Transparent Web Caching with Load Balancing

by

ZHENGANG LIANG

A thesis submitted to

the Department of Computing and Information Science

in conformity with the requirements

for the degree of Master of Science

Queen's University

Kingston, Ontario, Canada

February 2001

Abstract

Web caching is a technique that temporarily stores Web objects (such as Hypertext documents) for later retrieval in order to improve the performance and scalability of the Web. Layer 5 switching-based transparent Web caching schemes intercept HTTP requests and redirect requests according to their contents. This technique not only makes the deployment and configuration of the caching system easier, but also improves its performance by redirecting non-cacheable HTTP requests to bypass cache servers.

In this thesis, we propose a Load Balancing Layer 5 switching-based (LB-L5) Web caching scheme that uses the transparent Web caching technique to support distributed Web caching. In LB-L5, information about proxy cache server workload, network link delay, cache content and access-frequency is used to redirect HTTP requests in order to achieve cache server workload balance and better response times. LB-L5 uses a weighted Bloom Filter to represent cache content and access-frequency information, which enables LB-L5 to implement access-frequency-aware cache cooperation. LB-L5 extends ICP, the most popular Web caching protocol, to support communication between cache servers and Layer 5 switches, and ensure compatibility with existing Web cache systems.

A number of simulation experiments were conducted under different HTTP request intensities, network link delays and populations of cooperating cache servers. Simulation results show that LB-L5 outperforms existing Web caching schemes, namely ICP, Cache Digest, and basic L5 transparent Web caching, in terms of cache server workload balancing and response time. LB-L5 is also shown to adapt better to high HTTP request intensity than the other schemes.

Acknowledgments

I would like to thank my supervisors Professor Hossam Hassanein and Professor Patrick Martin, to whom I am greatly indebted, for their continuous guidance, keen supervision, constant encouragement and great assistance throughout the course of my study.

The financial support provided by the Department of Computing and Information Science at Queen's University and Communication and Information Technology Ontario (CITO) is greatly appreciated.

I would like to thank my friends in the Telecommunications Research Laboratory and the Database Systems Laboratory for being supportive. I will cherish their friendship forever. Especially, I am very grateful to Ahmed Safwat for all his help.

Special thanks to my wife for her selfless support, which allowed me to concentrate on my studies.

Finally, I would like to make special mention of my parents. I would not be doing higher studies if they did not teach me the value of hard work and dedication. Although they are thousands of miles away from me, I always feel their presence and blessings.

Table of Content

List of Figures

List of Tables

Chapter 1

Introduction

The World-Wide Web [1,2] (the Web) is an Internet-based globally distributed information system that was originally developed at CERN (Conseil Européen pour la Recherche Nucleaire) for sharing information among collaborating researchers. The Web uses Hypertext for structuring information [1,2]. Hypertext is text with links (called Hyperlinks) to other information, such as text and multimedia. The clients and servers on the Web use the HyperText Transfer Protocol (HTTP) to communicate [3,4]. A Web client accesses information on a Web server by sending an HTTP request. The server parses the request, retrieves the requested information, and returns it to the client.

Since its first complete implementation in 1991 [2], the Web has experienced phenomenal growth because of its friendly user interfaces and effective information dissemination capability. However, the growth of the Web has contributed significantly to the traffic on the Internet, and raised several problems such as long HTTP request/response times, heavy Web

server workloads and network congestion [5]. These problems have motivated several projects on improving the performance and scalability of the Web.

Web caching, a technique that temporarily stores Web objects (such as Hypertext documents) for later retrieval, is considered one of the most efficient approaches [6,7]. Web caching can be performed at Web proxies. A Web proxy consists of application level software that accepts HTTP requests from a set of clients, fetches the requested objects from original Web servers, caches the requested objects and sends these objects back to the clients. Proxy Web caching increases document availability and enables download sharing. It reduces overall access delay and saves network bandwidth by caching frequently requested Web objects.

In addition to local proxy Web caching, distributed cache cooperation, a mechanism for sharing documents between caches, can further improve system performance by providing a shared cache to a large user population [8,9,10,11,12]. In a cooperative Web caching system, if a cache miss occurs at a local cache server, the request can be forwarded to one of a set of cooperating servers. Therefore, if any one of the servers has a cached copy of the requested object then the request will result in a cache hit.

In the past few years, several Web caching schemes were proposed and widely deployed to support distributed cache cooperation. The pioneer project CERN [13], and its direct successor Harvest [14], introduced the Internet Cache Protocol (ICP) to achieve proxy server cooperation [15,16,17]. After Harvest was commercialized in 1995, Squid [18] became its public domain successor. One of the most recent improvements implemented in Squid is the

Cache Digest (CD) Web caching scheme [19], which uses a Bloom Filter [19] to represent the cache content and performs directory-based proxy cache server cooperation. A similar scheme, Summary Cache [20], was proposed by Pei Cao in 1998.

Recently, Layer 4 and Layer 5 (L4/L5) switching-based transparent Web caching techniques have drawn lots of attention from academic and industrial researchers [21,22,23,24,25]. IBM, ArrowPoint (acquired by CISCO in June 2000) and Alteon (acquired by Nortel in July 2000) have announced L5 switches that can transparently redirect non-cacheable HTTP requests to original Web servers and cacheable requests to caches by using TCP spoofing and TCP splicing techniques. Transparent Web caching not only makes the deployment and configuration of the caching system easier, but also improves its performance by redirecting non-cacheable HTTP requests to bypass cache servers [21].

However, no Web caching scheme has yet been proposed for the transparent Web caching to support distributed cache cooperation, which is widely supported by proxy Web caching systems and has proven effective [8,9,10,12,17,18,19,26,32]. The main thesis of this research is that transparent Web caching can be combined with distributed cache cooperation to provide improved cache performance.

In this thesis, we propose the Load Balancing Layer 5 (LB-L5) switching-based Web caching scheme. LB-L5 uses transparent Web caching techniques to support distributed Web caching. Moreover, cache server workload, network link delay, cache content and access-frequency information, are used in LB-L5 to redirect HTTP requests in order to achieve cache server

workload balancing and better response times. LB-L5 uses a weighted Bloom Filter to represent cache content and access-frequency information, which enables LB-L5 to implement access-frequency-aware cache cooperation. In order to achieve backward compatibility with existing Web caching systems, LB-L5 extends, but does not replace, ICP - the most popular Web caching protocol - to support communication between cache servers and L5 switches.

The rest of the thesis is organized as follows. In Chapter 2 we provide a review of related Web caching technologies. We first describe the hierarchical and distributed proxy Web caching models. We then introduce typical Web caching protocols representing query-based, hash-based and directory-based approaches. Finally, we discuss emerging transparent Web caching techniques, which use Layer 4 or Layer 5 switches to transparently redirect HTTP requests to cache servers.

Chapter 3 presents an overview of the proposed LB-L5 scheme followed by a discussion of the design decisions. The use of a weighted Bloom Filter to represent cache content and access-frequency information is described. A detailed description of the scheme and algorithms is given, where Layer 5 switches and proxy cache servers cooperate by using extended ICP messages.

A performance evaluation of LB-L5 is presented in Chapter 4. The adopted simulation model is described. Proxy traces from the NLANR caching project [27,28] are used to drive our simulator. Simulation experiments are conducted in order to study the effect of network link delay, HTTP request intensity, and the number of cooperating cache servers on the

performance of LB-L5. These results are compared to existing Web caching schemes, namely ICP, Cache Digest, and basic Layer 5 transparent Web caching. The results show that LB-L5 outperforms existing schemes in terms of overall HTTP request/response time and cache server workload balancing. Chapter 5 concludes the thesis, and provides some suggestions for further research.

Chapter 2

Web Caching Techniques

In this chapter we provide an overview of various related Web caching techniques. Section 2.1 describes the hierarchical, distributed, and hybrid Web caching models. Section 2.2 introduces query-based, hash-based and directory-based approaches to proxy server cooperation by describing a typical protocol of each category. Section 2.3 discusses the emerging transparent Web caching techniques with an emphasis on the structure and function of Layer 4 and Layer 5 switching.

2.1 Proxy Web Caching Models

In existing proxy Web caching systems, cooperative proxy cache servers are organized either hierarchically, in a fully distributed mesh, or in a hybrid structure. This section introduces these structures and points out their advantages and disadvantages.

2.1.1 Hierarchical Web Caching

One category of approaches to cooperative Web caching sets up a caching hierarchy [11,14], as shown in Figure 2.1. With hierarchical caching, caches are placed at multiple levels of the network. For the sake of discussion, we assume that there are four levels of caches: bottom, institutional, regional, and national. The client caches are at the bottom level of the hierarchy. When a client cache does not satisfy a request, the request is redirected to an institutional cache. If the document is not found at the institutional level, the request is then forwarded to the regional level cache, which in turn forwards unsatisfied requests to the national level cache. If the document is not found at any cache level, the national level cache contacts the original Web server. When the document is found, either at a cache or at the server, it travels down the hierarchy, leaving a copy at each of the intermediate caches. Further requests for the same document travel up the caching hierarchy until the document is found at some cache level.

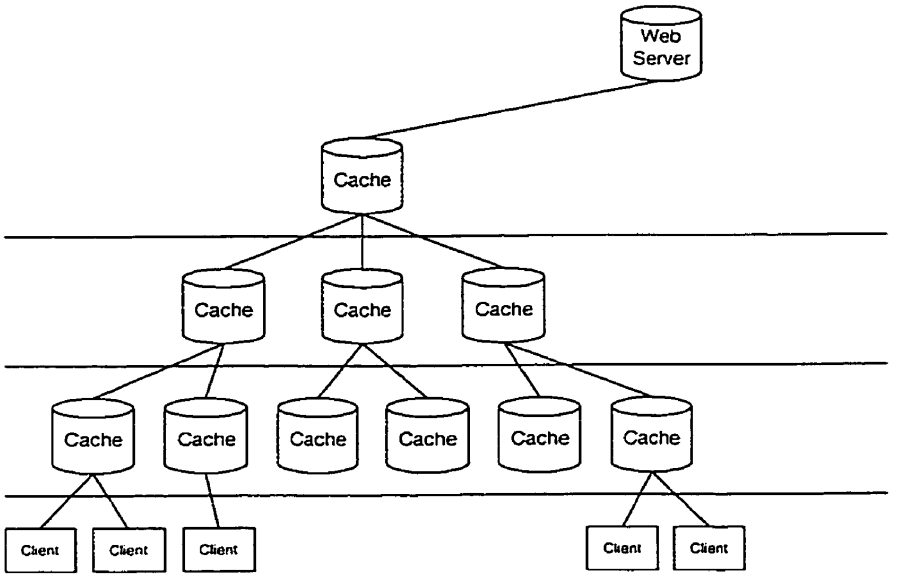


Figure 2.1 Hierarchical Web Caching Model

Hierarchical Web caching was first proposed in the Harvest project [14]. Other examples of hierarchical caching include Adaptive Web caching [29] and Access Driven cache [30]. A hierarchical architecture is bandwidth efficient, particularly when some cooperating cache servers do not have high speed connectivity. In such a structure, popular Web pages can be efficiently diffused towards the demand. However, there are several problems associated with a caching hierarchy:

1. Every hierarchy level introduces additional delays.
2. Higher level caches may become bottlenecks and may have long queuing delays.
3. Multiple copies of documents are stored at different cache levels.
4. To set up such a hierarchy, cache servers need to be placed at key access points in the network. This often requires significant coordination among participating cache servers.

2.1.2 Distributed Web Caching

Recently, a number of researchers have proposed an alternative to hierarchical caching, called distributed caching [8,9,10,11,31,35,36]. In distributed Web caching systems, no intermediate caches are set up. There are only institutional caches that serve each other's misses. In order to decide from which institutional cache to retrieve a document, institutional caches keep metadata information about the content of every other cooperating cache. To make the distribution of the metadata information more efficient and scalable, a hierarchical distribution can be used. However, the hierarchy is only used to distribute information about the location of the documents and not to store document copies. The structure of the distributed Web

caching model is shown in Figure 2.2.

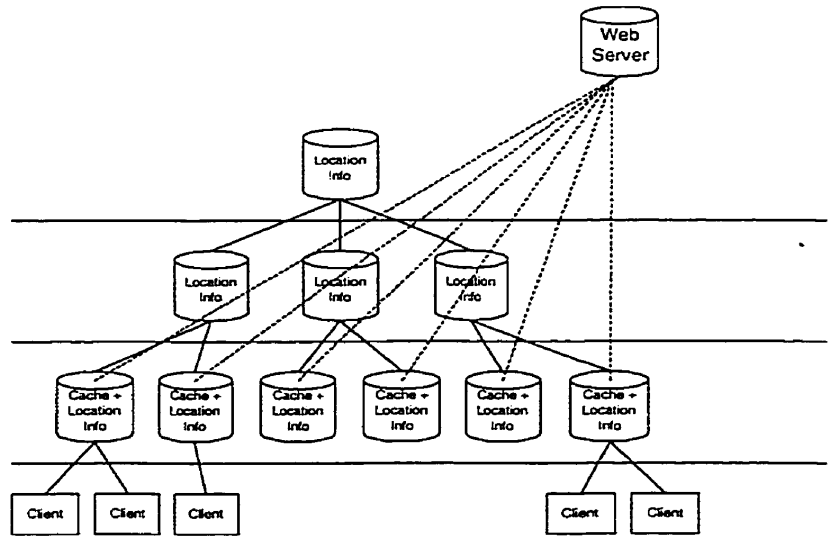


Figure 2.2 Distributed Web Caching Model

With distributed caching most of the traffic is in the lower levels of the hierarchy, which are less congested. The nodes at the intermediate levels only require little additional disk space. In addition, distributed caching allows better load sharing and more fault tolerance. Nevertheless, a large-scale deployment of distributed caching may encounter several problems, such as high connection delays, higher bandwidth usage, and administrative issues.

There are several approaches to distributed caching. The Harvest [14] group designed the Internet Cache Protocol (ICP), which supports discovery and retrieval of documents from neighbouring caches, as well as parent caches [14,15,16]. Another approach to distributed caching is the Cache Array Routing Protocol (CARP) [32], which divides the URL-space among an array of loosely-coupled caches and lets each cache store only the Web objects

whose URLs are mapped to it.

Provey and Harrison [31] also proposed a distributed caching scheme. In their scheme, directory servers that contain location hints about the documents kept at every cache replace the upper level caches of the other schemes. A metadata hierarchy is used to make the distribution of these location hints more efficient and scalable. Tewari et al. [8,9] proposed a similar approach to implement a fully distributed Internet caching system where location hints are replicated locally at the institutional caches.

In the central directory approach CRISP [33,34], a central mapping service interconnects a certain number of caches. In the Cachemesh system [35], cache servers establish a cache routing table among themselves, and each cache server becomes the designated server for a number of Web sites. Client requests are then forwarded to the proper cache server according to the cache routing table. In Cache Digest [19], Summary Cache [20], and the Relais project [36], caches interchange messages indicating their cache contents and keep local directories to facilitate finding objects in other caches.

2.1.3 Hybrid Web Caching

In a hybrid scheme, a certain number of proxy cache servers cooperate at every level of a caching hierarchy by using distributed caching techniques. For example, ICP [15,16,17] can be used for cache cooperation at every level of a caching hierarchy. The requested object is fetched from the parent/neighbour cache server that has the lowest round trip time. Rabinovich [37] proposed to limit the cooperation between neighbouring cache servers to

avoid obtaining documents from distant or slower caches. In this case, requested objects can be retrieved directly from the original Web server at a lower cost.

2.2 Cooperative Proxy Web Caching Protocols

To support distributed cache server cooperation, various protocols are used in existing Web caching systems. In this section, we introduce three approaches for cache cooperation, namely query-based, directory-based, and hash-based approaches. We describe a typical protocol for each approach.

2.2.1 Query-Based Approach – ICP (Internet Cache Protocol)

ICP [15,16] is the most popular protocol that uses the query-based technique to coordinate a set of cooperating proxy Web caches. The caches can be organized either hierarchically or in a distributed model. ICP is an application layer protocol running on top of UDP (User Datagram Protocol). Both Harvest [14] and Squid [18] use ICP to coordinate proxy Web caches.

In a distributed proxy Web caching system, ICP works as follows. A client sends a request to its configured proxy cache server. If that cache server cannot find the requested object in its own cache, it broadcasts an ICP query message to all the other cooperating cache servers. If at least one cooperating cache server has the object, the configured cache server sends an HTTP request for the object to the first server that responds to the query with an ICP hit message. Upon receiving the object, the configured cache server stores a copy in its cache, and then

sends the object back to the requesting client. If no cooperating cache server responds to the query with an ICP hit message before a time-out period, the configured cache server fetches the requested object from the original Web server.

2.2.2 Directory-Based Approach – Cache Digest

Although query-based approaches, such as ICP, work well when cooperating proxy cache servers are located close to each other, the query/response delay becomes significant in a wide area network. Directory-based approaches allow cache servers to make information about their cache content available to peers in order to avoid the query/response delay.

However, using an uncompressed directory of cache content can result in huge memory consumption on the cache servers and high directory update traffic on the network. For example, if we use the entire list of cache keys (URLs) to represent the cache content of a proxy server holding 1 million objects, and the average URL length is 50 bytes [19], the directory will be 50 megabytes. If we have 16 cooperating cache servers, 15x50=750 megabytes memory will be allocated in each proxy server for keeping siblings' directory information. Thus, compressed representations of the cache content directory have been used in several proposed approaches [8,9,19,20].

Summary Cache [20] and Cache Digest [19] are very similar approaches. Both use a Bloom Filter to represent the directory of cache content. The major difference between them is that Summary Cache extends ICP to update the directory, while Digest Cache uses HTTP to transfer directory information.

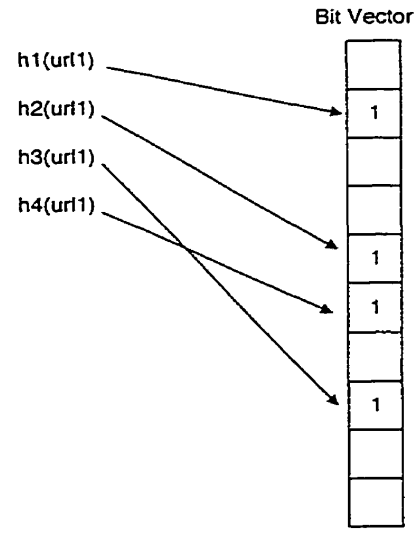


Figure 2.3 Bloom Filter

As shown in Figure 2.3, a Bloom Filter is an array of bits, some of which are set to 1 and the rest are 0's. To add an entry to the Bloom Filter, several independent hash functions are computed for the entry's key (URL). The hash values specify which bits in the filter are set to 1. To check if a specific entry is in the filter, we use the same hash functions to compute hash values for the entry's key, and check the corresponding bits in the filter. If any one of the bits is set to 0, the entry is not in the filter. If all the bits are set to 1, then we can predict that the entry is in the filter. A detailed description of the Bloom Filter is presented in Chapter 3.

The salient feature of a Bloom Filter is that there is a trade off between the prediction accuracy and the size of the filter. By adjusting the number of bits allocated for each entry and the number of hash functions, a low false prediction probability is achieved in the Summary Cache [20].

2.2.3 Hash-Based Approach – CARP (Cache Array Routing Protocol)

Another category of Web cache cooperation approaches use hash-based HTTP request redirection to avoid the inter-proxy query/response delay. Cache Array Routing Protocol (CARP) [32] is a typical example.

CARP was designed by Microsoft Corporation and the University of Pennsylvania. In CARP, a proxy cache server deterministically redirects HTTP requests to neighbouring caches by using a mapping function that maps the hash values of the requested URLs to cache server IDs. All requests for the same URL are redirected to the same cache server. Each cache server stores only the Web objects whose URLs are mapped to it.

In CARP, HTTP requests are redirected to cache servers without explicit knowledge of the cache content on these servers or the network link delay to these servers. It implicitly considers cache hit rate by redirecting the same requests to the same cache server. CARP works well for Intranet hierarchies, but less so for loosely-coupled Internet cache peers [19].

2.3 Transparent Web Caching Techniques

Transparent Web caching uses network devices to redirect HTTP traffic to cache servers. The technique is called transparent because Web browsers do not have to be explicitly configured to point to a cache server, that is the caches are transparent to the browsers [38].

2.3.1 L4-Switching-Based Transparent Web Caching

A Layer 4 switching device can be used to redirect TCP/IP packets destined to HTTP ports to

cache servers, and forward all other network traffic directly to the WAN router. They are called Layer 4 (L4) switches because their switching decisions are based on information in the TCP header, and TCP is a protocol for Layer 4 in the OSI 7-layer model [22].

L4-switching-based transparent Web caching systems partition the client's Web requests into separate hash buckets. The hash function maps the TCP session's destination address into a hash bucket, effectively mapping specific Web server URLs to specific caches. Most of these hash functions operate on subnet boundaries, and typically map the replicas of a Web server to a single cache.

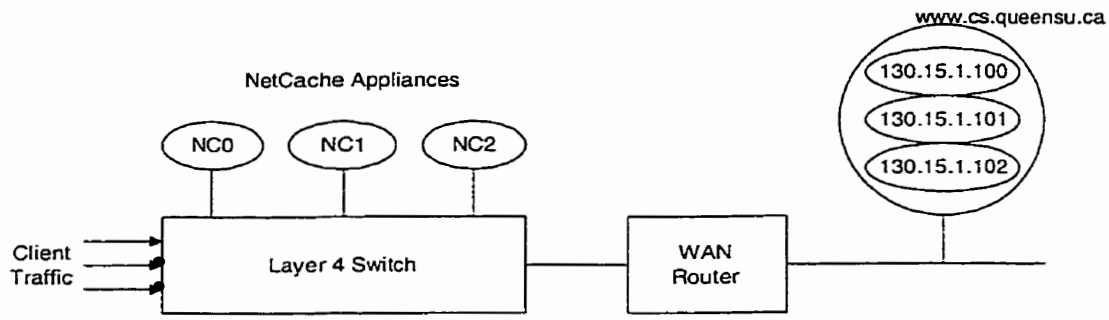


Figure 2.4 Transparent Caching supported by an L4 switch (adopted from [22])

Figure 2.4 shows NetCache Appliances NC1 to NC3, an Alteon's Layer 4 switch, a router, and a subnet 130.15.1.*. Suppose a client requests http://www.cs.queensu.ca from Web server 130.15.1.100 and that the L4 switch's hash function maps the entire 130.15.1.0 subnet to NetCache proxy server nc2. The client establishes a TCP session with NetCache nc2, thinking that it has established a connection with the Web server 130.15.1.100. A NetCache proxy server accepts all connections routed to it, regardless of the destination address. In

accepting these connections, the NetCache proxy server masquerades as the remote Web server.

2.3.2 L5-Switching-Based Transparent Web Caching

While L4 switches are optimized for the transport layer, they are completely unaware of the Application Layer (Layers 5 – 7), which in the Internet includes protocols such as HTTP and FTP. Layer 5 switches are networking devices that provide high speed switching of traffic. Information in the TCP and HTTP request header is used to make routing decisions based on the actual content, for example URL, being requested, and manage request/response flows from beginning to end [23].

The HTTP request header, which includes the URL, comes from the client browser, but the client does not send this until the TCP connection is set up. For a direct connection between a client browser and a Web server, the normal flow is shown in Figure 2.5.

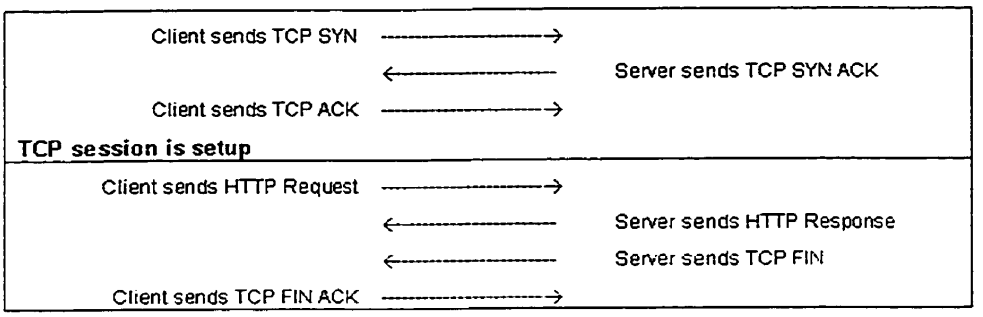


Figure 2.5 Normal HTTP traffic flow (adopted from [23])

A Layer 5 switch sits between the client and the Web servers. In order to obtain the HTTP

request header, the switch performs delayed binding (or TCP spoofing). Delayed binding means that the switch, after receiving the initial TCP SYN, sends the SYN ACK prior to establishing the TCP session to the server, thus "tricking" the client browser into sending it's HTTP request. After receiving the HTTP request, the Layer 5 switch has all of the information it needs to make routing decisions based on the content being requested, and can select the best site and server to service the request. The switch then initiates a new TCP connection to that server and sends the HTTP request. The Web server responds back to the client via the Layer 5 switch. In the flow switching stage the Layer 5 switch is providing Network Address Translation and wire-speed forwarding of packets for all traffic going between the Web Server and the client (TCP splicing). In the final stage, the Web switch tears down the connection, freeing resources allocated for the flow. These steps are shown in Figure 2.6.

2.4 Summary

In this chapter, we reviewed various related Web caching techniques. In Section 2.1, we described the hierarchical, distributed, and hybrid Web caching models and pointed out the advantages and disadvantages of each model.

In Section 2.2, query-based, hash-based and directory-based approaches to proxy cache server cooperation, were reviewed by describing a typical protocol of each category. Although query-based approaches work well when cooperating proxy cache servers are located close to each other, the inter-proxy query/response delay becomes significant in a wide area network. Directory-based approaches allow cache servers to make information about their cache

content available to peers in order to avoid inter-proxy query/response delays. Therefore, they require an efficient cache content representation. Hash-based approaches deterministically redirect HTTP requests to cooperating caches by using a mapping function that maps the hash values of the requested URLs to cache server IDs. In hash-based approaches, HTTP requests are redirected to cache servers without explicit knowledge of the cache content on these servers or the network link delays to these servers. Thus, they work less efficiently for loosely-coupled Internet cache peers.

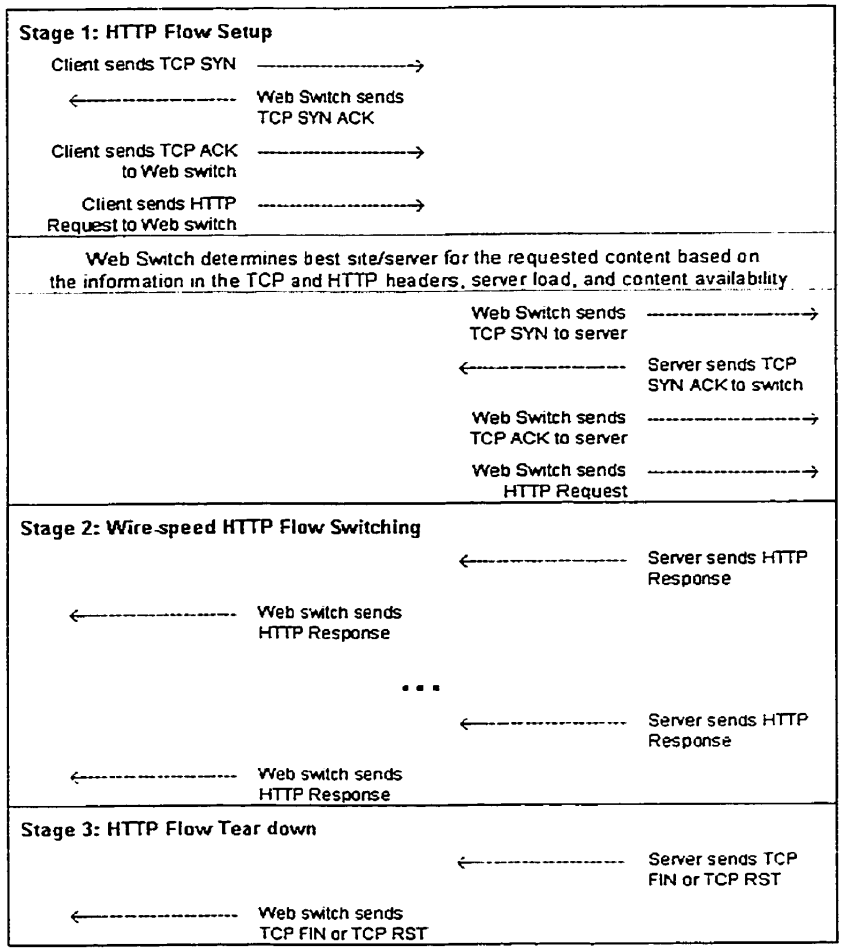


Figure 2.6 Layer 5 Switching traffic flows (adopted from [23])

In Section 2.3, we introduced transparent Web caching techniques and described the function of Layer 4 and Layer 5 switching. Layer 4 switching-based transparent Web caching can transparently redirect all HTTP traffic to cache servers. Layer 5 switching-based transparent Web caching redirects HTTP requests according to their content. In Layer 5 switching-based transparent Web caching, non-cacheable HTTP requests are redirected to bypass cache servers. Transparent Web caching not only makes the deployment and configuration of the caching system easier, but also improves its performance. However, no transparent Web caching scheme supports distributed cache cooperation, which is widely supported by proxy Web caching systems and has proven effective.

Chapter 3

LB-L5 Web Caching

Existing L5 switching-based Web caching schemes use TCP spoofing to inspect the content of the HTTP header of a client request, and then redirect non-cacheable requests to bypass the cache servers. This increases the cache hit rate and improves system performance because only cacheable requests are directed to cache servers. Existing approaches, however, do not support distributed Web caching. A Layer 5 switch and its associated cache server can only be placed at the gateway of a network domain. This results in problems such as congestion/latency caused by TCP spoofing and limited cache sharing between two or more domains.

In this chapter, we propose a fully distributed Web caching scheme that extends the capabilities of the Layer 5 switching-based approaches to support distributed Web caching. The goals of the proposed L5-based scheme are to balance proxy cache server workload and improve response time for client requests. We call the scheme the LB-L5 (Load-Balancing Layer-5-switching-based) Web caching scheme.

3.1 LB-L5 Web Caching Scheme Overview

LB-L5 uses information about the proxy cache server workload, network link delay, cache content and access-frequency to redirect HTTP requests, which in turn balances cache server workload and reduces average response times. LB-L5 uses a weighted Bloom Filter to represent cache content and access-frequency information, which enables LB-L5 to implement access-frequency-aware cache cooperation. LB-L5 extends ICP, the most popular Web caching protocol, to support communication between cache servers and Layer 5 switches, and so is compatible with existing Web cache systems.

In addition to the transparency of existing L5 switching-based schemes, LB-L5 provides the following benefits:

1. **Balanced cache server workload.** In LB-L5, client requests are directed with the intention of balancing the workload of cooperating cache servers.
2. **Reduced response time.** In a fully distributed scheme cache servers can be placed closer to clients. The number of hops for clients to access the caches is reduced. Moreover, LB-L5 can balance workload among cooperating cache servers and use network link delay information to redirect the http request in order to avoid high-cost remote hits.
3. **Improved cache sharing.** The cooperation among distributed cache servers can increase the overall hit rate by allowing more clients to share caches.
4. **Reduced possibility of congestion caused by TCP spoofing.** Layer 5 switches can

be distributed within the network. Therefore, the number of TCP flows each switch needs to handle is reduced.

5. **Avoidance of a single point of failure.** When a cache server stops running the L5 switches can redirect client requests to other cache servers.

The properties of LB-L5 are summarized as follows:

- **Fully distributed architecture**

As shown in Figure 3.1, LB-L5 uses a fully distributed architecture. HTTP requests from a cluster of clients are first inspected by a L5 switch, which then redirects the requests to one of a set of cooperating proxy cache servers or to the original Web servers.

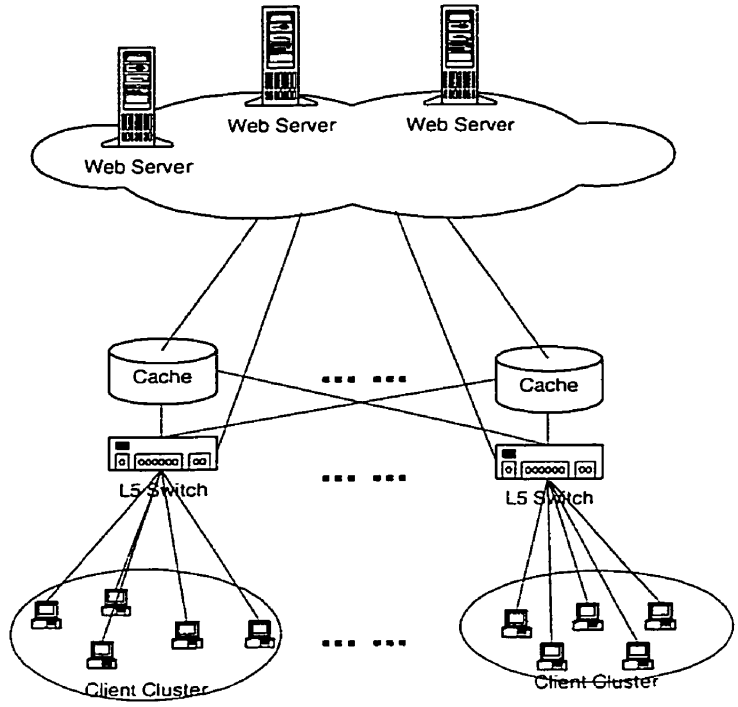


Figure 3.1 LB-L5 Web caching architecture

In LB-L5, to redirect client requests to the most suitable cache server, a L5 switch should have information about the cache content and the workload of every cooperating cache server, as well as the network link delay between the L5 switch and every cache server. The compressed cache content information is generated and published periodically to the L5 switches by cache servers. The L5 switches also query cache server workload and measure network link delay by using extended ICP messages. A detailed description of the cache content representation method and the use of extended ICP messages are provided in Section 3.2.

- **Balance between cache content distribution, workload and network latency**

LB-L5 uses a weighted Bloom Filter to represent the cache content and the access frequencies of cached objects at a cache server. The weighted Bloom Filter is smaller in size than a complete directory of cached objects. Moreover, it reflects the characteristics of the requests handled by the server since we use the weighted Bloom Filter to carry cache object access-frequency information.

A Bloom Filter is used to represent cache content in the Cache Digest [19] and Summary Cache [20] schemes. Every cached object is assigned a fixed number of bits in the filter. In the weighted Bloom Filter, cached objects are assigned different numbers of bits according to their access frequencies. This enables the filter to carry access-frequency information, and more accurately indicate whether an object is in a cache. A description of the weighted Bloom Filter is presented in Section 3.2.1.

The access-frequency information of cached objects is one factor that influences LB-L5's

HTTP request routing function. In cases where several cache servers have the requested object cached, and have similar workload and network link delay, L5 switches redirect the request to the cache server where the requested object has highest access-frequency. Because cached objects with lower access frequencies will expire and be evicted sooner, the routing decision based on object access-frequency is helpful to reduce caching of redundant copies.

Other factors of LB-L5's HTTP request routing function are the cache server workload and the network link delay. LB-L5 does not purely emphasize the cache hit rate, because in a fully distributed environment, a remote hit or a hit on a very busy cache server can be slower than fetching the requested objects from the original Web server. Thus, LB-L5 makes routing decisions by considering the combined effects of cache content (hit rate), cache server workload, and network link delay.

In LB-L5, the workload of a cache server is measured by the number of concurrent TCP sessions established to the server. L5 switches use extended ICP messages to query workload information, and, at the same time, measure network link delays by the message round trip time between the switch and a cache server.

- **Backward compatibility**

LB-L5 extends, but does not replace, ICP (Internet Cache Protocol), which is the most popular protocol in existing Web caching systems. Unused OP Codes in ICP are used for LB-L5 messages. This makes LB-L5 cache servers can cooperate with switches and/or cache servers that are not LB-L5 aware, and LB-L5 messages transparent to these switches and/or

cache servers. Because Cache servers in LB-L5 fully support ICP, LB-L5 is backward compatible with ICP aware Web caching systems.

3.2 LB-L5 Detailed Description

The following sections contain a detailed description of the LB-L5 scheme. We first introduce the weighted Bloom Filter, which is used to represent cache content and access-frequency information. We then describe the ICP extension, and explain how to use extended ICP messages to exchange cache content information, query proxy server workload and measure the network link delay. Finally, we provide pseudo code for the algorithms describing the cooperation of L5 switches and cache servers.

3.2.1 Cache Content Representation

The use of a Bloom Filter to compactly represent cache content was proposed in Cache Digest [19] and Summary Cache [20]. In these schemes, every object in a cache is represented in a Bloom Filter by using a fixed set of hash functions to compute hash values for its key (URL), and then setting corresponding bits of the filter to 1. To check if a specific object is in the cache, the same set of hash functions are computed and corresponding bits are checked. If one or more of the bits is 0, then the object is not in the cache. But if all bits are 1, we have two cases:

1) true prediction: the object is in the cache;

2) false prediction: the object is not in the cache while the filter indicates it is.

By adjusting the filter size and the number of hash functions, Summary cache achieves a false prediction probability around 4.7% or lower [20].

However, a desirable property of a cache content representation for LB-L5 is the ability to carry access-frequency information. As mentioned in the previous section, this information is used to route the HTTP requests. In LB-L5, if an object is cached at more than one cache server having similar values for workload and network link delay, requests for the object should be directed to a server where the object has the highest access-frequency. This routing policy helps to reduce the duplication of object caching because cached objects with lower access-frequency will expire and be evicted sooner.

Our cache content representation, which we call the weighted Bloom Filter, is based on signature files proposed for general text file retrieval by Faloutsos [39]. A signature file is fundamentally the same as a Bloom Filter. The signature file method builds a weighted Bloom filter that uses a varying number of hash functions for objects with different access frequencies.

The weighted Bloom Filter divides all cache objects into different sets and assigns a weight to each set according to their access-frequencies. The weight of a set is the number of hash functions that should be used to compute hash values for the key (URL) of an object belonging to the set, or the number of bits set to 1 in the filter for the object. We call this method weighted Bloom Filter because it assigns a weight to each cache-object set. The weighted Bloom Filter assigns a heavier weight to a set with higher access-frequency. Thus,

objects with higher access frequencies are represented with more bits set to 1 in the filter. At the time of looking up an object, the number of bits set to 1 indicates the access-frequency rank of the object.

An example of using the weighted Bloom Filter to represent cache content information is shown in Figure 3.2. In this example, a cache server divides all cache objects into two sets according to their access frequencies. The filter size is 16. "www.yahoo.com" belongs to a high access-frequency set, S_1, whose weight is 6. "www.beowulf.org" and "www.delphi.net" belong to a low access-frequency set, S_2, whose weight is 4.

Case (a) in Figure 3.2 illustrates how to use the weighted Bloom Filter to represent cache content. We compute 6 hash functions for the key of every object in S_1 (4 for objects in S_2), and set the corresponding bits in the filter to 1. Figure 3.2 (cases b-e) describes the procedure of looking up an object from a particular cache and its access-frequency rank. We first compute the 4 hash functions, and check the corresponding bits in the filter representing the cache content. If one or more of the 4 bits is 0, the object is not in the cache. Otherwise, we compute the additional $6 - 4 = 2$ hash functions and check the corresponding bits. If both bits are 1, we can predict that this object is in the cache and belongs to the high access-frequency subset. On the other hand, if any one of the 2 bits is 0, the object is in the cache but belongs to the low access-frequency subset.

a)cache content representation:

Object URL	Access freq.	Set	Weight	Representation
www.yahoo.com	high	S1	6	1001 0110 0001 0010
www.beowulf.org	low	S2	4	1000 0100 1001 0000
www.delphi.net	low	S2	4	0000 0100 0101 0010

weighted Bloom Filter
1001 0110 1101 0010

b)look up www.yahoo.com

Steps	Weight	Representation	Results
1.check if it is in S2	4	1000 0100 0001 0010	(in S2)
2.check if it is in S1			
2.1 compute additional hash	6-4=2	0001 0010 0000 0000	
2.2 check			(in S1)

(in the cache, high access freq.)

c)look up www.beowulf.org

Steps	Weight	Representation	Results
1.check if it is in S2	4	1000 0100 1001 0000	(in S2)
2.check if it is in S1			
2.1 compute additional hash	6-4=2	0010 0001 0000 0000	
2.2 check			(not in S1)

(in the cache, low access freq.)

d)look up www.whouse1.com

Steps	Weight	Representation	Results
1.check if it is in S2	4	0001 0010 1001 0000	(not in S2)

(not in the cache)

e)look up www.whouse2.com (false prediction)

Steps	Weight	Representation	Results
1.check if it is in S2	4	1001 0110 0000 0000	(in S2)
2.check if it is in S1			
2.1 compute additional hash	6-4=2	0100 0000 0000 0010	
2.2 check			(not in S1)

(in the cache, low access freq.)

Figure 3.2 Cache content representation based on weighted Bloom Filter

The weighted Bloom Filter can be used to represent cache content and carry cache access-frequency information at the same time. In addition, it gives lower false prediction probability than the basic Bloom Filter used in Cache Digest and Summary Cache. By using the weighted Bloom Filter to represent cache content, LB-L5 can support directory-based HTTP request routing in order to avoid the query/response delay in query-based schemes such as ICP. Meanwhile, the access-frequency information carried in a weighted Bloom Filter enables LB-L5 to support access-frequency-aware cache cooperation, which helps to reduce the duplication of object caching, since cached objects with low access-frequency will expire and be evicted sooner. A detailed analysis of the weighted Bloom Filter is presented in Appendix A.

The hash functions used for building the weighted Bloom Filter are based on MD5 [40], which is also used in Summary Cache and Cache Digest. MD5 is a one-way hash function designed by Ron Rivest. MD stands for Message Digest; the algorithm produces a 128-bit hash value, or message digest, for an arbitrary-length input message. MD5 has a good collision-resistant property, which means that it is difficult to find two random messages sharing a common hash value. The collision-resistant property makes MD5 suitable for building the weighted Bloom Filter, since we want different objects to have different representations.

The hash functions used in LB-L5 take groups of bits from the 128-bit MD5 hash value of a URL. This method was recommended by Cao in Summary Cache [20]. The computational overhead of MD5 is negligible compared with the user and system CPU overhead incurred by

caching [20,41]. The MD5 algorithm is described in Appendix B.

3.2.2 ICP Extension

An ICP message includes a 20-byte header and a variable-sized payload, which typically contains a URL. Figure 3.3 shows the ICP message format.

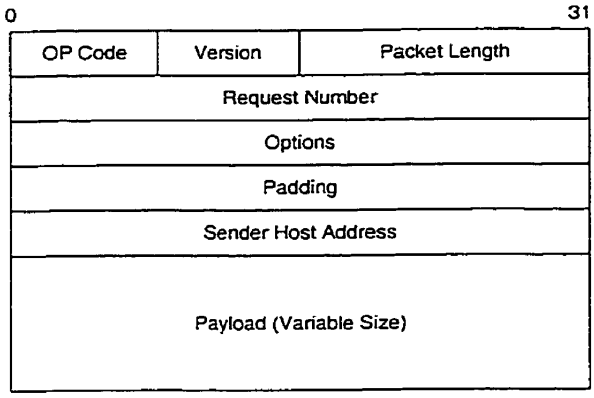


Figure 3.3 ICP Message Format

The contents of the header are as follows:

OP Code: the type of message. For example, a query message's OP Code is ICP_QUERY, a reply message's OP Code is ICP_MISS or ICP_HIT.

Version: ICP version to maintain backward compatibility.

Packet Length: the total size of the ICP message.

Request Number: an opaque integer identifier to match queries and responses.

Options: bitfield to support optional features and new additions to ICP.

Padding: unused. (In Harvest, Options and Padding fields are slated to be used for

authorization.)

Sender Host Address: originally intended to hold the IPv4 address. However, since the originating address is also available from the socket API, this field is redundant and often unused.

Table 3.1 shows the currently defined ICP OP Codes. The spacious OP Code field and the variable length payload allow ICP to be easily extended.

Value	Name
0	ICP_OP_INVALID
1	ICP_OP_QUERY
2	ICP_OP_HIT
3	ICP_OP_MISS
4	ICP_OP_ERR
5-9	UNUSED
10	ICP_OP_SECHO
11	ICP_OP_DECHO
12-20	UNUSED
21	ICP_OP_MISS_NOFETCH
22	ICP_OP_DENIED
23	ICP_OP_HIT_OBJ

Table 3.1 ICP OP Codes

LB-L5 defines the following four new ICP messages for updating content frequency distribution information and querying cache server workload information:

ICP_UPDATE_CONTENT: used by cache severs to inform L5 switches of changes of their cache content and access-frequency.

ICP_UPDATE_CONTENT_ACK: used by L5 switches to acknowledge an update of cache content and access-frequency information.

ICP_QUERY_WORKLOAD: used by L5 switches to query cache server workload.

ICP_UPDATE_WORKLOAD: used by cache servers to answer a workload query from a L5 switch.

These four new ICP messages are assigned the OP Codes 12 to 15, respectively.

Figure 3.4 shows the procedure for updating cache content information with extended ICP messages. Each cache server periodically computes its cache content information and publishes it to every L5 switch. To do this, a cache server multicasts an extended ICP message, **ICP_UPDATE_CONTENT**, to the L5 switches. The OP Code is set to **ICP_UPDATE_CONTENT** and the content information and a timestamp are put in the payload field of the message.

When a L5 switch receives an **ICP_UPDATE_CONTENT** message and successfully updates the content information of the sending cache server, it sends an **ICP_UPDATE_CONTENT_ACK** message back to the cache server to acknowledge the update. If the cache server receives **ICP_UPDATE_CONTENT_ACK** responses from all L5 switches before it times out, then the update is successfully completed. Otherwise, the cache server sends out the **ICP_UPDATE_CONTENT** message, with the same timestamp, to all L5 switches that did not respond. After sending the second **ICP_UPDATE_CONTENT** message, the cache server finishes this round of updates, that is, it does not wait for another response. A L5 switch may, however, receive two **ICP_UPDATE_CONTENT** messages with the same timestamp from a particular cache server. In this case, only the first one is accepted.

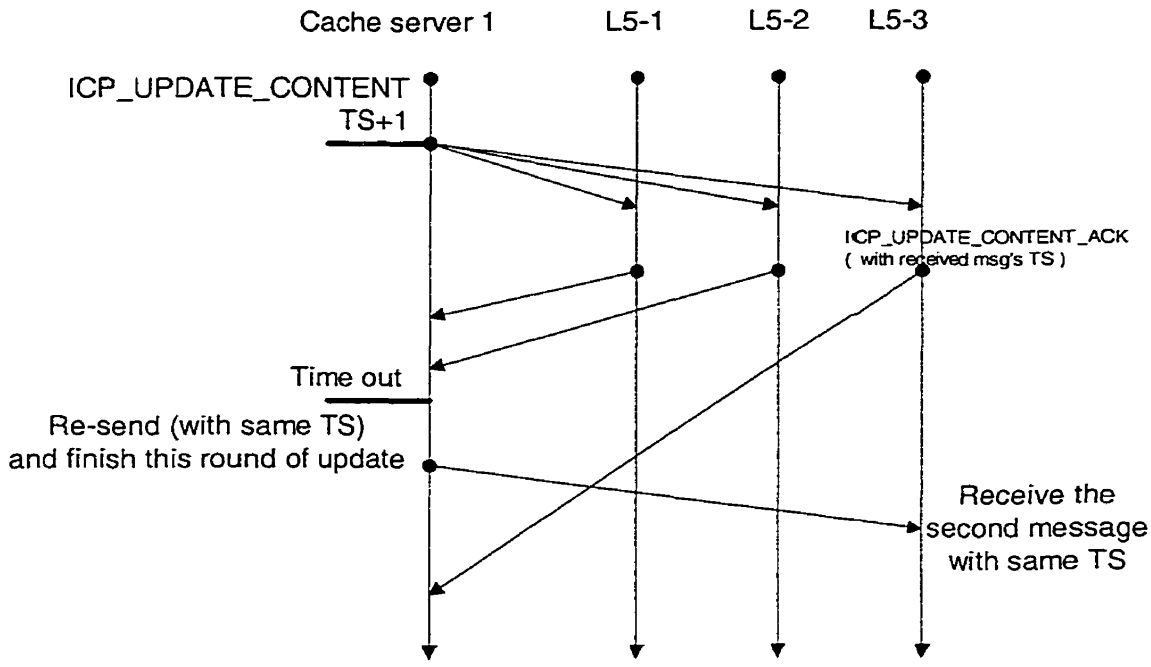


Figure 3.4 Updating cache content information with extended ICP messages

Figure 3.5 shows the procedure of a L5 switch querying workload information with extended ICP messages. Each L5 switch periodically queries all cache servers for workload information by multicasting an **ICP_QUERY_WORKLOAD** message to all cache servers. Every time a L5 switch sends out an **ICP_QUERY_WORKLOAD** message, it increases the timestamp for every cache server by 1. When a cache server receives an **ICP_QUERY_WORKLOAD** message, it sends back an **ICP_UPDATE_WORKLOAD** message, with its workload information and the timestamp provided by the switch in the payload field of the **ICP_QUERY_WORKLOAD** message.

If the L5 switch receives **ICP_UPDATE_WORKLOAD** messages from all cache servers before it times out, this round of queries is successfully completed. Otherwise, it resends the

ICP_QUERY_WORKLOAD message to all the cache servers that had not responded, and waits for responses until it receives **ICP_UPDATE_WORKLOAD** messages from these cache servers or times out.

In a round of workload queries, a L5 switch sends out at most two **ICP_QUERY_WORKLOAD** messages to a cache server. If a cache server fails to respond to two consecutive **ICP_QUERY_WORKLOAD** messages, the L5 switch sets the server's workload to be infinite, in order to avoid redirecting client requests to this cache server. Upon receiving an **ICP_UPDATE_WORKLOAD** message from a cache server with a ***newer*** timestamp than in previous **ICP_UPDATE_WORKLOAD** messages from that server, or if the previously recorded workload for the cache server is infinite, the L5 switch updates the workload information of the sending cache server.

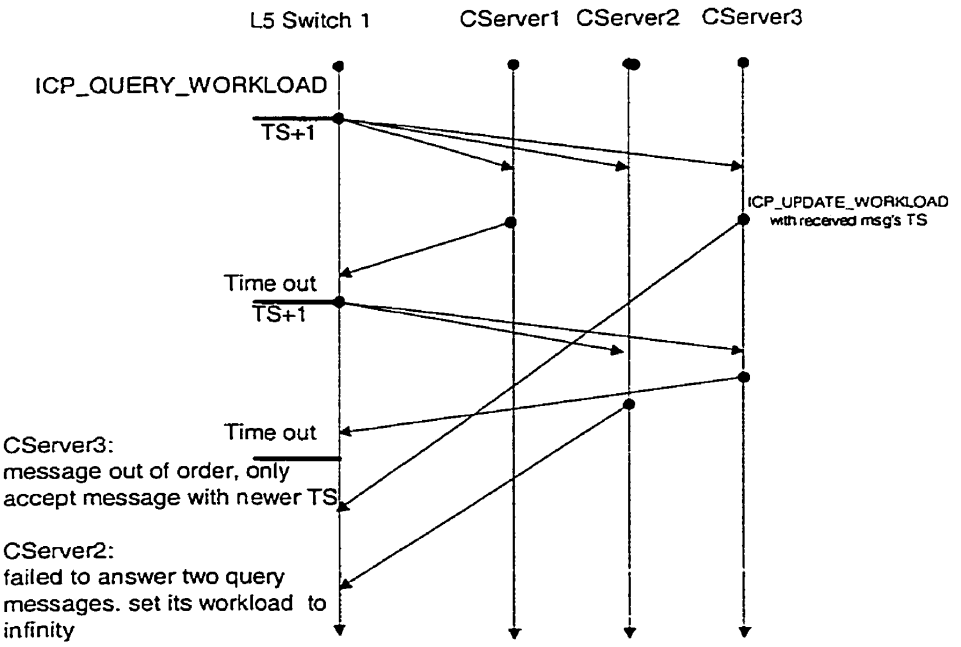


Figure 3.5 Querying workload information with extended ICP messages

3.2.3 L5 Switches working with Extended ICP

In the LB-L5 scheme, the Layer 5 switch uses cache server workload, network link delay, cache content and access-frequency information, in addition to the HTTP header information inspected from the content of client requests, to route a request to the most suitable cache server or its destination Web server. The information is obtained as follows:

- Cache content and access-frequency information: Cache content and access-frequency information is obtained and represented with a weighted Bloom Filter by each cache server. It is sent to each Layer 5 switch using the extended ICP message **ICP_UPDATE_CONTENT**.
- Cache server workload information: A Layer 5 switch obtains the workload information from a cache server by sending each cache server an **ICP_QUERY_WORKLOAD** message, which is answered by the receiving cache server with an **ICP_UPDATE_WORKLOAD** message, whose payload field carries the server's workload information. The workload of a cache server is measured as $\frac{Num\_of\_TCP\_Sessions}{Max\_TCP\_Sessions}$, where *Num_of_TCP_Sessions* is the number of current TCP sessions established at the server, and *Max_TCP_Sessions* is the maximum number of TCP sessions that can be handled by the server.
- Network link delay information: There are several proposed approaches to measuring network latency. One category of approaches uses tools such as ping, traceroute, and Zone transfer from a DNS server [42]. Another category of approaches uses network services

such as SONAR [43], IDMAPS [44] and ReMoS [45]. However, we did not find any commonly supported approach to obtain instantaneous latency information. In LB-L5, the message round trip time between a Layer 5 switch and a cache server is used to measure the network latency. This message round trip time includes the propagation delay, packet transmission delay and network access delay.

Upon receiving a HTTP request, a L5 switch makes a routing decision as follows:

1. If the request is non-cacheable, the switch redirects it to the original Web server.
2. For every cache server, the switch estimates the time needed for fetching the requested object from that server as follows:
   2.1 The switch computes hash values for the request's URL, and then checks the weighted Bloom filters representing the server's cache content to see if the server has the requested object in its cache.
   2.2 The switch then estimates the time (T) needed for fetching the requested object from the server as:
      2.2.1 If the requested object is available at the cache server, then T includes the time for connecting to the server, sending the request from the switch to the cache server, searching for the object at the server, moving the object from disk to memory, and sending the object from the server to the switch.
      2.2.2 If the object is not available at the cache server, then the time for the cache

server to fetch the object from the Web server is added.

3. The switch then pre-selects a set of cache servers that have a reasonably short response time. A threshold can be used for this purpose.

4. The switch chooses the cache server from the pre-selected set that has the highest frequency for the requested objects, and redirects the request to the server.

To make the routing decisions and redirect client requests, a L5 switch maintains the following information for every cache server:

IPAddress: IP address of the cache server.

ContentDistribution: a weighted Bloom Filter representing the cache content of the cache server.

WorkLoad: the workload of the cache server

NetworkLatency: the message round trip time between the cache server and the switch.

WorkLoad_Query_Time: the time the last **ICP_QUERY_WORKLOAD** message was sent to the cache server.

WorkLoad_Query_Response_Time: the time of the most recently received **ICP_UPDATE_WORKLOAD** message from the cache server.

WorkLoad_Query_TS: the timestamp, represented with a sequence number, carried in the most recent **ICP_QUERY_WORKLOAD** message sent to the cache server.

Last_WorkLoad_UpdateMsg_TS: the timestamp, represented with a sequence number,

carried in the most recent **ICP_UPDATE_WORKLOAD** message received from the cache server.

A L5 switch uses an array, CacheServerArray, to store the above information about all cooperating cache servers. As described in Figure 3.6, a L5 switch has a process to receive ICP messages from the cooperating cache servers and update the workload and network link delay information of these servers. Upon receiving an ICP message, the ICP-message-receiving process first locates the sending server in its CacheServerArray according to the message's SenderAddress (lines 2-7), and then processes the ICP message according its OP Code.

If the message is **ICP_UPDATE_CONTENT**, the L5 switch updates the content information of the server sending the message, and sends the server an **ICP_UPDATE_CONTENT_ACK** message (using the timestamp from the **ICP_UPDATE_CONTENT** message) to acknowledge the update (lines 9-14). If the message is **ICP_UPDATE_WORKLOAD**, the L5 switch updates the workload information of the server if the message carries more recent workload information or if the previously recorded workload information is not valid (lines 15-20). An **ICP_UPDATE_WORKLOAD** message (from a cache server to a L5 switch) is an immediate response to an **ICP_QUERY_WORKLOAD** message (from a L5 switch to a cache server). A L5 switch uses these two messages to measure the message round trip time between the switch and a cache server (lines 21-23).

A L5 switch periodically queries the workload information of all cache servers by sending **ICP_QUERY_WORKLOAD** messages to the servers. As described in Figure 3.7, a L5 switch

increments the workload query timestamp and records the query sending time before sending an **ICP_QUERY_WORKLOAD** to a cache server (lines 4-6 and 14-16). The timestamp is used to match up the query and responding messages. Recording the query send time facilitates measuring the message round trip time between the switch and the cache server. In a round of queries, a L5 switch sends at most two **ICP_QUERY_WORKLOAD** messages to a particular cache server. If a server does not respond to two consecutive queries, the L5 switch sets the workload and network link delay to infinity (lines 25-28). Therefore, a L5 switch can avoid redirecting client requests to non-responding server.

```
1. Process OnReceiveMessage(msg: ICPMessage)
2. for i:= 1 to NumOfCacheServers do
3.    if( CacheServerArray[i].IPAddress == msg.SenderAddress) then
4.       cserver := CacheServerArray[i]
5.       break
6.    endif
7. endfor
8. switch (msg.OPCode)
9.   case ICP_UPDATE_CONTENT:
10.      if( msg.TS > cserver. Last_Content_UpdateMsg_TS) then
11.            cserver.ContentDistribution := msg.ContentDistribution
12.            cserver.Last_Content_UpdateMsg_TS := msg.TS
13.            SendMessage(ICP_UPDATE_CONTENT_ACK, msg.TS,
                           cserver.IPAdderss)
14.      endif
15.   case ICP_UPDATE_WORKLOAD:
16.      if((msg.TS > cserver.Last_WorkLoad_UpdateMsg_TS)
            OR ( cserver.WorkLoad == INFINITE ) ) then
17.            cserver.WorkLoad := msg.WorkLoad
18.            cserver.Last_WorkLoad_UpdateMsg_TS := msg.TS
19.            cserver.WorkLoad_Query_Response_Time:= time()
20.      endif
21.      if (msg.TS == cserver.WorkLoad_Query_TS) then
22.            cserver.NetworkLatency :=
                 cserver.WorkLoad_Query_Response_Time -
                 cserver.WorkLoad_Query_Time
23.      endif
24. endswitch
25. end
```

Figure 3.6 The algorithm of the ICP-message-receiving process on a L5 switch

```
1. Procedure QueryWorkLoad ()
2. for i:=1 to NumOfCacheServers do
3.   cserver := CacheServerArray[i]
4.   cserver.WorkLoad_Query_TS += 1
5.   cserver.WorkLoad_QueryTime := time()
6.   SendMessage(ICP_QUERY_WORKLOAD, cserver.WorkLoad_Query_TS,
                 cserver.IPAddress)
7. endfor
8.  SendTime := time()
9.  Wait until ( time() > ( SendTime + TIME_OUT_THRESHOLD))
10. flagSendAgain:= FALSE
11. for i:=1 to NumOfCacheServers do
12.  cserver := CacheServerArray[i]
13.  if(cserver.Workload_Query_Response_Time <
      cserver.WorkLoad_QueryTime) then
14.        cserver.WorkLoad_Query_TS += 1
15.        cserver.WorkLoad_QueryTime := time()
16.        SendMessage(ICP_QUERY_WORKLOAD, cserver.WorkLoad_Query_TS,
                       cserver.IPAddress)
17.        flagSendAgain :=TRUE
18.  endif
19. endfor
20. if (flagSendAgain) then
21.   SendTime := time()
22.   Wait until ( time() > ( SendTime + TIME_OUT_THRESHOLD))
23.   for i:=1 to NumOfCacheServers do
24.      cserver := CacheServerArray[i]
25.      if(cserver.Workload_Query_Response_Time <
          cserver.WorkLoad_QueryTime)then
26.        cserver.WorkLoad := INFINITE
27.        cserver.NetworkLatency := INFINITE
28.      endif
29.   endfor
30. endif
31. end
```

Figure 3.7 The algorithm of querying cache server workload information

3.2.4 Cache Servers working with Extended ICP

To cooperate with Layer 5 switches, cache servers must be extended to support new ICP messages, and use these messages to inform the L5 switches about their cache content and workload information.

To cooperate with L5 switches, a cache server maintains the following information for every L5 switch:

IPAddress: IP address of this cache server.

Content_Update_ TS: the timestamp, represented with a sequence number, carried in the most recent **ICP_UPDATE_CONTENT** message sent to this L5 switch.

Content_Update_AckMsg_TS: the timestamp, represented with a sequence number, carried in the most recent **ICP_UPDATE_CONTENT_ACK** message received from this L5 switch.

A cache server uses an array, SwitchArray, to store the above information about all cooperating L5 switches. As described in Figure 3.8, a cache server has a process to receive ICP messages from the cooperating L5 switches. Upon receiving an ICP message, the process first locates the sending L5 switch in its SwitchArray according to the message's SenderAddress (lines 2-7), and then processes the ICP message according its OP Code. If the message is **ICP_UPDATE_CONTENT_ACK**, the cache server records the acknowledgement timestamp, which is used to check if the content update on the switch is successfully finished. If the message is **ICP_QUERY_WORKLOAD**, the cache server responds with an

ICP_UPDATE_WORKLOAD message using the timestamp from the L5 switch, so that the switch can keep track of the query and the response (lines 11-12). The cache server's workload information is put in the payload field of the responding message.

```
1. Process OnReceiveMessage(msg: ICPMessage)
2. for i:= 1 to NumOfSwitches do
3.   if( SwitchArray[i].IPAddress == msg.SenderAddress) then
4.       sw := SwitchArray[i]
5.       break
6.   endif
7. endfor
8.  switch (msg.OPCode)
9.   case ICP_UPDATE_CONTENT_ACK:
10.      sw.Content_Update_AckMsg_TS := msg.TS
11.  case ICP_QUERY_WORKLOAD:
12.      SendMessage(ICP_UPDATE_WORKLOAD, msg.TS, sw.IPAddress)
13. endswitch
14.end
```

Figure 3.8 The algorithm of the ICP-message-receiving process on a cache server

Each cache server periodically publishes its cache content information to every cooperating L5 switch. Similar to the algorithm used in a L5 switch, a cache server also uses a sequence number as a timestamp to keep track of **ICP_UPDATE_CONTENT** and **ICP_UPDATE_CONTENT_ACK** messages. As described in Figure 3.9, a cache server may send out one or two **ICP_UPDATE_CONTENT** messages to a particular L5 switch in every round of content update, but it only increases the timestamp of this switch by 1 (line 4). This is because a L5 switch might receive two **ICP_UPDATE_CONTENT** messages in the same round. Thus, if a L5 switch receives two **ICP_UPDATE_CONTENT** messages with a same timestamp, it can simply ignore the second message.

```
1. Procedure UpdateContentDistribution(content:WEIGHTED_BLOOM_FILTER)
2. for i:=1 to NumOfSwitches do
3.    sw := SwitchArray[i]
4.    sw.Content_Update_TS += 1
5.    SendMessage(ICP_UPDATE_CONTENT,sw.Content_Update_TS sw.IPAddress)
6.  endfor
7.  SendTime := time()
8.  Wait until ( time() > ( SendTime + TIME_OUT_THRESHOLD))
9.  for i:=1 to NumOfSwitches do
10.  sw := SwitchArray[i]
11.  if( sw.Content_Update_TS ! = sw.Content_Update_AckMsg_TS) then
12.       SendMessage(ICP_UPDATE_CONTENT,sw.Content_Update_TS
                     sw.IPAddress)
13.  endif
14. endfor
15. end
```

Figure 3.9 The algorithm of updating cache content information

3.3 Summary

In this chapter, we introduced the Load Balancing Layer 5 (LB-L5) switching-based Web caching scheme. LB-L5 uses a weighted Bloom Filter to represent cache content in order to support directory-based cache cooperation. The weighted Bloom Filter has two salient properties. First, the weighted Bloom Filter can be used to represent cache content and carry cache access-frequency information at the same time. Second, the weighted Bloom Filter gives lower false prediction probability than the basic Bloom Filter used in Cache Digest and Summary Cache. By using the weighted Bloom Filter to represent cache content, LB-L5 can support directory-based HTTP request routing in order to avoid the query/response delay existing in query-based schemes such as ICP. Meanwhile, the access-frequency information carried in a weighted Bloom Filter enables LB-L5 to support access-frequency-aware cache

cooperation, which helps to reduce the duplication of caching.

LB-L5 uses cache content and access-frequency information, cache server workload information and network link delay information to route HTTP requests to one of a set of cooperating cache servers. This routing policy makes LB-L5 suitable to support distributed Web caching since it avoids fetching objects from remote or busy cache servers whenever a less expensive copy can be found. LB-L5 achieves backward compatibility with existing Web caching methods by extending ICP, the most popular Web caching protocol, to facilitate communication between the cache servers and the switches. This means that LB-L5 cache servers can cooperate with switches and/or cache servers that are not LB-L5 aware, and that LB-L5 messages are transparent to these switches and/or cache servers.

Chapter 4

Performance Evaluation

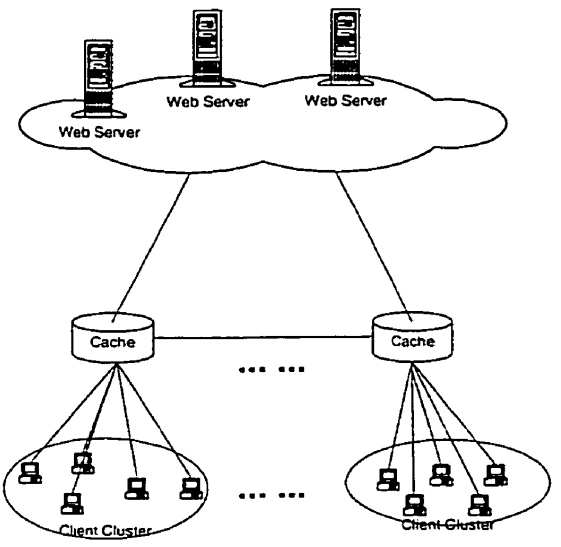
In this chapter, we evaluate the performance of our proposed LB-L5 Web caching scheme. The results are compared with those of ICP, Cache Digest, and basic L5 transparent Web caching. Section 4.1 explains the simulation model adopted in this study, which includes the network model, proxy traces and the simulation software implementation. The effects of network link delay, HTTP request intensity, and the number of cooperating proxy servers on the performance of the Web caching schemes are reported in Section 4.2. Finally, Section 4.3 provides a summary of the results obtained in the simulation study.

4.1 Simulation Model

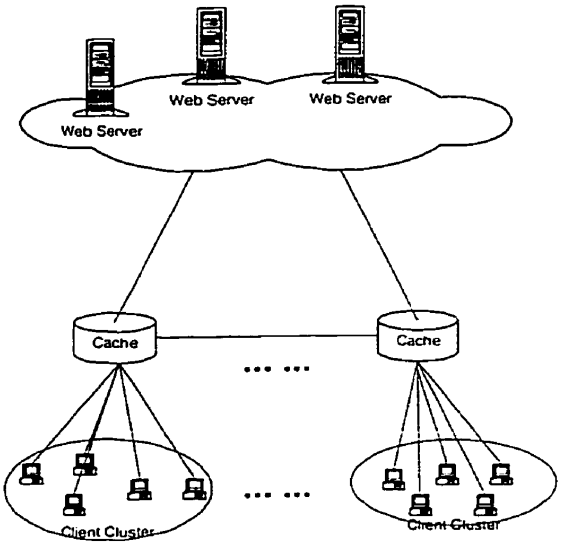
We first describe the simulation model, including the network model and the proxy traces used to generate HTTP request traffic. The simulation of the Web caching schemes is then described, followed by the necessary parameter settings and the simulation software structure.

4.1.1 Network Model

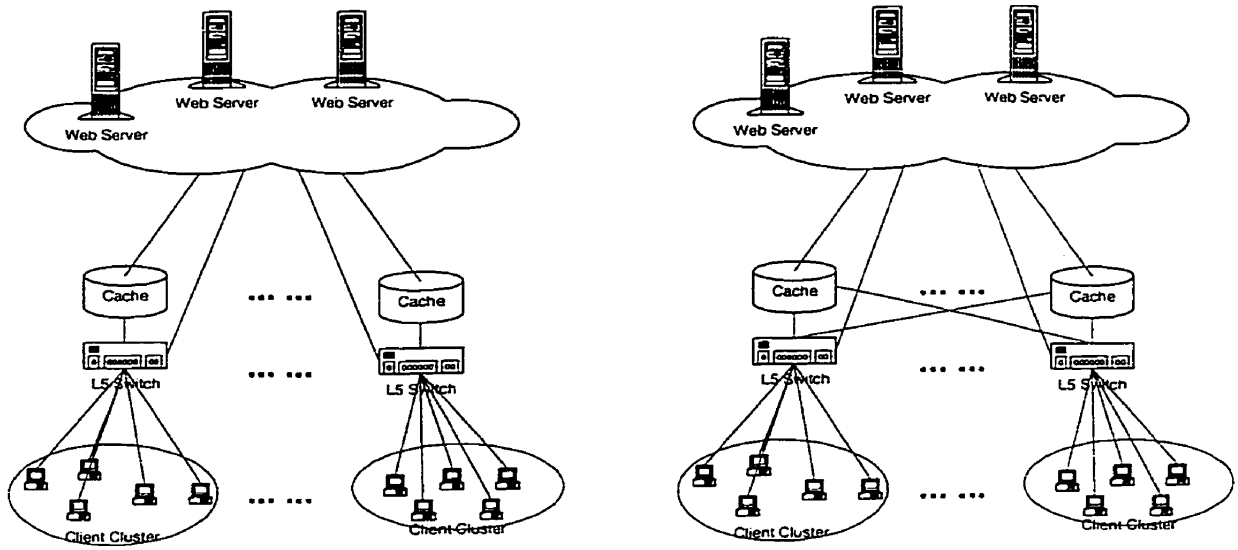
In this study, a fully distributed cache cooperation architecture is simulated. The network model for each of the four simulated Web caching schemes is shown in Figure 4.1. In the ICP and Cache Digest schemes shown in Figure 4.1 (a) and (b), respectively, each proxy cache server accepts HTTP requests from a cluster of clients, and has a link to every other cooperating proxy server. In the basic L5 and LB-L5 Web caching schemes shown in Figure 4.1 (c) and (d), respectively, a L5 switch transparently intercepts HTTP requests from a cluster of clients. The L5 switch redirects a cacheable request to a cache server. Non-cacheable requests are routed directly to the Web server. The difference between the basic L5 and LB-L5 is that LB-L5 supports distributed cache cooperation. In the LB-L5 scheme, a L5 switch can make routing decisions and redirect a HTTP request to one of a set of cooperating cache servers.



(a) ICP



(b) Cache Digest



(c) Basic L5 (d) LB-L5

Figure 4.1 The network model for the Web caching schemes

4.1.2 Proxy Traces

We use publicly available proxy traces from the National Laboratory for Applied Network Research (NLANR) [27] cache servers to generate HTTP requests in the simulation. NLANR network topology is shown in Figure 4.2. The proxy traces from BO (Boulder) and UC (Urbana-Champaign) are used in the simulation.

The proxy trace files are in the Squid native format. An entry in the trace files has the following fields:

- **Timestamp**: the time when the client socker is closed. The format is "Unix time" (seconds since January 1, 1970) with millisecond resolution.

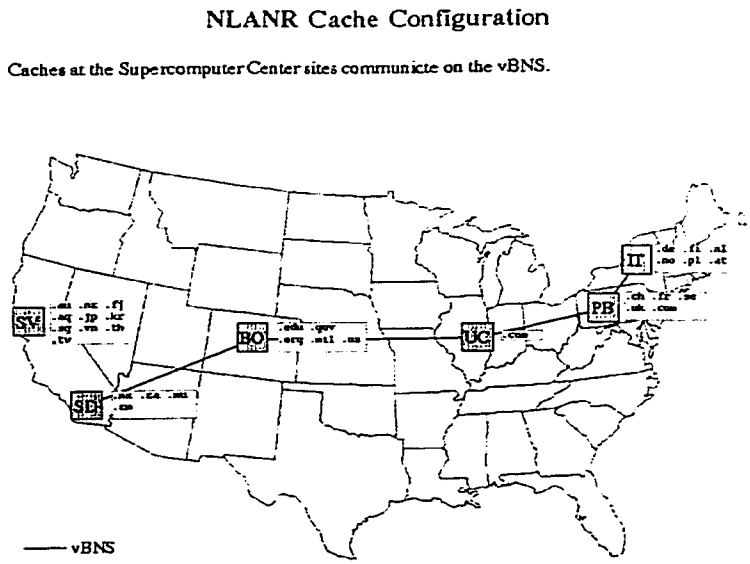


Figure 4.2 NLANR network topology (adopted from [27])

- **ElapsedTime**: The elapsed time of the request, in milliseconds. This is the time between the accept() and close() of the client socket. For persistent HTTP connections, this is the time between reading the first byte of the request, and writing the last byte of the reply.
- **ClientAddress**: the client IP address.
- **LogTag/HTTPCode**: The LogTag describes how the request was treated locally, for example, hit or miss. The HTTP code is obtained from the first line of the HTTP reply header. Non-HTTP requests may have zero reply codes.
- **Size**: the number of bytes transferred from the proxy to the client.
- **RequestMethod**: the HTTP request method. For example, GET or GIMS (Get If Modified Since).

- **URL**: the URL of the requested object.
- **HierarchyData/HostName**: a description of how and where the requested object was fetched.
- **ContentType**: the content type of the request object.

In our experiments, proxy trace files are used in two ways: raw-trace and controlled. In the raw-trace data simulations, the traces are used to simulate the requests from client clusters. In the controlled-parameter simulations, we modify the traces to examine the effects of different parameters. We condense or expand the traces with different factors (shorten or enlarge the interval between requests proportionally), and use different network link delays and different numbers of cooperating cache servers to investigate their effects on the performance of Web caching schemes.

4.1.3 Web Caching Schemes

We evaluate the performance the ICP, Cache Digest, basic L5 and LB-L5 Web caching schemes. We compare LB-L5 with ICP because it is the most popular Web caching protocol in existing Web caching systems. In addition, LB-L5 is compared with Cache Digest because it is also used in the NLANR network together with ICP. The performance comparison of LB-L5, ICP, and Cache Digest can show the improvement achieved by adopting the L5 switch to support distributed caching. Moreover, LB-L5 is compared with the basic L5 scheme to investigate to what extent the workload balancing and cache cooperation features of LB-L5 improve upon the performance of the basic L5.

Figures 4.3 to 4.5 illustrate the basic HTTP request processing procedure in LB-L5. A non-cacheable HTTP request is redirected to a Web server as shown in Figure 4.3.

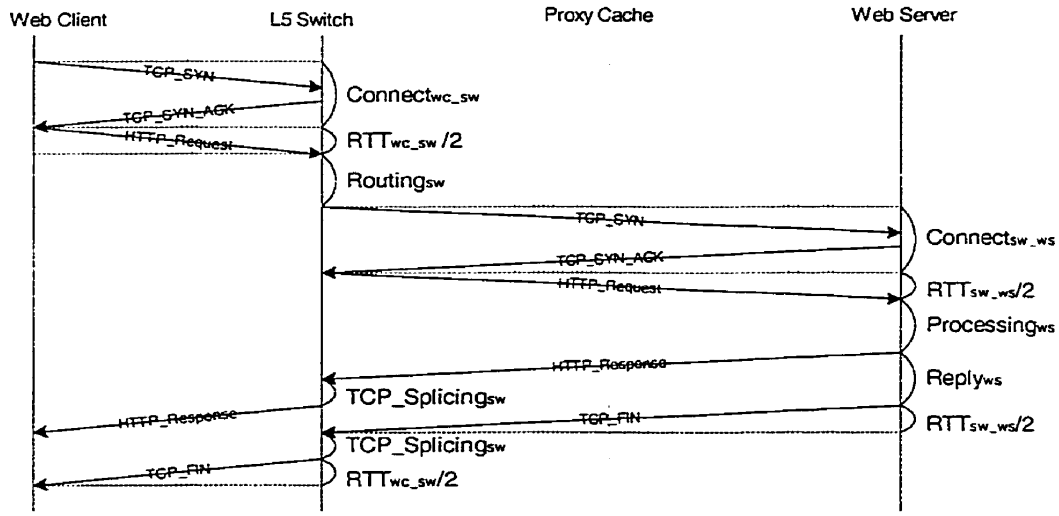


Figure 4.3 Non-cacheable HTTP requests in LB-L5

Cacheable HTTP requests are processed as shown in Figures 4.4 and 4.5. The switch redirects a cacheable request to one of a set of cooperating cache servers. If the receiving cache server finds the requested object in its cache, it replies with a HTTP response message and the object (Figure 4.4). Otherwise, it fetches the objects from a Web server (Figure 4.5).

The detailed HTTP request processing flow charts of the four Web caching schemes are described in Appendix C. These flow charts are drawn according to the scheme descriptions in ICP [15,16], Cache Digest [19], basic L5 [23] and LB-L5. The simulation method is also used by Chiang to model and evaluate hierarchical Web caching schemes [46]. The parameters used in the simulation, such as the request processing time at proxy servers and the round trip time

between sibling proxy servers, are summarized in the next section.

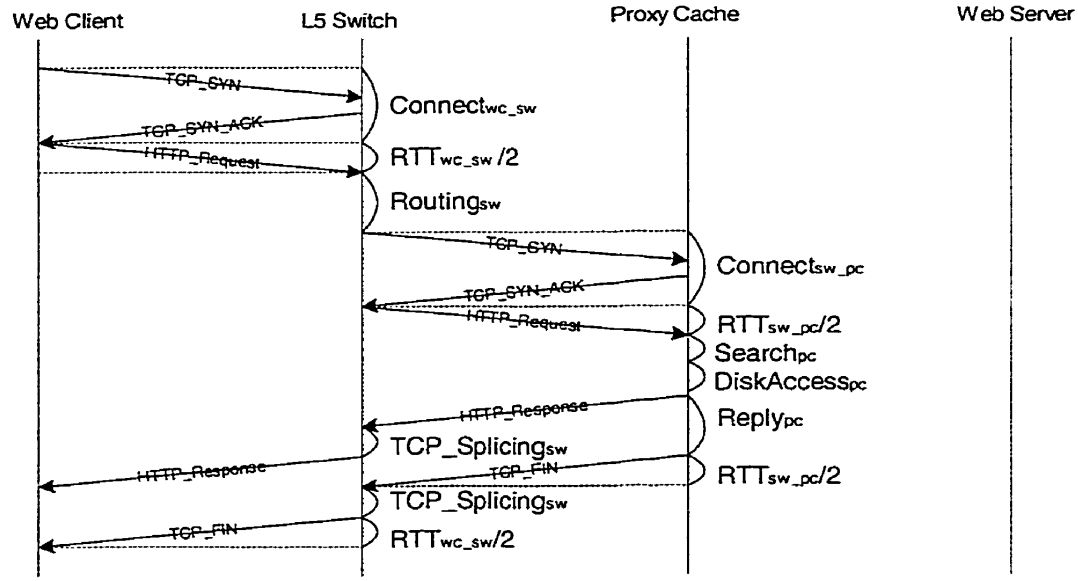


Figure 4.4 Cacheable HTTP GET requests in LB-L5 (proxy cache hit)

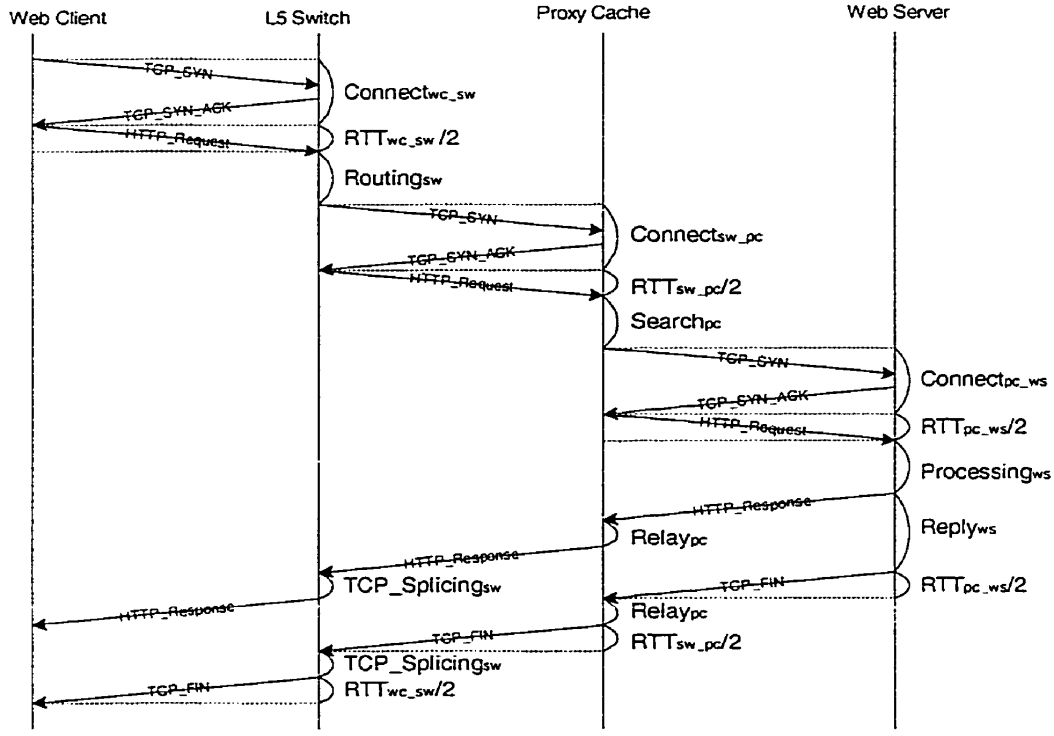


Figure 4.5 Cacheable HTTP GET request in LB-L5 (proxy cache miss)

4.1.4 Simulation Parameter Settings

The parameters used in the simulation are chosen according to data measured by Rousskov [47,48] and IBM's technical report on its L5 switch [25]. These parameters are summarized in Tables 4.1 and 4.2.

Parameter	Meaning	Nominal Value	Source
Connect pc_ws	The elapsed time since a proxy cache sends TCP_SYN to a Web server to the proxy receiving TCP_SYN_ACK from the Web server	350ms	[46,47]
Connect sw_pc	The elapsed time since a L5 switch sends TCP_SYN to a proxy cache server to the switch receiving TCP_SYN_ACK from the cache server	10~450ms	♣♦
Connect sw_ws	The elapsed time since a L5 switch sends TCP_SYN to a Web server to the switch receiving TCP_SYN_ACK from the Web server	350ms	♦
Connect wc_pc	The elapsed time since a Web client sends TCP_SYN to a proxy cache server to the client receiving TCP_SYN_ACK from the cache server	0~30ms	[46,47]
Connect wc_sw	The elapsed time since a Web client sends TCP_SYN to a L5 switch to the client receiving TCP_SYN_ACK from the switch	0~30ms	♦
DiskAccess pc	The time it takes a proxy cache server to retrieve a cache object from disk to memory	100ms	[46,47]
Processing ws	The time it takes a Web server between receiving a request and returning the first byte of the requested object	350ms	[46]
Processing ws_TimeOut	The time it takes a proxy server to abort an outgoing HTTP connection setup request for a Web server	15000ms	[46]
Processing ws_GIMS	The time it takes a Web server between receiving a GIMS request and returning the first byte of the requested object	250ms	[46]
Relay pc	The time it takes a proxy cache server to relay a response to the requesting party	50ms	[46]
Reply pc	The time it takes a proxy cache server to reply an object in memory to the requesting party	150ms	[46,47]
Reply pc_304	The time it takes a proxy cache server to reply a "304: not modified" message	50ms	[46,47]
Reply ws	The time it takes a Web server to reply an object in memory to the requesting party	150ms	[46]
Routing sw	The time it takes a L5 switch to make a routing decision	50ms	♦
RTTpc_ws	The round trip time between a proxy cache server and a Web server	300ms	[46]
RTTsw_pc	The round trip time between a L5 switch and a proxy cache server	10~400ms	♣♦
RTTsw_ws	The round trip time between a L5 switch and a Web server	300ms	♦
RTTwc_pc	The round trip time between a Web client and a proxy cache server	0~20ms	[46]
RTTwc_sw	The round trip time between a Web client and a L5 switch	0~20ms	♦
Search pc	The time it takes a proxy cache server to search an object from its cache	250ms	[46]
SearchDigest pc	The time it takes a Cache Digest proxy cache server to search an object from cache digests	50ms	[9,46]
TCP_Splicing sw	The time it takes a L5 switch's port controller to translate TCP sequence number	<129μsec	[25]

Table 4.1 Simulation parameters (Time values)

♣ Varying according to distance.
♦ Assumed value.

Parameter	Meaning	Nominall Value	Source
Prob pc_GIMS_fresh	The probability of object not modified (304) for a GIMS request to a proxy cache server	50%	[46,47]
Prob ws_GIMS_fresh	The probability of object not modified (304) for a GIMS request to a Web server	60%	[46]
Prob ws_TimeOut	The probability of a request not gaining access to a Web server because of timeout	5%	[46]

Table 4.2 Simulation parameters (Probability values)

We assume that the request processing time at a proxy cache server, which includes the time to search for a requested object in a cache and the disk access time for moving the object from disk to memory, is proportional to the number of concurrent requests. This assumption is supported by the data collected by Rousskov [47] as shown in Figure 4.6.

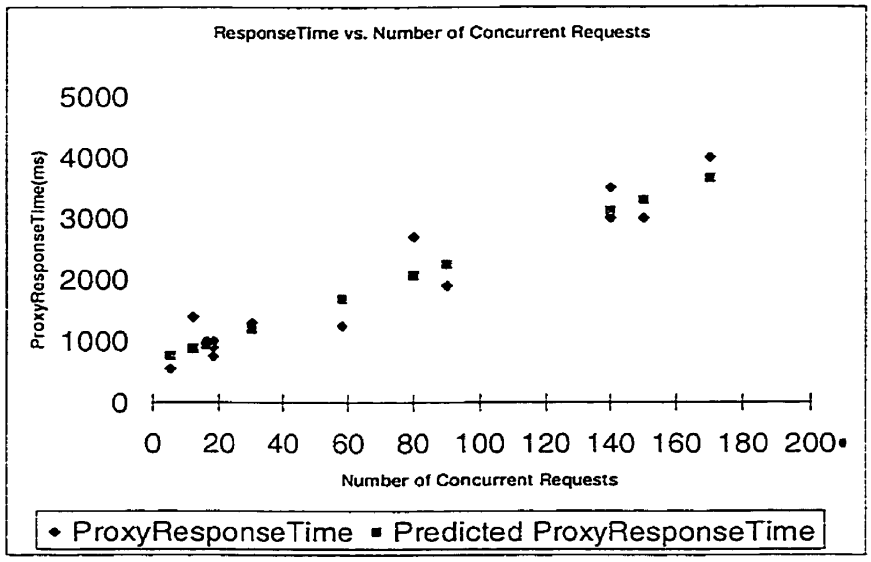


Figure 4.6 Proxy response time vs. number of concurrent requests

The predicted response time plot in Figure 4.6 is obtained through linear regression analysis

using the least squares method to fit a line through a set of observations.

4.1.5 Simulation Software Implementation

The simulator used in this thesis conducts discrete event driven simulation. It is developed using the Java programming language. The simulation software consists of the following six major components:

- Client Cluster: responsible for simulating a cluster of clients. It generates HTTP request traffic using the request logs from proxy trace files.
- Basic L5 and LB-L5 Switch: responsible for simulating the basic L5 switch and LB-L5 switch. The basic L5 switch redirects a cacheable request to its associated cache server and a non-cacheable request to a Web server. The LB-L5 switch supports extended ICP messages and communicates with cooperating cache servers. It redirects a cacheable request to one of a set of distributed cache servers based on the cache content and access-frequency information, server workload, and network link delays.
- ICP/CacheDigest/L5/LB-L5 Proxy Servers: responsible for simulating a proxy cache server. They use the Least Recently Used (LRU) replacement algorithm to maintain their caches. The basic L5 proxy server does not support cache cooperation. Therefore, it only performs the LRU cache management function. Other proxy cache servers perform additional functions. The ICP proxy server uses the ICP protocol to support query-based cache cooperation. The Cache

Digest uses a Bloom Filter to represent cache content and supports directory-based cache cooperation. The LB-L5 proxy server uses a weighted Bloom Filter to represent cache content. It uses extended ICP messages to communicate with LB-L5 switches to publish workload, cache content and access-frequency information.

- Web Server: responsible for simulating a Web server. It accepts HTTP requests and then sends back HTTP responses and requested objects.
- Network Link: responsible for simulating a network link connecting proxy servers, L5 switches, Web server and client clusters. It passes messages from one end to the other with a specified link delay.
- Event Manager: responsible for simulation event queuing and dispatching. All simulation events are handled by the event manager.

The detailed software structure and class description of the simulator are given in Appendix D.

4.2 Simulation Results

In this section, we describe and analyze raw-trace data and controlled-parameter simulation experiments. In the raw-trace data simulations, proxy traces are used to generate HTTP requests to drive the simulation. In the controlled-parameter simulations, proxy traces are condensed or expanded using different factors (the interval between requests is shortened or enlarged proportionally) to simulate different HTTP request intensities. The network link delay (including propagation delay, packet transmission delay and network access delay) and the number of cooperating proxy cache servers are other parameters used in both raw-trace data

and controlled-parameter simulations.

4.2.1 Raw-trace Data Simulations

In the raw-trace data simulations, we use the proxy traces from proxy servers on the NLANR network. The proxy traces used in the simulations are described in Table 4.3.

ProxyName	Location	NumOfRequests	Date
BO1	Boulder	146211	Sept. 16, 2000
UC	Urbana-Champaign	374093	Sept. 16, 2000

Table 4.3 Proxy traces used in raw-trace data simulations

The two proxy servers are at the root level of the NLANR network. They accept HTTP requests from proxies at lower levels. Approximately 90 proxies use BO1 as their parent cache and 60 use UC. We choose HTTP requests from network domains by the client IP addresses logged in the trace files, so that we can simulate the cache cooperation among different network domains.

4.2.1.1 HTTP request intensity and response time

The number of HTTP requests received by a proxy determines the workload of the proxy. As we shall see, the response time of a proxy cache server follows the HTTP request intensity. Meanwhile, if a scheme adapts well to high HTTP request intensities, it should have a flatter response time curve.

Figure 4.7 plots the HTTP request intensity in an experiment, where four proxy cache servers cooperate in a 24-hour duration (The results after simulation warm-up time, 0:00am - 6:00am,

are adopted). The request intensity ranges from 127 to 1043 messages per minute. The average intensity is 465 messages per minute.

The average response times under different link delays are shown in Figures 4.8 – 4.11. The simulation results show that LB-L5 outperforms the other three schemes, and has a better adaptability to high HTTP request intensities.

Under a small link delay (5 milliseconds), as shown in Figure 4.8, LB-L5 outperforms ICP by 31%, Cache Digest by 23%, and basic L5 by 13% on average. However, under very large link delays, LB-L5's performance is similar to that of the basic L5 scheme. LB-L5 avoids redirecting requests to remote cache servers when the response time improvement is less than the cost. As shown in Figure 4.11, when the link delay is 200 milliseconds, LB-L5 outperforms ICP by 41%, Cache Digest by 17%, and basic L5 by only 2% on average.

The results also show that LB-L5 has a better adaptability to high HTTP request intensity, especially for small link delays. As shown in Figure 4.8, under a peak request intensity (time of day =19:50) LB-L5 outperforms ICP and Cache Digest by 53%, and basic L5 by 28%.

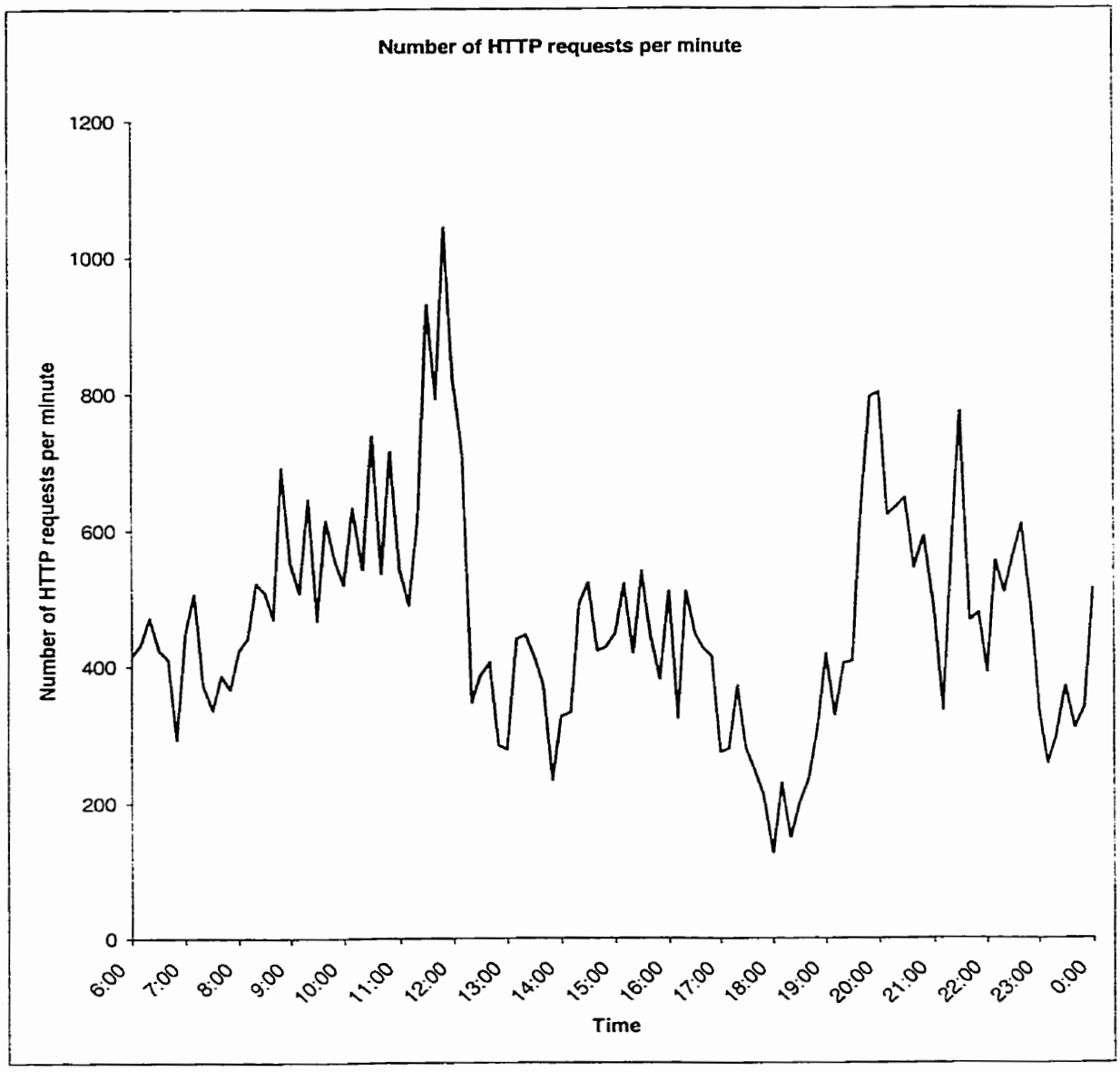


Figure 4.7 HTTP requests intensity from 6:00 to 0:00

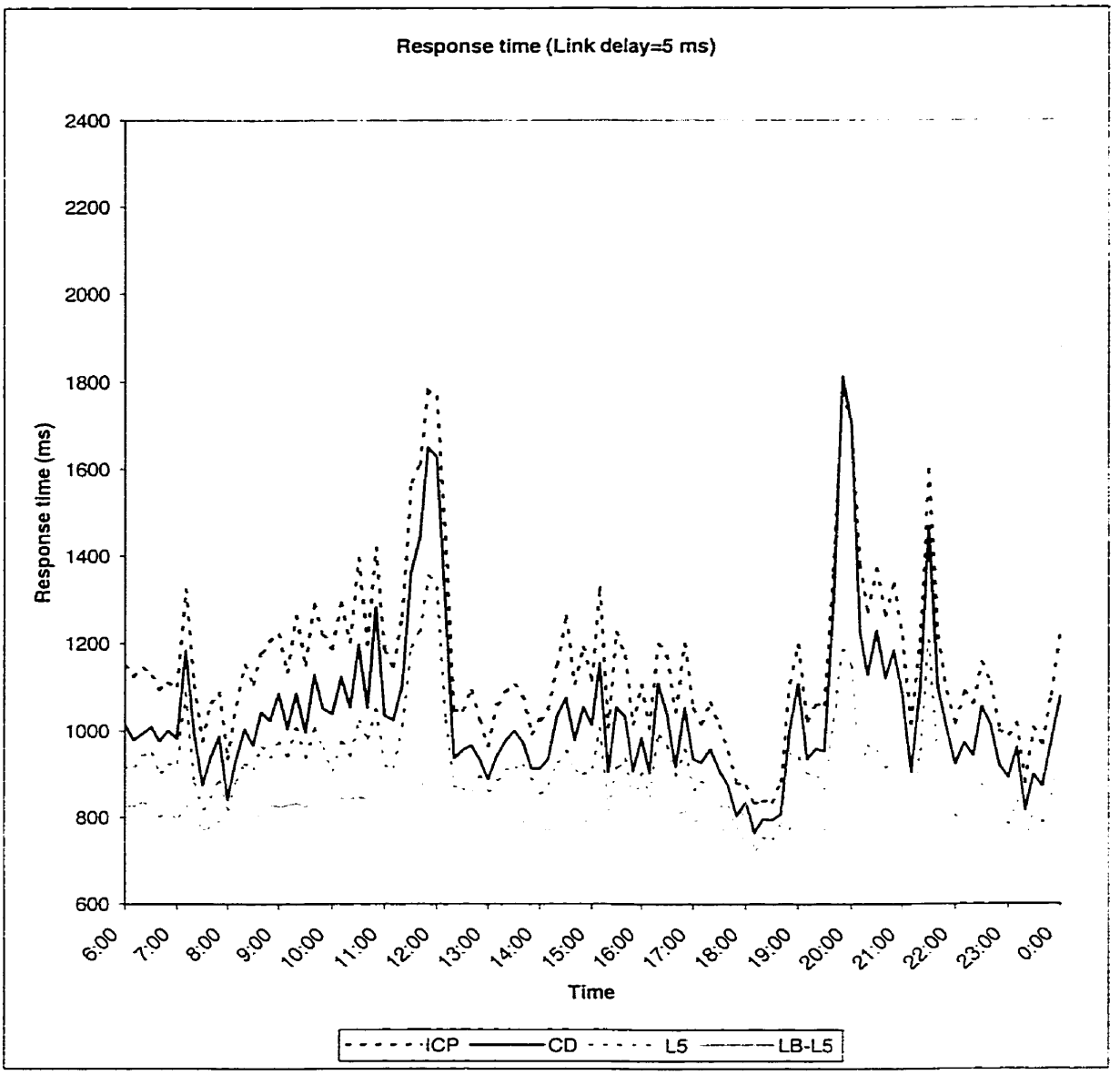


Figure 4.8 Average response time at link delay =5 ms

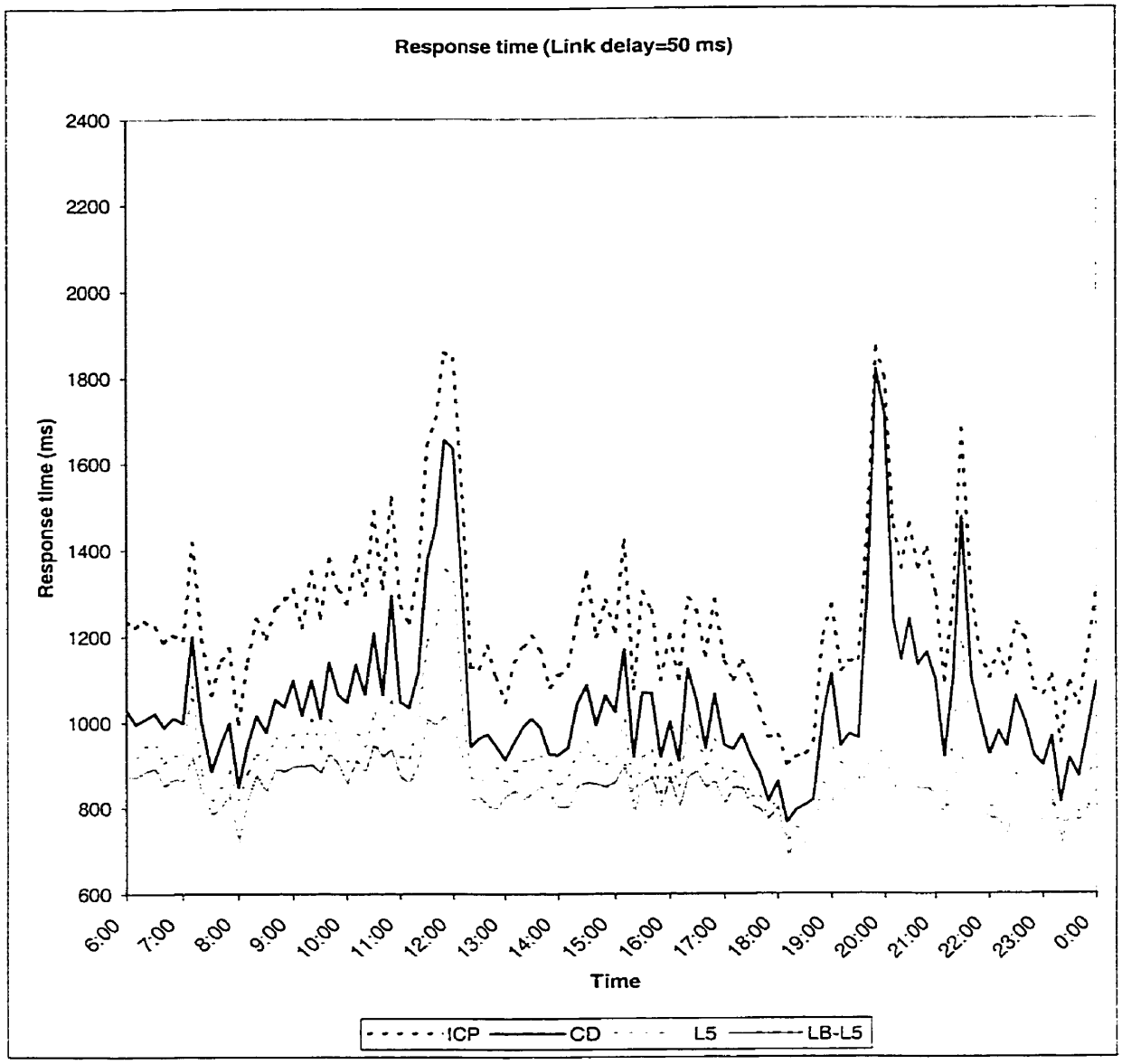


Figure 4.9 Average response time at link delay =50 ms

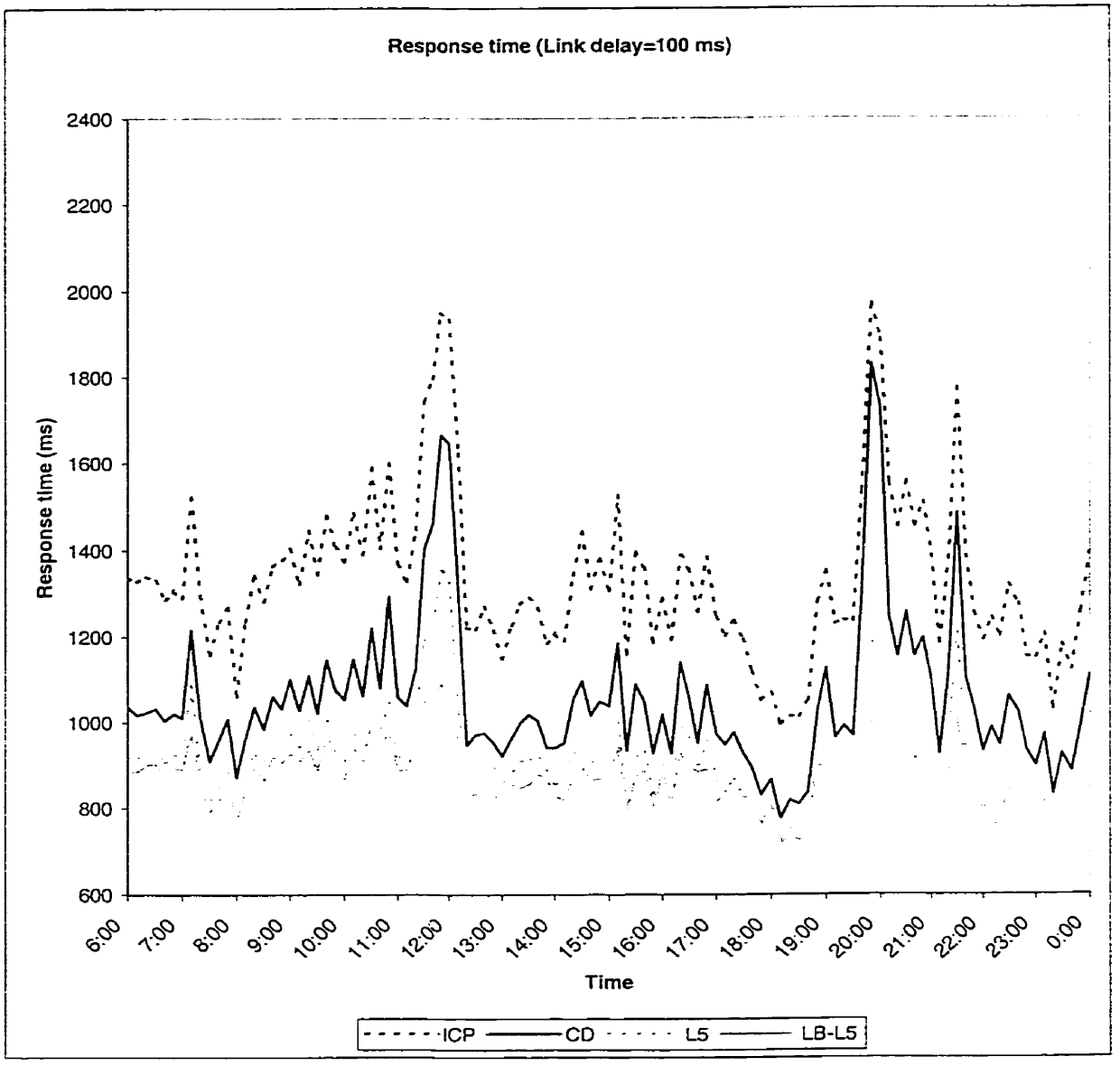


Figure 4.10 Average response time at link delay =100 ms

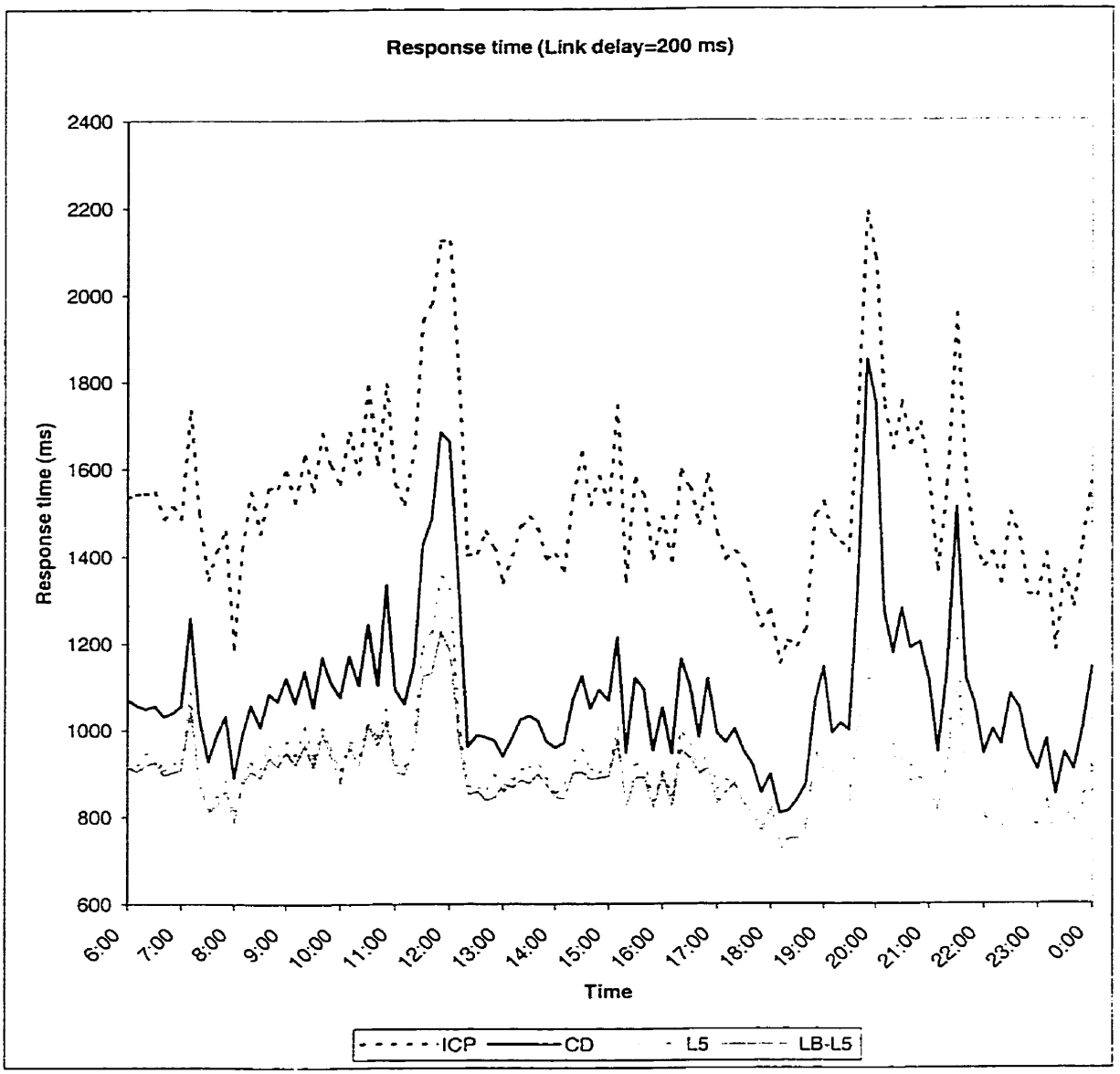


Figure 4.11 Average response time at link delay =200 ms

4.2.1.2 Hit rate

The cache hit rates of the four schemes under a link delay of 50 milliseconds (about the delay of a 100 kilometer network link) are shown in Figure 4.12. The results show that ICP achieves

the highest hit rate. The average hit rates for ICP, Cache Digest, basic L5 and LB-L5 are 22.79%, 22.20%, 17.16% and 21.07%, respectively.

The results show that ICP has the highest hit rate, since it is a query-based protocol. When an ICP proxy server receives a request and cannot find the requested object in its own cache, it asks all cooperating cache servers for that object. Therefore, if any one of a set of cooperating servers has a cached copy of the requested object then the request will result in a cache hit. In the directory-based Cache Digest, the directory update delay prevents a proxy server from having the most up-to-date information about the cache content on cooperating servers, and thus decreases the hit rate. The basic L5 scheme does not support cache sharing, and, therefore, its hit rate is the lowest of all the schemes. Like Cache Digest, LB-L5 is a directory-based approach, so the directory update delay is one of the reasons for its lower hit rate. The second reason for LB-L5's lower hit rate is that it sacrifices hit rate when it is cost-effective not to redirect a request to a remote server or a server that has a high workload.

We have a hypothesis that using a weighted Bloom Filter to represent cache content improves cache hit rate, because the weighted Bloom Filter has lower false prediction probability than the basic Bloom Filter used in ICP and Cache Digest. However, the hit rate improvement is offset because LB-L5 sacrifices hit rate to balance server workload and to avoid remote cache hits. The combined effect of these factors on LB-L5's hit rate needs further study.

Although simulation results show that LB-L5 has a lower hit rate than ICP and Cache Digest, it has a better response time. This is because LB-L5 takes into consideration other factors

affecting the performance of Web caching systems, such as network link delay, HTTP request intensity and proxy server workload balancing. The effects of these factors are further investigated using controlled-parameter simulations in the next section.

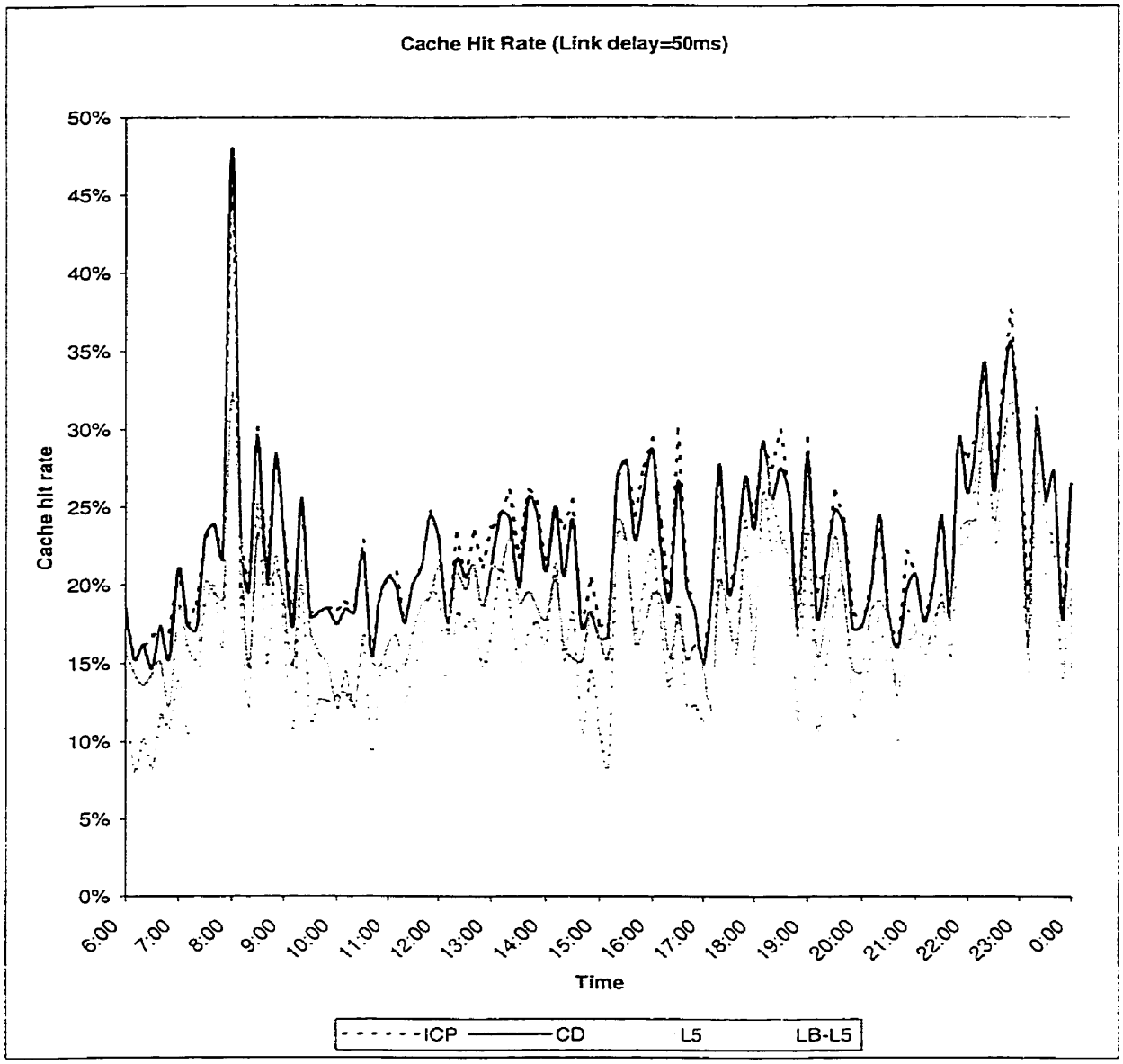


Figure 4.12 Hit rate comparison of ICP, CD, basic L5, and LB-L5

4.2.2 Controlled-Parameter Simulations

In this section, we study the performance of the proposed LB-L5 Web caching scheme in a controlled-parameter environment. LB-L5 is again compared with ICP, Cache Digest, and basic Layer 5 transparent Web caching.

The parameters used in the experiments are network link delay, HTTP request intensity, and the number of cooperating cache servers. These parameters are set as follows:

- Network link delay: in the experiments, the network link delays are varied from 5 to 200 milliseconds, which represent a wide range of link distance and/or network congestion levels.
- HTTP request intensity: the different HTTP request intensities are simulated by condensing or expanding proxy traces with a controlled factor (shortening or enlarging the interval between requests proportionally), and then using the modified traces to generate HTTP requests. In the experiments, the HTTP request intensities are set from 50% to 250%. The intensity range is chosen according to the previous raw-trace data simulation results, where the peak request intensity (1043 requests per minute) is about 224% of the average request intensity (465 requests per minute).
- Number of cache servers: we vary the number of cooperating cache servers from a small number (2) to a large number (12).

In the simulation experiments, results are sampled every minute in 30-minute durations after warm-up time. Simulations are run large enough times to obtain 90% confidence level with 10% confidence intervals. Plots in this section do not show the confidence intervals because they are too small.

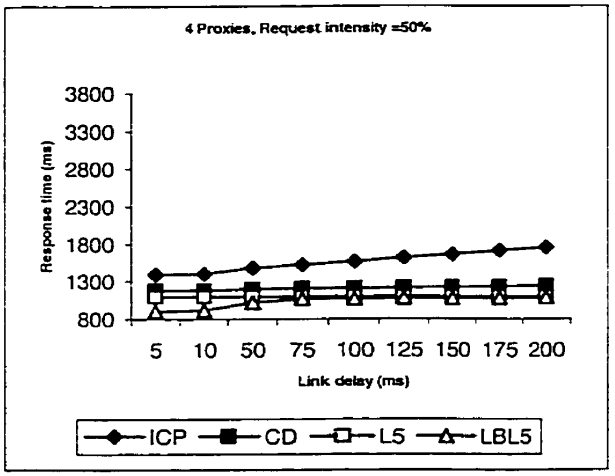
4.2.2.1 Effect of network link delay

Figure 4.13 plots the response time versus network link delay. The experiments are conducted with different HTTP request intensities and different numbers of cooperating proxy cache servers. We first note that LB-L5 outperforms the other schemes for all values of link delay and HTTP request intensities.

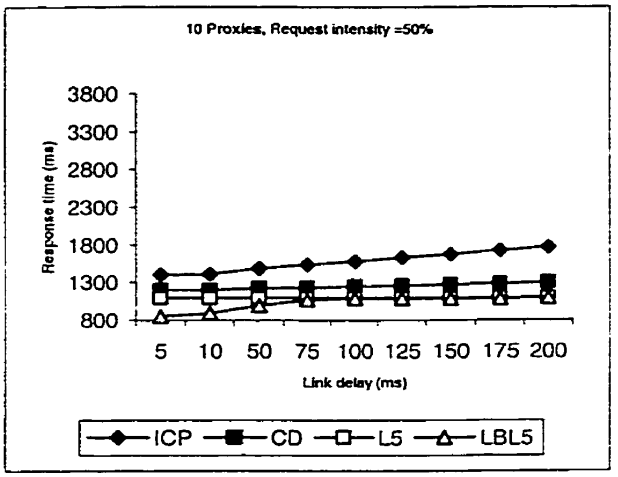
Except for the basic L5 scheme, the response times of the Web caching schemes increase as the network link delay increases. However, the extent to which the times increase is different in each scheme. The response time of the basic L5 scheme is not affected by link delay since it does not support cache server cooperation.

ICP is highly affected by the network link delay since it is a query-based scheme. As described in Chapter 2, if an ICP proxy cache server cannot find a requested object in its own cache, it queries all other cooperating cache servers to find a cached copy of the object. As the network link delay increases, the inter-proxy query/response time increases. As shown in Figure 4.13 (e.2), the response time of ICP increases up to 370 milliseconds as the link delay increases from 5 to 200 milliseconds.

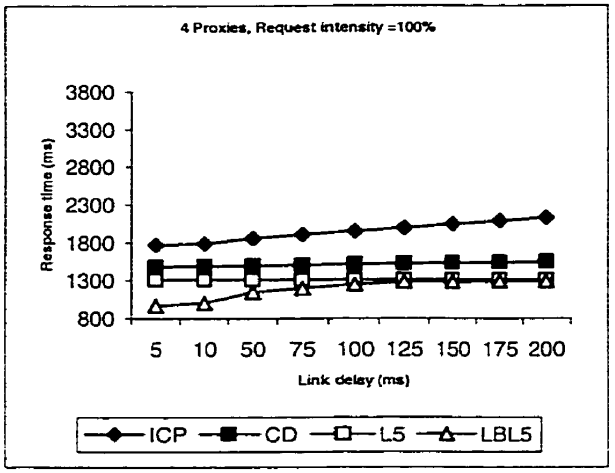
In Cache Digest, when a proxy cache server cannot find the requested object from its own cache, it searches the digests of all other cooperating servers, and if it finds the object then it fetches the object from other cache server. The Cache Digest scheme does not involve inter-proxy query/response. Fetching an object from a remote server, however, makes the HTTP request response time still susceptible to network link delay. As shown in Figure 4.13 (e.2), the response time of Cache Digest increases up to 84 milliseconds as the link delay increases from 5 to 200 milliseconds.



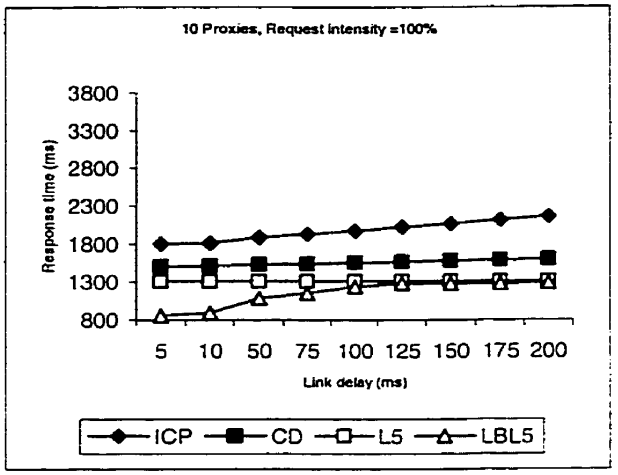
(a.1)



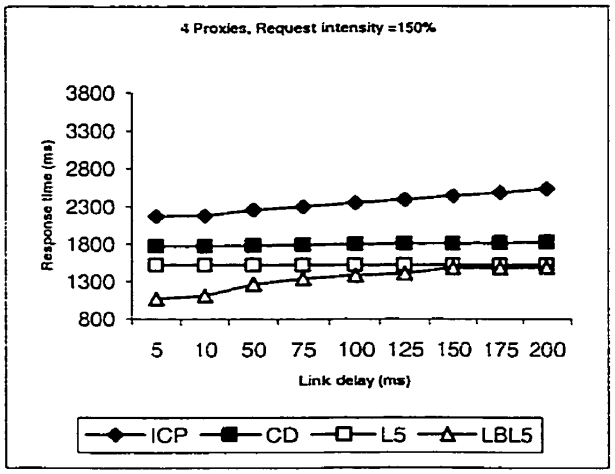
(a.2)



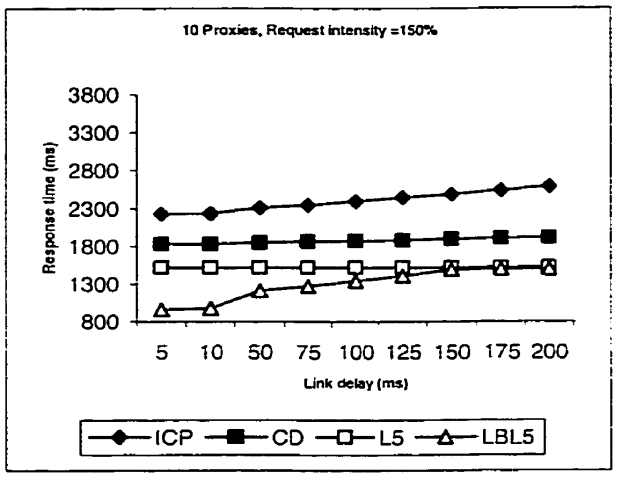
(b.1)



(b.2)



(c.1)



(c.2)

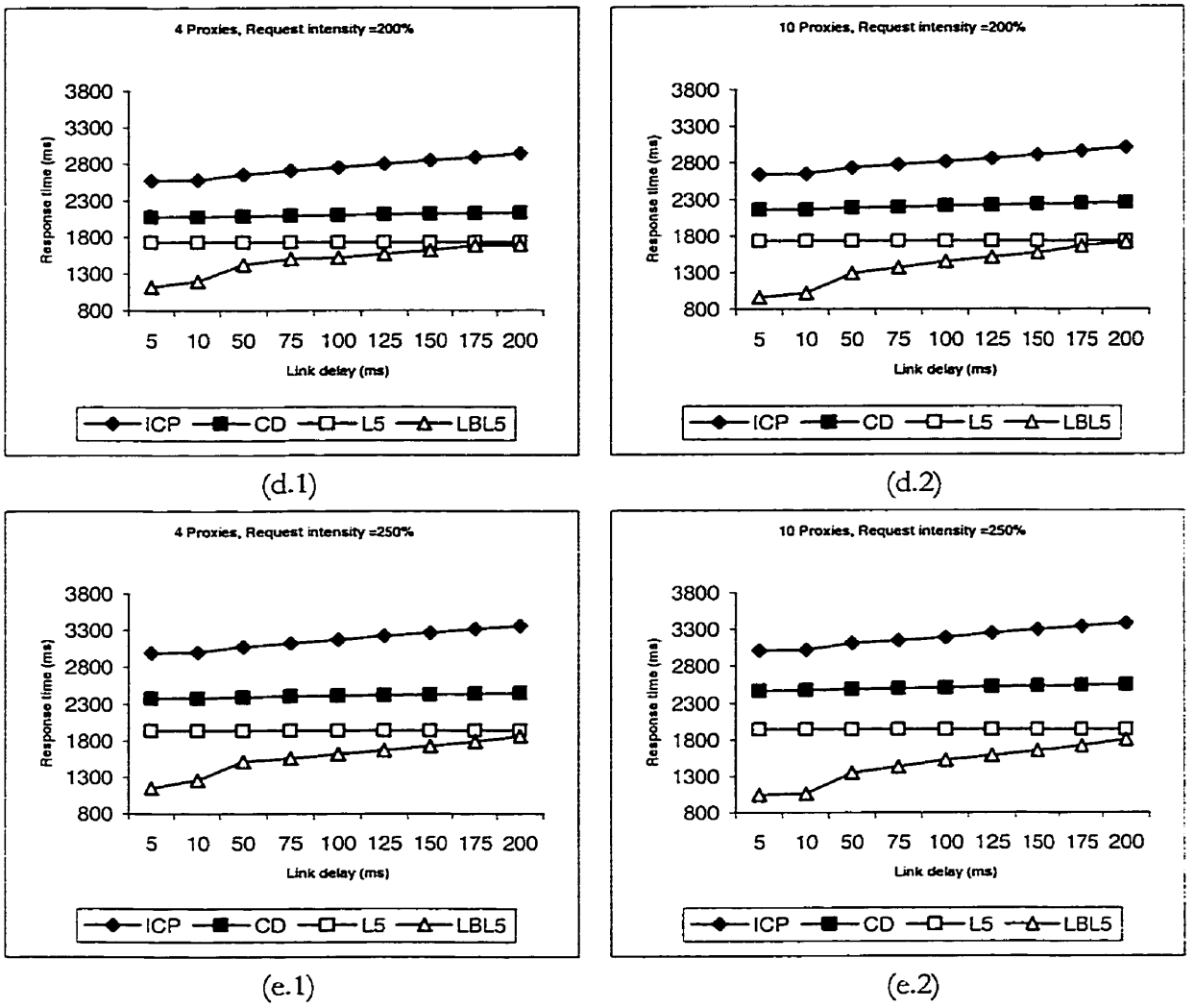


Figure 4.13 Effect of link delay on response time

The basic L5 scheme is not affected by network link delays because it does not support distributed cache cooperation. On the other hand, the response time of LB-L5 is greatly affected by link delay. When the link delay is small, LB-L5's routing decision is based mainly on the cache content and workload information of cache servers. HTTP requests can be redirected to any one of the cooperating cache servers to increase cache hit rate or to balance

server workload. As the link delay increases, the response time improvement achieved by redirecting requests to remote cache servers decreases. Therefore, when the link delay is very large, LB-L5's performance closely resembles the basic L5 scheme. As shown in Figure 4.13 (e.2), the response time of LB-L5 increases up to 750 milliseconds as the link delay increases from 5 to 200 milliseconds. However, LB-L5 always outperforms the other schemes.

Simulation results show that performance advantage of LB-L5 over the other schemes is even more apparent when the request intensity is high. This is because, as the request intensity increases, the imbalance of the server workloads increases, which means that the performance improvement achieved by LB-L5's server workload balancing increases. As shown in Figure 4.13 (a.2), when the request intensity is 50%, the response time of LB-L5 is lower than that of ICP, Cache Digest, and the basic L5 schemes by up to 560, 353, and 250 milliseconds, respectively. Figure 4.13 (e.2) shows that under a request intensity of 250%, the response time of LB-L5 is lower than that of ICP, Cache Digest, and the basic L5 scheme, up to 1966, 1422, and 895 milliseconds, respectively. The effect of request intensity on the response time under different link delays is further studied in the next section.

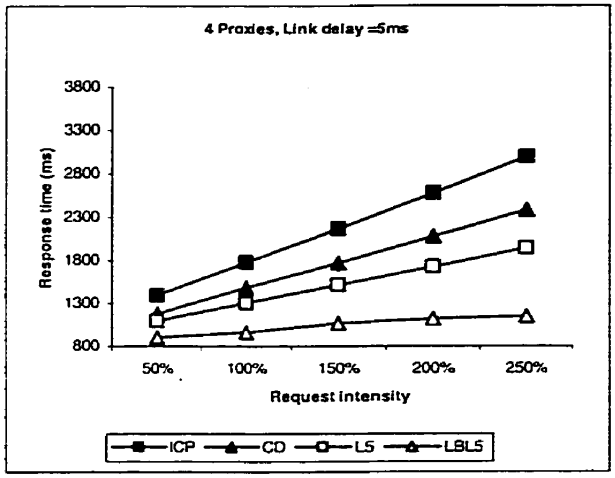
4.2.2.2 Effect of request intensity

Figure 4.14 plots the response time versus HTTP request intensity under different link delays. The experiments are conducted for different numbers of cooperating cache servers. We first note that regardless of the network link delay, the HTTP request response time for all Web caching schemes increases as we increase the request intensity. Again, LB-L5 outperforms the other three schemes under all combinations of request intensity and link delay, as shown in

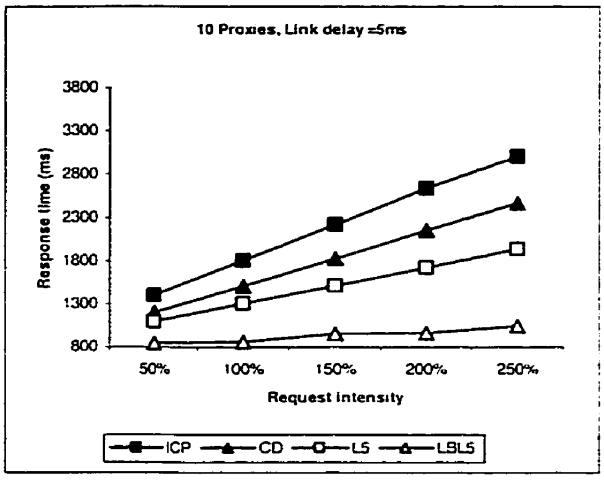
Figure 4.14.

Simulation results also show that LB-L5 adapts better to high request intensities than the other schemes. As shown in Figure 4.14 (a.2), under a small link delay (5 milliseconds), LB-L5's response time increases by only 23% as the request intensity increase from 50% to 250%. The corresponding response time increases for ICP, Cache Digest and basic L5 are 114%, 106%, and 77%, respectively.

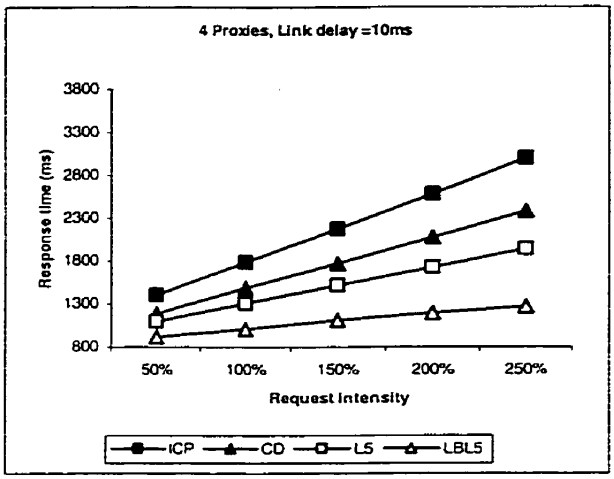
These results again reflect the performance improvement achieved by LB-L5's server workload balancing. As the network link delay increases, the cost of redirecting requests to remote servers increases. Therefore, when the link delay is very large, requests are less likely to be redirected to remote servers and the response time of LB-L5 is very close to that of the basic L5 scheme. This case is shown in Figure 4.14 (f.1, f.2).



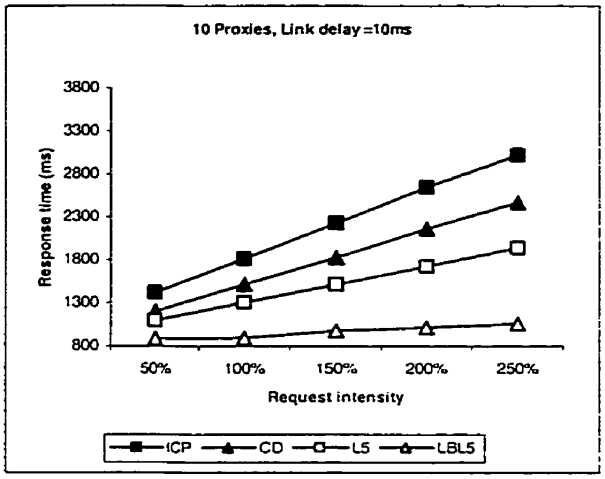
(a.1)



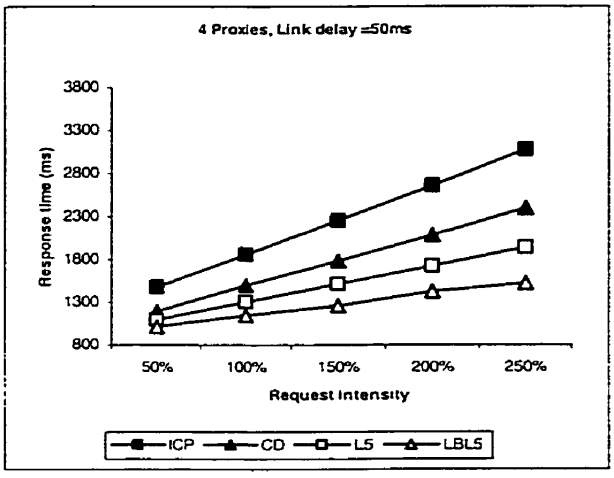
(a.2)



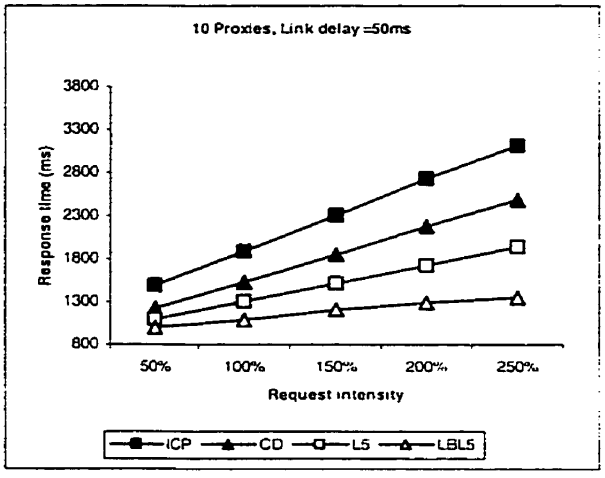
(b.1)



(b.2)



(c.1)



(c.2)

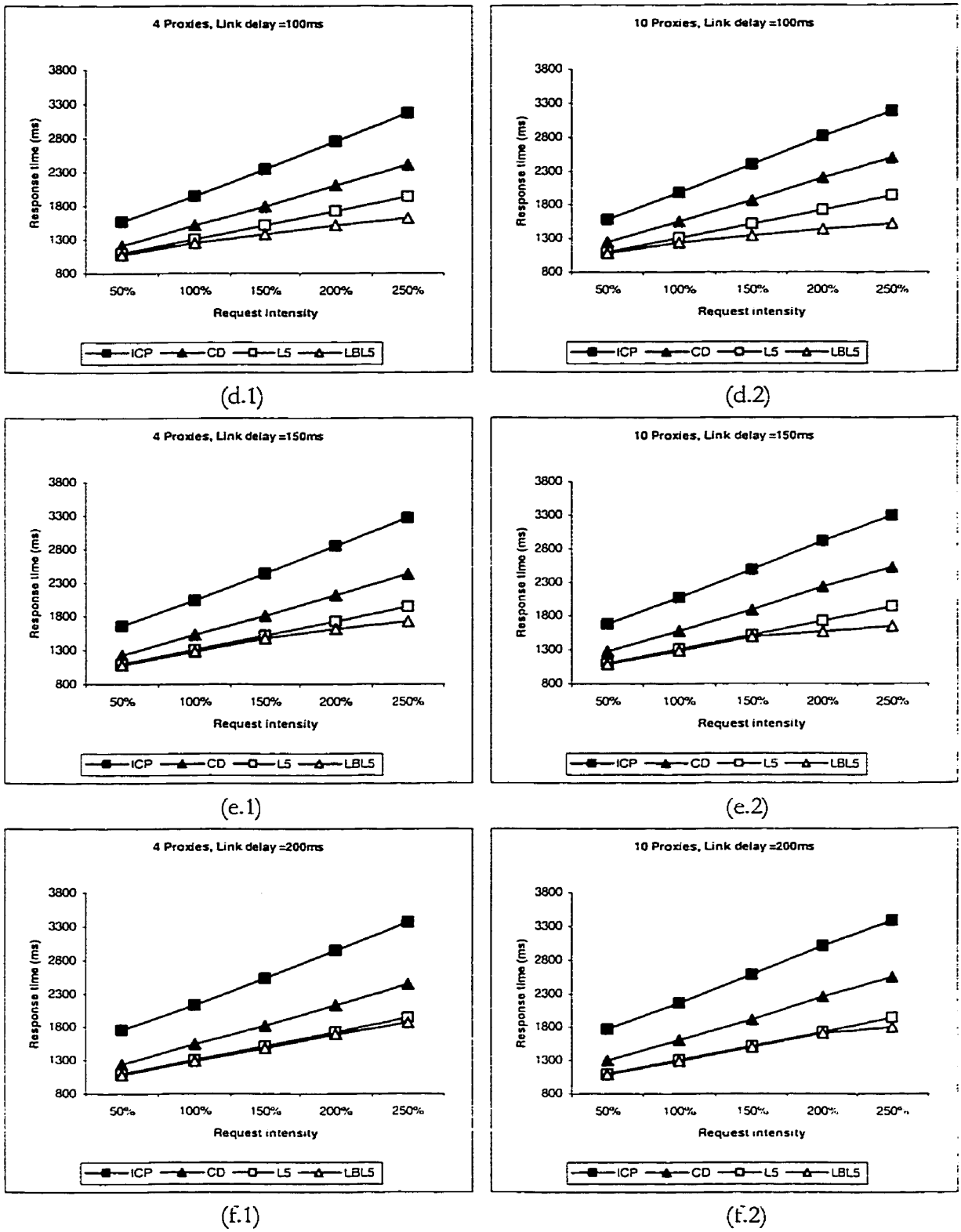


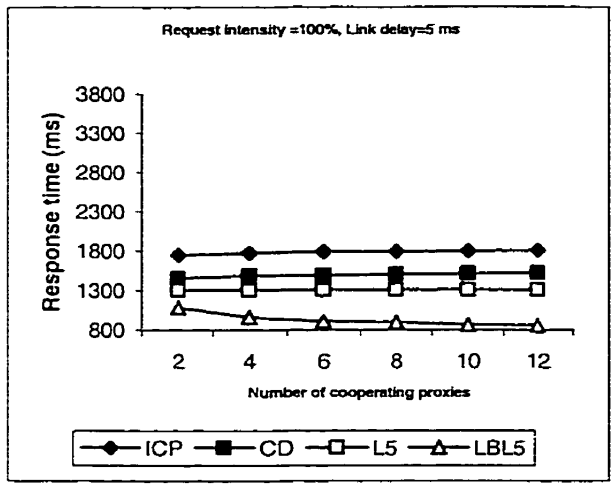
Figure 4.14 Effect of HTTP request intensity on response time

4.2.2.3 Effect of number of cooperating cache servers

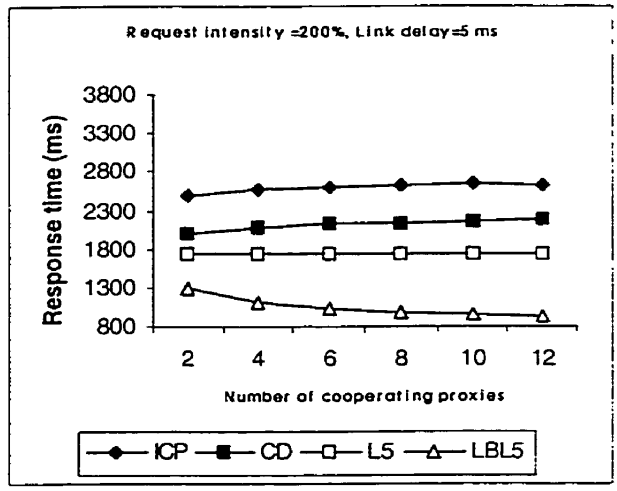
Figure 4.15 plots the response times of the various schemes versus the number of cooperating cache servers. These experiments are conducted under different request intensities and link delays.

The basic L5 scheme does not support cache server cooperation. Therefore, it is not affected by the number of cache servers and link delays. Increasing the number of cooperating cache servers does not guarantee performance improvements in ICP and Cache Digest. This is because ICP and Cache Digest try to achieve the best hit rate but do not consider cache server workload and network link delay. Simulation results show that the response times of ICP and Cache Digest increase as more requests are redirected to remote or busy cache servers to achieve higher hit rates.

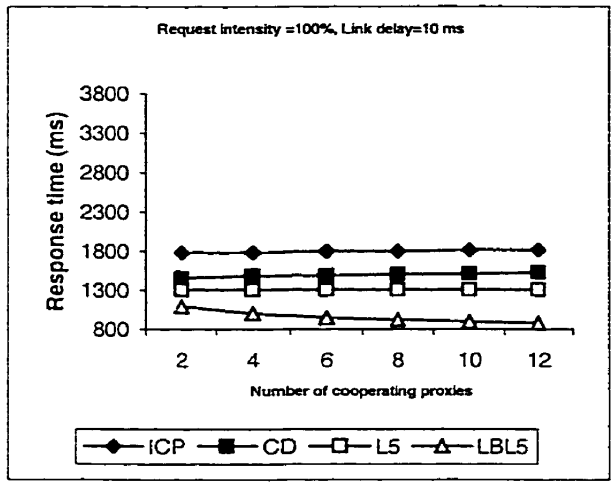
On the other hand, in LB-L5, as the number of cooperating cache servers increases, L5 switches are better able to redirect requests to balance server workloads. Thus the performance gain increases. When the link delay is small, the performance improvement is significant. As shown in Figure 4.15 (a.1, a.2), the response time is reduced by 29% as the number of cooperating cache servers increases from 2 to 12. However, under very large link delays, requests are not likely to be redirected to remote servers. LB-L5 cannot gain the benefits of server workload balancing and cache sharing in this case, as shown in Figure 4.15 (f.1, f.2).



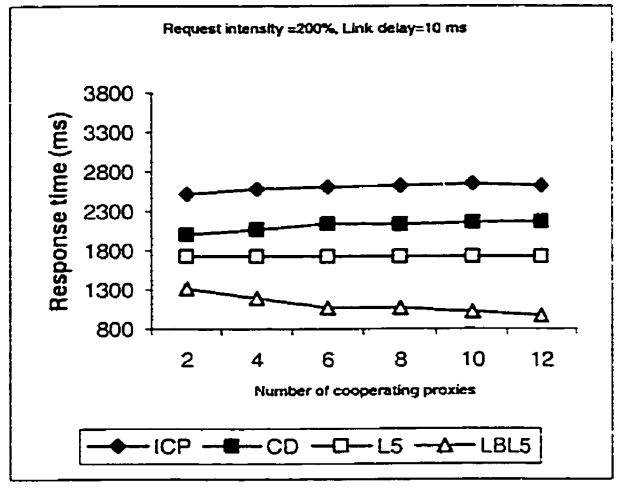
(a.1)



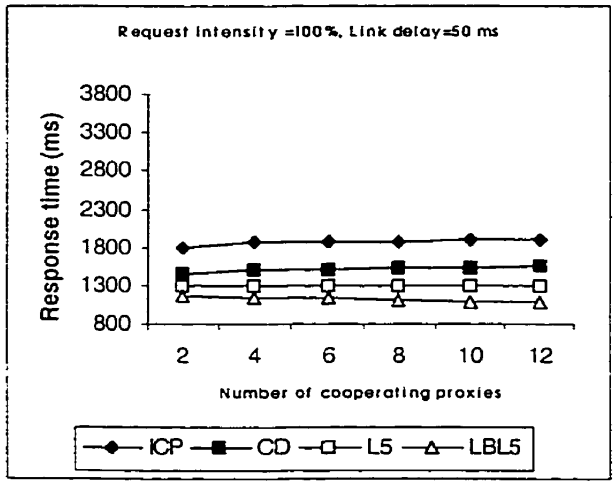
(a.2)



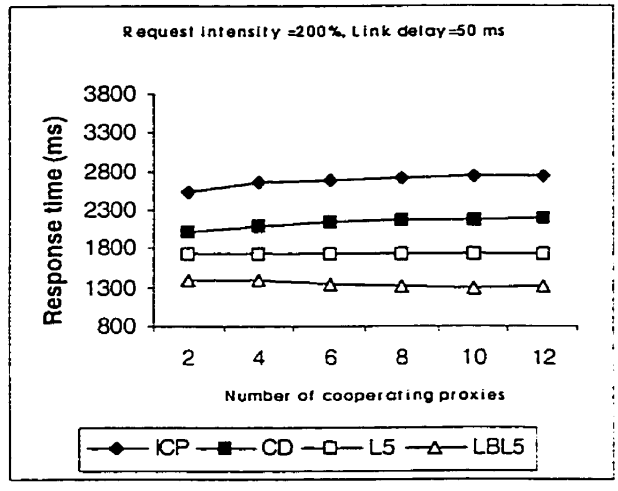
(b.1)



(b.2)



(c.1)



(c.2)

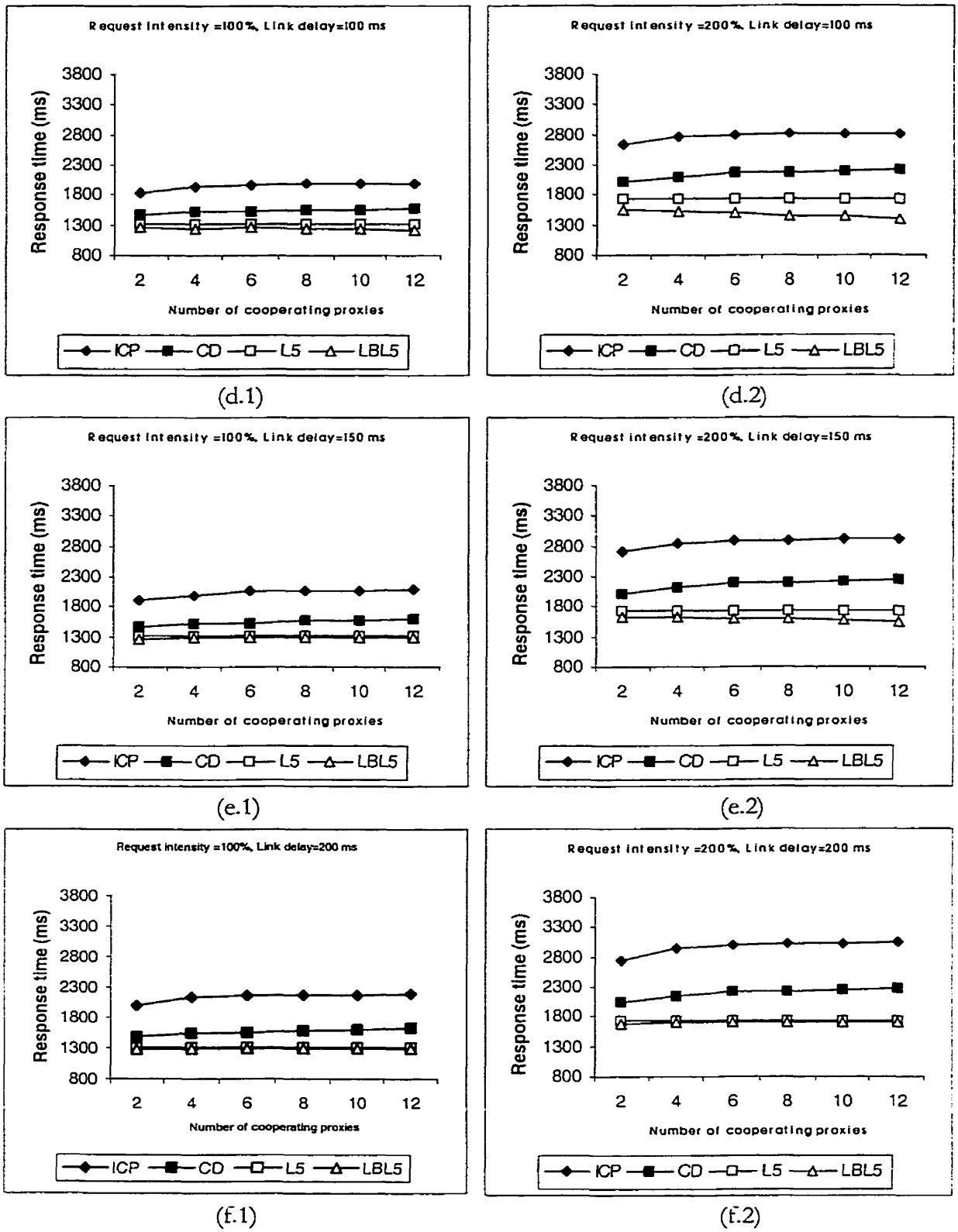


Figure 4.15 Effect of the number of cooperating proxies on response time

4.2.2.4 Effect of request imbalance

The experiments in this section study the server workload balancing capability of LB-L5. Four client clusters generating different intensities of requests are used in the simulations. Figure 4.16 (a-d) plots the number of messages processed per minute by each proxy cache server for each scheme. Figure 4.16 (e) plots the standard deviations of these numbers. These experiments are conducted under different network link delays.

As shown in Figure 4.16, server workload is not balanced in ICP, Cache Digest or the basic L5 scheme, since they do not have a server workload balancing capability. LB-L5, on the other hand, ensures balanced server workloads when the network link delay is relatively small. As the network link delay increases, the cost of redirecting requests to remote servers increases, and the benefits of workload balancing decreases. LB-L5 avoids redirecting requests to remote servers when the link delay is very large.

Simulation results show that the server workload balancing capability of LB-L5 is apparent when the network link delay is not larger than 75 milliseconds. Therefore, by balancing server workload, LB-L5 can achieve performance improvement for distributed cache systems deployed on a local area network or a metropolitan area network.

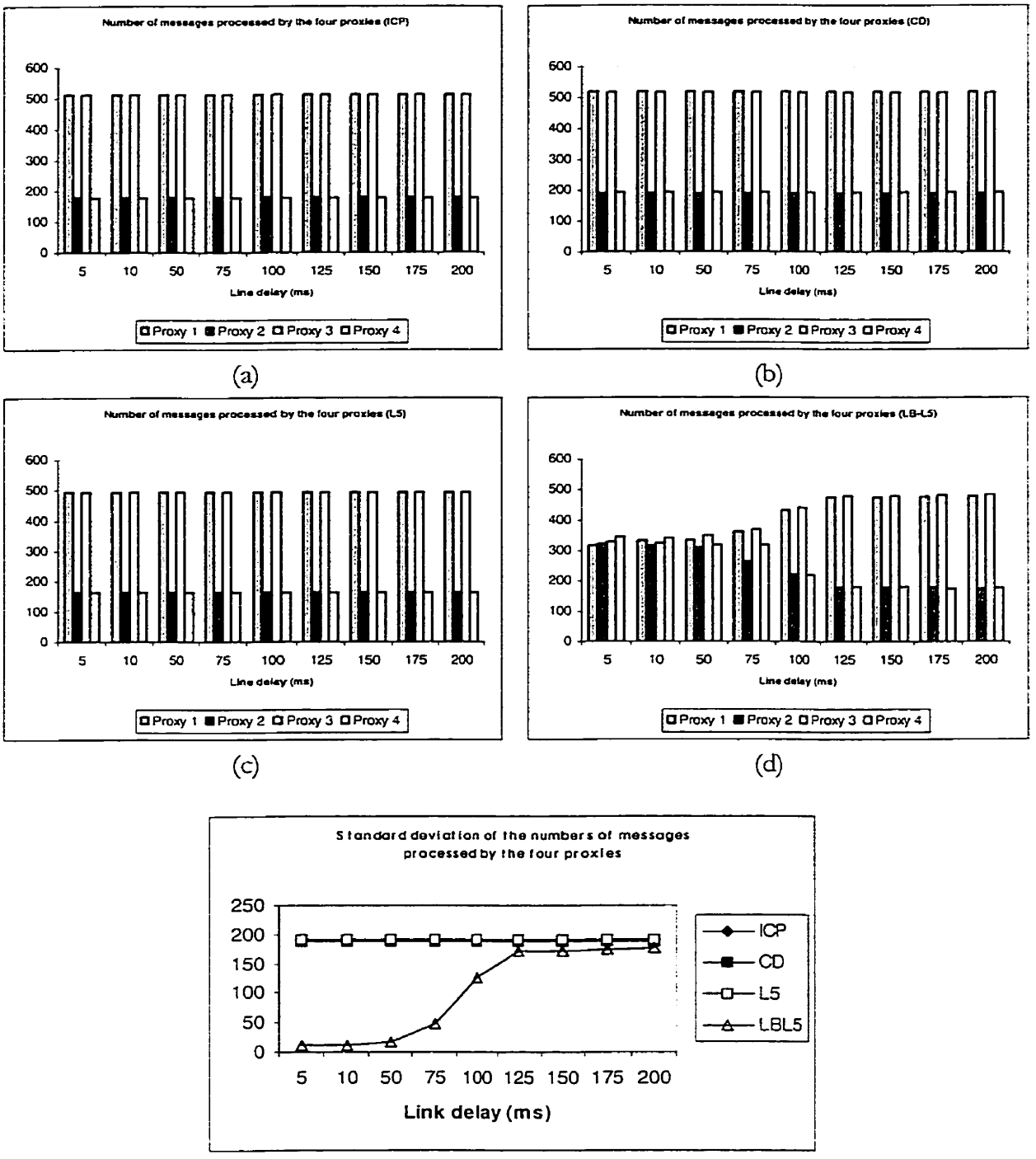


Figure 4.16 Workload balancing in LB-L5

4.3 Summary

In summary, this chapter evaluated the performance of the proposed LB-L5 Web caching scheme. The performance of LB-L5 was compared to that of ICP, Cache Digest, and the basic L5 Web caching scheme via simulation. In addition, the adopted network model, simulation experiment settings, and simulation software implementation were described. Two types of simulation experiments, raw-trace data and controlled-parameter, were conducted to investigate the effects of network link delay, HTTP request intensity, and the number of cooperating proxy servers on the performance of the Web caching schemes.

The simulation experiments showed that LB-L5 outperforms ICP, Cache Digest, and the basic L5 scheme, with respect to HTTP request response time under various network link delays. Regardless of the HTTP request intensity, the response time of ICP, Cache Digest, and LB-L5, increases as the network link delay increases, whereas the basic L5 scheme is not affected by link delay. Under large link delays, the response time of LB-L5 is close to that of the basic L5 scheme, since LB-L5 avoids redirecting requests to remote servers when the cost is too high.

The results obtained from the experiments conducted on the effect of HTTP request intensity showed that LB-L5 adapts better to high request intensities than the other three schemes. Under high request intensity, LB-L5's server workload balancing produces significant performance improvement.

Likewise, LB-L5 demonstrated a better capability of supporting cache cooperation than the

other three schemes. As the number of cooperating cache servers increases, LB-L5 has more opportunities to redirect requests, which helps to balance server workloads and increase cache sharing. These two factors mean that the performance improves.

It is noteworthy that the experiments conducted to investigate LB-L5's server workload balancing showed that the workloads of cooperating cache servers are well balanced when the link delay is not very large. However, the workload is not well balanced when the link delay is very large, since the cost of redirecting a request becomes too high. The obtained results also reflected that LB-L5 does not over-emphasize any one of the factors that determine the performance of a Web caching system.

During the simulation experiments it was found that the cache hit rate of LB-L5 is not as high as that of ICP and Cache Digest, although it is better than that of the basic L5 scheme. The main reason is that LB-L5 sacrifices hit rate when the cost of redirecting a request to a remote or high-workload server is too high. This again showed that LB-L5 tries to give balanced consideration to the multiple factors affecting performance. Thus, we can conclude that LB-L5 is more adaptable than the other schemes.

Chapter 5

Conclusion

5.1 Concluding Remarks

Web caching is considered one of the most effective approaches to improving the performance and scalability of the Web. The emerging Layer 5 switching-based transparent Web caching not only makes the deployment and configuration of the caching system easier, but also improves its performance by redirecting non-cacheable HTTP requests to bypass cache servers.

The main thesis of this research is that transparent Web caching can be combined with distributed cache cooperation to provide improved cache performance. We proposed the Load Balancing Layer 5 switching-based (LB-L5) Web caching scheme, which uses transparent Web caching techniques to support distributed Web caching.

LB-L5 uses a weighted Bloom Filter to represent cache content in order to support directory-

based cache cooperation. The weighted Bloom Filter has two salient properties. First, the weighted Bloom Filter can be used to represent cache content and carry cache access-frequency information at the same time. Second, the weighted Bloom Filter gives lower false prediction probability than the basic Bloom Filter used in Cache Digest and Summary Cache. By using the weighted Bloom Filter to represent cache content, LB-L5 can support directory-based HTTP request routing in order to avoid the query/response delay existing in query-based schemes such as ICP. Meanwhile, the access-frequency information carried in a weighted Bloom Filter enables LB-L5 to support access-frequency-aware cache cooperation, which helps to reduce the duplication of caching.

LB-L5 uses cache content and access-frequency information, cache server workload information and network link delay information to route HTTP requests to one of a set of cooperating cache servers. This routing policy makes LB-L5 suitable to support distributed Web caching since it avoids fetching objects from remote or busy cache servers whenever a less expensive copy can be found. LB-L5 achieves backward compatibility with existing Web caching methods by extending ICP, the most popular Web caching protocol, to facilitate communication between the cache servers and the switches.

A detailed simulation model was developed to study the performance of LB-L5. LB-L5 was compared with three existing Web caching schemes, namely ICP, Cache Digest, and basic Layer 5 transparent Web caching. The results show that LB-L5 outperforms these existing schemes in terms of HTTP request/response time and proxy server workload balancing. LB-

L5 is also shown to adapt better to high HTTP request intensity.

5.2 Future Work

While LB-L5 Web caching has a number of advantages that enable it to outperform existing schemes, several aspects of the research need further investigation. The hash functions used in LB-L5 to generate cache content representation are based on MD5, which is a one-way hash function that produces a 128-bit hash value, or message digest, of an arbitrary-length input message. MD5's collision-resistance property makes it suitable for building the weighted Bloom Filter, since we want different objects to have different representations. However, we believe it is possible to find a more computationally efficient hash function for the Web caching context. This is because, first, MD5 was designed to process input messages of much larger size (n 512-byte blocks) than a typical URL (50 bytes on average). Second, the avalanche effect, which is bringing the previous blocks hash values to the following blocks, in MD5 is used to strengthen MD5's one-way property and may be simplified in our case. In addition, a hash function that produces hash values reflecting the similarity of input messages (that is, URLs) may be used to more precisely represent cache content access characteristics.

In LB-L5, the cache servers periodically publish the cache content and access-frequency information to the switches. Using compression techniques and incremental updates is possible, since the content and access-frequency information on a particular cache server changes gradually.

We have a hypothesis that using a weighted Bloom Filter to represent cache content improves cache hit rate, because the weighted Bloom Filter has a lower false prediction probability than the basic Bloom Filter used in ICP and Cache Digest. However, the hit rate improvement is offset because LB-L5 sacrifices hit rate to balance server workload and to avoid remote cache hits. The combined effect of these factors on LB-L5's hit rate needs further study.

With some modifications, LB-L5 can be used in a Web server cluster scenario. The Web servers can either contain duplicated Web documents, or can be optimized for different document types. For example, some Web servers can be optimized for image documents, while other Web servers are optimized for text documents. This makes the Web server cluster easy to maintain, and possess better performance. A Layer 5 switch can be integrated into a Web server cluster to redirect incoming HTTP requests to one of the Web servers according to the workload and the content information of the Web servers. A weighted Bloom Filter can be used to represent a Web server's content and preference of the different sets of document requests. For example, the Web server optimized for image documents can also contain some text documents, but it has a better response time for image document requests, and thus prefers these requests.

Bibliography

[1] T. Berners-Lee, R. Cailliau, J. Groff, and B. Pollermann, "World-Wide Web: The Information Universe". In *Electronic Networking: Research, Applications, and Policy*, Spring 1992.

[2] CERN – European Laboratory for Particle Physics, "An Overview of the World-Wide Web". Available at: http://www.cern.ch/Public/ACHIEVEMENTS/WEB/

[3] T. Berners-Lee, R. Fielding, and H. Nielson, "Hypertext Transfer Protocol - HTTP/1.0". RFC 1945, May 1996.

[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1". RFC 2616, June 1999.

[5] M. Seltzer, "Issues and challenges facing the World Wide Web". Available at: http://www.eecs.harvard.edu/~margo

[6] A. Luotonen, and K. Altis, "World Wide Web proxies". In *Proceedings of the First International Conference on the World-Wide Web, WWW '94*, 1994.

[7] M. Abrams, C. Standridge, G. Abdulla, and S. Williams, "Caching Proxies: Limitations and Potentials". In *The Fourth International World Wide Web Conference*, Boston, Massachusetts, USA , December 1995.

[8] R. Tewari, M. Dahlin, H. Vin and J. Kay,"Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet"'. Technical Report TR98-04, Department of Computer Sciences, University of Texas at Austin, February 1998.

[9] R. Tewari, M. Dahlin, H. Vin and J. Kay, "Design Considerations for Distributed Caching on the Internet"'. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, Austin, Texas, May1999

[10] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. Levy, "On the scale and performance of cooperative web proxy caching". In *Proceedings of the 17th Symposium on Operating Systems Principles*, December 1999.

[11] P. Rodriguez, C. Spanner, and E. Biersack, "Web caching architectures: Hierarchical and distributed caching". In *Proceedings of the 4th International Web Caching Workshop*, April 1999.

[12] C. Jeffery, S. Das, and G. Bernal, "Proxy Sharing Proxy Servers". In *Proceedings of the IEEE ETA-COM Conference*, Portland, OR, May 1996.

[13] CERN – European Laboratory for Particle Physics, "CERN httpd Web Server". Available at: http://www.w3.org/Daemon.

[14] C. Bowman, P. Danzig, D. Hardy, U. Manber, and M. Schwartz, "The Harvest information discovery and access system". In *Proceedings of the Second International World Wide Web Conference*, October 1994.

[15] D. Wessels and K. Claffy, "Application of Internet Cache Protocol (ICP), version 2". RFC 2187, September 1997.

[16] D. Wessels and K. Claffy, "Internet Cache Protocol (ICP), version 2". RFC 2186, September 1997.

[17] D. Wessels and K. Claffy, "ICP and the Squid Web Cache". In *IEEE Journal on Selected Areas in Communication*, Vol 16, No.3, pp 345-357, April 1998.

[18] D. Wessels, "Squid and ICP: Past, Present, and Future". In *Proceedings of the Australian Unix Users Group*, Brisbane, Australia. August 1997.

[19] A. Rousskov and D. Wessels, "Cache digests". In *Proceedings of the Third International WWW Caching Workshop*, Manchester, England, June 1998.

[20] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol". In *Proceedings of ACM SIGCOMM*, September 1998.

[21] E. Johnson, "Increasing the Performance of Transparent Caching with Content-aware Cache Bypass", Arrowpoing communications. In *The Fourth International WWW Caching Workshop*, 1999. Available at: http://www.ircache.net/Cache/Workshop99/Papers/johnson-0.ps.gz

[22] B. Williams, "Transparent Web Caching Solutions", Director of Strategic Business Planning, Alteon Networks .White Paper. In *The Third International WWW Caching Workshop*, 1998. Available at: http://www-sor.inria.fr/mirrors/wcw98/33/cachpaper.html

[23] ArrowPoint Communications, "Content Smart™ Cache Switching", White paper. Available at: http://www.arrowpoint.com/solutions/white_papers/

[24] ArrowPoint Communications, "A Comparative Analysis of Web Switching Architecture", White paper. Available at: http://www.arrowpoint.com/solutions/white_papers/

[25] G. Apostolopoulos, V. Peris, P. Pradhan, and D. Saha, "L5: A self learning layer 5 switch". Technical Report RC21461, IBM, T.J. Watson Research Center, 1999.

[26] R. Malpani, J. Lorch, and D. Berger, "Making World Wide Web caching servers cooperate". In *Proceedings of the Fourth International World Wide Web Conference*, December 1995.

[27] National Laboratory for Applied Network Research (NLANR). Ircache project. Available at: http://ircache.nlanr.net

[28] D. Wessels, "Evolution of the NLANR cache hierarchy: Global configuration challenges". Available at: http://www.nlanr.net/Papers/Cache96/, November 1996.

[29] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd and V. Jacobson, "Adaptive Web caching: towards a new caching architecture". In *The Third International WWW Caching Workshop*, June 1998.

[30] J. Yang, W. Wang, R. Muntz, and J. Wang, "Access Driven Web Caching". UCLA CSD, Technical Report TR990007, 1999.

[31] D. Povey and J. Harrison, "A distributed Internet cache". In *Proceedings of the 20th Australian Computer Science Conference*, Sydney, Australia, February 1997.

[32] V. Valloppillil and K. Ross, "Cache Array Routing Protocol v1.0". Internet Draft, draft-vinod-carp-v1-03.txt , February 1998.

[33] S. Gadde, J. Chase, and M. Rabinovich, "A Taste of Crispy Squid". In *Workshop on Internet Server Performance (WISP'98)*, Madison, WI, June 1998.

[34] S. Gadde, M. Rabinovich, and J. Chase, "Reduce, Reuse, Recycle: An Approach to Building Large Internet Caches". In *The Sixth Workshop on Hot Topics in Operating Systems (HotOS-VI)*, pp 93-98, May 1997.

[35] Z. Wang, "Cachemesh: a Distributed Cache System for World Wide Web", *Web Cache Workshop*, 1997.

[36] Realis Project, "Relais: cooperative caches for the World Wide Web", 1998. Available at: http://www-sor.inria.fr/projects/relais/

[37] M. Rabinovich, J. Chase, and S. Gadde, "Not all hits are created equal: cooperative proxy caching over a wide-area network". In *Computer Networks And ISDN Systems*, 30, 22-23, pp. 2253-2259, November 1998.

[38] G. Barish and K. Obraczka, "World Wide Web Caching: Trends and Techniques". In *IEEE Communications Magazine, Internet Technology Series*, May 2000.

[39] C. Faloutsos and S. Christodoulakis, "Design of a Signature File Method that Accounts for Non-Uniform Occurrence and Query Frequencies". In *11th*

International Conference on VLDB, pp. 165-1 70, Stockholm, Sweden, August 1985.

[40] R. Rivest, "The MD5 Message-Digest Algorithm". RFC 1321, 1992.

[41] A. Menezes, P. Oorshot, and S. Vanstone, "Handbook of Applied Cryptography". CRC Press, 1997.

[42] R. Siamwalla, R. Sharma, and S. Keshav, "Discovering Internet Topology", July 1998. Available at: http://www.cs.cornell.edu/skeshav/

[43] K. Moore, J. Cox, and S. Green, "SONAR – a network proximity service". Internet Draft, February 1996.

[44] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. Gryniewicz, and Y. Jin, "An architecture for a global internet host distance estimation service". In *Proceedings of IEEE INFOCOM '99*, March 1999.

[45] T. Dewitt, T. Gross, B. Lowekamp, N. Miller, P. Steenkiste, J. Subhlok, AND D. Sutherland, "ReMoS: A resource monitoring system for network aware applications". Technical Report CMU-CS-97-1 94, School of Computer Science, Carnegie Mellon University, December 1997.

[46] C. Chiang, M. Ueno, M. Liu and M. Muller, "Modeling Web Caching Hierarchy Schemes", Technical Report, Ohio State University, OSU-CISRC-6/99-TR17, 1999.

[47] A. Rousskov and V. Soloviev, "On performance of caching proxies". In *Proceedings of the Joint International Conference on Measurement and Modeling*

of Computer Systems (SIGMETRICS '98/PERFORMANCE '98), pages 272-273, Madison, WI, June 1998.

[48] A. Rousskov and V. Soloviev, "A performance study of the squid proxy on http/1.0". In *World Wide Web*, 2(1-2):47-67, January 1999.

[49] W. Li, "References on Zipf's Law". Available at: http://linkage.rockefeller.edu/wli/zipf

[50] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications". In *Proceedings of INFOCOM'99*, 1999.

[51] L. Breslau, P. Cao, L. Fan, and G. Phillips, "On the Implications of Zipf's Law for Web Caching". Technical Report 1371, Computer Science Department, University of Wisconsin-Madison, April 1998.

[52] B. Duska, D. Marwood, and M. Feely, "The Measured Access Characteristics of World-Wide-Web Client Proxy Caches". In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS '97)*, December 1997.

[53] I. Marshall, and C. Roadknight, "Linking cache performance to user behaviour". In *Proceedings of the Third International WWW Caching Workshop*, June 1998.

[54] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira, "Characterizing reference locality in the WWW". In *Proceedings of the IEEE Conference on Parallel and Distributed Information Systems (PDI,)*, Miami Beach, FL, December 1996.

[55] P. Barford, A. Bestavros, A. Bradley, and M. Crovella, "Changes in Web client access patterns: Characteristics and caching implications". In *World Wide Web*, 2(1):15-28, January 1999.

Appendix A

Weighted Bloom Filter

LB-L5 uses a weighted Bloom Filter to represent cache content. Each cache server divides all cache objects into different sets and assigns a weight to each set according to their access-frequencies. The weight of a set is the number of hash functions that should be used, or the number of bits set to 1 in the filter, for an object belonging to the set. We call this method weighted Bloom Filter because it assigns a weight to each cache-object set.

The weighted Bloom Filter assigns a heavier weight to a set with higher access-frequency. Thus, objects with higher access frequencies are represented with more bits set to 1 in the filter. At the time of looking up an object, the number of bits set to 1 indicates the access-frequency rank of the object.

To analyze the false prediction probability and derive an optimum weight assignment, we formalize the problem using an approach similar to that in [39].

Assume that the set S of all possible objects in a cache is partitioned into n subsets S_1, S_2, ..., S_n, which are disjoint and whose union is S, that is

$$S = S_1 \cup S_2 \ldots \cup S_n$$

and

$$S_i \cap S_j = \varnothing \quad \text{, where } 1 \le i \le n, 1 \le j \le n, \text{and } i \ne j$$

Let D_i be the number of objects in S_i, and $D = D_1 + D_2 \ldots + D_n$ be the total number of objects in the cache. We define the access probability for objects in S_i to be P_i, and the weight for S_i to be W_i. Assume the filter length is F.

In a filter representing D objects, the probability that a particular bit is 0 is:

$$R_0 = \left(\frac{F-1}{F}\right)^{W_1D_1+W_2D_2+\ldots+W_nD_n} = \left(1-\frac{1}{F}\right)^{W_1D_1+W_2D_2+\ldots+W_nD_n}$$

or

$$R_0 \approx \left[1 - e^{-\frac{W_1D_1+W_2D_2+\ldots+W_nD_n}{F}}\right] \qquad (\text{when } 1 \ll F)$$

Hence, the probability that a particular bit is 1 is:

$$R = 1 - R_0 \approx 1 - e^{-\frac{W_1D_1+W_2D_2+\ldots+W_nD_n}{F}} \qquad (1)$$

The false prediction probability is:

$$F_p = P_1 R^{W_1} + P_2 R^{W_2} + \ldots + P_n R^{W_n} \tag{2}$$

To find the optimum W_i for each subset S_i such that the false prediction F_p is minimized, we differentiate F_p with respect to W_i's:

$$\frac{\partial F_p}{\partial W_i} = 0 \text{ , where } 1 \le i \le n \tag{3}$$

Because $\frac{\partial F_p}{\partial W_i} = \frac{1-R}{R}\frac{D_i}{F}\left[\frac{R}{1-R}\frac{F}{D_i}P_i R^{W_i}\ln R + \sum_{j=1}^{n} P_j W_j R^{W_j}\right]$, we have

$$\frac{R}{1-R}\frac{F}{D_i}P_i R^{W_i}\ln R + \sum_{j=1}^{n} P_j W_j R^{W_j} = 0 \qquad (1 \le i \le n) \tag{4}$$

Equation (4) is equivalent to

$$\frac{P_1 R^{W_1}}{D_1} = \frac{P_2 R^{W_2}}{D_2} = \ldots = \frac{P_n R^{W_n}}{D_n} = \frac{F_p}{D} = K \quad (K \text{ is a constant independent of } i) \tag{5}$$

Substituting equation (5) into (4), we have:

$$K\left[F\frac{R}{1-R}\ln R + \sum_{j=1}^{n} W_j D_j\right] = 0$$

or

$$\frac{R}{1-R} = -\frac{\sum_{j=1}^{n} W_j D_j}{F \ln R} \tag{6}$$

From equations (1) and (6), we have:

$$\frac{R}{1-R} = \frac{\ln(1-R)}{\ln R}$$

$$R = \frac{1}{2} \tag{7}$$

Substituting equation (7) into (6), we have:

$$\sum_{j=1}^{n} W_j D_j = F \ln 2 \tag{8}$$

Substituting equation (7) into (5), we have:

$$W_i = \frac{1}{\ln 2}\left[\ln \frac{P_i}{D_i} - \ln K\right] \qquad (1 \le i \le n) \tag{9}$$

Substituting equation (9) into (8), we have:

$$\ln K = \frac{-F(\ln 2)^2 + \sum_{i=1}^{n} D_i \ln \frac{P_i}{D_i}}{D} \tag{10}$$

Substituting equation (10) into (9), we give the optimum values for the W_i's:

$$W_i = \frac{F \ln 2}{D} + \frac{1}{\ln 2}\left[\ln \frac{P_i}{D_i} - \frac{\sum_{j=1}^{n} D_j \ln \frac{P_j}{D_j}}{D}\right] \qquad (1 \le i \le n) \tag{11}$$

From equation (5) and (10), we give the solution for F_p:

$$F_p = e^{\ln D - \frac{F(\ln 2)^2}{D} + \frac{\sum_{i=1}^{n} D_i \ln \frac{P_i}{D_i}}{D}} \tag{12}$$

The Bloom Filter used in ICP and Cache Digest is a special case of the weighted Bloom Filter. If we do not divide objects into subsets (n=1), equations (11) and (12) reduce to the corresponding formulas of the Bloom Filter without weights, as shown in equations (13) and (14), respectively.

$$W = \frac{F \ln 2}{D} \qquad (n = 1) \tag{13}$$

$$F_p = e^{-(\ln 2)^2 \frac{F}{D}} \qquad (n = 1) \tag{14}$$

To compare the weighted Bloom Filter with the Bloom Filter without weights, let us first investigate Web access characteristics. Researchers have found that a small fraction of the objects receives a large fraction of the accesses. More precisely, HTTP request distributions follow a Zipf's distribution [49,50,51,52,53,54,55]. The number of accesses for the i^{th} most popular object is: $R_i = \frac{N}{i^{\alpha}}$. The Zipf exponent α reflects the degree of popularity skew, and N represents the number of requests for the most popular object. In the reported Web access analysis, the exponent α ranges from 0.6 to 0.8.

For example, we assume that 30% of objects receive 70% of access requests. We divide S into subset S_1 and S_2 with P_1=0.7, P_2=0.3, and D_1/D_2=3/7. The comparison of the false prediction probability of these two Bloom Filters is shown in Table A.1 and Figure A.1.

	Bloom Filter without weights		Weighted Bloom Filter		
F/D	W	Fp	W1	W2	Fp
2	1.386294361	0.382546379	3.097643751	0.652858908	0.272579682
3	2.079441542	0.236606212	3.790790932	1.346006089	0.168591443
4	2.772588722	0.146341732	4.483938112	2.039153269	0.10427437
5	3.465735903	0.09051285	5.177085293	2.73230045	0.064494046
6	4.158883083	0.0559825	5.870232473	3.425447631	0.039889783
7	4.852030264	0.034625363	6.563379654	4.118594811	0.024671964
8	5.545177444	0.021415902	7.256526834	4.811741992	0.015259692
9	6.238324625	0.013245807	7.949674015	5.504889172	0.00943817
10	6.931471806	0.008192576	8.642821195	6.198036353	0.00583754
11	7.624618986	0.005067136	9.335968376	6.891183533	0.003610538
12	8.317766167	0.00313404	10.02911556	7.584330714	0.00223313
13	9.010913347	0.001938414	10.72226274	8.277477894	0.001381198
14	9.704060528	0.001198916	11.41540992	8.970625075	0.000854276
15	10.39720771	0.000741533	12.1085571	9.663772256	0.000528372
16	11.09035489	0.000458641	12.80170428	10.35691944	0.0003268
17	11.78350207	0.000283671	13.49485146	11.05006662	0.000202127
18	12.47664925	0.000175451	14.18799864	11.7432138	0.000125016
19	13.16979643	0.000108517	14.88114582	12.43636098	7.73229E-05
20	13.86294361	6.71183E-05	15.574293	13.12950816	4.78245E-05

Table A.1 Weighted Bloom Filter vs. Bloom Filter without weights

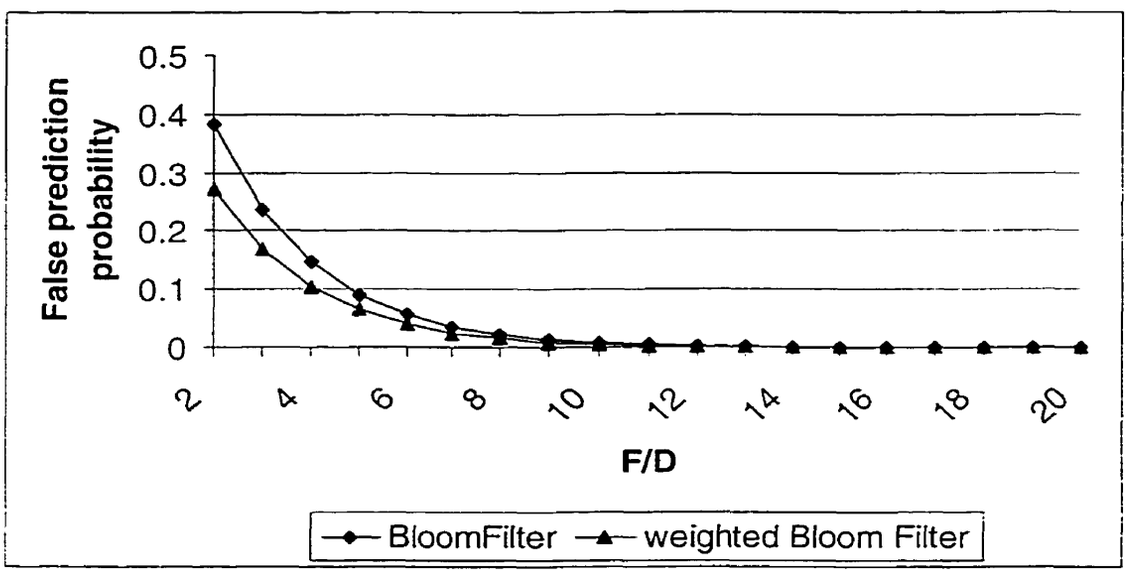


Figure A.1 Weighted Bloom Filter vs. Bloom Filter without weights

Appendix B

MD5 Hash Function

MD5 processes the input text in 512-bit blocks, which are divided into 16 32-bit sub-blocks. The output of the algorithm is a set of four 32-bit blocks, which concatenate to form a single 128-bit hash value. The algorithm consists of following steps:

- **Step 1: Pad the message, make the message length = 512 x n**

First, the message is padded so that its length is just 64 bits short of being a multiple of 512. This padding is a single 1-bit added to the end of the message, followed by as many zeros as are required. Then, a 64-bit representation of the message's length (before padding bits were added) is appended to the result. These two steps serve to make the message length an exact multiple of 512 bits in length (required for the rest of the algorithm), while ensuring that different messages will not look the same after padding.

- **Step 2: Initialize chaining variables**

Four 32-bit chaining variables are initialized:

```
A = 01234567(Hex)
B = 89abcdef(Hex)
C = fedcba98(Hex)
D = 76543210(Hex)
```

- **Step 3: Main loop**

As shown in Figure B.1, the main loop has four rounds. The loop continues for every 512-bit block in the message.

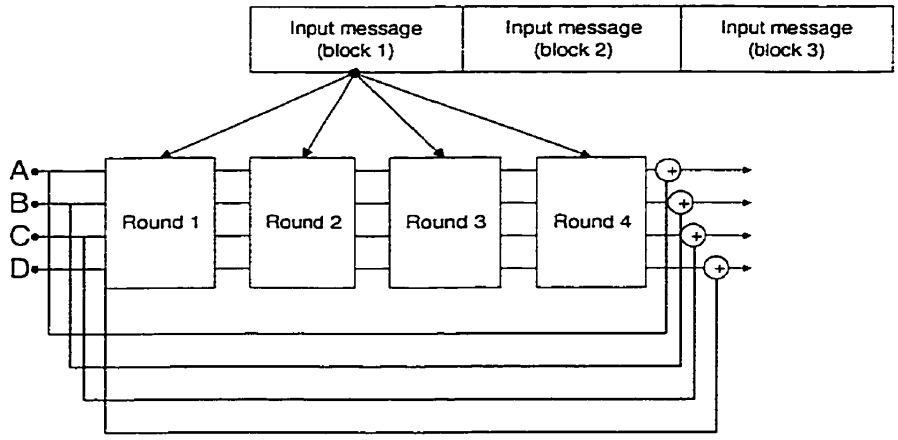


Figure B.1 MD5 main loop

Each round uses a different operation 16 times. As shown in Figure B.2, each operation performs a nonlinear function on three variables of A, B, C, and D. Then it adds the result to the fourth variable, a sub-block of the text and a constant. Then it rotates that result to the right a variable number of bits and adds the result to one of A, B, C, or D. Finally the result replaces one of A, B, C, or D.

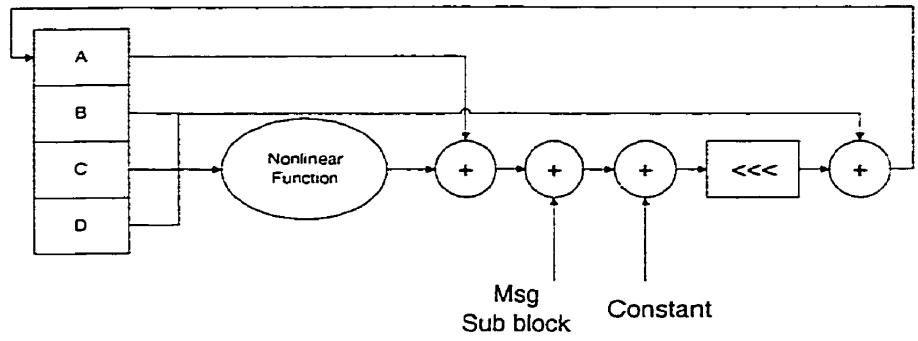


Figure B.2 One operation in one round in MD5

There are four nonlinear functions, one used in each operation (a different one for each round).

$F(X,Y,Z) = (X \text{ AND } Y) \text{ OR } ((\neg X) \text{ AND } Z)$
$G(X,Y,Z) = (X \text{ AND } Z) \neg (Y (\neg Z))$
$H(X,Y,Z) = X \text{ XOR } Y \text{ XOR } Z$
$I(X,Y,Z) = Y \text{ XOR } (X \text{ OR } (\neg Z))$

These functions are designed so that if the corresponding bits of X, Y, and Z are independent and unbiased, then each bit of the result will also be independent and unbiased. The function F is the bit-wise conditional: If X then Y else Z. The function H is the bit-wise parity operator.

Appendix C

The Flow Charts of Web Caching Schemes

In this study, we evaluate the performance of ICP, Cache Digest, basic L5 and LB-L5 Web caching schemes. The HTTP request processing flow charts of the four schemes are shown in Figure B.1 to Figure B.8. These flow charts are drawn according to the scheme descriptions in ICP [15,16], Cache Digest [19], the basic L5 caching [23], and LB-L5 as described in Chapter 3. A similar simulation model was used by Chiang to model and evaluate hierarchical Web caching schemes [46].

As shown in Figure C.1, in an ICP fully distributed cache server mesh, HTTP requests generated by clients using the GET method are processed as follows. When a proxy cache server receives a HTTP GET request, it searches its cache for the requested object. If the requested object is found in the cache, the proxy cache server returns the object to the requesting client. Otherwise, the proxy cache server sends an ICP query message to all the siblings, which in turn search their own caches for the requested object and reply with an ICP

HIT or MISS according to the search result. If none of its sibling replies with an ICP HIT message, the proxy cache server forwards the request to the original Web server. If at least one of its siblings replies with an ICP HIT message, the proxy server fetches the requested object from the sibling whose reply comes first, and then sends the requested object to the requesting client.

HTTP requests generated by clients using the GIMS (Get If Modified Since) method are processed as shown Figure C.2. The HTTP GIMS request is used when a client has a cached copy of the requested object. The GIMS request carries a timestamp indicating when the client has cached the object. A proxy cache server or the original Web server should reply to the GIMS request with the requested object if and only if the object is modified after the timestamp. If the requested object has not been modified after the timestamp, a short HTTP message, Code 304, should be sent to the client. Thus HTTP GIMS requests are processed differently from the GET requests after the reques ted objects are found: an extra step is taken to check if the object has been modified since the client cached it.

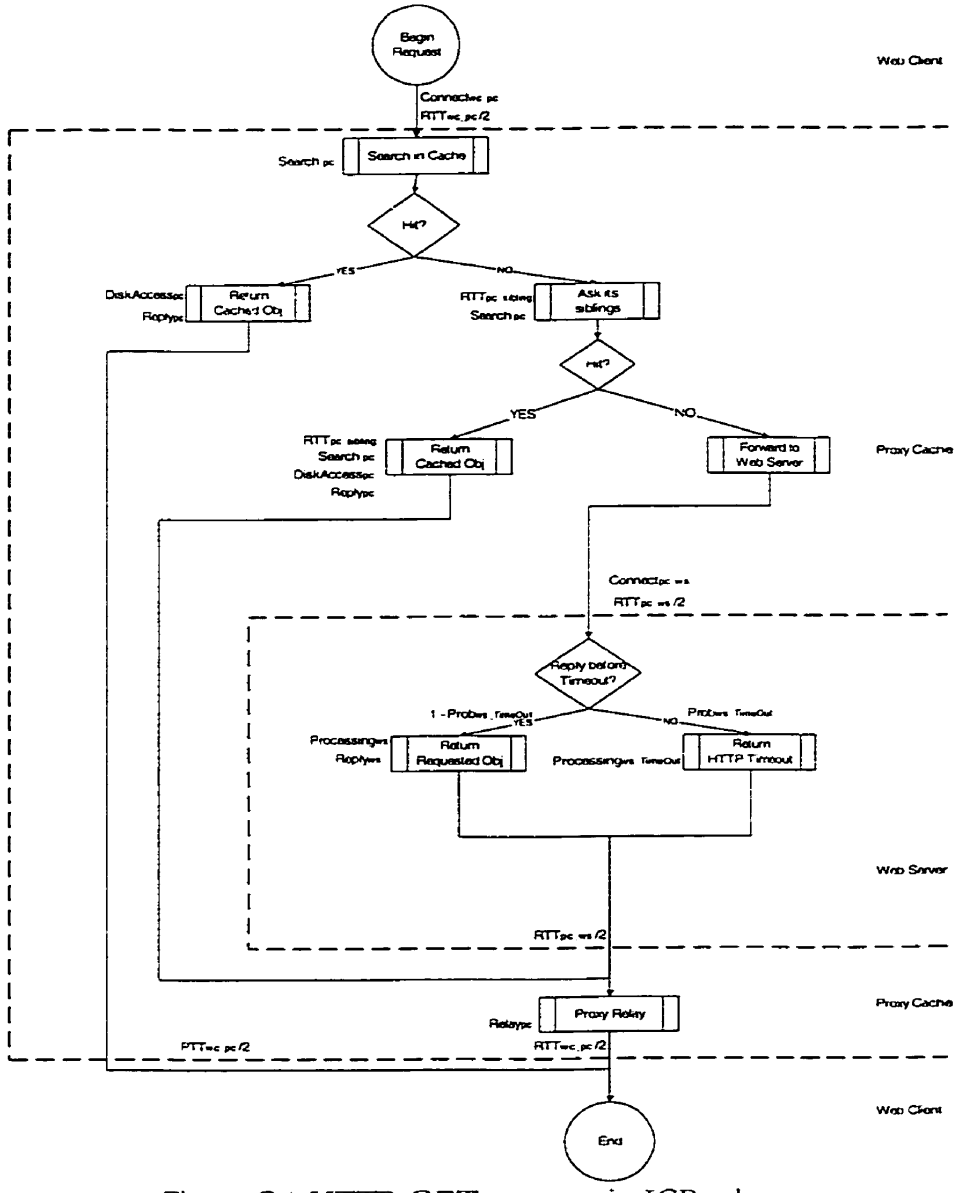


Figure C.1 HTTP GET requests in ICP scheme

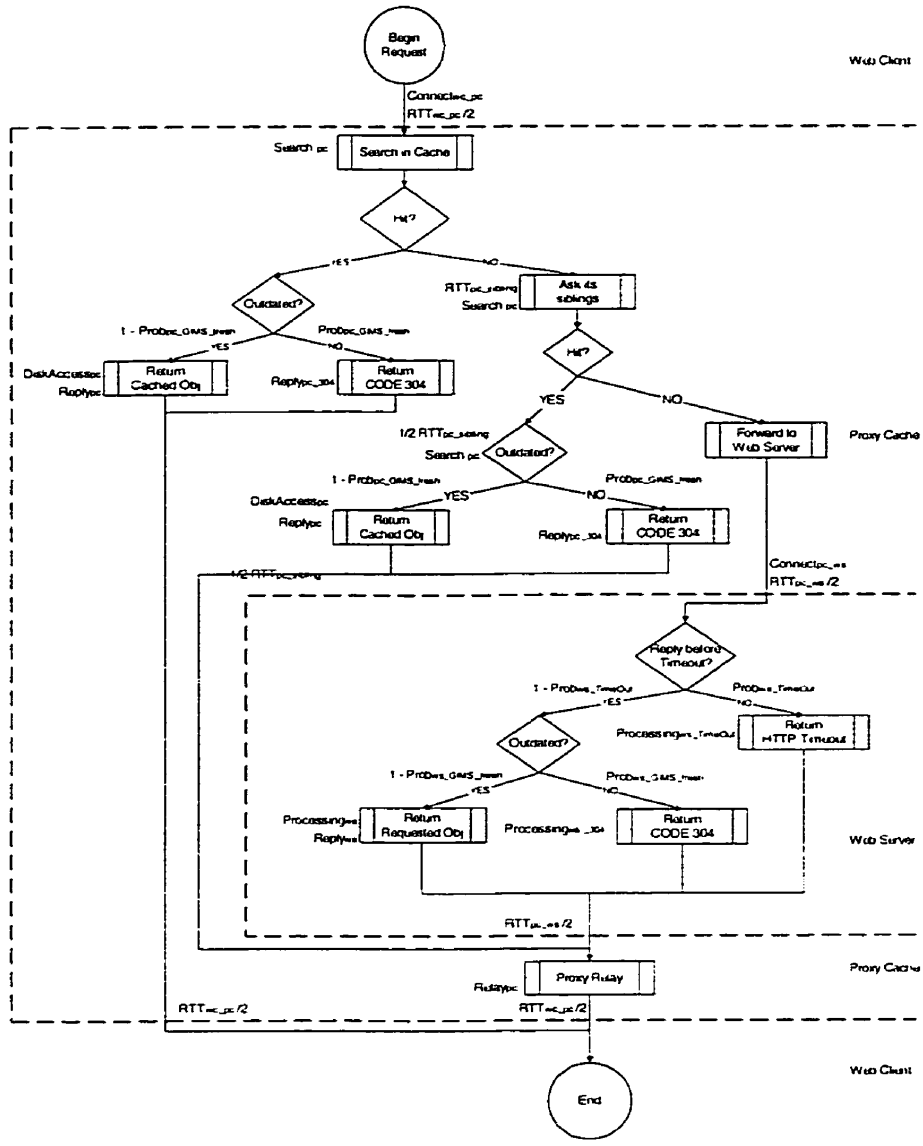


Figure C.2 HTTP GIMS requests in ICP scheme

Unlike query-based ICP, Cache Digest is a directory-based Web caching scheme. When a proxy cache server cannot find a requested object in its cache it searches the siblings' digests (the directory of cache contents). If the requested object is found in the digest of a sibling cache, the proxy cache server requests the object from the sibling. As described in Chapter 2,

Cache Digest uses a Bloom Filter to represent the cache contents, which can introduce false predictions, that is an object is shown in the filter but is not actually in the cache. If the sibling server receiving the request does not find the requested object, then it forwards the request to the original Web server. The processing flow charts of HTTP GET and GIMS requests are shown in Figure C.3 and Figure C.4, respectively.

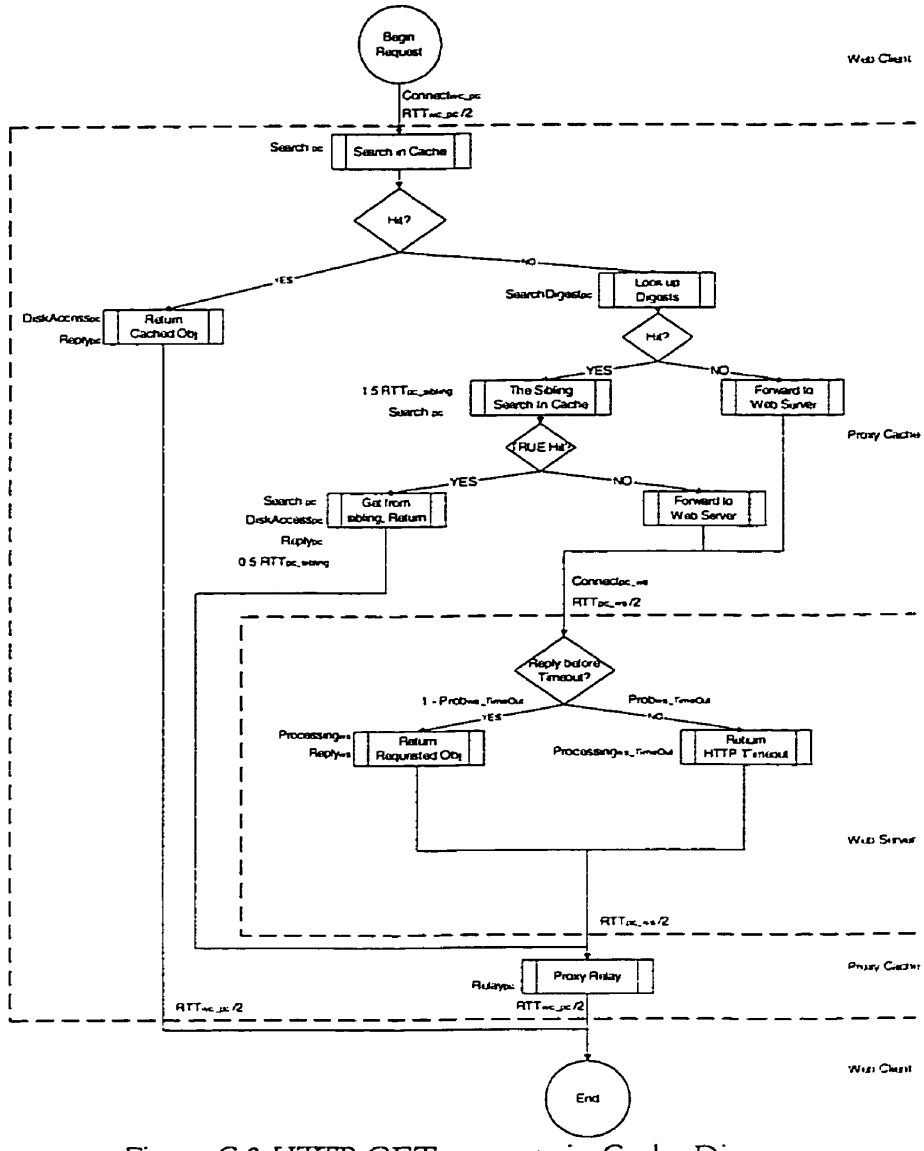


Figure C.3 HTTP GET requests in Cache Digest

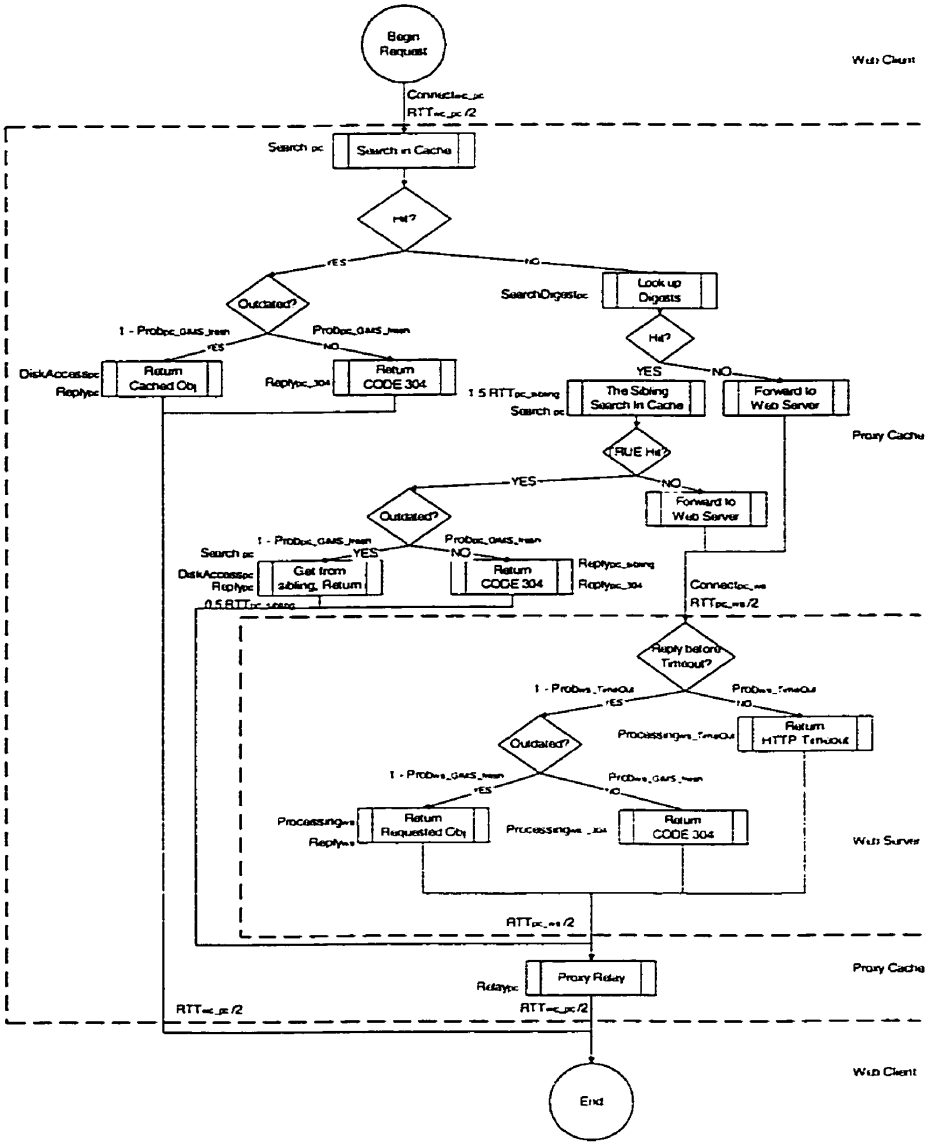


Figure C.4 HTTP GIMS requests in Cache Digest

In the basic L5 Web caching scheme, every L5 switch transparently inspects HTTP requests from a client cluster. It redirects the cacheable requests to its associated proxy cache server, and non-cacheable requests to the original Web servers. The HTTP GET and GIMS request

processing flow charts are shown in Figure C.5 and Figure C.6 respectively.

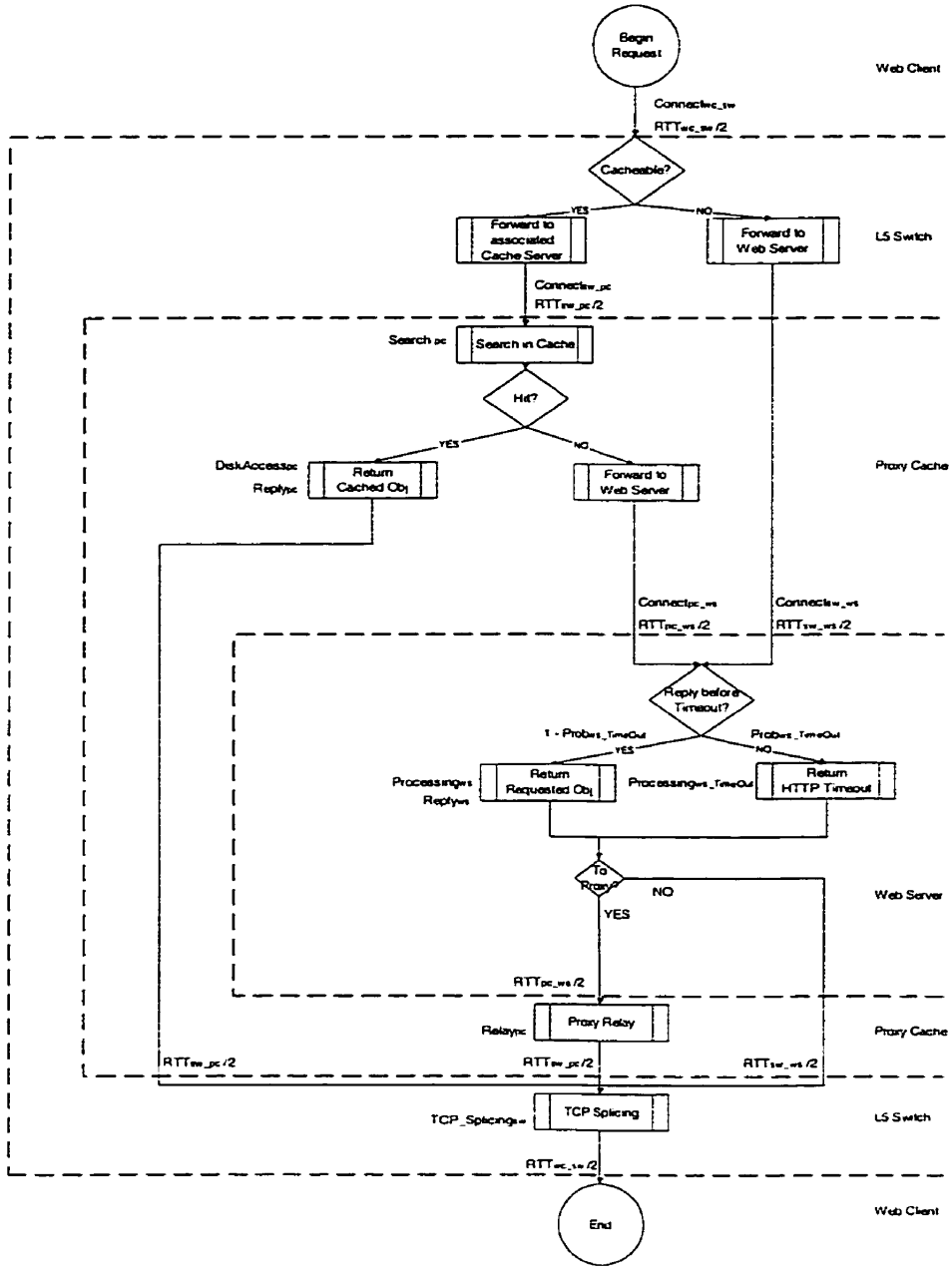


Figure C.5 HTTP GET requests in the basic L5 scheme

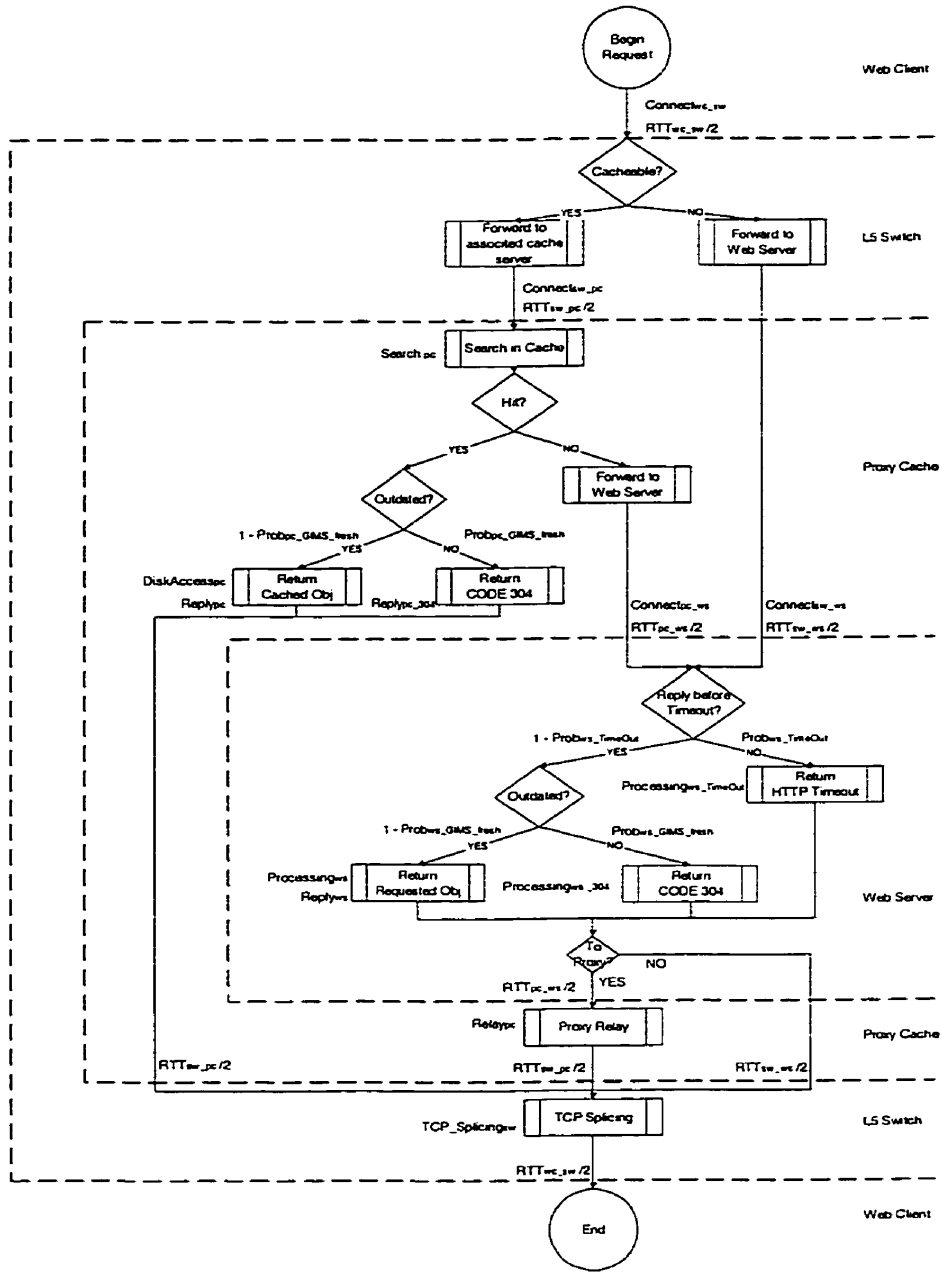


Figure C.6 HTTP GIMS requests in the basic L5 scheme

In LB-L5, the switch inspects the HTTP requests in the same way as in the basic L5 scheme. The non-cacheable HTTP requests are redirected to the original Web servers. However, The

cacheable HTTP requests are redirected to one of a set of cooperating cache servers according the switch's routing decision, which is based on the cache content and workload information of cache servers, as well as the network link delay between the routing switch and cache servers. Figure C.7 and Figure C.8 show the HTTP GET and GIMS requests processing flow chats in the LB-L5 scheme.

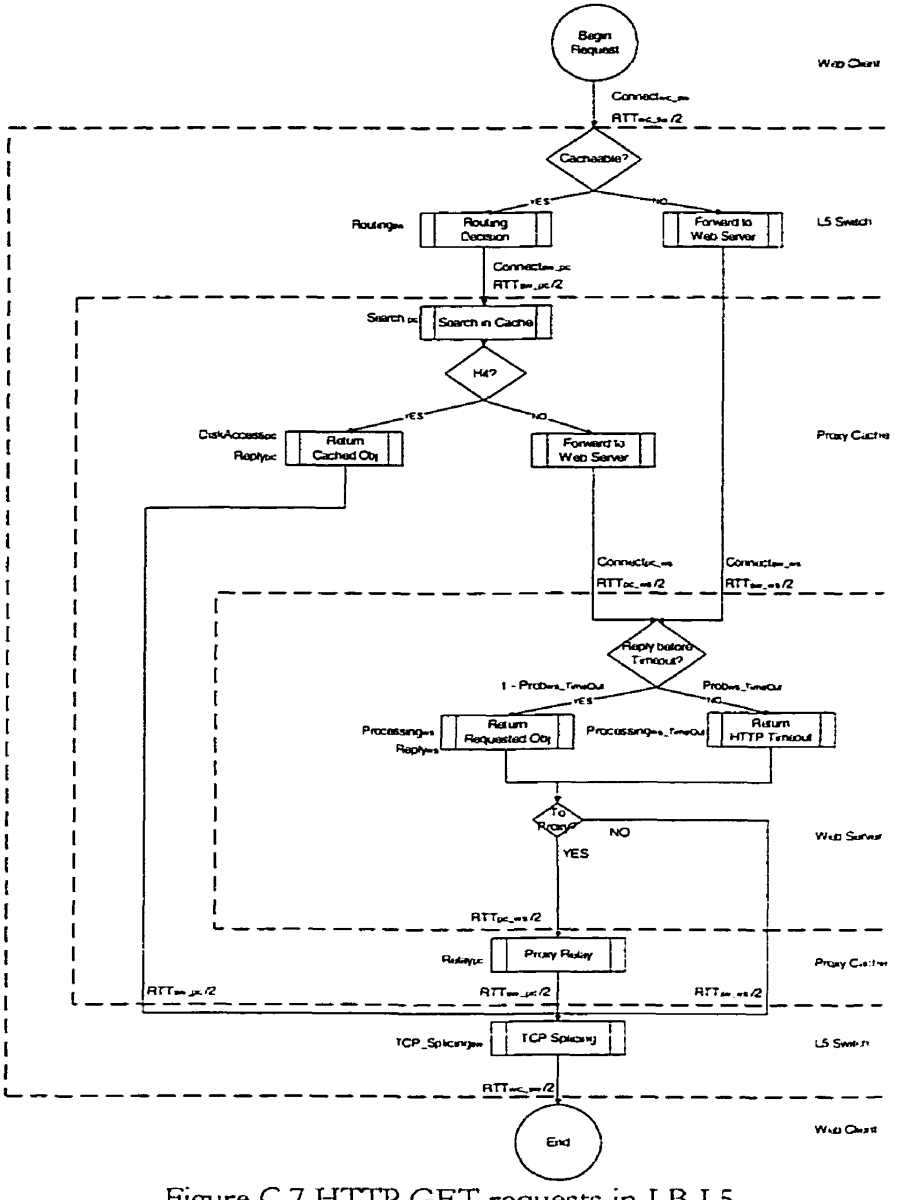


Figure C.7 HTTP GET requests in LB-L5

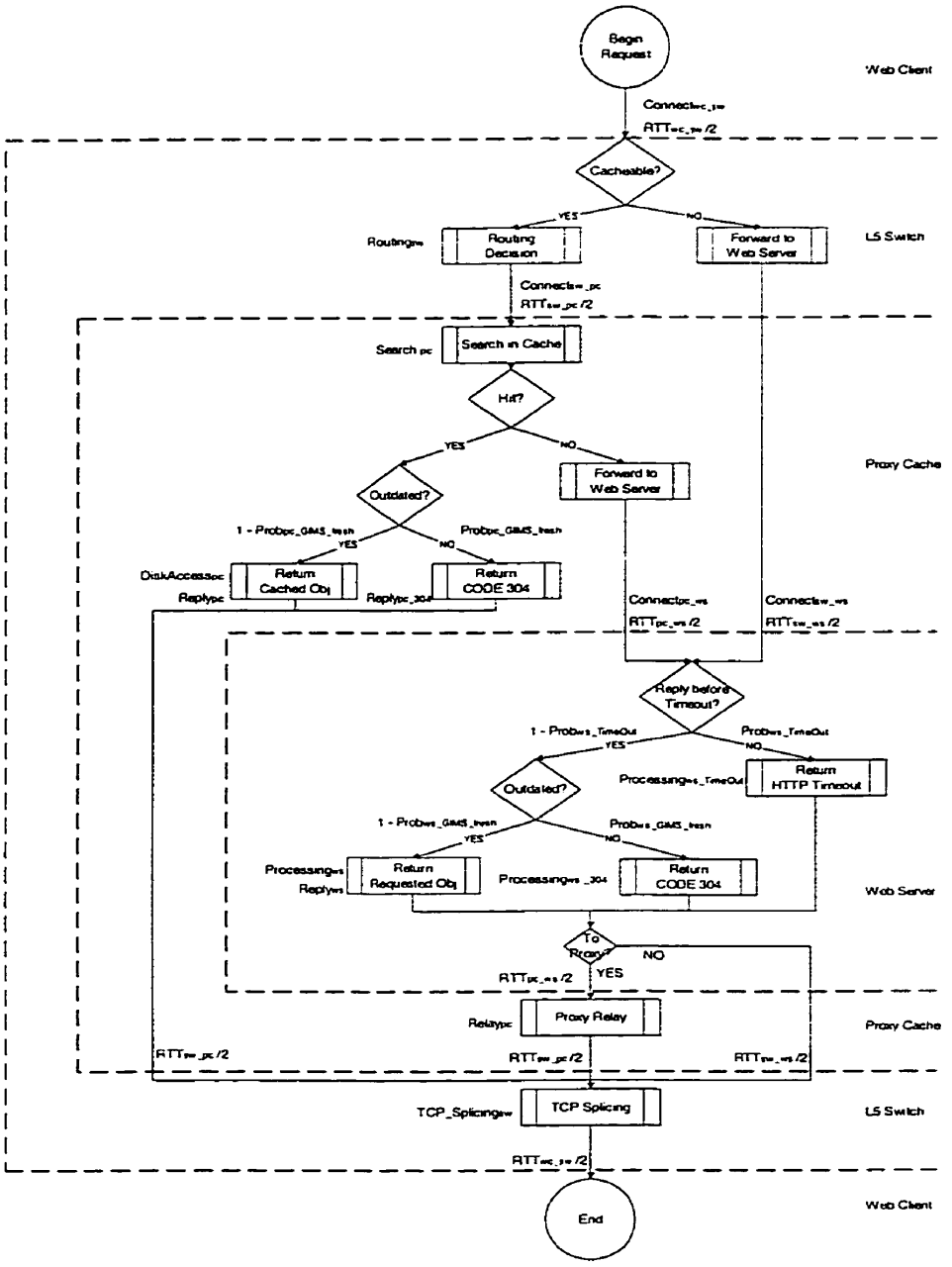


Figure C.8 HTTP GIMS requests in LB-L5

Appendix D

The Simulation Software Structure

The simulator used in this study uses discrete event driven simulation to simulate ICP, Cache Digest, basic L5, and the LB-L5 Web caching schemes. The simulation software is developed using the Java programming language.

As shown in Figure C.1, client clusters, L5 switches, proxy cache servers, Web servers, and network links are modeled and simulated with different simulation objects. Both the communication between objects and tasks performed are modeled as simulation events. Every simulation object has an event handler to process received events, and can schedule events for its associated objects. All events are sent to the EventManager module, where events are queued and then dispatched to handle objects according to time ordering.

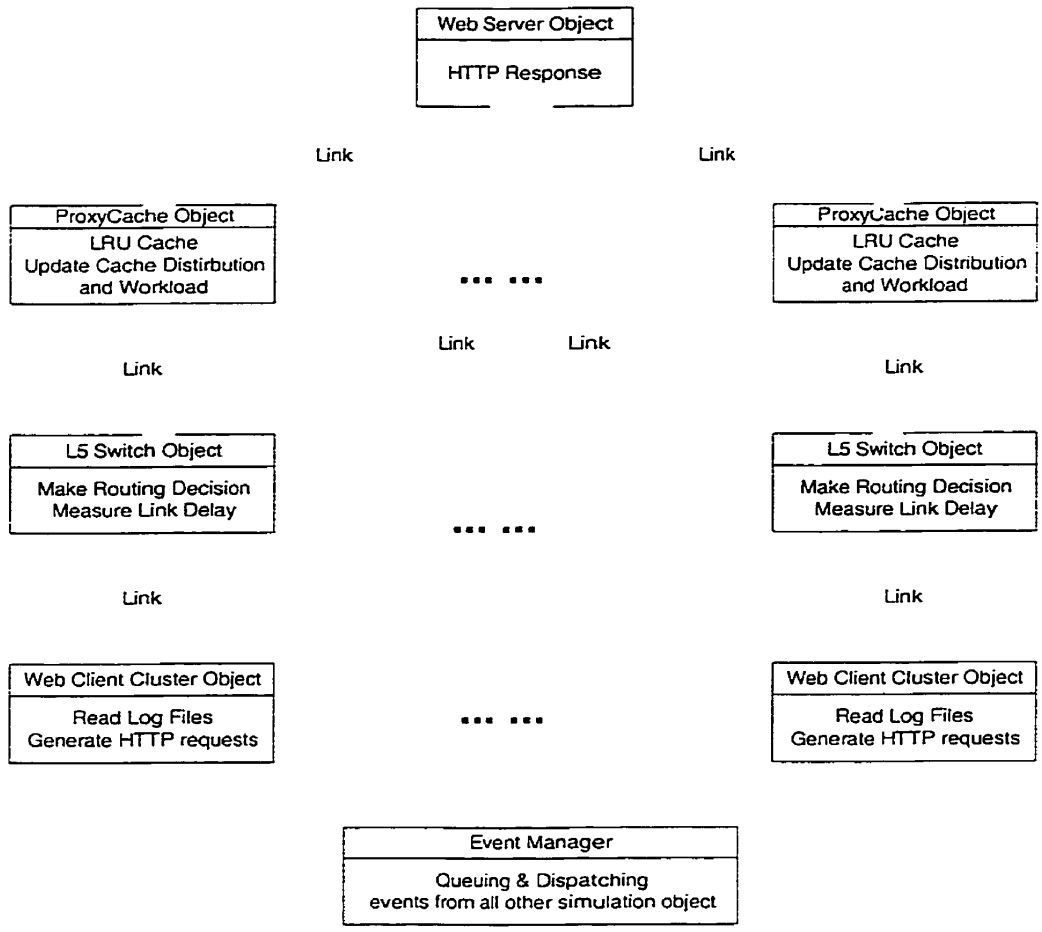


Figure D.1 Simulation software structure

The simulation software package consists of 31 classes. The major classes are described as follows:

- **Interface SimuObject**

Interface *SimuObject* defines the virtual method *public boolean eventHandle(SimuEvent se).* All simulation object classes implement this interface, and process received events in their own *eventHandle* methods.

- **Class ClientCluster**

Class *ClientCluster* simulates a client cluster by reading proxy trace files and generating HTTP requests. For example, it can schedule a *ReadNextLogEntry* to itself for getting a log entry from trace files and generate a HTTP request. It can also schedule *TPC_SYN* or *HTTP_Request* events for sending a request to a proxy server. *ClientCluster* handles events from its associated links. For example, *TCP_ACK*, *HTTP_Response* for processing messages from proxy servers.

- **Class L5Switch**

Class *L5Switch* simulates a basic L5 switch. It conducts TCP Spoofing to inspect the HTTP requests from a client cluster, and then directs cacheable requests to its associated cache server and non-cacheable requests to the Web server. *L5Switch* can handle events from its associated links. For example, it handles *TCP_SYN* and *HTTP_Requesst* events from a link to a *ClientCluster*; and *TCP_ACK* and *HTTP_Response* events from a link to a *WebServer*. It also schedules events for its associated links. For example, it schedules *TCP_ACK* and *HTTP_Request* events to a *WebServer*, and *TCP_ACK* and *HTTP_Response* events to a *ClientCluster*.

- **Class LB_L5Switch**

Class *LB_L5Switch* simulates a LB-L5 switch. In addition to the functions of a basic L5 switch, it communicates with cache servers to obtain their workload, cache content and access frequency information. It also measures network link delay between itself and cache servers. *LB_L5Switch* uses the obtained information to redirect HTTP requests to one of the cooperating cache servers.

LB_L5Switch supports extended ICP messages in its *eventHandle* method, and schedules events corresponding to such messages on its associated links. For example, it handles *ICP_UPDATE_CONTENT* events from links to *L5ProxyCaches*; and schedules *ICP_QUERY_WORKLOAD* events to *L5ProxyCaches*.

- **Class ProxyCache**

Class *ProxyCache* is the super class of all proxy classes. It implements the Least Recently Used (LRU) cache replacement algorithm to simulate a LRU cache. All proxy classes derived from this class inherit its methods implementing LRU cache.

- **Class LBL5_ProxyCache**

Class *LBL5_ProxyCache* simulates a proxy cache server supporting LBL5 switches. In addition to the functions of a basic LRU proxy cache server, it communicates with L5 switches for updating its workload, cache content and access frequency information.

LB_L5Switch supports extended ICP messages in its *eventHandle* method, and schedules events corresponding to extend ICP messages for its associated links. For example, it handles *ICP_QUERY_WORKLOAD* events from links to *L5ProxyCaches*; and schedules *ICP_UPDATE_CONTENT* events for links to *L5ProxyCaches*.

- **Class ICP_ProxyCache**

Class *ICP_ProxyCache* simulates a proxy cache server supporting the ICP protocol. In addition to the functions of a basic LRU proxy cache server, it communicates with cooperating ICP proxy cache servers to share cached Web objects.

ICP_ProxyCache supports ICP messages in its *eventHandle* method, and schedules events corresponding to ICP messages for its associated links. For example, it handles *ICP_QUERY* events from links to other *ICP_ProxyCaches*; and responds the query by scheduling *ICP_HIT* or *ICP_MISS* events for the link to the querying proxy.

- **Class CD_ProxyCache**

Class *CD_ProxyCache* simulates a proxy cache server supporting the Cache Digest protocol. In addition to the functions of a basic LRU proxy cache server, it communicates with cooperating Cache Digest proxy cache servers to share cached Web objects.

A CD_ProxyCache computes its cache digest represented with a Bloom Filter periodically, and publishes the digest to other cooperating Cache Digest proxy servers. Whenever a *CD_ProxyCache* receives a request and cannot find the requested object in its own cache, it searches the cache digests of cooperating cache servers, and fetches the object from a cooperating server that has this requested object.

- **Class WebServer**

Class *WebServer* simulates a Web server. It accepts HTTP requests and sends back HTTP responses. *WebServer* supports TCP and HTTP messages in its *eventHandle* method, and schedules events corresponding to TCP and HTTP messages for its associated links. For example, it can handle a *TCP_SYN* event from a link; and respond the TCP message by scheduling a TCP_ACK event to the link.

- **Class EventManager**

Class *EventManager* uses a linked list to queue simulation events scheduled by all simulation objects, and then dispatch the events to their receiving objects in time order.

- **Class MD5**

Class *MD5* implements MD5 hash function. The *computeHashValue* method of this class is used for building the Bloom Filter in the Cache Digest scheme and the weighted Bloom Filter in the LB-L5 Web caching scheme.

Appendix E

Confidence Intervals

The accuracy of simulation results can be described in terms of confidence intervals placed on the mean values of the results. The confidence interval calculation procedure is described as follows:

Let $Y_1, Y_2, \ldots, Y_N$ be the statistically independent results from N different runs of the same simulation. The sample mean, $\overline{Y}$, of these results is:

$$\overline{Y} = \frac{\sum_{i=1}^{N} Y_i}{N} \tag{1}$$

The variance of the distribution of the sample values, S_Y^2, is:

$$S_Y^2 = \frac{\sum_{i=1}^{N} (Y_i - \overline{Y})^2}{N-1} \tag{2}$$

The standard deviation of the sample mean is:

$$\frac{S_Y}{\sqrt{N}} \tag{3}$$

Under the assumption of independence and normality, the sample mean is distributed in accordance to the t-distribution. The upper and lower limits of the confidence interval regarding the simulation results are:

$$LowerLimit = \overline{Y} - \frac{S_Y t_{\alpha/2,N-1}}{\sqrt{N}} \tag{4}$$

$$UpperLimit = \overline{Y} + \frac{S_Y t_{\alpha/2,N-1}}{\sqrt{N}} \tag{5}$$

where $t_{\alpha/2,N-1}$ is the upper $\alpha/2$ percentile of the t-distribution with $N-1$ degrees of freedom.

The simulation experiments in this thesis were run large enough times to ensure a 90% confidence level with 10% confidence intervals.