publisher moves the contents to the ISPs so that web traffic will not enter content publisher's network. As we know, Internet routing has an inherent hierarchy similar to a tree. That means that when a web site is accessed, the client (leaf) first sends the packet to the local ISP (intermediate node) and the ISP forwards the traffic to a regional ISP. Then, the regional ISP forwards the packet to the backbone ISP (root of the tree). After that, traffic is forwarded back to another regional ISP, and forwarded again to another ISP where the web site is connected. In short, the packet is sent upward till it reaches the root of the tree and is sent down to another leaf of the tree. As we may imagine, a good place for the web hosting service is in the root of the tree, that is, backbone network. Statistically this will cut the traffic path by half.

What a CDN does is to provide reverse proxies for the web sites near the client locations. A CDN does this by putting servers at the intermediate nodes of the tree as we previous described, and tries to cache documents from web sites so that documents accessed by clients are in the cache and much closer to the clients. A CDN provides the content delivery service so that web site operators focus their effort on the publishing without worrying too much about the performance and management of the web sites.

7.4.1 How Does A CDN Work?

Different CDNs have different offerings. Here we show how Akamai (one of the major CDNs) provides content delivery service. Akamai claims it has put about 8,000 servers near the POPs (Points Of Presence) of ISPs where the client networks are connected to the Internet.

When a web site decides to take advantage of Akamai service, the site uses the program Free Flow Launcher provided by Akamai to customize the URLs to be served by Akamai server. As we mentioned before, typical web pages contain a lot of embedded images that constitute about 80% of the web traffic One observation is that although the HTML document containing the embedded images changes frequently, the embedded images themselves change less frequently. Typically a web site akamaizes the embedded images and still maintains either static HTML or dynamically generated files. This partially helps the consistency problem.

Any document served by Akamai CDN has a URL in the following format:

http://a388.g.akamaitech.net/7/388/21/fc35ed7f236388/cnn.com/images/hub2000/ad.info.gif

In this URL, the A388 refers to a customer, CNN. The URL of the document in the original server is also included (cnn.com/images/hub2000/ad.info.gif).

In the case the embedded images are akamaized and the HTML documents are still served by the original site, the client first accesses the HTML document. The browser parses this document and gets the URLs like above. The client side DNS server consults the akamai DNS server to get the nearest akamai servers available (IP address list) which most likely have the document or are appropriate to serve the document. The selection process is done by proprietary algorithms. If an Akamai's server does not have that document, the server retrieves the document from the original server and caches it in the Akamai network.

7.4.2 Benefits of CDN

The benefits of content delivery networks may be summarized as follows:

- *latency reduction.* Because a CDN may reduce the load on the original servers and avoid a lot of long network round trip latency, the CDN might greatly reduce the latency perceived by the end-users.

- *scalability* . CDNs also provide better scalability. It has been reported that the web site peak rate might be as much as 10 times the average rate. A typical practice is that the web sites do capacity planning and try to provide enough capacity to accomodate the peak rate of web requests. This obviously leads to waste of the computing resources and increases the web site operation

costs both in the hardware and management. Another observation is that since different web sites might have peak rates at different times, the aggregrate resource requirement will be much smaller. In an extreme case, a CDN might allocate resources to serve only one site. Thus CDNs provide almost unlimited scalability for today's typical web applications.

- *availablility.* A CDN may also be used to enhance the availability of documents pushed on the CDN. CDN servers may cache the documents at multiple locations and serve directly from the content servers even under the failure of the original sites.

- *maintenance cost.* Under current CDN practice, the embedded images are pushed to the content delivery network, which greatly reduces the burden on the publiser who operates the web site. This greatly reduces the hardware and management overhead of the web sites.

7.4.3 Content Delivery Network and End-to-End QoS

Streaming media is believed to be the next thing in the industry. Streaming media is much more sensitive to the network conditions. A stream has a fixed bandwidth requirement and is very sensitive to the network latency variation. Network QoS, both integrated service and differentiated service, is still a dream in some sense, because the technology requires a lot of changes in the Internet infrastructure. A typical Internet path usually involves several ISPs. It is difficult for application developers to benefit from the technology if only one or several ISPs deploy such technologies.

The deployment of such technologies will take some time. Application developers may not be able to take advantages of these solutions in the near future.

By placing the content servers at POP's, where the client network connects to the Internet, a CDN bypasses the whole Internet network. Because clients are just about a few hops away from the content servers, network latency is much more stable and predictable. So CDNs will be an interesting technology to deliver streaming media documents.

7.5 Summary

In this chapter, end-to-end WebQoS is discussed. We described some of the kernel and user-level mechanisms to achieve web differentiated service at the web system side to differentiate web requests and how these mechanisms may be supported both on the FreeBSD network stack and a STREAMS-based protocol stack. Network QoS is briefly described. Content Delivery Networks (CDN) are presented. We showed CDNs do address some of the need of the today's web service, namely, latency reduction, scalability, availability and maintenance cost. We also showed that because CDN servers are deployed at the network edge, it is an interesting technology for the emerging streaming media delivery.

Chapter 8

Summary

The web has experienced phenomonal growth during the last couple of years. As the Internet evolves into client/server based computing, where millions of clients (browsers) access a comparatively few number of popular sites, web site performance and scalability is an important issue to be addressed. In this dissertation, we address some problems related to the development of *high performance, scalable web server systems.* Our achievement may be divided into two parts: a number of technologies that target a *single web server* and technologies for *multiple replicated web sites.*

For a single web server, *Browser Initiated Pushing* (BIP) is proposed to improve performance based on the observation that today's typical web page has one or more embedded images. Measurement shows that BIP reduces the user perceived response time by 40% under normal load and improves response time 3-4 times under heavy load for a typical file in a LAN environment. BIP increases server's throughput by 150% under our measurement configuration in a LAN environment. Trace-driven

simulation shows that enhanced BIP reduces the overhead of traditional pushing from 32% to 10%. BIP is an important tool to improve single server performance.

For replicated web sites, we proposed two approaches and developed a framework to address the scalability of such a system. A non-dispatcher approach *Static Allocation and Client Redirection SSCR* to share load between replicated web servers is developed. SSCR is mainly targeted for web server replication at global scale. Trace file simulation shows that SSCR offers competitive performance to a dispatcher based approach without one-point bottleneck and failure in the cluster environment.

Smart Server Selection (S3) addresses the server selection problem. In S3, client DNS server is extended to prioritize a pool of IP addresses based on routing metric information collected from routers and other information it collects (geographical location of servers and clients). An efficient scheme to collect routing-metric information from routers is proposed. The overhead of such a scheme is independent of the number of replicated servers. This is a big achievement over existing approaches because the overhead of these approaches suffers a linear increment as the number of replicated web server increases.

A framework to support *Content-Aware* request distribution in a STREAMS-based TCP/IP implementation is developed and prototyped. Content-Aware request distribution provides the ability to support partial replication, flexible web site arrangements, web Quality of Service, and security. Our framework is based on the TCP handoff mechanism. The TCP handoff mechanism is designed as a STREAMS module(s) in the protocol stack. Three different designs are reported according to

workload charateristics. Differentiated web service support in the STREAMS-based TCP/IP implementation is discussed.

Chapter 9

Further Work

There are several opportunities to extend and improve the work that we have already accomplished. This section outlines some of the areas of further research.

- Performance evaluation of scalable proxy servers based on TCP handoff mechanisms. In chapter 6, we outlined the approach to use a content-aware protocol stack to support a request resolution in a partition-based cooperative proxy environment. It would be nice to have the performance evaluation of such an approach.

- Prototype web differentiated service in a STREAMS based TCP/IP environment. We outlined the design in chapter 7. A performance evaluation will strongly augment this work.

- For the BIP approaches, we plan to develop new Apache modules to give a tighter bound on the benefits of BIP, and possibly measure the performance of the BIP under realistic workloads.

- For our S3 approaches, a special tool may be developed which prioritizes the IP address list. Such a tool can measure the latency and the number of hops information from the network, because we do not have access to a BGP routing table directly. However, we may use dumped BGP routing tables to identify the number of groups of the servers, so that the number of entries processed by this tool is limited to the size of a backbone router of the Internet, which has about 400,000 entries.

- For the SSCR approach, the evaluation is done for local clusters. It would be nice if the ideas may be further validated over wide area replicated web site logs.

Bibliography

[1] Jeffrey Mogul. What's wrong with http and why it doesn't matter. In *Proceedings of 1999 USENIX Annual Conference*, June 1999.

[2] Akamai. *http://www.akamai.com.*

[3] Inktomi. *http://www.inktomi.com.*

[4] Digital Island. *http://www.digitalisland.com.*

[5] Amin Vahdat, Thomas Anderson, Michael Dahlin, David Culler, Eshwar Belani, Paul Eastham, and Chad Yoshikawa. Webos: Operating system services for wide area applications. In *The 7th IEEE Symposium on High Performance Distributed Computing*, July 1998.

[6] A. Bestavros and C. Cunha. Server-initiated document dissemination for the www. In *IEEE Data Engineering Bulletin*, September 1996.

[7] Michael Rabinovich and Amit Aggarwal. Radar: A scalable architecture for a global web hosting service. In *Proceedings of 8th World Wide Web conference*, May 1999.

[8] V. N. padmanabhan and J. C. Mogul. Using predictive prefetching to improve world wide web latency. In *Computer communication Review, vol 26(3)*, July 1996.

[9] Crovella Mark and Barford Paul. The network effects of prefetching. Technical Report 1997-002, Department of Computer Science, Boston University, February 1997.

[10] Cunha Carlos R. and Jaccoud Carlos F.B. Determining www user's next access and its applications to pre-fetching. Technical Report 1997-004, Department of Computer Science, Boston University, March 1997.

[11] Stephen Williams, Marc Abrams, Charles R. Standridge andGhaleb Abdulla, and Edward A. Fox. Removal policies in network caches for world-wide web documents. In *Proceedings of the 1997 ACM SIGCOMM*, August 1997.

[12] Pei Cao and Sandy Irani. Cost-aware www proxy caching algorithms. In *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, December 1997.

[13] L. Cherkasova. Improving www proxies performance with greedy-dual-size-frequency caching policy. Technical Report HPL-98-125R1, HP Laboratories Report, July 1998.

[14] P. Cao and C. Liu. Maintaining strong cache consistency in the world wide web. *IEEE transactions on computers*, 47(4), April 1998.

[15] Balachander Krishnamurthy and Craig E. Wills. Piggyback server invalidation for proxy cache coherency. *Computer Networks and ISDN systems*, 30, April 1998.

[16] Radhika Malpani, Jacob Lorch, and David Berger. Making world wide web caching servers cooperate. In *Proceedings of 4th World Wide Web conference*, 1995.

[17] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. Technical Report Technical Report 1361, Department of Computer Science University of Wisconsin-Madison, February 1998.

[18] J. Cohen, N. Phadnis, V. Valloppillil, and K. W. Ross. *Cache Array Routing Protocol 1.0*, September 1997.

[19] David Karger, Alex Sherman, and Andy Berkheimer. Web caching with consistent hashing. In *Proceedings of 8th World Wide Web conference*, May 1999.

[20] C. Yoshikawam, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler. Using smart clients to building scalable services. In *USENIX'97*, January 1997.

[21] P. Scheuermann M. Sayal and P. Vingralek. Selection algorithms for replicated web servers. In *The 1998 SIGMETRICS/Performance Workshop on Internet Server Performance*, June 1998.

[22] J. Heidemann and V. Visweswaraiah. Automatic selection of nearby web servers. Technical Report 98-688, USC/Information Science Institute, 1998.

[23] K. Moore, Jason Cox, and Stan Green. Sonar - a network proximity service. In *Internet Draft, Network Working Group, draft-moore-sonar-01.txt*, February 1996.

[24] Cisco Systems Inc. *Distributed director*. http://www.cisco.com/univercd/cc/td/doc/product/iaabu/distrdir/.

[25] K. Obraczka and F. Silva. Looking at network latency for server proximity. Technical Report 99-714, USC/Information Science Institute, 1999.

[26] Foundry Networks. *http://www.foundrynet.com.*

[27] Alteon. *Layer 7 switching.* http://www.alteon.com.

[28] ipivot. *Intelligent Broker.* http://www.ipivot.com.

[29] E. Anderson, D. Patterson, and E. Brewer. The magicrouter, an application of fast packet interposing. In *OSDI*, 1996.

[30] Cisco Systems Inc. *Localdirector.* http://www.cisco.com/warp/public/cc/cisco/mkt /scale/locald.

[31] G. D. H. Hunt, G. S. Goldszmidt, R. P. King, and R. Mukherjee. Network dispatcher: A connection router for scalable internet services. In *Proc. of 7th International World Wide Web conference*, Brisbane, Austrilia, April 1998.

[32] P. Damani, P. E. Chung, Y. Huang, C. Kintala, and Y.-M. Wang. ONE-IP: Techniques for hosting a service on a cluster of machines. In *Sixth International World Wide Web Conference*, Santa Clara Convention Center, California, April 1997.

[33] Resonate Inc. *Central Dispatcher.* http://www.resonate.com.

[34] Arrowpoint. *Content-Aware Switching.* http://www.arrowpoint.com.

[35] V.Pai, M.Aron, G.Banga, M.Svendsen, P.Drushel, W. Zwaenepoel, and E.Nahum. Locality-aware request distribution in cluster-based network servers. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VIII)*, 1998.

[36] Mohit Aron, Darren Sanders, Peter Druschel, and Willy Zwaenepoel. Scalable content-aware request distribution in cluster-based network servers. In *Proceedings of the USENIX 2000 Annual Technical Conference*, June 2000.

[37] Chu-Sing Yang and Mon-Yen Luo. Efficient support for content-based routing in web server clusters. In *Proceedings of the 2nd Usenix Symposium on Internet technologies and Systems*, October 1999.

[38] M. Colajanni, P. S. Yu, and D. M. Dias. Analysis of task assignment policies in scalable distributed web-server systems. *IEEE Trans. Parallel Dist. Syst.*, 9(6), June 1998.

[39] M. Colajanni, P.S. Yu, and V. Cardelliini. Dynamic load balancing in geographically distributed heterogeneous web servers. In *Proc. of 18th International Conference on Distributed Computer Systems*, Amsterdam, May 1998.

[40] V. Cardellini, M. Colajanni, and P. Yu. Redirection algorithms for load sharing in distributed web-server systems. In *Proc. of the 19th IEEE International Conference on Distributed Computing Systems*, Austin, Texas, June 1999.

[41] A. Bestavros, M. Crovella, J. Liu, and D. Martin. Distributed packet rewriting and its application to scalable web server architectures. In *Proc. of ICNP'98: The 6th IEEE International Conference on Network Protocols,*, Austin, Texas, October 1998.

[42] Jussara Almeida, Mihaela Dabu, Anand Manikutty, and Pei Cao. Providing differentiated quality-of-service in web hosting services. In *1998 SIGMETRICS Workshop on Internet Server Performance*, March 1998.

[43] Hewlett Packward. *HP WebQoS v2.0 white paper.*

[44] Nina Bhatti, Anna Bouch, and Allan Kuchinsky. Integrating user-perceived quality into web server design. In *Proceedings of 9th World Wide Web conference*, May 2000.

[45] E. Cohen, B. Krishnamurthy, and J. Rexford. Efficient algorithms for predicting requests to web servers. In *IEEE Infocom'99*, New York, March 1999.

[46] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext transfer protocol – http/1.1. In *RFC 2068*, January 1997.

[47] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of ACM*, 13(2), July 1970.

[48] David Mosberger and Tai Jin. httperf - a tool for measuring web server performance. In *1998 Workshop on Internet Server Performance*, June 1998.

[49] Wenting Tang and Matt W. Mutka. Intelligient browser initiated pushing. In *19th IEEE International Performance,Computing,and Communica tions Conference*, February 2000.

[50] Wenting Tang and Matt W. Mutka. Supporting global replicated services by a routing-metric-aware dns. In *2nd International Workshop on Advanced Issues of E-commerce and Web-Based Information Systems*, San Jose, CA, June 2000.

[51] J. Doyle. *Routing TCP/IP*, volume I. Macmillan Technical Publishing, first edition, 1998.

[52] J. Moy. Ospf version 2. In *RFC 2328*, April 1998.

[53] J. Postel. Internet control message protocol. In *RFC 777*, April 1981.

[54] W. R. Stevens. *TCP/IP illustrated*, volume I: The Protocols. Addison-Wesley Professional Computing Series, first edition, 1994.

[55] Wenting Tang and Matt W. Mutka. Load distribution via static scheduling and client redirection for replicated web servers. In *Proceedings of the First Workshop on Scalable Web Services*, Toronto,Canada, August 2000.

[56] L. Cherkasova. Flex: Load balancing and management strategy for scalable web hosting service. In *Fifth International Symposium on Computers and Communications (ISCC'00)*, July 2000.

[57] A.Cohen, S.Rangarajan, and H.Slye. One the performance of tcp splicing for url-aware redirection. In *Proceedings of the 2nd Usenix Symposium on Internet technologies and Systems*, October 1999.

[58] Wenting Tang, Lumilda Cherkasova, Lance Russell, and Matt W. Mutka. A customized library of plug-ins to support content-aware request processing in streams-based tcp/ip implementation. In *3rd International Workshop on Advanced Issues of E-commerce and Web-Based Information Systems*, June 2001.

[59] Wenting Tang, Lumilda Cherkasova, Lance Russell, and Matt W. Mutka. Modular tcp handoff design in streams-based tcp/ip implementation. In *International Conference on Networking 2001*, July 2001.

[60] UNIX International, OSI Special Interest Group. *Transport Provider Interface (TPI).*

[61] UNIX International, OSI Work Group. *Data Link Provider Interface specification.*

[62] N. Batti and R. Friedrich. Web server support for tiered services. *IEEE network*, 13(5), 1999.

[63] P. Phaal L. Cherkasova. Session based admission control: a mechanism for improving performance of commercial web sites. In *Seventh International Workshop on Quality of Service*, June 1999.