Transparent Web Caching with Minimum Response Time

By Qing Zou

A thesis submitted to the

Department of the Computing and Information Science

in conformity with the requirements

for the degree of the Master of Science

Queen's University

Kingston, Ontario, Canada

January 2002

Abstract

Distributed Web caching systems allow better load sharing and more fault tolerance in Web caching systems. Layer 5 switching-based transparent Web caching schemes intercept HTTP requests and redirect them according to their content. Employment of these schemes in distributed Web caching systems provides balanced server workload, reduced response time and improved cache sharing. However, none of the existing schemes attempt to minimize the HTTP request response time.

In this thesis, we propose a Minimum Response Time (MRT) Layer 5 switching-based Web caching scheme for distributed Web caching systems. MRT distinguishes non-cacheable requests from cacheable requests based on HTTP request header. It intelligently redirects cacheable requests to the cache server with the minimum HTTP request response time based on the information about cache server content, cache server workload, Web server workload and network latency. MRT extends ICP to support the communication between cache servers and Layer 5 switches. The MRT heuristic is a solution to optimize the performance of the distributed Web caching.

A number of simulation experiments are conducted under different HTTP request intensities, network latency factors, object expiration time values and number of cooperating cache servers. Simulation results show that MRT outperforms existing switching-based Web caching schemes, namely Content, Workload, RTT and LB_L5 in terms of HTTP request response time.

Acknowledgements

I would like to express my gratitude to my supervisors Professor Patrick Martin and Professor Hossam Hassanein, for their excellent guidance, great advice and support.

I would like to thank Wendy Powley and all the other members in the Database System Laboratory and the Telecommunications Research Laboratory at Queen's University for their suggestions, assistance, comments and friendship.

Special thanks to my husband Gang, for his selfless support and great advice. Thanks to Jasmi John and Hongzhi Li at the Computer Science Department, Queen's University for their generous help. Thanks to Zhenggang Liang for help with the simulation code.

Finally, I would also like to thank the Department of Computing and Information Science at Queen's University for offering me such an invaluable opportunity to pursue my Master's Degree. The financial support provided by Communications and Information Technology Ontario (CITO) and Kingston Software Factory is appreciated.

Table of Contents

List of Figures

List of Tables

List of Acronyms

CGI	Common Gateway Interface
CSS	Content Smart Switch
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
ICP	Internet Cache Protocol
LB_L5	Load Balancing Layer 5 switching-based transparent
NLANR	National Laboratory for Applied Network Research
MRT	Minimum Response Time
RTT	Round Trip Time
SSH	Secure Shell Remote Login Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator
W3C	World Wide Web Consortium

Chapter 1

Introduction

The rapidly increasing number of Web applications, coupled with the rapidly increasing number of documents accessible by Web clients, has resulted in an explosive increase in Web traffic expressed both in terms of HTTP requests and HTTP replies. HTTP Web traffic has grown to account to 75-80% of all Internet traffic [1]. There is no indication that this increase will abate in the near future. In fact, the number of Web users keeps increasing and the Web is used in ever more different ways to access a wide variety of text, still images, audio and video documents. This popularity is raising an urgent need for solutions aimed at improving the quality of the service provided by the Web.

Web caching [1] [2] is one of the most popular solutions to the problem mentioned above. It is a technique that uses caches over the Internet for replication of the most frequently accessed data. Various approaches have been examined in order to increase the performance of Web caching. These include the use of large caches and of more efficient cache management techniques. However, the effectiveness of a single cache remains poor as it is, in general, no higher than 40% [3]. Furthermore, the use of large

caches raises financial and technical problems. Other efforts have focused on pre-fetching of data to caches but the resulting traffic overhead is too costly [4]. Another way to increase Web caching performance is to expand solutions from the level of a single cache to the level of a set of cooperating caches. Cache cooperation provides a mechanism to share documents among caches and to share one cache among a number of clients [5] [6] [7] [8].

The most popular types of cooperative cache systems are the hierarchical and the distributed systems, which are both implemented by the Squid software [1] [9] as part of the Harvest project [10]. Several Web caching schemes are deployed by Squid to support cache cooperation. Internet Cache Protocol (ICP) [11] is employed to exchange the messages between cache servers. A Bloom Filter [12] is used in Squid to represent the cache content compactly. In this type of cooperation, approaches for inter-proxy cooperation try to maximize the global hit ratios. A Web client's local proxy redirects requests to one of its cooperative cache servers when it is a cache-miss on the local cache server. The redirection is based on the query results of the contents of the cooperative cache servers.

Traditional hierarchical Web caching systems [1][9] have several drawbacks. Shared higher-level cache servers may be too far away from the client. Cache misses are significantly delayed by having to traverse the hierarchy. As well, redundant data are stored on higher-level cache servers and the higher-level cache servers may become a bottleneck. Distributed Web caching systems [1][9][13] [14] [15] [16] rely on replicated

objects and services to improve performance and reliability. There are no hierarchies among cache servers. All cache servers are employed at the same level. So distributed Web caching systems overcome the drawbacks of hierarchical Web caching systems. Moreover, they have better fault tolerance, distribution of server loads and improvement of client performance by bringing cache servers closer to Web clients.

In both traditional hierarchical or distributed Web caching systems, the redirection of HTTP requests is done by cache servers. There are cases in which copies of objects in some distant cache servers may not be worth fetching. Instead the original Web server itself may be a better choice. Sometimes copies in a heavily loaded cache may be costly to fetch and instead a lightly loaded cache may be a better choice. It is difficult for a cache server to collect and process the load information of all the cooperative cache servers and network load information. The packet processing functions and packet forwarding functions may not be efficient if performed at cache servers.

Recently, a new type of Web caching technique has emerged. It is called switching-based transparent Web caching [17] [18]. A switch sits in the data path between the Web clients and the server cluster. It intercepts the Web traffic and transparently redirects the HTTP requests to different cache servers or to the Web server. Transparent Web caching makes the configuration of the caching system easier. Switches can rapidly process and forward the packets. This switching-based transparent Web caching technique can use content-aware Layer 5 switches in a distributed Web caching system with enhanced cache cooperation [19] [20] [21] [22]. The switches perform content checking based on

Layer 5 header information of the HTTP request packets. A HTTP request is redirected by switches to the cache server that can best service the request.

From the perspective of the Web clients, the request response time is an essential component of quality of service. However, fluctuations in network congestion and server load make it difficult to collect the information and to predict response times. In the existing switching-based transparent Web caching system, the switch uses several performance estimators to approximate the HTTP request response time. For example, a ping probe [23] measures current network latency but does not measure server workload. Another example is Cisco CSS 11000 [22], which requires the cooperative cache servers be within the same LAN. Z. Liang proposes a Load Balancing Layer 5 switching-based transparent Web caching scheme (LB_L5) [20]. LB_L5 considers both workload and network latency, but LB-L5 has one main drawback in that it cannot guarantee minimum response time for Web requests.

In this thesis, we propose the Minimum Response Time (MRT) switching-based Web caching scheme. The main goals of our research are:

- To propose a solution to optimize the performance of distributed switching-based transparent Web caching systems. The proposed scheme should minimize the HTTP request response time and balance the workload among the caches based on a combination of request content, cache server content, network latency and server workload.

- To develop a trace driven simulation to evaluate the performance of the proposed scheme.

The rest of the thesis is organized as follows. Chapter 2 surveys existing work on distributed Web caching systems. In Chapter 3, we describe the proposed MRT scheme. Simulation results and analysis are reported in Chapter 4. Finally, in Chapter 5 we conclude the thesis, list the contributions of our work and discuss future research directions.

Chapter 2

Related Work

Web caching improves the quality of the service provided by the Internet. A single cache, however, has a finite size and there is a limit to the number of objects that can be cached. A group of cache servers can be used to realize cache cooperation. The two most popular types of cooperative cache systems are the hierarchical and the distributed systems. In hierarchical Web caching architecture, cache servers are placed at multiple levels of the network. On the other hand, in distributed caching architecture, caches are placed at the bottom levels of the network and there are no intermediate caches. Such distributed systems rely on replicated objects at different locations and services to improve performance and reliability. The design of efficient server selection algorithms is critical for distributed Web caching schemes.

This chapter presents a literature review of server selection algorithms in distributed Web caching systems. Section 2.1 gives a brief introduction to the hierarchical and the distributed Web caching models. The server selection methods fall into three categories:

client-initiated [23] [24] [25] [26] [27], switch-based [17] [18] [19] [20] [21] [22] and server-initiated [28] [29] methods (depending on who makes the selection). Both client-initiated and switch-based approaches are aimed at a group of servers that are heterogeneous. Section 2.2 discusses current client-side selection algorithms. Section 2.3 presents selection methods that rely on network switches, which are the focus of our work. Switches choose among the interfaces by deciding which is "best", where best is defined by request contents and the switch metrics. In server-initiated methods, the servers decide where to send the requests. The server side algorithms are used for Web server clusters, which typically contain members with similar resources and a shared local network. As they do not directly pertain to the work in this thesis, they are not discussed further. Finally a summary is given in section 2.4.

2.1 Hierarchical and Distributed Web Caching

Hierarchical Web caching is one form of cooperation among cache servers. Traditional hierarchical cache server architectures such as Squid [1][9] define parent-sibling relationships among cache servers. A parent cache server is essentially one level up in a cache hierarchy. A sibling cache server is on the same level. Each cache server in the hierarchy is shared by a group of clients or by a group of children cache servers, as shown in Figure 2.1.

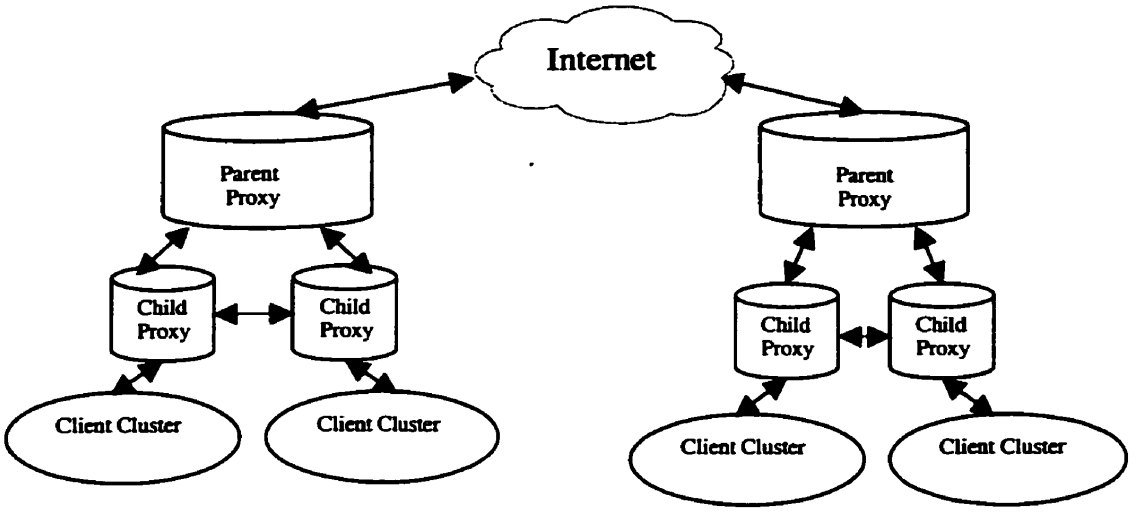


Figure 2.1 Hierarchical Architecture for Web Caching

Data access proceeds as follows: if the lowest-level cache server contains the data requested by a client, it sends the data to the client, otherwise, the cache server asks each of its siblings for the data. If none of the siblings possess the data, then the cache server sends a request to its parent. This process recursively continues up the hierarchy until the data is located or the root cache server fetches the data from the Web server. The cache servers then send the data down the hierarchy and each cache along the path stores the data.

Traditional cache hierarchies have several problems. First, a request may have to travel many hops in a cache hierarchy directory to get to the data, and the data may traverse several hops back to get to the clients. Second, cache misses are significantly delayed by having to traverse the hierarchy. Third, there is little sharing of data among caches. Fourth, shared higher-level cache servers may be too far away from the client and the time for the object to reach the client is simply unacceptable.

Distributed Web caching allows the distribution of caching proxies geographically over large distances and attempts to overcome some of the drawbacks of traditional hierarchical Web caching. Cache servers are organized into cache clusters with no definite hierarchy among them, as shown in Figure 2.2. A device, such as a switch or a local cache of the client cluster, sits between the client cluster and the cache server cluster.

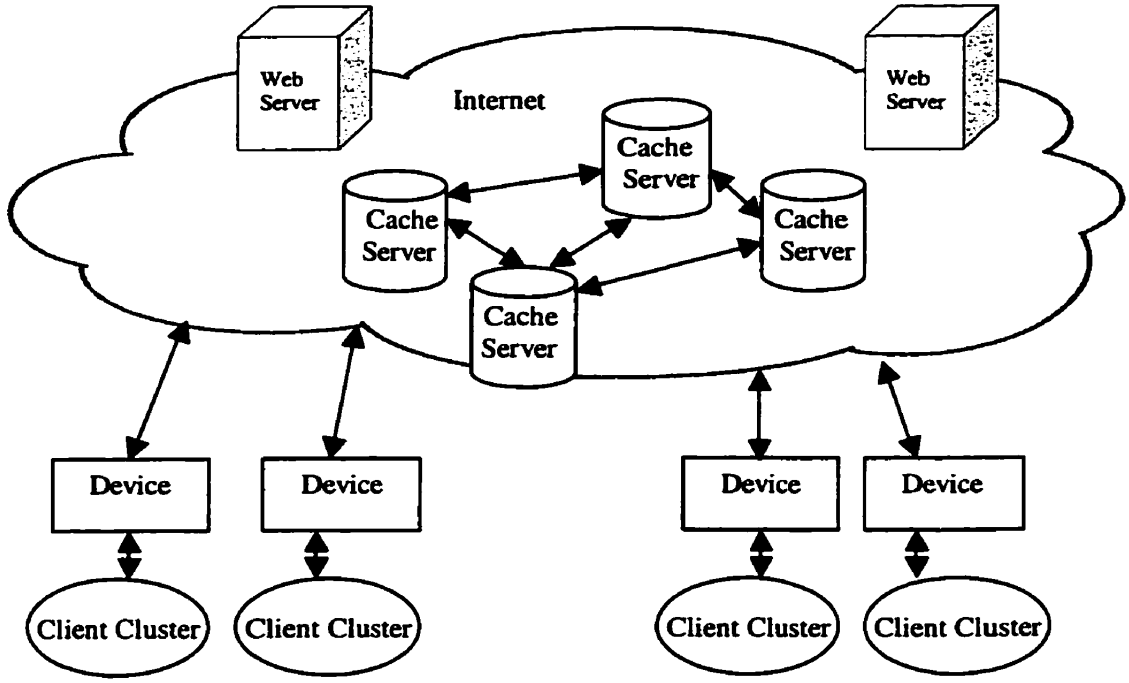


Figure 2.2 Distributed Architecture for Web Caching

Data access proceeds as follows: if a local cache server contains the data requested by a client, it sends the data to the client. Otherwise, the local cache server or a switch device redirects the client request to one of the cache servers. If that cache server has a copy of the requested object, it sends the data back to the client. Otherwise the request is redirected to the Web server. Distributed Web caching systems have several benefits

including fault tolerance, distribution of server loads and improvement of client performance by bringing cache servers closer to the client.

2.2 Client-Initiated Selection Algorithms

Client-initiated methods are a class of cache server selection algorithms in which the clients or their local proxies select the server. The algorithms are designed for a set of heterogeneous, topologically dispersed servers, whose response times depend upon both server and network effects. The HTTP request response time is the most appropriate performance metric from the users' point of view. The request response time is the sum of connection establishment time, latency and transmission time:

$$T = T_{connect} + T_{latency} + T_{remaining} \quad (2.1)$$

where, $T_{connect}$ is the time to establish a TCP/IP connection, $T_{latency}$ is from the time of sending the request to the time of receiving the first packet of the reply, and $T_{remaining}$ is the time to receive the remaining reply packets. It is not easy to measure the request response time. $T_{latency}$ is related to the network load, network propagation delays, cache server load and cache server speed. $T_{remaining}$ is determined by the size of objects. Measurement of these times may lead to substantial overhead. Different selection methods that have been used to approximate the HTTP request response time, and upon which the clients or their local caches make their selections.

2.2.1 Minimum Number of Hops

The number of hops between a client and a cache server is one common approximation of the request response time [23] [24] [25]. It can be used by clients to determine the

proximity of distributed servers. The fewer the number of hops then the smaller the distance between the cache server and the client. The number of hops can be obtained directly from the routing tables without incurring any additional network load. The approach is very simple and easy to implement.

For example, J.Guyton [24] and R.Cater *et al* [23] [25] investigate this approach under the assumption that each cache contains contents that are also held in other caches. A client sends all requests to the closest server in terms of the number of hops [24]. The problem with this approach is that the number of hops cannot reflect the varying network load. Even for homogeneous server sets with well-balanced load, response times can differ significantly because network routes between the client and the servers have different bandwidths and congestion patterns. The correlation between the number of hops and the HTTP request response time has been shown to be relatively low [26].

2.2.2 Minimum Round Trip Time

The round trip time (RTT) of the packets sent by the ping utility is another common metric for determining the proximity of distributed servers. The standard ping utility uses the Internet Control Message Protocol (ICMP) [30] to send ECHO_REQUEST datagrams to the cache server's echo port and listens for the ECHO_RESPONSE. Unlike the number of hops, the ping round trip time reflects the actual network load on the route between the client and the server.

Internet Cache Protocol (ICP) [5] and National Laboratory for Applied Network Research (NLANR) [6] use RTT. The Squid cache severs of NLANR are configured into a tree-structured hierarchy. In such a hierarchy, every participating cache server is organized with a connection of neighboring peers and a parent. ICP is used for communication. When a client's local proxy cannot service the request from its cache, it uses a set of configuration rules to determine if the Web server is local. If not, the proxy issues a set of simultaneous ICP_OP_Query messages to all its peers. When a peer receives the query message and finds that it has the requested object, it sends back ICP_OP_HIT. The client's local proxy forwards the request to the peer that responds first. If all peers reply with ICP_OP_MISS, the following three situations apply:

1) If the peers are using the ICP_FLAG_SRC_RTT feature, the request is forwarded to the peer with the lowest RTT to the origin Web server.
2) If there is a parent available, the request will be forwarded to the local cache's parent.
3) If the ICP query/reply exchange does not produce any appropriate parents, the request is sent directly to the origin Web server.

The drawback of this approach is that the ping round trip time does not provide any indication of the cache server load and the speed of the cache server. The correlation between the round trip time and the HTTP request response time is found to be slightly higher than for the Minimum Number of Hops, which is still not indicative of the request response time [26].

2.2.3 Minimum HTTP Request Latency

Under the assumption that the HTTP request response times are stable within a short period of time, the response time of a new HTTP request can be estimated from the response times of HTTP requests previously sent to the same server. However, the response time also depends on the size of the requested object, which is not known at the time the object is requested. Instead, HTTP request latency, which is the time from sending the request until the first byte of a response is received, can be used as a substitute for estimating the HTTP request response time. Unlike RTT, the HTTP request latency reflects not only the actual network load on the route between the client and the server but also the server workload and speed. Although it is independent of the size of objects it is still a reasonable predictor of the HTTP request response time because most web objects tend to be small [26].

For example, S.Dykes *et al* [27] use HTTP request latency to approximate the HTTP request response time, under the assumption that each cache contains contents that are also held in other caches. With the HTTP request latency algorithm, a client sends requests to the server with the lowest median HTTP request latency in prior transfers. The problem with this approach is that prior latency does not successfully estimate the current response time because network load and server load change all the time.

2.2.4 Hybrid Approach of Bandwidth and RTT

S.Dykes *et al* propose a hybrid approach [27] for client-side selection. They combine the RTT approach with bandwidth. First, the client selects n servers with the fastest median

bandwidth from prior transfers. Then it sends a dynamic ping to each of these servers and selects the first to reply. It immediately forwards the request to that server without waiting for replies from other servers. The bandwidth measured only reflects past cache server workload and provides no information about current cache server load.

2.3 Switch Selection Algorithms

Cache server selection can be done by a networking device such as a switch. This kind of selection algorithm is used in a distributed switching-based transparent Web caching system, which is shown in Figure 2.3.

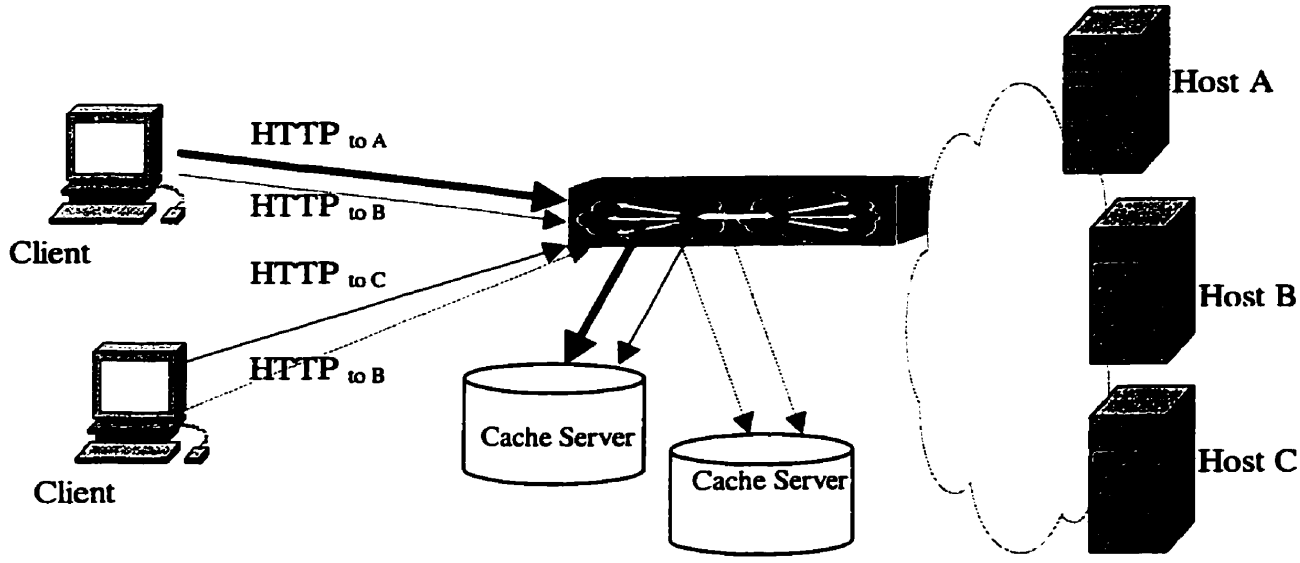


Figure 2.3 Transparent Proxy Web Caching with Redirection

A switch, running special software acting as a redirector, sits in the data path and examines all packets bound for the Internet. It sends the HTTP traffic to cache servers for processing and passes on the remaining traffic. In particular, cache servers are not

involved in the network functions, such as network address translation and routing (usually performed by cache servers in the other Web caching schemes).

The Layer at which the switch operates is determined by how much header detail the switch reads as data passes through. The switch-based redirectors may operate at Layer 4 (network level) or Layer 5 and above (application level). The redirectors that provide Layer 4 services use TCP or UDP transport layer information, e.g., port numbers found in TCP/UDP headers, in making packet-forwarding decisions. A Layer 4 switch can be configured to direct all traffic with particular destination TCP ports to a particular network port. For example, it may switch all traffic going to port 80 (used for HTTP traffic) to a particular port on the switch where a cache server is attached. More sophisticated Layer 4 switches may provide additional functionalities, such as load balancing among a cluster of caches.

Layer 5 switches [31] add the ability to use information found in the payload of HTTP request header packets. In order to obtain the HTTP request header, a Layer 5 switch sends a TCP SYN ACK message to the client and tricks it into believing that there is a TCP connection established between the client and the server. The client then sends the HTTP request to the Layer 5 switch. The information in the HTTP request can be used to provide more sophisticated capabilities. For example, the URL found in HTTP GET requests can be examined to determine whether an image is being requested. If so, all packets belonging to the TCP connection corresponding to this request can be switched to a server that is optimized to deliver images. Another common use for parsing the content

of HTTP requests at the switch is directing requests for non-cacheable content, e.g., results of a CGI script, to the original web server instead of a cache server, thus eliminating unnecessary load on the cache server. Besides content checking, Layer 5 switches can use other information metrics such as server workload and network latency to pick the best cache server to service the client HTTP request.

2.3.1 Content-Based Selection

A Layer 5 switch can make the routing decisions based on the content of the request. One example is the Arrowpoint Content SmartSwitch (CSS) [18]. On the client side CSS can be configured to redirect static HTTP requests to one cache server cluster since it can distinguish among different "higher-level" protocols, like HTTP [32] and The Secure SHell (SSH) remote login protocol [33], and divert them to the appropriate server or group of servers that service the type of requested content. The CSS also bypasses dynamic HTTP requests and redirects them to the Web server. Arrowpoint CSS makes the routing decision based on the availability and type of the content. Each cache server cluster stores one specific type of content. Inside one cache cluster, other approaches such as round robin or random or workload are needed to assist in selecting the appropriate server.

2.3.2 Workload-Based Selection

Some switches can intelligently redirect HTTP requests to lightly loaded cache servers. For example, the Extreme Networks switches [21] and Cisco CSS 11000 [22], use the following load–balancing algorithms to redirect the HTTP requests:

- **Round robin** - A simple algorithm that distributes each new connection/session to the next available server.
- **Weighted round robin** with response time as weight – An enhancement of the round robin method where response times for each server within the virtual service are constantly measured to determine which server will take the next connection/session.
- **Fewest connections** – determines which server gets the next connection by keeping a record of how many connections each server is currently providing. The server with fewest connections gets the next request.

Extreme Networks switches and the cache servers are usually deployed within a LAN, so they do not need to consider the network delay or congestion. If some servers have huge network latency to the switches, then it will lead to serious performance problems, since the network delay greatly affects the performance. Extreme Networks switches route at the Layer 2 and Layer 3 levels. They are blind to the content of the objects. Cisco CSS 11000 series switches learn where specific content resides. The server selection done by the CSS 11000 switches is based on server load and number of connections or round-robin algorithms. CSS11000 is, therefore, only suitable for a local cache cluster.

2.3.3 RTT-Based Selection

Global server selection algorithms allow mirrored servers or server farms to be distributed around the world, which enables requests to be directed to the best cache server. Switches determine the best cache server based on the cache server content, the proximity to the client and the round trip time to the cache server. An example is the

Alteon WebSystems' ACEdirector [23]. This kind of Layer 5 switch automatically exchanges the above information with all other ACEdirector Layer 5 switches. With a global view of every cache server's performance, each switch develops a list of candidate cache servers. The switches then direct traffic to cache servers in proportion to the servers' performance measurements. As a result, the best performing cache server receive more connections than others, due to their ability to handle more connections.

This approach has the same problem as client-initiated algorithms. It is difficult to measure or estimate the actual HTTP request response time. ACEdirector uses the RTT or proximity to the client as the performance measurement to approximate the HTTP request response time. However, this works only when the workload of cache servers is fairly distributed.

2.3.4 LB_L5 Selection

Z.Liang [20] proposes a fully distributed Web caching scheme that extends the capabilities of Layer 5 switching to improve the response time and balance cache server workload. In LB_L5, a Layer 5 switch selects the best server based on cache content, cache server workload, network load and the HTTP header information. If the network latency between a cache server that stores the object and the Layer 5 switch is smaller than some threshold, then that cache server is considered as a candidate for access and the Layer 5 switch uses load balancing algorithms to choose the best server from which to retrieve the object. The drawback of this approach is that it is difficult to set the threshold value. If the value is too large, then the network delay affects the performance. If the

value is too small, then the advantage of cache cooperation is lost. LB_L5, therefore, cannot guarantee the minimum request response time.

2.4 Summary

A distributed Web caching system uses a cluster of servers to provide load balancing, fault tolerance and reduced redundant copies of content. One important problem of a Web caching scheme for distributed systems is to find the best server to service the request. In this chapter we provide descriptions of several cache server selection algorithms for distributed Web caching systems. We classify the algorithms according to where the cache server selection is made.

Some researchers, like C. Yoshikawa *et al* [34], argue that the client, rather than the server or a switch, is the right place to implement transparent access to distributed network services. They believe this approach offers increased flexibility. For example, clients are aware of the relative load on a number of servers and can easily reduce the load on heavily loaded servers compared to server-initiated algorithms. Clients also do not require special network topology to do the selection. Finally, unlike a single switch, different clients do not represent a bottleneck.

We believe, however, that offloading the selection function from clients to a switch is much better. Client-side selection has several drawbacks. First, if the client browser does the selection, it requires a program such as an applet running on the client side [34]. This kind of application program has to process packets and may result in an inefficient

selection decision. Moreover, if a client local proxy does the selection, it adds load to the local cache and the local proxy becomes the single point of failure.

Compared to client selection algorithms, switching-based selection algorithms have the following advantages. First, a switch is optimized for examining and processing packets, so there is minimal impact on non-Web traffic. Second, removing the packet examination, server selection, network address translation and routing functions from the cache server frees up CPU cycles for serving Web pages. Third, using a switch redirector that is separate from the cache servers allows the client load to be dynamically spread over multiple cache servers, which, in turn, can reduce response time. Further, redundant redirectors can be deployed, eliminating any single point of failure in the system.

Our research focus is distributed Layer 5 switching-based transparent Web caching. Our research objective is to design an efficient switching-based transparent Web scheme to optimize the performance of distributed Web caching systems (minimize response time and balance the workload). Among all existing switching-based Web schemes there are no effective methods to estimate the actual HTTP request response time, while at the same time, to balance the workload among the cache server cluster for a global distributed web caching system. Some switches, such as Cisco CSS 11000 series, are only suitable for a local cache cluster and do not consider the network load. Other switches, like Alteon WebSystems' ACEdirector, use RTT to approximate the request response time and not concern the server workload. The Layer 5 switch used in the proposed LB_L5 scheme has difficulty in setting its threshold.

Chapter 3

Minimum Response Time Scheme

Layer 5 Switching-based schemes support distributed Web caching systems and can intelligently redirect a client request to the proper server (cache server or the Web server) using the content information in the HTTP header. However, these existing schemes cannot guarantee optimized performance in terms of request response time. This may result in huge request response times when the caching system is heavily loaded or the network is highly congested.

In this chapter, we present the Minimum Response Time (MRT) scheme, which is a distributed transparent load-balanced Web Caching scheme that uses the client request header, cache server content, cache server workload, Web server workload and network latency to intelligently redirect requests. The goal of our work is to optimize the performance of distributed Web caching systems to achieve the minimum response time and balanced load. Section 3.1 is an overview of the MRT scheme. Section 3.2 contains a detailed description of the MRT scheme. Finally, section 3.3 provides a summary of the chapter.

3.1 Overview of the MRT

The proposed MRT scheme is intended for response-time-sensitive Web caching systems. It is based on the switching-based transparent distributed Web caching systems, as shown is Figure 3.1. There is no communication among cache servers in the cache cluster. The cache sharing is achieved through Layer 5 switches [30]. Layer 5 switches directly connect to the Web server.

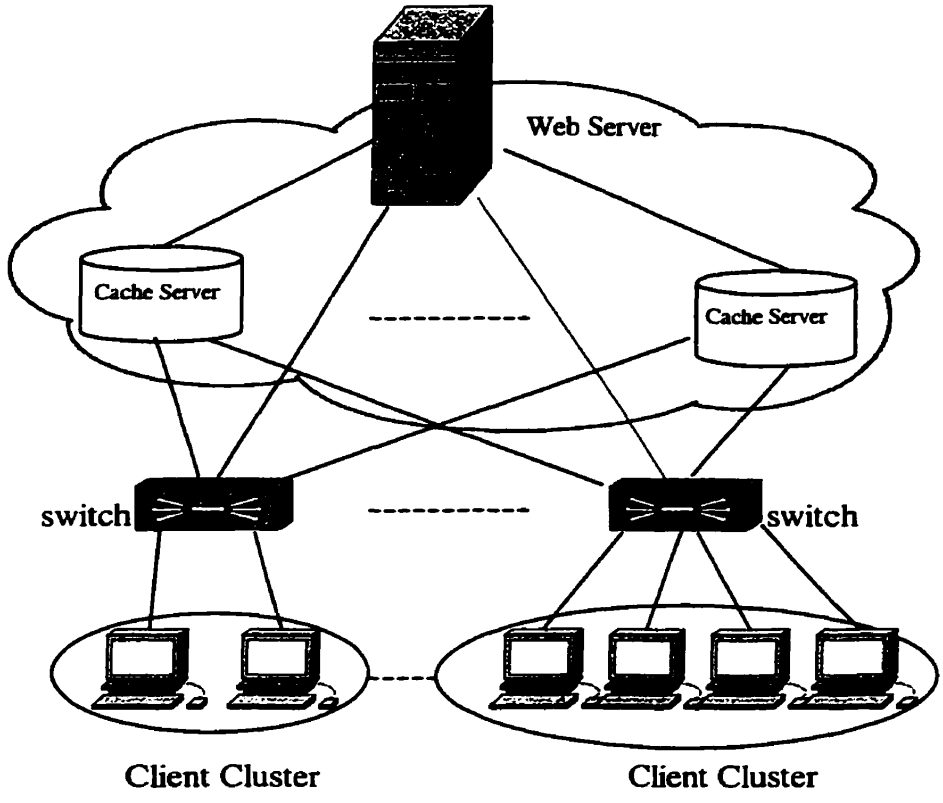


Figure 3.1 Distributed Switching-Based Transparent Web Caching System

MRT uses Layer 5 switches to check the HTTP request content. It sends the non-cacheable requests directly to the Web server. It redirects the cacheable requests to the most appropriate cache server. MRT predicts the request response time for each cache server using some information (to be described later) and chooses the cache server with

the minimum predicted response time. The MRT scheme has two components: content checking and server selection.

The following Table 3.1 summarizes all symbols used in the reminder of this chapter.

Symbol Name	Symbol Meaning
F	The size of a Bloom Filter
D	The number of objects stored in a cache sever
W	Number of hash functions. It is the number of bits to represent an object in a cache server
F_p	False hit rate
$E(RT)_{cs}$	The expected value of the http request response time if the switch sends the request to the cache server CS
P_{cs_miss}	The probability that a request is a cache-miss on the cache server CS. It is approximate to the false hit rate of CS in our discussion.
T_{cs_miss}	The request response time when the request to cache server CS is a cache-miss.
T_{cs_hit}	The request response time when the request to cache server CS is a cache-hit.
RTT_{sw_cs}	Network latency between switch SW and cache server CS.
RTT_{cs_ws}	Network latency between cache server CS and Web server WS.
Max_{cs}	Maximum Number of TCP connections cache server CS can service.
Max_{ws}	Maximum Number of TCP connections of Web server WS.
WL_{cs}	Workload of cache server CS.
WL_{ws}	Workload of the Web server.
PT_{ws}	Average processing time at Web sever WS
PT_{cs}	Average processing time at cache server CS

TABLE 3.1 SYMBOLS

3.1.1 Content Checking

MRT uses a Bloom Filter to represent cache server contents. The use of Bloom Filter to compactly represent cache server contents is proposed in Cache Digest [12] and Summary Cache [35]. Objects in a cache server can be represented by a Bloom Filter, which is an array of bits. To represent an object in a Bloom Filter, a fixed number of independent hash functions are computed for the object's key, which is the URL. The number of hash functions specifies how many bits are used to represent one object. Their hash values specify the bit positions that should be set to1 in the Bloom Filter.

A cache server in MRT can inform the switches about its content by sending them its content information in the form of a Bloom Filter. A switch stores the content information for each cache server. When a switch needs to check whether a cacheable requested object is in a cache server, it uses the same set of hash functions for the request's URL and examines the corresponding bits in the server's Bloom Filter. If all of the matching bits are 1's then the requested object is assumed to be in that cache server. Otherwise the object is not in the cache server.

If we know the size of the Bloom Filter of a cache server as F and the number of the objects stored in that cache server as D, then we can calculate the optimum number of hash functions, W, for a Bloom Filter as follows:

$$W = \frac{F \ln 2}{D} \tag{3.1}$$

The detailed proof of equation (3.1) can be found in Appendix A.

A switch in MRT determines the cacheability of a requested object using the URL information in the request's HTTP header. In this way, only the requests for cacheable objects are presented to the cache servers. Since an object may be placed at different cache servers, a Layer 5 switch may find a number of cache servers that contain the requested object. How to choose the best server to service the request is very important in a distributed Web caching system. The major part of our work is to find the best cache server and optimize the performance of a distributed Web caching system.

3.1.2 Cache Server Selection

A Layer 5 switch can use information like cache server contents, server workload and network latency to estimate the request response time for each cache server and choose the cache server with the minimum response time. A potential problem with such an approach is that the content prediction cannot always be correct, since a Bloom Filter size is not infinitely large. A request predicted to be a cache-hit might be a cache-miss. It results in the incorrect predicted request response time and the wrong cache server selection.

In our proposed MRT scheme, cache server selection is based on the expected value of response time in case of HTTP request cache-hit and in case of HTTP request cache-miss. MRT selects the cache server with the minimum expected value of a HTTP request response time. The cache server selection algorithm used in MRT has to determine three factors:

1) P_{cs_miss}, the probability that a predicted cache-hit HTTP request is a cache-miss on cache server CS.
2) The delay components for a cache-miss HTTP request, T_{cs_miss},
3) The delay components for a cache-hit HTTP request, T_{cs_hit}.

MRT then estimates the expected value of the request response time for CS as follows:

$$E(RT)_{cs} = P_{cs\_miss} * T_{cs\_miss} + (1-P_{cs\_miss}) * T_{cs\_hit} \quad (3.2)$$

False Hit Rate

The probability that a predicted cache-hit request is a cache-miss can be approximated to be the false hit rate of a cache server. This is the probability that an object is not actually stored in the cache server, when the cache server's Bloom Filter indicates it is there. We use the terms hit and miss to indicate whether the bits of the Bloom Filter predict that a given object is in the cache server or not, respectively. There are two types of hits and misses:

True hit: The Bloom Filter correctly predicts an entry is in the cache server.

False hit: The Bloom Filter incorrectly predicts the entry is in the cache server.

True miss: The Bloom Filter correctly predicts the entry is not in the cache server.

False miss: The Bloom Filter incorrectly predicts the entry is not in the cache server.

A Bloom Filter size is not infinitely large so URLs cannot be hashed to unique bits. A Bloom Filter, therefore, always has a non-zero number of false hits. The size of a Bloom Filter and the number of objects in a Bloom Filter determine the probability that lookup is correct. A smaller filter size results in higher false rate than a large one for the same number of objects. As A. Rousskov pointed out the number of false misses is negligible, while the number of false hits is relatively high [12] when a Bloom Filter is small.

In a Bloom Filter representing D objects, if each object is represented by W bits, the false hit rate is derived as [36]:

$$Fp = (1 - e^{-WD/F})^W \tag{3.3}$$

Fp is a function of F/D and W. In Figure 3.2, we plot the relationship between Fp and F/D for different values of W according to the equation (3.3).

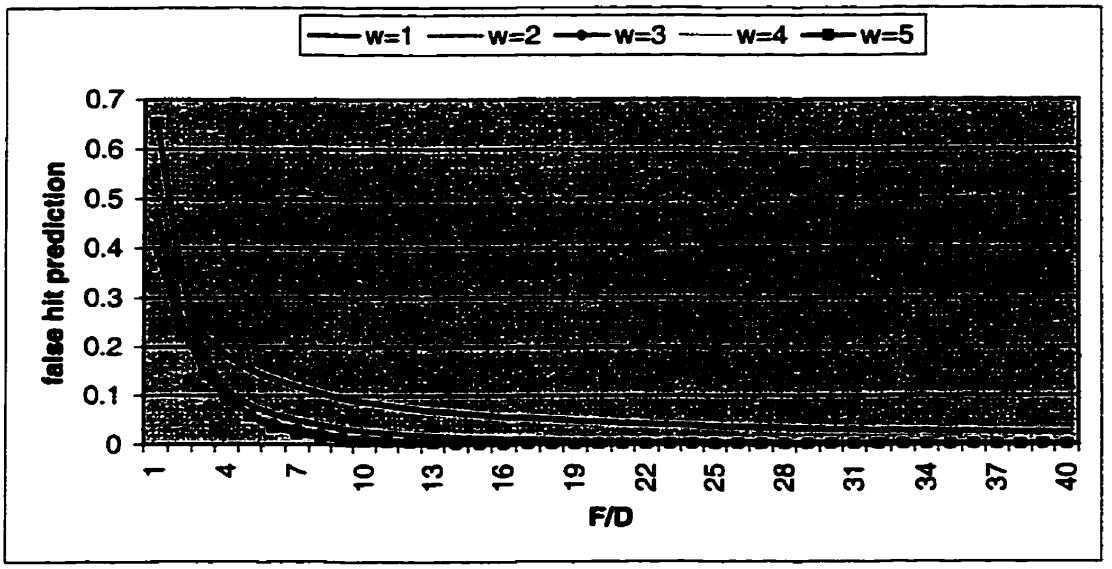


Figure 3.2 False Hit Rates

From Figure 3.2 we can see that false hit rate decreases when F/D increases. The amount of decrease is more apparent for larger values of W. When W is greater than 2 and F/D is greater than 10, the false hit rate is close to 0. If a Bloom Filters is not very large and if the F/D is smaller than 10, the false hit rate highly varies with changes in F/D. It is possible that one cache server has a low false hit rate while another has a high false hit rate. This information is very useful when a Layer 5 switch in MRT makes the routing decision. In our scheme, if an object can be cached at more than one cache server and these cache servers have similar values for workload and network link delay, the requests for an object should be directed to a server where the false hit rate is low.

HTTP Request Response Time Components

From the previous section we know that a Bloom Filter may result in false hits. When a false hit occurs the response time to retrieve the requested object will increase because the cache server has to retrieve the object from the original Web server. Since the request response time is predicted from the Layer 5 switch's point of view, we do not consider the time spent between the Web client and the Layer 5 switch in our discussion.

The basic processing procedure for a cache-hit HTTP request in MRT is illustrated in Figure 3.3. After a switch receives a HTTP request from the Web client,

(1) The switch sends a TCP-SYN signal to a Proxy Cache Server for a connection request. The Cache Server sends back a TCP_ACK to accept the connection. The time spent is the round trip time between the switch and the cache server.

(2) The switch then relays the HTTP request to the cache server. The time required is half of the round trip time between the switch and the cache server.

(3) The cache server processes the request. The time for the processing is proportional to the cache server's workload.

(4) Since the request is a cache hit, the cache server immediately relays the requested objects to the switch. The time spent is half of the round trip time between the cache server and the switch.

A cache-hit request response time is equal to the sum of the above components.

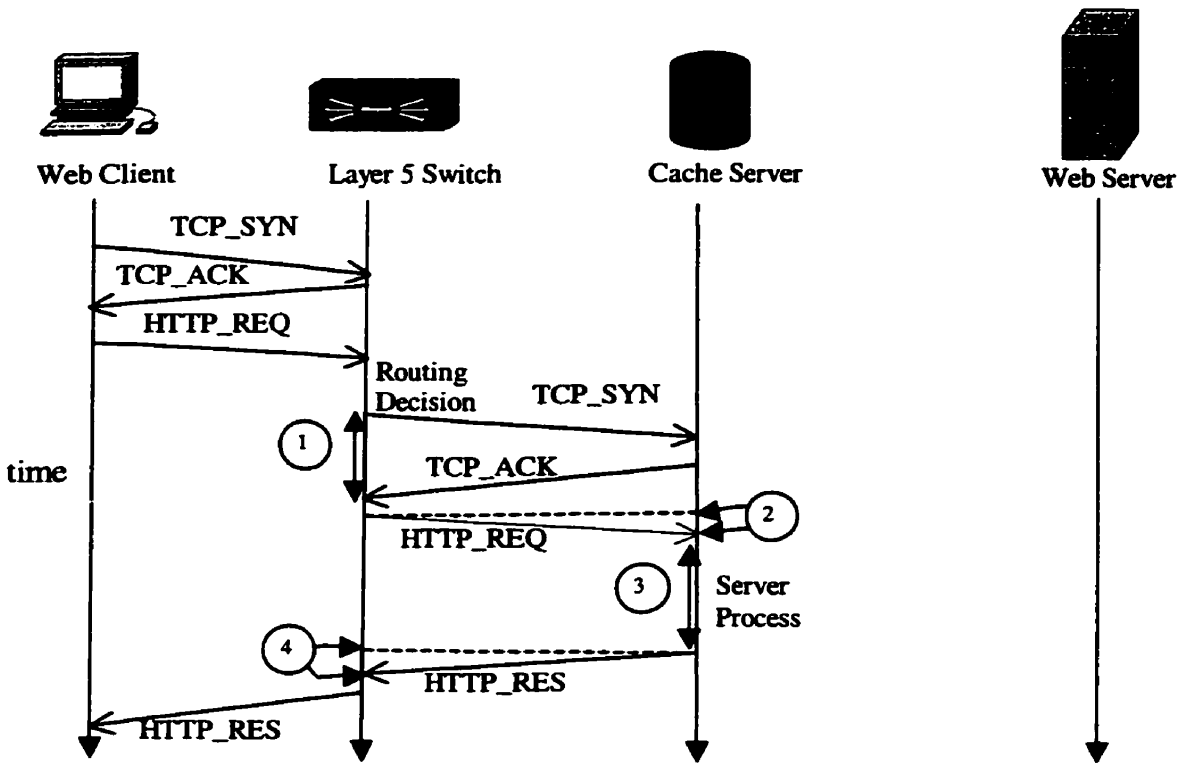


Figure 3.3 A Cache-Hit Request in MRT

The basic processing procedure for a cache-miss HTTP request in MRT is illustrated in Figure 3.4. After a switch receives a HTTP request from the Web client,

(1) The switch sends a TCP-SYN signal to a Proxy Cache Server for connection request. The Cache Server sends back TCP_ACK to accept the connection. The time spent is the round trip time between the L5 Switch and the cache server.

(2) The switch relays the HTTP request to the cache server. The time required is half of the round trip time between the L5 Switch and the cache server.

(3) The cache server processes the request. The time spent is proportional to the cache server's workload.

(4) Since the request is a cache-miss, the cache server makes a TCP connection request to the original Web server. The Web server sends back TCP_ACK to accept the connection. The time spent is the round trip time between the cache server and the Web server.

(5) The cache server then relays the HTTP request to the Web server. The time spent is half of the round trip time between the cache server and the Web server.

(6) The Web server processes the request. The time spent is proportional to the Web server workload.

(7) The Web server sends back the requested object to the cache server. The cache server receives the object and stores a copy. The time spent is half of the round trip time between cache server and the Web server.

(8) The cache server immediately relays the object to the switch. The time spent is half of the round trip time between the cache server and the switch.

A cache-miss request response time is equal to the sum of the above components.

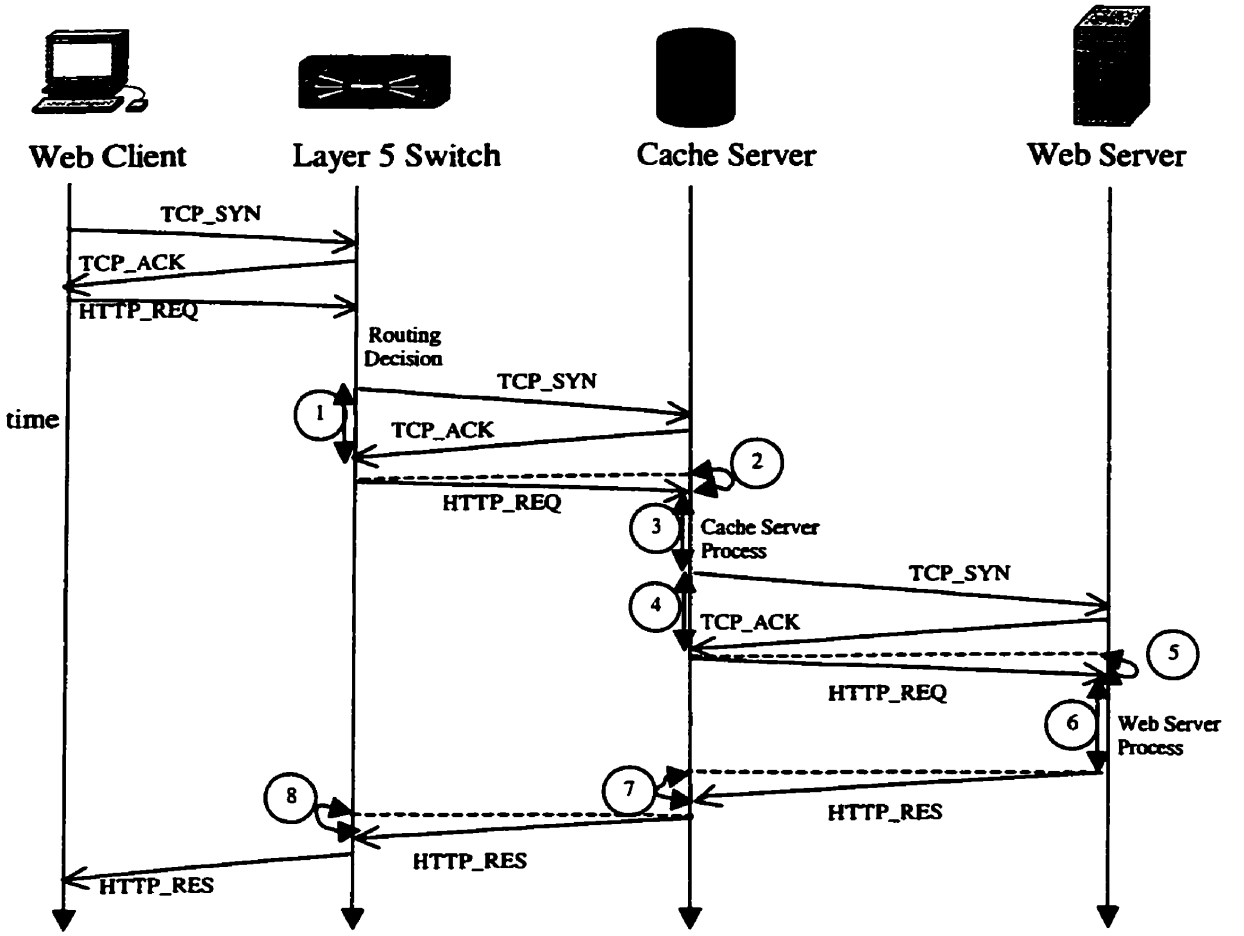


Figure 3.4 A Cache-Miss Request in MRT

We need to represent the relationship between the server (both cache server and Web server) processing time and the workload of the server. We define the workload as the number of active concurrent requests at a given time divided by the maximum number of concurrent requests that can be serviced. The server processing time, which includes the request queuing time, the time to search for the requested object and the disk access time to move the requested object from the disk to the memory, measures the *total* delay from the time a request arrives the server until the server responds. The processing time on the server is proportional to the number of concurrent requests at the proxy cache server.

This assumption is supported by the data collected by A. Rousskov [37] and also used by Z. Liang [20]. As shown in Figure 3.5, if the average time to process one request at the cache server is PT and the cache server's current workload is N/Max, then the time T needed to process the N_{th} request on the proxy cache server is:

$$T = (\frac{N}{Max}) * Max * PT \tag{3.4}$$

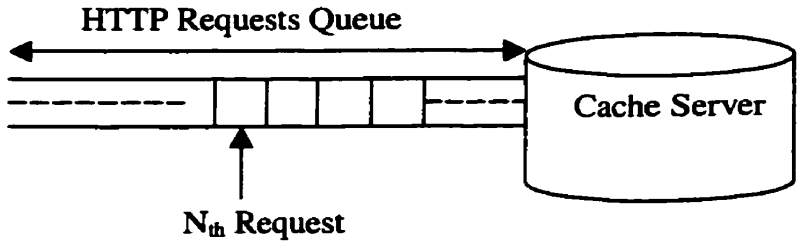


Figure 3.5 Cache Server Processing Time and Number of Concurrent Requests

A server can send its workload to a switch in the MRT periodically and the switch records the maximum number of concurrent requests for each server. The switch can then use the server workload to calculate the server processing time.

3.1.3 MRT Routing Scheme

The MRT scheme is intended for distributed architectures. Each cache server sends a representation of its content and number of objects to the switches. The switches periodically query the workload of each cache server. Web servers periodically send their workload to switches. A switch in MRT uses the HTTP request header, cache server content, cache server workload, Web server workload and network latency to route HTTP requests, and hence minimizes the average HTTP request response time and balances cache server workload.

A Layer 5 switch in MRT makes a routing decision as follows:

(1) If a request is non-cacheable, then redirect it to the Web server.

(2) If a request is cacheable then for each cache server

(2.1) If cache server CS is predicted to store the requested object, the probability that the request is a cache-miss is calculated as:

$$P_{cs\_miss} \approx Fp = (1 - e^{-WDcs / Fcs})^{W}$$

(2.2) Otherwise: $P_{cs\_miss} = 1$

(2.3) The expected response time for CS is calculated as:

$$E(RT) = P_{cs\_miss} * (2*RTT_{sw\_cs} + WL_{cs}*Max_{cs}*PT_{cs} + 2*RTT_{cs\_ws} + WL_{ws}* Max_{ws}*PT_{ws}) + (1 - P_{cs\_miss})*(2*RTT_{sw\_cs} + W_{cs}*M_{cs}*PT_{cs}) \quad (3.5)$$

(3) Select the cache server with the minimum estimated response time.

In equation (3.5), the time calculated within the first bracket represents T_{cs_miss}. $2*RTT_{sw\_cs}$ is the sum of time components (1), (2) and (8) shown in Figure 3.4. $WL_{cs}*Max_{cs}*PT_{cs}$ and $WL_{ws}* Max_{ws}*PT_{ws}$ respectively represent time components (3) and (5) shown in Figure 3.4. $2*RTT_{cs\_ws}$ is the sum of time components (4), (5) and (7) shown in Figure 3.4. The time calculated within the latter bracket represents T_{cs_hit}. $2*RTT_{sw\_cs}$ is the sum of time components (1), (2) and (4) shown in Figure 3.3. $WL_{cs}*Max_{cs}*PT_{cs}$ represent time component (3) shown in Figure 3.3.

3.2 Detailed Description of MRT Scheme

This section provides a detailed description of the MRT scheme. It includes the extended ICP messages used between the switch and the cache server and the extended table used by the switch for routing decision, as well as the routing mechanism of the MRT scheme.

3.2.1 Extended ICP Messages

As in LB_L5 [20], we use four extended ICP (Internet Cache Protocol) messages for exchanging content and workload information between the switch and the proxy cache in MRT scheme. A single extended ICP message is sent periodically by the Web server to the Layer 5 switch with its workload.

The ICP [11] message format consists of a 20-byte fixed header plus a variable sized payload, as shown in Figure 3.6.

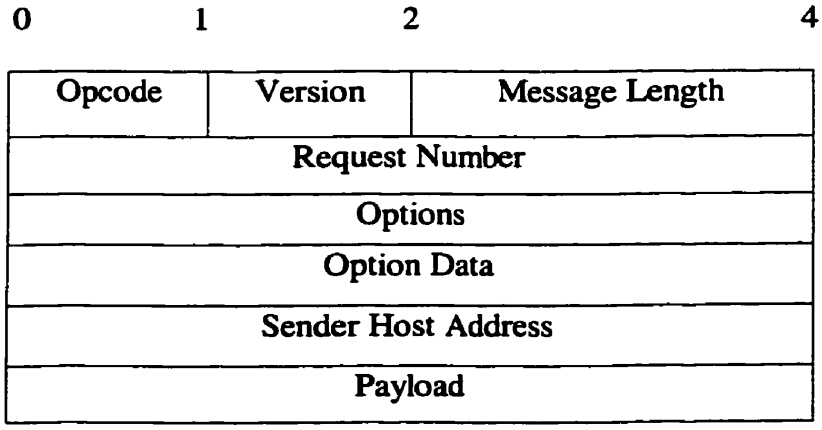


Figure 3.6 ICP Message Format

Opcode specifies the type of an ICP message. Table 3.2 shows currently defined ICP opcodes in ICP version2 [11]:

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 3.2 ICP OPCODE

From table 3.2 we see that there are some unused Opcodes, so our four new ICP messages use these Opcodes. The four messages are following:

ICP_UPDATE_CONTENT message is used by a cache server to periodically inform a switch about the changes in its cache contents and the changes in the number of stored objects in that cache server. The format of the message is shown in Figure 3.7. The Sender Address is the IP address of the cache server sending the message. The content of the Payload field has three parts: 1) a Bloom Filter, which represents the cache server contents. 2) a count to record the number of objects that are stored in the sender cache server. 3) A timestamp, which is used for error control.

Opcode		Sender Address	Payload		
ICP_UPDATE_CONTENT	...	IP Address	Bloom Filter	Count	TimeStamp

Figure 3.7 The Format of ICP_UPDATE_CONTENT

ICP_UPDATE_CONTENT_ACK message is used by a switch in MRT to acknowledge the ICP_UPDATE_CONTENT message. The format of the message is shown in Figure 3.8. The sender Address is the IP address of the switch that sends this message. The

content of the Payload field includes a timestamp, which is used with the timestamp in the ICP_UPDATE_CONTENT to deal with the situation when the ICP_UPDATE_CONTENT message is lost in the network.

Opcode		Sender Address	Payload
ICP_UPDATE_CONTENT_ACK	...	IP Address	TimeStamp

Figure 3.8 The Format of ICP_UPDATE_CONTENT_ACK

ICP_QUERY_WORKLOAD is used by a switch to periodically query the workload of cache servers. The format of the message is shown in Figure 3.9. The sender Address is the IP address of a switch. The content of the Payload field contains a timestamp, which is used to deal with lost messages.

Opcode		Sender Address	Payload
ICP_QUERY_WORKLOAD	...	IP Address	Time Stamp

Figure 3.9 The Format of ICP_QUERY_WORKLOAD

ICP_UPDATE_WORKLOAD is used by a cache server to send its workload information to the switch after it receives the ICP_Query_Workload message. It can also be used by a Web server to periodically send its workload information to the switches. The format of the message is shown in Figure 3.10. The Sender Address is the IP address of a cache server sending the message. The content of the Payload consists of two parts: 1) workload of the cache server and 2) a timestamp, which is used, along with the timestamp in the ICP_QUERY_WORKLOAD to deal with lost messages.

Opcode		Sender Address	Payload	
ICP_UPDATE_WORKLOAD	...	IP Address	Workload	TimeStamp

Figure 3.10 The Format of ICP_UPDATE_WORKLOAD

3.2.2 Extended Information Table

In MRT, we use an extended information table in a Layer 5 switch to assist the switch in making the routing decision. Figure 3.11 below illustrates the format of the table. Each entry in the table consists ten fields:

IP_ Add	Bloom Filter	Count	Workload	Network_ Latency	Max_Co nnection	Workload_ QueryTime	Workload_ QueryRes_ Time	Last_Content _UpdateMsg _TS	Workload_ Query_TS

Figure 3.11 CacheServerArray Information Table

- IP_Add: IP address of a cache server. It is also an identification of the cache server.
- BloomFilter: The representation of contents of the cache server. It is updated periodically.
- Count: The number of objects stored in that cache server. It is updated periodically.
- Workload: The workload of the cache server. It is updated periodically.
- NetworkLatency: Half of the round trip time between the switch and the cache server. It is updated periodically.
- Max_Connection: The max number of concurrent TCP connections of the cache server.
- Workload_QueryTime: The time measured in milliseconds when the switch sends the query workload message.
- Workload_QueryRes_Time: The time measured in milliseconds when the switch receives the updated workload.

- Last_Content_UpdateMsg_TS: The Timestamp of the most recent content update. It is used for message loss control.
- Workload_Query_TS: The timestamp when the switch sends the query workload message. It is used for message loss control.

3.2.3 Routing Mechanism of MRT

In the MRT scheme, every time a Layer 5 switch receives a HTTP request from the Web client, it uses the HTTP header, cache server contents, cache server workload, Web server load and network latency to route the request. For cacheable requests, the switch calculates the expected value of the request response time for each cache server based on the above information stored in the switch's CacheArrayTable. It sends the request to the cache server with the minimum estimated request response time. A detailed pseudo code and explanation of the operation of the switch and cache server can be found in Appendix B. Following we explain how a switch in MRT obtains and updates all the information needed for routing:

- HTTP header information:

  When a switch receives a TCP connection request from a Web client, it accepts the connection by sending back a TCP_ACK prior to establishing the TCP connection with the server. Thus it tricks the Web client into believing there is a connection between the client and the server. When the switch receives a HTTP request from the client, it unpacks the request message to get the HTTP header information. Using the URL, the switch knows whether the request is cacheable or non-cacheable.

- Cache Server Content information:

  Each cache server computes a representation of its cache content information. It periodically multicasts the Bloom Filter and the number of objects to every switch with the extended ICP message ICP_UPDATE_CONTENT. When a switch receives an ICP_UPDATE_CONTENT message, it finds the sending cache server in the switch's CacheArrayTable, as shown in Figure 3.11, and updates that cache server's content and the number of objects. After the update, it sends an ICP_UPDATE_CONTENT_ACK back to the cache server to acknowledge the update.

- Cache Server Workload Information:

  A switch obtains the workload information from a cache server by periodically sending each cache server an ICP_QUERY_WORKLOAD message. When a cache server receives an ICP_QUERY_WORKLOAD message, it measures its current workload and replies to the switch with an ICP_UPDATE_WORKLOAD message, whose payload field carries that cache server's most recent workload information. When the switch receives an ICP_UPDATE_WORKLOAD, it finds the sending cache server in the switch's CacheArrayTable, as shown in Figure 3.11, and updates that cache server's workload.

- Web Server Workload Information:

  The Web server periodically sends its current workload to every switch with an ICP_UPDATE_WORKLOAD message.

- Network Latency Information:

  The duration from the time a switch sends an ICP_QUERY_WORKLOAD to a cache server to the time the switch receives an ICP_UPDATE_WORKLOAD from that cache server is measured as the network latency between the switch and the cache server. Whenever a switch receives an ICP_UPDATE_WORKLOAD message from a cache server, it updates both the cache server workload and the network latency in its CacheArrayTable, as shown in Figure 3.11, for the corresponding cache server.

- False hit rate information:

  Every time a switch makes a routing decision, it calculates the false hit rate for each cache server based on the size of the Bloom Filter and the number of objects on the cache server.

3.3 Summary

In this chapter, we provide a detailed description of the operation of the MRT scheme. The MRT scheme is a network latency sensitive, dynamical load-balancing cache server selection protocol that is intended for distributed and transparent Web caching systems. The desirable characteristics of MRT are the following:

- The most prominent characteristic of MRT is that it can avoid redirecting the requests to remote cache servers. At the same time it tries to balance the workload on all cache servers. MRT can adjust its routing decision dynamically based on the cost of network latency, the cost of the workload and the cost of false hits to optimize the performance of the whole Web caching systems.

- MRT is completely distributed. In such a fully distributed architecture a single point of failure can be avoided and the cache sharing can be achieved by allowing more clients to share multiple cache servers.

- The MRT server selection algorithm is done by a Layer 5 switch compared to other server selection algorithms that are done by the client or cache servers. Layer 5 switches can perform the routing very efficiently.

- MRT extends ICP protocols and is compatible with existing Web caching systems. The extended ICP messages allow the switch and cache server in the MRT scheme to cooperate with switches and cache servers that do not support MRT.

Chapter 4

Performance Evaluation

In this chapter, the performance of the MRT scheme is studied. The results are analyzed and compared with the Content, Workload, RTT and LB_L5 schemes. We chose those schemes for comparison because they represent the range of cache sever selection algorithms used in distributed switching-based transparent Web caching systems. Section 4.1 describes the simulation model, which includes the network model, the network latency model, the workload model, the invalidation-checking model and the simulation parameter settings. Section 4.2 describes how the Content, Workload, RTT and LB_L5 schemes work in our simulation. Section 4.3 presents the performance metrics that we use to represent the performance of the different schemes. The effects of the network latency, request intensity, expiration time and the number of cooperating cache servers on the performance of the MRT are investigated and the results are reported in Section 4.4. Finally, a summary of this chapter is provided in section 4.5.

4.1 Simulation Model

To evaluate the MRT scheme and to compare it with other schemes, we construct a simulator that allows us to observe and measure the performance of the schemes under different network latencies, different HTTP request intensities, different values of object expiration time and different number of cache servers. The simulator consists of a network model, a network latency model, a server workload model and a cache server content validation checking model.

4.1.1 Network Model

The network model for each of the five Web caching schemes simulates a fully distributed switching-based Web caching system, as shown in Figure 4.1. A client cluster connects to its own Layer 5 switch, which connects to all the cache servers and the Web server. A cache server connects to all the switches and the Web server. There are no direct connections between cache servers and there is no direct connection between a client cluster and a cache server. The cache sharing is achieved through the Layer 5 switches.

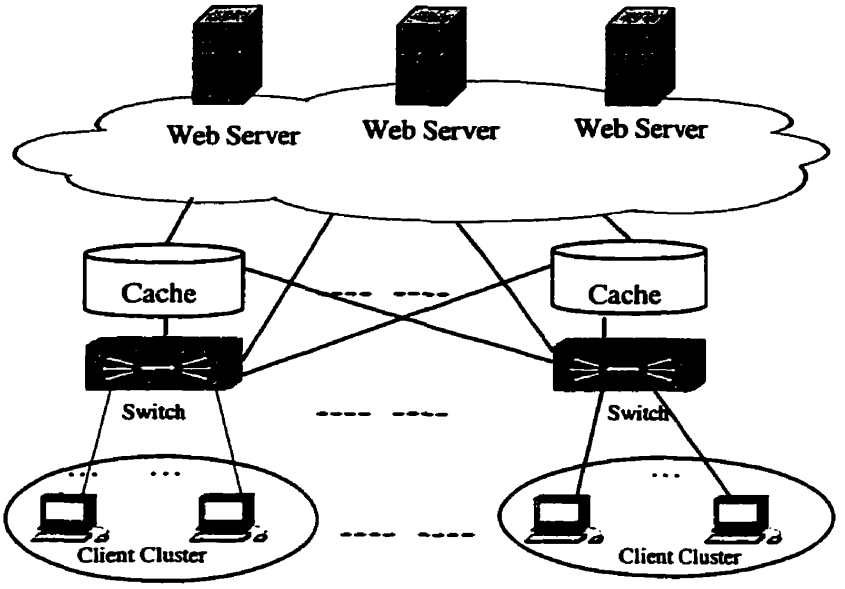


Figure 4.1 Network Model

4.1.2 Network Latency Model

It is difficult to measure the network latency in a simulation. M. Rabinovich *et al* [38] use a distance metric to reflect the costs of transferring data between any two nodes. The cost may include monetary costs, time, bandwidth of the connection, etc. The cost of transferring data between two nodes is proportional to the distance between that pair of nodes. We adopt a similar method in our simulation. Our network latency model is based on a symmetric architecture with n client clusters, n switches and n cache servers, as shown in Figure 4.2. In our simulation, n varies from 2 to 8.

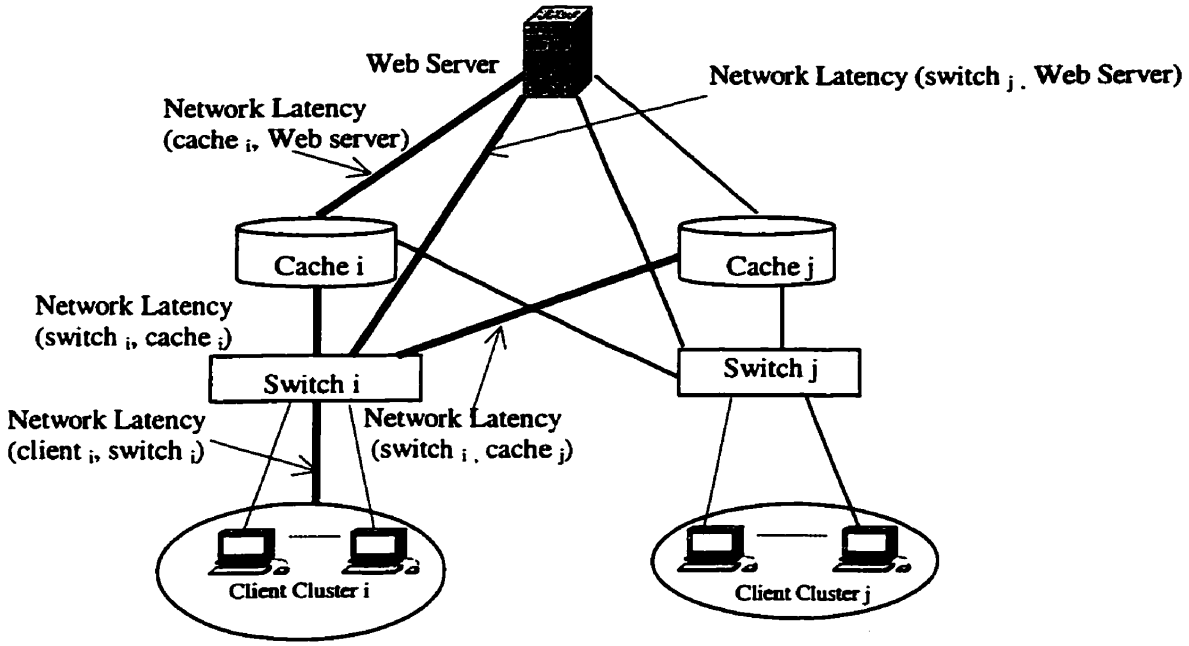


Figure 4.2 Network Latency Model

We calculate the network latency between node i and node j, which is the cost of transferring data between them as follows:

$$Distance\ (i, j) = |i-j| \tag{4.1}$$

$$NetworkLatency\ (i, j) = Distance\ (i, j) * LatencyFactor \tag{4.2}$$

NetworkLatency (i,j):The time spent on the network when data are transferred from node i to node j, which is measured in milliseconds.

LatencyFactor: The time spent on the network when data are transferred for one unit of distance. It is measured in milliseconds.

In order to investigate the effect of the network latency between the cache servers and the switches on the performance of all the schemes, we vary the LatencyFactor from 5 milliseconds to 125 milliseconds in our simulation and the results are studied in Section 4.4.

4.1.3 Workload Model

The research community has made progress towards characterizing workload patterns for Web servers and proxy cache servers using Benchmarks, such as those by Almeida and Cao [39] and Barford and Crovella [40]. Proxy traces can also provide an alternative means of workload generation that is able to account for client access patterns and the requested content [2].

We use publicly available proxy traces from the NLANR to generate HTTP requests [2]. In our simulation we use the July 26, 2001 and August 27, 2001 traces from the NLANR Boulder cache servers. Client IP addresses are randomized daily and are consistent within a trace but not between traces. Each trace spans 24 hours and contains from 100,000 to 400,000 total requests. Each entry in the trace has 9 fields. In our simulation the TimeStamp, ClientAddress, Size and URL fields are used. They are defined as follows:

Timestamp: specifies the time when the client generates the HTTP request. The format is "Unix time" with millisecond resolution

Client Address: The IP address of the client cluster

Size: The number of bytes transferred from the proxy to the client

URL: The uniform resource locator, which is a character string describing the location and access method of a resource on the Internet.

In raw proxy traces HTTP requests are generated at the time specified by the TimeStamp field. The controlled proxy traces vary the request inter arrival times to simulate the different HTTP request intensities. For example, if in the raw trace the average number of HTTP requests generated by the client clusters is 2000 per 10 minutes, the average inter

arrival time is: (10*60*1000) / 2000 milliseconds. In controlled proxy traces, for a 50% request intensity, we can enlarge the average interval by 2 and for a 200% request intensity, we can shorten the average interval by 2.

We measure the average workload of a server (cache server or the Web server) in our simulation as follows:

$$AverageWorkload = \frac{\text{Average Num of TCP connections Per Second}}{\text{Maximum Num of TCP connections of the Server Per Second}} \quad (4.3)$$

The average number of TCP connections is calculated as follows:

$$AvgTCPNum = (1-W_q) * AvgTCPNum + W_q * TCPNum \quad (4.4)$$

AvgTCPNum: The average number of TCP connections per second.

TCPNum: The active number of TCP connections.

W_q: A weight factor, $0 < W_q < 1$.

If W_q is too large, then the averaging procedure may not filter out transient congestion or busty traffic. If W_q is too low, then the average number of TCP connections responds too slowly to changes in the actual number of TCP connections. Many researchers use 0.002 as the value for W_q [41]. We ran in our simulation with W_q as 0.001, 0.002 and 0.25 and found that $W_q = 0.002$ is a reasonable choice. The average number of TCP connections in our simulation is therefore calculated as:

$$AvgTCPNum = 0.998 * AvgTCPNum + 0.002 * TCPNum \quad (4.5)$$

4.1.4 Validation Checking Model

Cache servers sometimes provide users with stale pages, which are out of date with respect to the copies on the Web servers. If a page is stale then the HTTP request is

redirected to the original Web server for the refreshed object. In order to reduce the time wasted on stale pages, a cache server must validate the pages in its cache so that it can give refresh pages to users. All known Web caching systems use validation check mechanisms to maintain cache consistency [2][43][44]. Some widely used proxy servers, such as the World Wide Web Consortium's httpd (formerly CERN httpd) [43] and Netscape's Proxy Server [44] - use an *expiration mechanism* to keep their pages up to date.

The expiration mechanism works as follows. Each cached page is assigned an expiration time. Any GET requests made before the expiration time elapses are answered with the cached page. After the expiration time elapses, the cache server directs the GET request to the Web server. After receiving the Web server's response, the cache server resets the page's expiration time to its default value. The detailed pseudo code for W3C httpd 3.0 cache consistency can be found in Appendix C.

Expiration-based caches use a variety of mechanisms to assign expiration times for cached pages. The field "**Expires: date**" in the HTTP request header, means that the document definitely will not change before the given date and probably will change soon after the given date. The Web server may return such a header with a document. It can be used directly as an expiration time for cached copies of the document. However, as Glassman [45] points out, the header **Expires: date** is rarely used. This is not surprising, since the author of a WWW page usually can't estimate a document's lifetime at the time it is created.

For documents with no **Expires: date** header, the simplest expiration time algorithm used by the cache server is to assign each object an expiration time equal to be ¼ to ½ of the average lifetime of that object [46]. The average lifetime of an object is an average interval between two successive modifications of the same URL. The average lifetime of an object is affected by the type of the object, such as HTML, GIF or AUDIO. It is also affected by the class to which the object belongs, such as the COM class, the EDU class or the NEWS class (The average lifetime in NEWS classes usually is 1 ~2 days). We assume that all objects studied in our simulation belong to the NEWS classes. We varied the expiration time from 3 hours to 24 hours to investigate the effect of different expiration times on the performance of the investigated Web caching schemes.

The implementation of the simulation software is based on the simulator used by Z. Liang [20]. We add the validation checking mechanism to the proxy cache server. The Web server supports ICP messages. We implement the layer 5 switches used in the Content, Workload, RTT and MRT schemes. The detailed software structure of the simulator is described in Appendix D.

4.1.6 Simulation Parameter Setting

The parameter settings used in the simulations are similar to the values used by Z. Liang [20]. The parameters are summarized in the following tables:

Name	Meaning	Value (milliseconds)
RTT_{wc-sw}	The round trip time between a Web client and a L5 switch	130
RTT_{wc-pc}	The round trip time between a Web client and a Cache server	130
RTT_{sw-pc}	The round trip time between a L5 switch and a Cache server	0~400
RTT_{sw-ws}	The round trip time between a L5 switch and Web server	300
RTT_{pc-ws}	The round trip time between a Cache server and Web server	300

TABLE 4.1 PARAMETERS FOR LINK

Name	Meaning	Value (milliseconds)
QueryWorkload_Interval	The interval between the QueryWorkload msgs	1000
ECHO_Interval	The interval between the Echo msgs	1000
TCP_Splicing	The time it takes a L5 switch port controller to translate TCP sequence number	0
Routing	The time it takes a L 5switch to make a routing decision	10
CheckCacheable	The time it takes a L5 switch to check the requested object is cacheable or not	5

TABLE 4.2 PARAMETERS FOR SWITCHES

Name	Meaning	Value (milliseconds)
WS_Processing	The time at a Web server between receiving a request and returning the first byte of the requested object	150
WS_Reply	The time it takes a Web server to send an object in memory to the requesting party	150
UpdateWorkload_Interval	The interval to send the updated workload	60*1000

TABLE 4.3 PARAMETERS FOR WEB SERVER

Name	Meaning	Value
PC_CacheSize	The physical size of a Cache server	1024*1024*64(bytes)
PC_Search	The time it takes a Cache server to search for an object in its cache	250(milliseconds)
PC_SearchDigest	The time it takes a Cache to search for an object in its cache digests	100(milliseconds)
PC_DiskAccess	The time it takes a Cache server to retrieve a cached object from disk to memory	100(milliseconds)
PC_Reply	The time it takes a Cache server to send an object in memory to the requesting party	150(milliseconds)
PC_Relay	The time it takes Cache server to relay a response to the requesting party	50(milliseconds)
PC_CacheDigest_Size	The size of a Bloom Filter for a Cache server	32*1024 (bytes)
PC_CacheDigest_Interval	The interval between consecutive content_update msgs	1*60*1000(milliseconds)
Objects expirationTime	The time for an object to expire	(3~24)*60*60*1000
PC_Latency_Threshold	The network latency threshold for LB_L5	100 milliseconds

TABLE 4.4 PARAMETERS FOR CACHE SERVERS

4.2 Content, Workload, RTT and LB_L5 Schemes

The Content scheme, Workload scheme, RTT Scheme and LB_L5 Scheme in our simulation represent certain classes of cache server selection algorithms that are introduced in chapter 2. The schemes are simulated as follows.

4.2.1 The Content Scheme

The Content scheme represents a class of cache server selection algorithms that redirect HTTP requests only based on content. When a Layer 5 switch in the Content scheme receives a HTTP request, it makes the routing decision as follows:

(1) If the request is non cacheable, the switch immediately redirects it to the Web server.

(2) If the request is cacheable then the switch checks the content of each cache server.

(2.1) If a set of the cache servers is predicted to store the object, the switch randomly picks one cache server from these cache servers.

(2.2) If none of the cache servers is predicted to store the object, the switch picks the cache server in a round robin manner.

4.2.2 The Workload Scheme

The Workload scheme represents a class of cache server selection algorithms that redirects HTTP requests based on both content and workload of the cache server. When a Layer 5 switch in the Workload scheme receives a HTTP request, it makes the routing decision as follows:

(1) If the request is non-cacheable, the switch immediately redirects it to the Web server.

(2) If the request is cacheable then the switch checks the content of each cache server.

(2.1) If a set of the cache servers is predicted to store the object, the switch picks the cache server with the minimum workload among these cache servers.

(2.2) If none of the cache servers is predicted to store the object, the switch picks the cache server with the minimum workload among all the cache servers.

4.2.3 The RTT Scheme

The RTT scheme represents a class of cache server selection algorithms that redirects HTTP requests based on both content and network latency. When a Layer 5 switch in the RTT scheme receives a HTTP request, it makes the routing decision as follows:

(1) If the request is non-cacheable, the switch immediately redirects it to the Web server.

(2) If the request is cacheable then the switch checks the content of each cache server.

(2.1) If a set of the cache servers is predicted to store the object, the switch picks the cache server with the smallest response time among these cache servers.

(2.2) If none of the cache servers store the object, the switch picks the cache server with the minimum response time to the switch among all cache servers.

4.2.4 The LB_L5 Scheme

The LB_L5 scheme represents a class of cache server selection algorithms that redirects HTTP requests based on the content, the cache server workload and the network latency. When a Layer 5 switch in the LB_L5 scheme receives a HTTP request, it makes the routing decision as follows:

(1) If the request is non cacheable, the switch immediately redirects it to the Web server.

(2) If the request is cacheable then the switch checks the content of each cache server.

(2.1) If a cache server potentially stores the object and the network latency from the cache server to the switch is not greater than some threshold, then the switch picks the cache server with minimum workload from these cache servers that satisfy the above conditions.

(2.2) If all the cache servers potentially store the object have the network latency to the switch greater than the threshold, then the switch picks its local cache server regardless whether that cache server has the requested object or not.

(2.3) If none of the cache servers store the object, the switch picks the cache server with the minimum workload from those cache severs whose network latency to the switch is not greater than the threshold.

4.3 Performance Metrics

The following two key performance metrics are evaluated:

1. Client Request Response Time

The duration from the time a client sends a TCP connection request to the time the client receives the TCP connection finished signal. It is affected by the workload of the cache servers and the Web server, the network latency and false prediction. The client's perception of Web performance is based on the response time. The smaller the average response time, the better the performance.

2. Average Cache Server Workload

A cache server may be shared by hundreds or thousands of users. At any given time, there are a number of concurrent HTTP requests to the cache server. Object retrieval times on the cache server vary with load, and consequently the request

response time varies as well. We define the load on the cache server as the average number of TCP connections per second divided by the maximum number of TCP connections per second that each cache server can service. The average workload can be used to indicate the relative balance among the cluster of cache servers.

The average response time indicates the quality of the schemes. The efficiency of any Web caching scheme can be measured in terms of how the workload is balanced across the cache servers. A balanced workload usually means lower response times, but sometimes the two performance metrics conflict. For example, to achieve workload balance a request has to be redirected to a remote cache server at the expense of higher response times. The ultimate goal of a Web caching system remains to minimize response times.

4.4 Simulation Results

In this section we describe both raw-trace driven and controlled-trace simulation experiments. For the raw-trace driven simulations, the HTTP requests are generated at the time specified by the timestamp field in the trace file. In the controlled-trace simulations, the HTTP requests are generated periodically and the request inter arrival time is controlled in order to vary the HTTP request intensity. All experiments are done with a 30-minute warm-up period to fill cache servers with objects until all cache servers are full and the simulated Web caching system is stable. A statistical analysis of the experiments results reveals that the performance is quite stable. The experiments were run with a 90% confidence level with 5% confidence intervals.

4.4.1 Raw-trace Driven Simulations

We ran experiments with two raw traces from proxy servers using NLANR traces [2]. In order to simulate the cache cooperation among different network domains we built four different network domains for four different client clusters based on client IP addresses. Each trace is for a 24 hours period. We compare the response times of MRT to the other schemes under different network latencies and HTTP request intensities during one day. When the network latency is low the workload of cache servers is the main factor affecting the response time. When the workload of the cache server is light, the network latency becomes the main factor affecting the response time. The workload of a cache server is determined by the number of HTTP requests generated by client clusters. Figure 4.3 plots the HTTP request intensity over one day versus the number of HTTP requests in10-minute periods. The request intensity is relatively high from 8 am to 4 pm. The peak intensity is 2950 requests per 10 minutes around 11 am, and significantly decreases after 5 pm. The minimum intensity is 950 requests per 10 minutes around 8 pm.

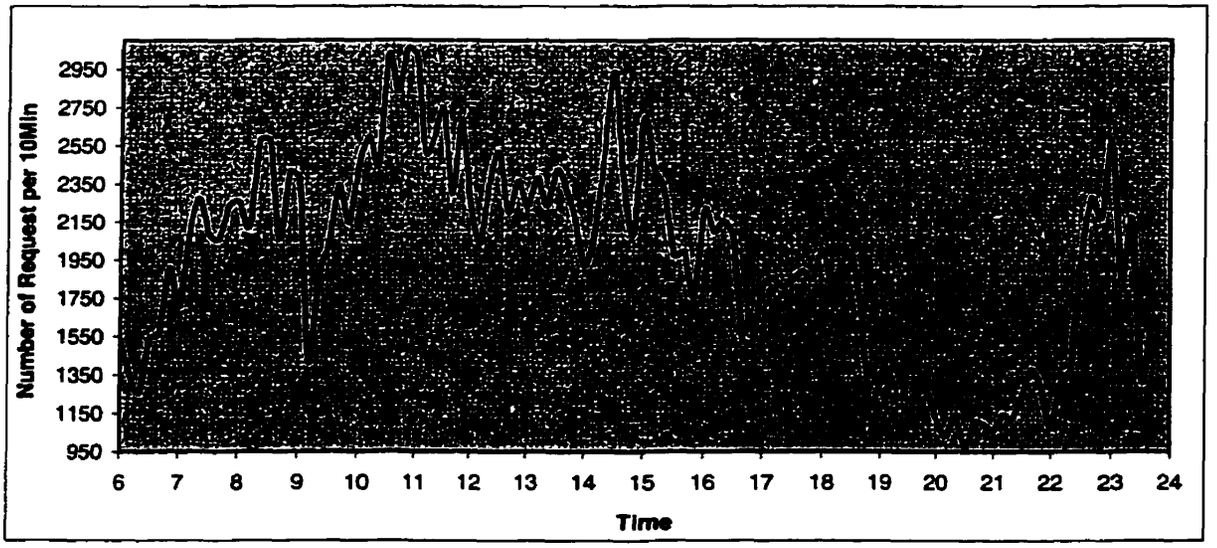


Figure 4.3 HTTP Request Intensity

The response times over one day under different network latencies are shown in Figures 4.4-4.6 below. The simulation results show that MRT outperforms the other four schemes and has a better adaptability to high HTTP request intensities and large network latencies. When the network latency is very small, the response times of five schemes follow the HTTP request intensity, as shown in Figure 4.4. The Workload scheme, LB_L5 scheme and MRT scheme consider the workload at the cache servers so they have better performance than the Content and the RTT schemes when the request intensity is high. In Figure 4.4, under a peak request intensity (10:50 am), MRT outperforms Content by 22% and RTT by 25%. MRT has similar performance to the Workload and LB_L5 schemes.

Under a higher network latency (Latency Factor = 75 ms) as shown in Figure 4.5, the average response time of MRT is much better than that of Content, Workload, RTT and LB_L5. It outperforms Content by 27%, Workload by 17%, RTT by 20% and LB_L5 by 13%. As the network latency increases, the corresponding increase in the average response times of Content, Workload and LB_L5 schemes is higher than that of RTT and MRT since the latter schemes respond better to high network latency factor.

When the network latency is very large (Latency Factor = 125ms) as shown in Figure 4.6, both LB_L5 and MRT avoid redirecting requests to remote cache servers. LB_L5 does not redirect the requests to cache servers whose network latency to the switch is larger than the predefined threshold (100 milliseconds in our simulation). MRT always redirects the requests to the cache server with the minimum estimated request response time. MRT

outperforms LB_L5 by 11%, while it outperforms Content by 44% and Workload by 34% since neither of them considers network latency when they make the routing decision. The average response time of RTT does not increase as much as that of Content or Workload. However, when all the cache servers that may store the requested object have large network latency, RTT has to redirect the request to one of them. The average response time of RTT increases as the network latency increases. MRT outperforms RTT by 30%.

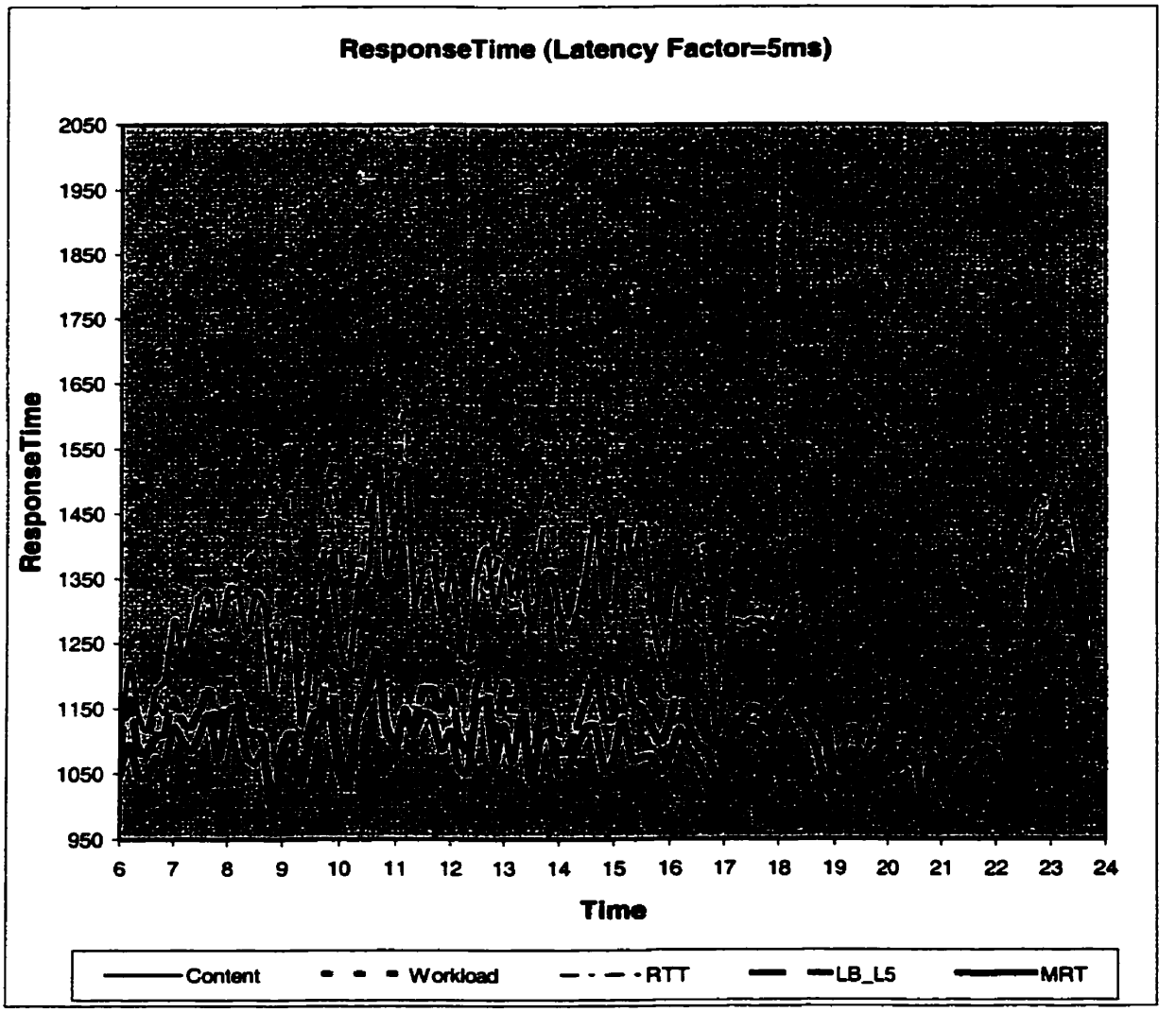


Figure 4.4 Response Time At Latency Factor = 5ms

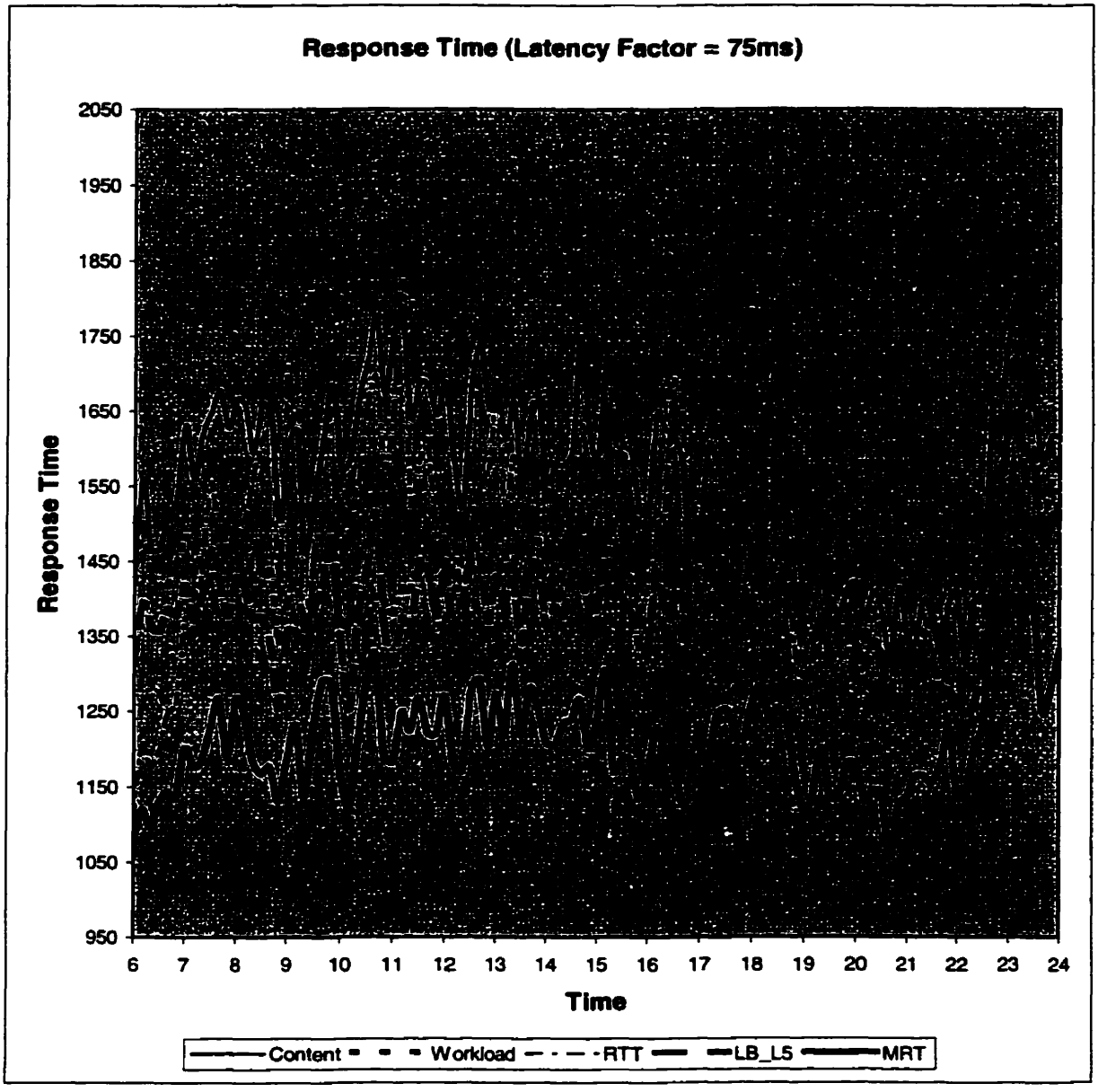


Figure 4.5 Response Time At Latency Factor = 75ms

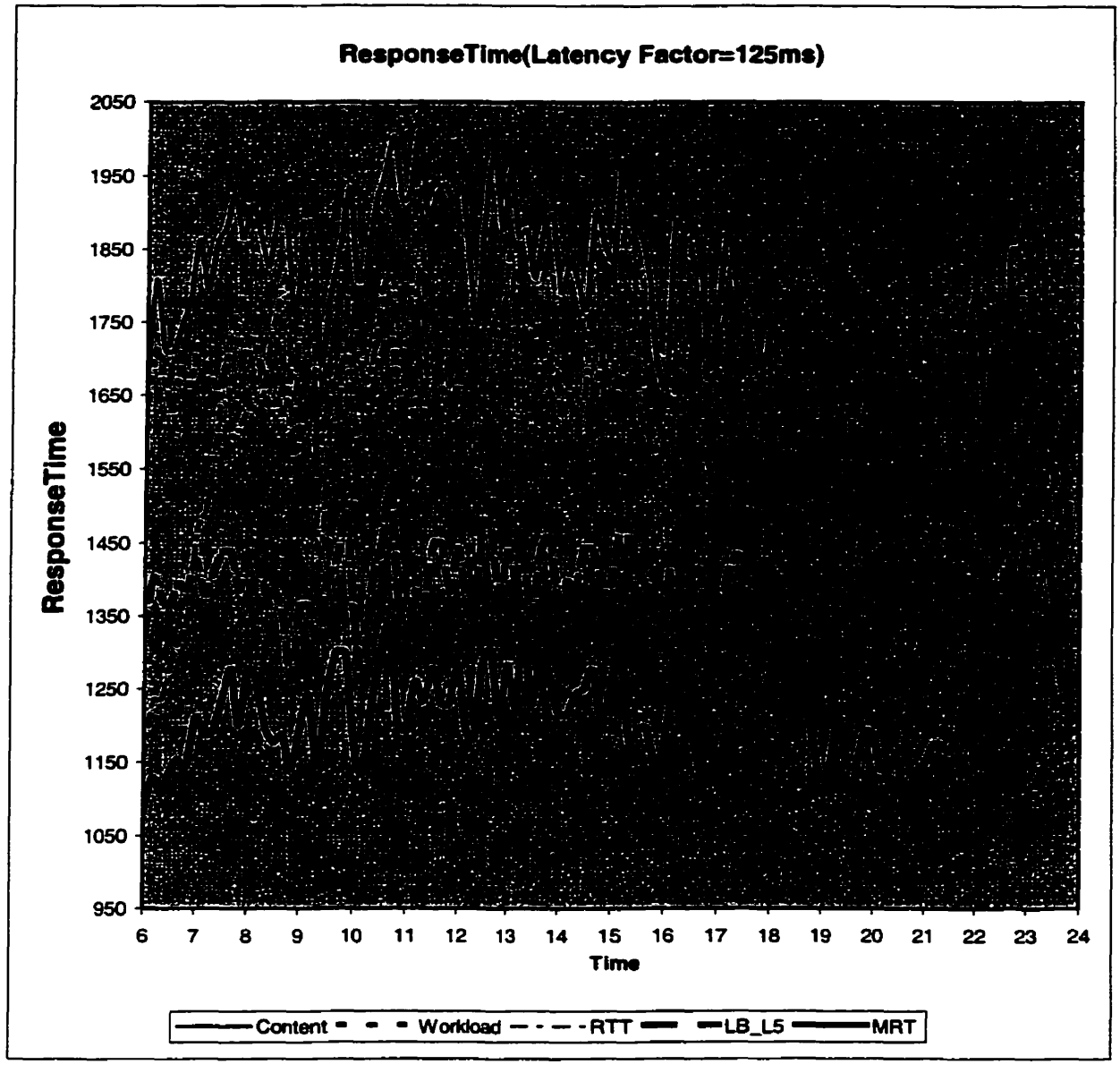


Figure 4.6 Response Time At Latency Factor = 125 ms

4.4.2 Controlled-Trace With Balanced Requests

In this section, we present experiments using controlled traces. We investigate the effect of the HTTP request intensity, the network latency and object expiration time on the response time of MRT. The results are compared with the performance of the Content, Workload, RTT and LB_L5 schemes. All experiments are done under the condition that each client cluster sends the same amount of http requests per 10-minute period. We run the simulator 8 to 24 hours and sample the request response time every 10 minutes and present the average of the values with a 90% confidence level.

4.4.2.1 Effect of Http Request Intensity

Figure 4.7 plots the response time versus HTTP request intensity under different network latency factors. As shown in Figure 4.7, we can see that MRT always outperforms the other schemes under different request intensities and network latency factors. The average HTTP request response time of all investigated schemes increases as the HTTP request intensity increases for all network latency factors. This is because the workload of each cache server increases as the HTTP intensity increases. The processing time of cache servers increases too. The extent of the increase on response time for each scheme is different.

The Content and RTT schemes are greatly affected by the HTTP request intensity since neither of them uses the workload information about cache servers when they make the routing decision. Content selects the cache server that is picked randomly or in a round robin fashion. RTT selects the cache server that stores the object and has the minimum

network latency to the switch. They may cause unbalanced load distribution on those cache servers, although originally client-clusters have balanced requests. The imbalance is more obvious when the HTTP request intensity increases.

The Workload, LB_L5 and MRT schemes adapt better to high request intensities because they are aware of the workload of all cache servers. These schemes always try to direct requests to lightly loaded cache servers. The request response times of Workload, LB_L5 and MRT are less than that of Content and RTT, since the processing time on a lightly loaded cache server is less than that on a heavily loaded cache server. The improvement on response time is more apparent when the request intensity increases, as shown in Figure 4.7. The average increase of response time is around 1400 millisecond for both Content and RTT when the HTTP request intensity increases from 50% to 250%, while the average increase of response time is only around 700 milliseconds for Workload, LB_L5 and MRT.

We also see that the performance advantage of MRT is more obvious as the network latency factor increases. For example, in Figure 4.7 (a) with 50% HTTP request intensity and 5 milliseconds network latency factor, MRT performs similarly to the Content, Workload and RTT schemes. On the other hand, when the network latency factor is 125 milliseconds, as shown in Figure 4.7 (f), MRT outperforms Content, Workload and RTT by 52%, 51% and 28%, respectively. For LB_L5 the situation is a little different. When the network latency factor is greater than 100 milliseconds, a switch redirects the HTTP requests to its local cache server. The average response time of LB_L5 does not increase

when the network latency factor is greater than 100 milliseconds. The average response time of MRT increases slowly as the network latency factor increases. As shown in Figure 4.7 (a) to (f), under 50% HTTP request intensity, when the network latency factor is 30, 50, 75, 100, 125 milliseconds, MRT outperforms LB_L5 by 7%, 15.5%, 20%, 21.2% and 19.2%, respectively.

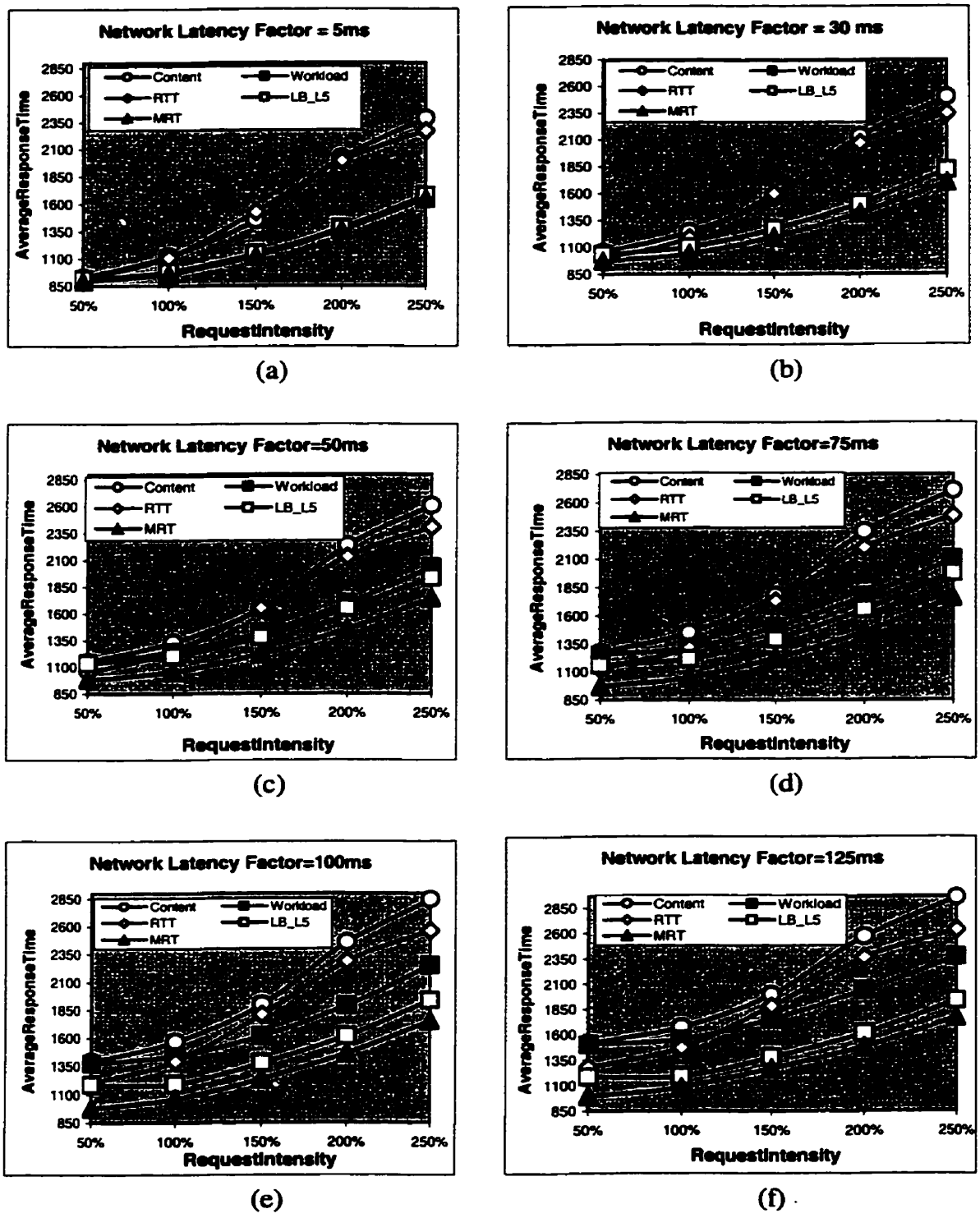


Figure 4.7 Average Response Time Versus HTTP Request Intensity

4.4.2.2 Effect of Network Latency

Figure 4.8 illustrates the average response time versus network latency factor. The experiments are conducted under different HTTP request intensities. MRT outperforms the other schemes for all values of network latency and for each HTTP request intensity. Basically, the average request response time for all investigated schemes increases as the network latency factor increases under different HTTP request intensities. However, the extent of the response time increase for each scheme is different.

The Content and Workload schemes are highly affected by the network latency since both of them do not consider the network latency when they make the selection decision. As we know, in the Content scheme, the cache server for a request is picked randomly or in a round robin fashion. In the Workload scheme, the selected cache server is the one that has the minimum workload. In both cases, if the network latency from the selected cache server to the switch is large, then there can be a noticeable increase in the average response time. As shown Figure 4.8, the average response time of Content and Workload respectively increases by 20% and 30% as the network latency factor increases from 5 to 125 milliseconds with 250% HTTP request intensity. The increase is more apparent when the HTTP request intensity is small. The average response time of Content and Workload respectively increases by 62% and 64% as the network latency factor increases from 5 to 125 milliseconds with 50% HTTP request intensity.

The RTT scheme knows the up-to-date network latency. It tries to redirect the requests to the cache server with minimum network latency, so the amount of increase in the average response time is smaller than in the Workload and the Content schemes. We found,

however, the average response time of RTT is highly affected by the network latency factor. If the cache servers that potentially store the requested object have huge network latency to the switch, the average response time increases as the network latency increases. As shown in Figure 4.8 the average response time of RTT increases by 40% and 14% as the network latency factor increases from 5 to 125 milliseconds for 50% and 250% HTTP request intensities, respectively.

For LB_L5, when the network latency factor is small, the selected cache server is the one that has the minimum workload among the servers storing the requested object. It is similar to the Workload scheme under small network latency, and the average response time highly depends on the network latency. In Figure 4.8 (a) we can see that the average response time increases around 30% when the network latency factor increases from 5 to 100 milliseconds. However, if the network latency factor is very large, the switch does not redirect requests to remote cache servers. Instead, it redirects the requests to the local cache servers no matter if it stores the requested object or not and no matter if it has the minimum workload or not. The average response time remains the same when the network latency factor increases further.

MRT knows the most recent information about the network latency and server workload. Using this information, MRT can predict the request response time and automatically adjusts the cost incurred by the workload and network latency. It chooses the server that can service the request fastest. As shown in Figure 4.8, we can see that the response time increases modest with the increase of network latency. The increase in the response time

is only about 11% and 7% as the network latency factor increases from 5 to 125 milliseconds for 50% and 250% HTTP request intensity, respectively.

MRT has a better performance when the Web caching system has high request intensity or large network latency. When the request intensity is 50% and the network latency factor is 5 milliseconds the average response time of MRT is similar to the Content, Workload, RTT and LB_L5 schemes. However, when the request intensity is 250% and the network latency factor is 125 milliseconds, the average response time of MRT is lower than that of Content, Workload, RTT and LB_L5 by 70%, 34%, 49% and 9%, respectively.

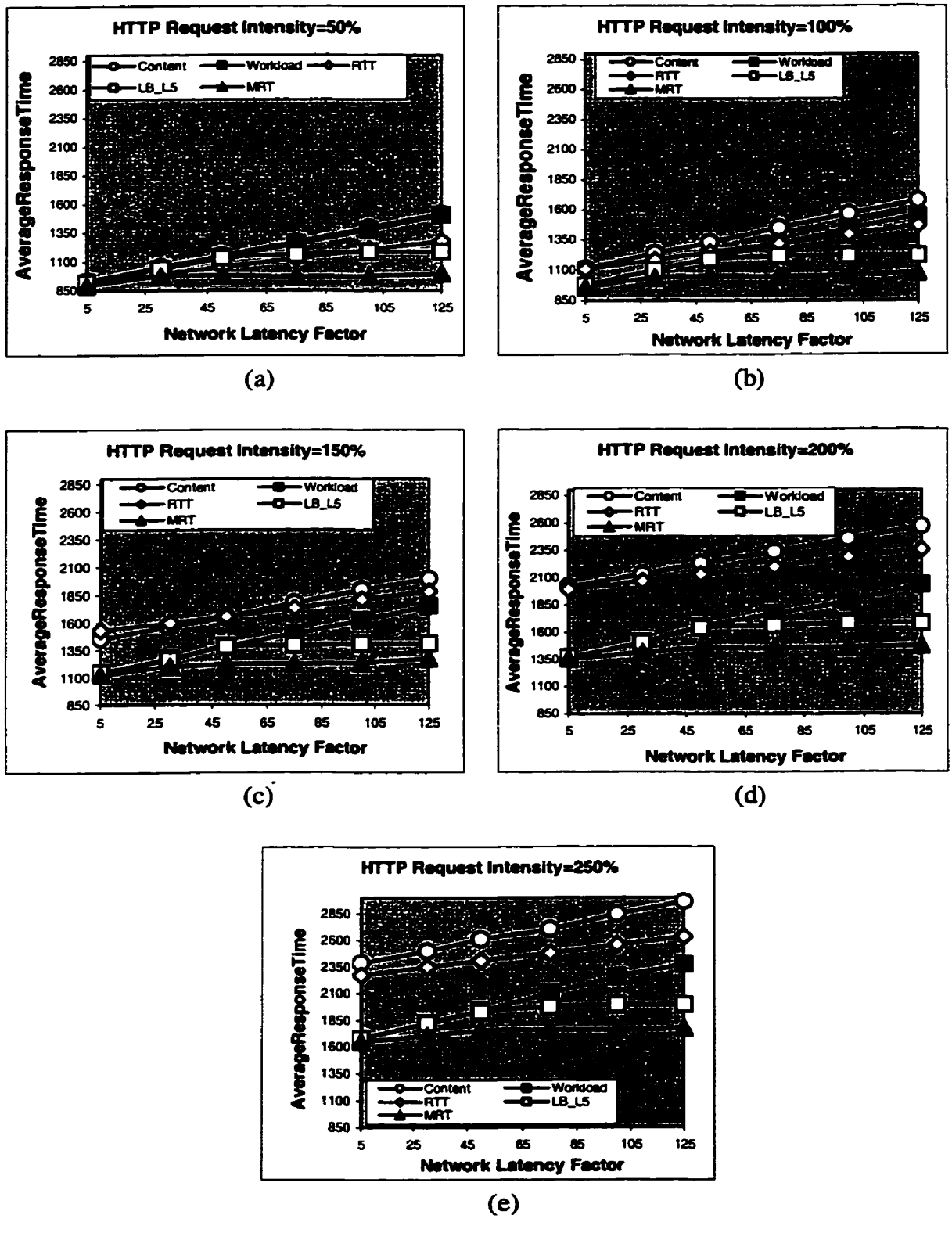
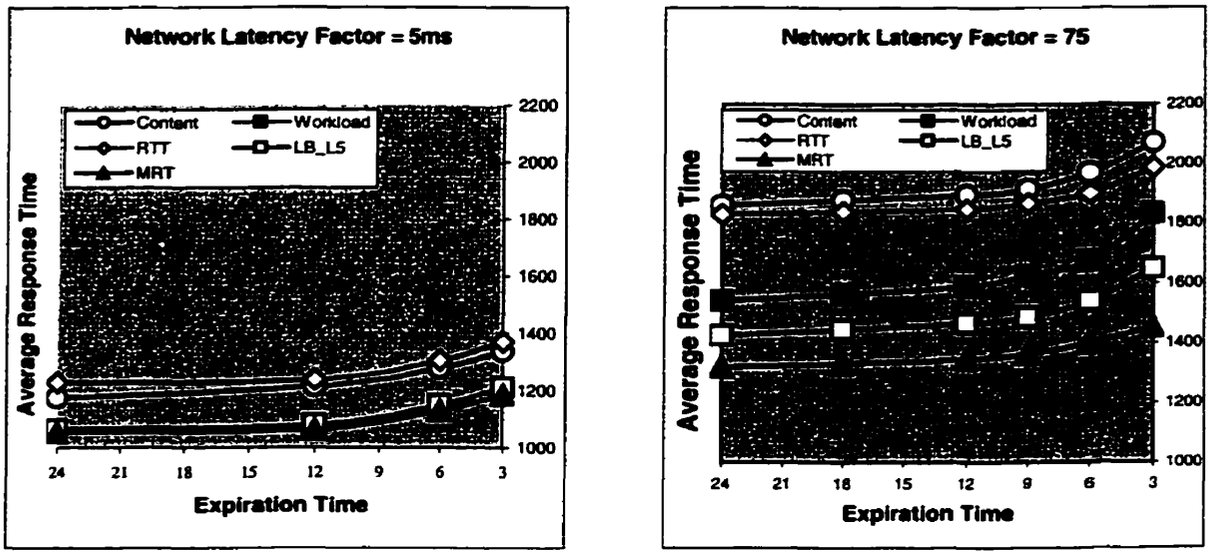


Figure 4.8 Average Response Time Versus Network Latency Factor

4.4.2.3 Effect of Object Expiration Time

The expiration time is the period during which the object is assumed valid in a cache server. If an object is out of date, it has to be retrieved from the original Web server. This can result in a huge increase in the request response time. Proxy cache servers in our simulation use the expiration time mechanism to do validation checking. In this way, the object need only be retrieved once from the original Web server within its expiration time.

Figure 4.9 illustrates the response time versus object expiration time. The experiments are done under 100% HTTP request intensity. The shorter the expiration time, the more often validation checking is performed. As shown in Figure 4.9, the average request response times of all investigated schemes increase as the object expiration time decreases. In Figure 4.9 (a), for a small latency factor (5ms), when the object expiration time decreases from 24 hours to 3 hours, the average response times of Content, Workload, RTT, LB_L5 and MRT increase by 13.5%, 14%, 11.5%, 14% and 12%, respectively. In Figure 4.9 (b), for a larger latency factor (75ms), when the object expiration time decreases from 24 hours to 3 hours, the average response time of Content, Workload, RTT, LB_L5 and MRT increases by 12%, 19%, 8%, 16% and 10%, respectively. It should be noted that the relative increase in average response time for al schemes is not significantly affected by the latency factor.



(a) (b)

Figure 4.9 Average Response Time versus Expiration Time

4.4.3 Controlled-Trace With Unbalanced Requests

In this section, we present experiments using controlled traces with unbalanced client request intensities. We investigate the effect of unbalanced workload and the network latency on the average response time of MRT. The results are compared with the performance of the Content, Workload, RTT and LB_L5 schemes. The experiments are conducted for 4 client clusters where client clusters 1 and 3 have 50% HTTP request intensity, while client clusters 2 and 4 have150% HTTP request intensity. We run the simulator 8 to 24 hours and sample the request response time every 10 minutes.

Figure 4.10 (a.1) to (e.1) plot the average workload of the four tested cache servers under different network latency factors for each scheme. Figure 4.10 (a.2) to (e.2) plot the average request response time versus the network latency factor for each scheme. The Content scheme selects the cache server in a round robin fashion. It may balance the

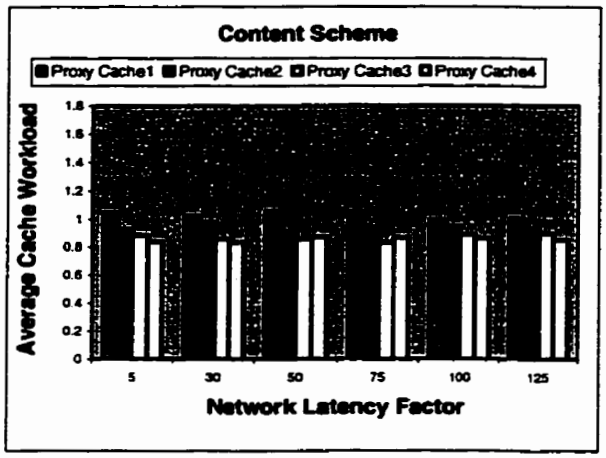
workload on all cache servers, as shown in Figure 4.10 (a.1). The Workload scheme balances the workload of the four cache servers for all network latency factors. Workload moves some load from the heavily loaded cache servers to the lightly loaded cache servers so that the workload on all cache servers is balanced, as shown in Figure 4.10 (b.1). This reduces the processing time on the heavily loaded cache servers. When the network latency is low, the communication cost to other cache servers is low and the balanced workload on the cache servers can reduce the average response time. For example, in Figure 4.10 (b.2) the average response time is only 1146 milliseconds when the network latency factor is 5 milliseconds. When the network latency factor increases, the communication cost to the remote cache servers increases, so the benefit from the balanced workload on each cache server cannot compensate for the communication cost. The average response time increases rapidly as the network latency increases. The average response time increases up to 1724 milliseconds when the network latency factor is 125 milliseconds.

The RTT scheme cannot balance the workload of the four cache servers since it does not use the cache servers' workload information and always redirects the request to the cache server that may store the object and has the minimum network latency. In Figure 4.10 (c.1) we observe that cache servers 2 and 4 are heavily loaded for all network latency factors. The queuing time on these two cache servers is very large, which means that the average response time is large. Figure 10 (c.2) shows the average response time is 1373 milliseconds when the network latency factor is 5 milliseconds, which is 17% more than that of Workload scheme. The average response time of RTT increases slower than that

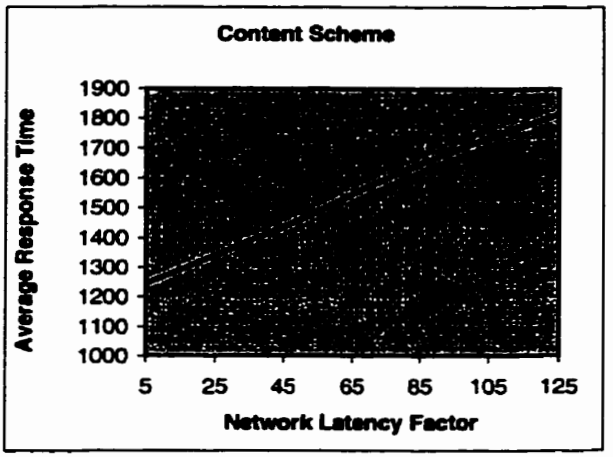
of Workload as the network latency factor increases, since it mainly depends on the network latency. The average response time is 1700 milliseconds when the network latency factor is 125 milliseconds, which is almost the same as the Workload scheme.

The LB_L5 scheme balances the workload of cache servers under small network latency factors. Figure 4.10 (d.2) shows the average response time is only 1146 milliseconds when the network latency factor is 5 milliseconds. When the network latency is greater than 100 milliseconds (the LB_L5 threshold), LB_L5 redirects the request only to its own cache server, so the cache server sharing capability and load balancing capability are lost. The average response time is 1339 milliseconds when the network latency factor is 125 milliseconds, which is 17% higher than that at a network latency factor of 5 milliseconds.

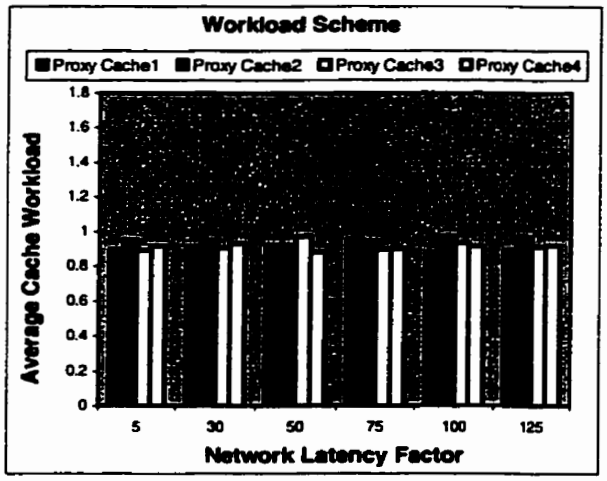
The MRT scheme balances the workload of cache servers for small network latency. The balancing capability decreases, as the network latency increases. MRT calculates the queuing cost and the communication cost. In Figure 4.10 (e.1) we see that MRT's workload balancing capability is better than that of LB_L5 under large network latency. The rate of the unbalanced workload of LB_L5 on cache server 3 and 4 reaches 300%, while that of MRT on cache server 3 and 4 is only about 100%. In Figure 4.10 (e.2) we can see the MRT's rate of increase of average response time is slower than that of LB_L5 and RTT. Although the workload balancing capability of MRT is not better than Workload or Content, the average response time of MRT is much better than that of the Workload and Content schemes.



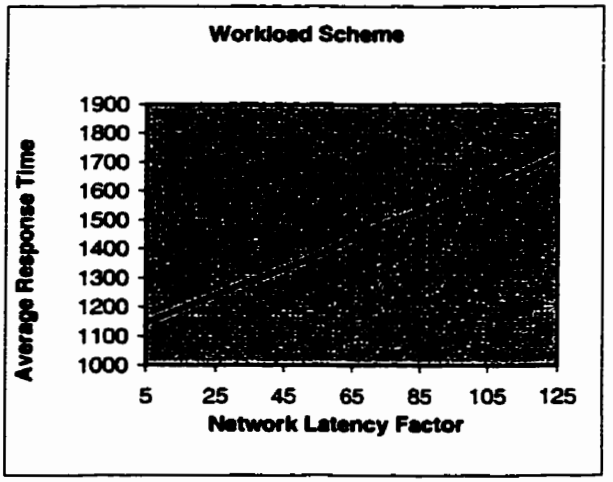
(a.1)



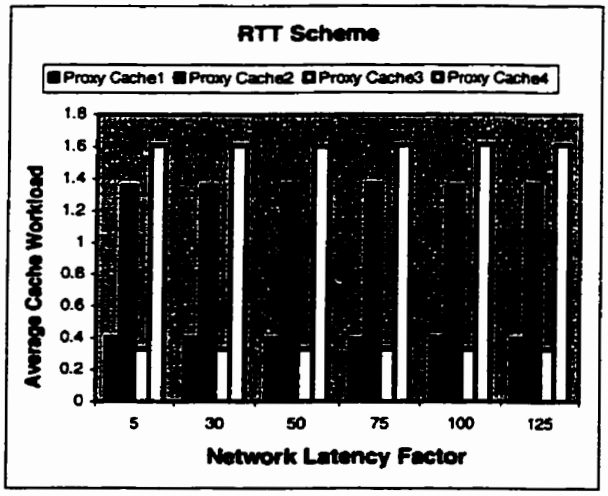
(a.2)



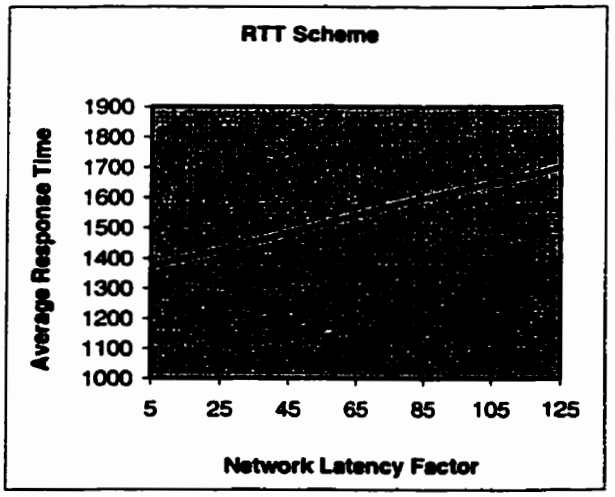
(b.1)



(b.2)



(c.1)



(c.2)

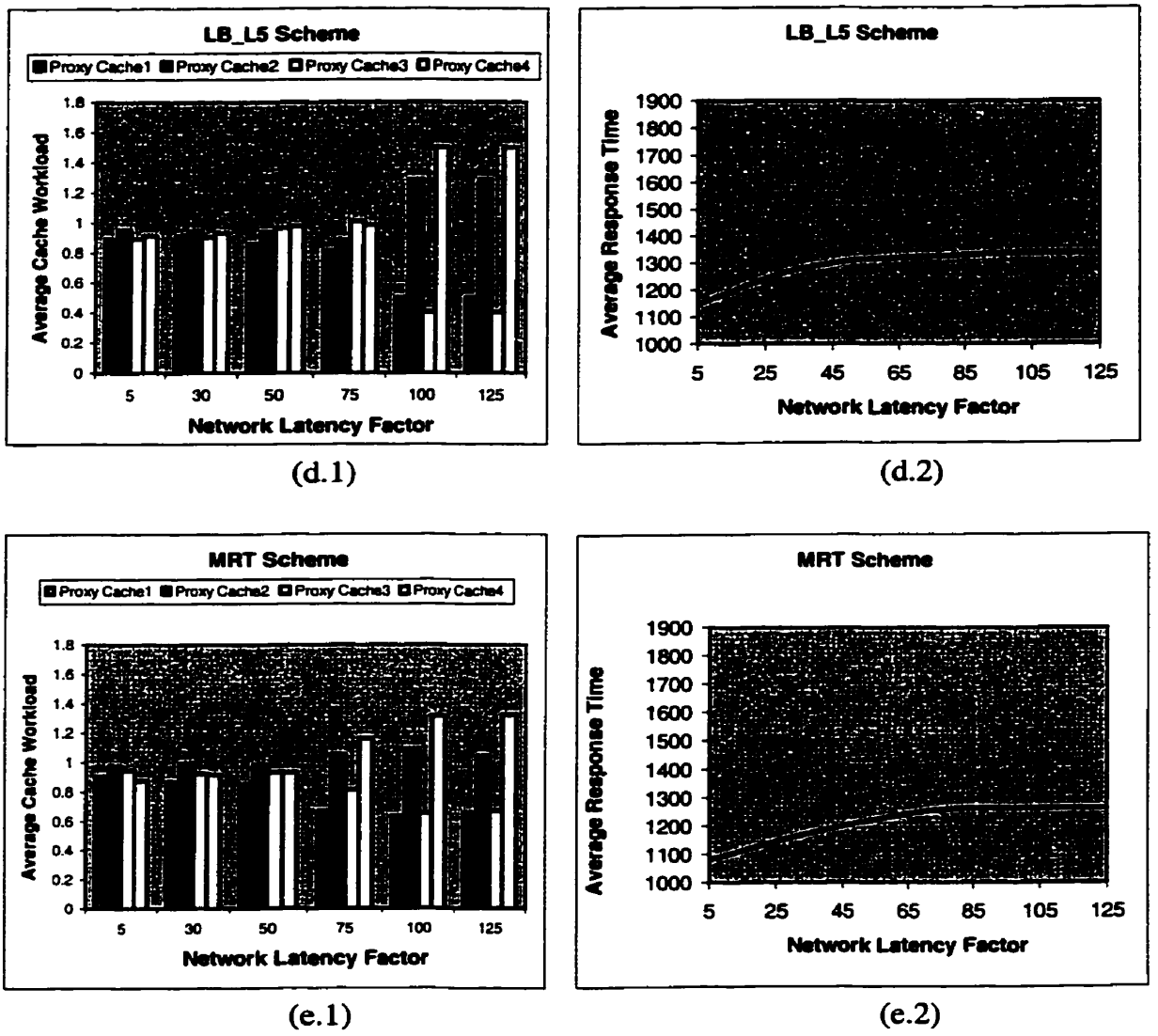


Figure 4.10 Workload and Average Response Time

4.4.4 Effect of Number of Cache Server

This section presents experiments under different number of cooperating cache servers for different HTTP request intensities and network latency factors. To provide a fair comparison when we change the number of coopering cache servers, we fix the total number of HTTP requests generated by client clusters and the total capacity of the Web caching system. We expect that, if the number of cooperating cache servers is small, then

the number of HTTP requests that each cache server services will be large and the communication cost between cache servers will be small. Similarly, if the number of coopering cache servers is larger, then the number of HTTP requests that each cache server services will be small and the communication cost between cache servers will be large.

Figure 4.11 plots the response time versus the number of cooperating cache servers. Figure 4.11 (a) and (b) show the impact of the number of cache severs for 100% HTTP request intensity. As the number of cooperating cache servers increases, the number of HTTP requests on each cache server decreases and the processing time on each cache server decreases. When the network latency factor is very small (5 milliseconds), the communication cost between cache servers can be ignored. The average response time is mainly determined by cache servers processing time. The average responses time of all schemes decrease as the number of cache servers increases.

When the network latency factor is large (50 milliseconds), the communication cost between cache servers cannot be ignored. As the number of cooperating cache servers increases, the communication cost also increases. The average response time is determined by both the communication cost and the cache server processing time. RTT tries to minimize the communication cost when it routes requests. MRT tries to minimize the sum of the communication time and the processing time. When the network latency factor is 50 milliseconds, the expense of the communication cost negates the benefit from reducing the processing time on each cache server. Content, Workload and LB_L5 can distribute the client requests to more cache servers as the number of cache servers

increases. Therefore, the workload on each cache server decreases. However, communication cost increases as the number of cache servers increases. The average response time of Content, Workload and LB_L5 increases when the number of cache servers is greater than 4, as shown in Figure 4.11 (b). For RTT and MRT, the average response time always decreases as the number of cache servers increases.

Figure 4.11 (c) and (d) show the results of experiments with 200% HTTP request intensity. The performance improvement by splitting the workload of cache server is more significant. In this situation regardless of the network latency (5 or 50 milliseconds), the average response time of all schemes decreases as the number of cache servers increases. As expected earlier, the rate of the decrease of the average response time under a small network latency factor is higher than that under a larger network latency factor. When the network latency factor is small (5 milliseconds), see Figure 4.11 (c), the average response time of Content, Workload, RTT, LB_L5 and MRT decreases by 45%, 48.6%, 40.4%, 48.6% and 49.5%, respectively as the number of cache servers increases from 2 to 8. When the network latency factor is large (50 milliseconds), see Figure 4.11 (d), the average response time of Content, Workload, RTT, LB_L5 and MRT decreases by 31.5%, 35%, 32%, 41.4% and 43.5%, respectively as the number of cache servers increases from 2 to 8.

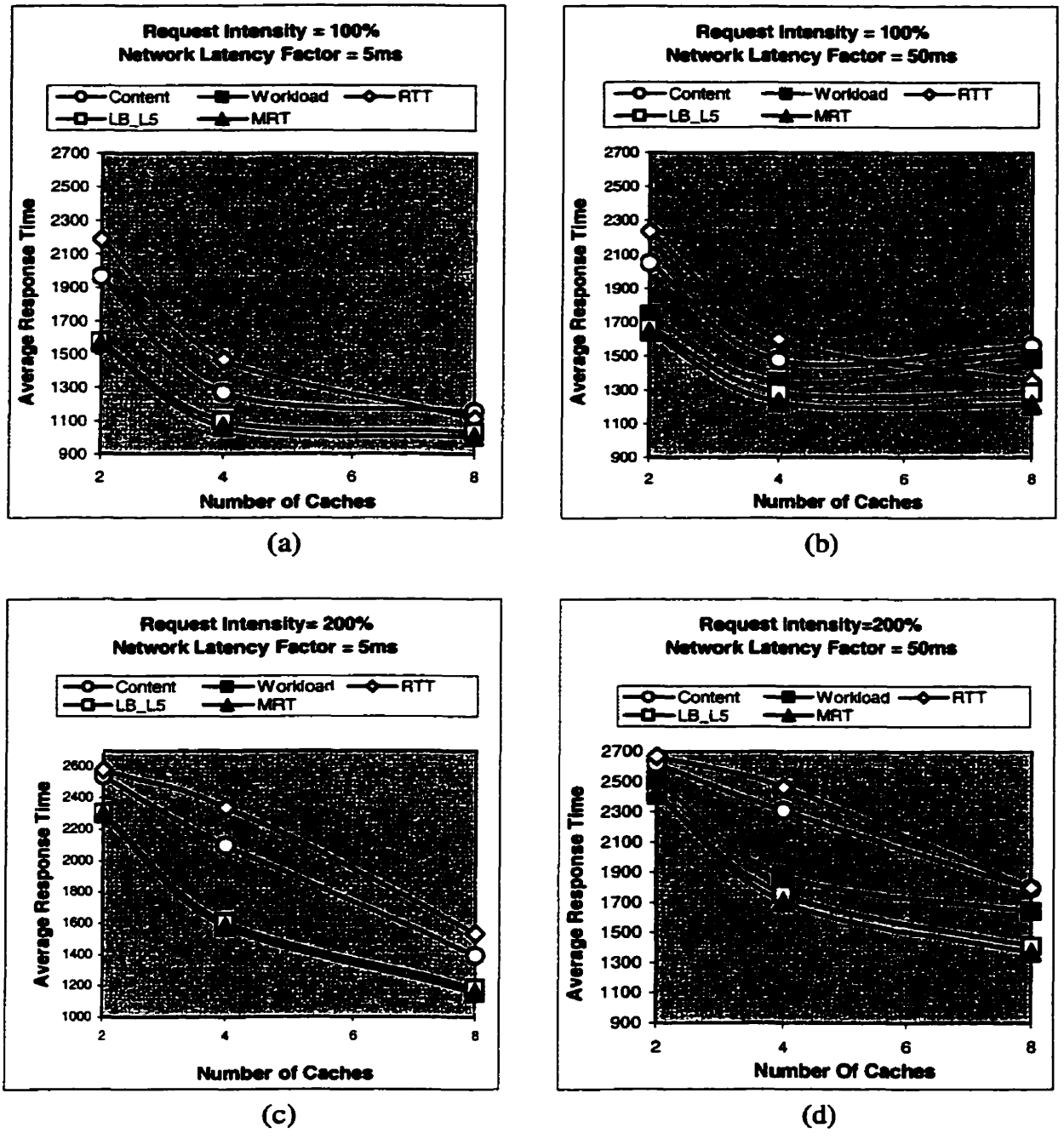


Figure 4.11 Average Response Time Versus The Number of Cache Servers

4.5 Summary

This chapter presents an evaluation of the performance of the proposed MRT Web caching scheme, through a comprehensive simulation. The performance of MRT is compared with that of the Content, Workload, RTT and LB_L5 schemes. The network model, the network latency model, the server workload model, the validation checking model and the simulation experimental settings are described. Two types of simulation experiments, namely raw trace and controlled trace were conducted to investigate the effects of the network latency, HTTP request intensity, expiration time and the number of cooperating proxy servers on the performance of the MRT. The results are compared with the other transparent distributed Web caching schemes.

The simulation experiments show that MRT outperforms Content, Workload, RTT and LB_L5 in the term of the HTTP request response time. MRT always achieves significantly lower request response time than that of other schemes, under different network latencies, HTTP request intensities and object expiration time values.

In the raw-trace experiments, MRT outperforms Content, RTT by an average of 13% and 12%, respectively and has a similar performance as Workload and LB_L5 under small network latency. The performance advantage is more prominent when the network latency is large. MRT outperforms Content, Workload, Response and LB_L5 by 44%, 34%, 30% and 11%, respectively.

In the controlled-trace experiments, it is shown that MRT, like Workload and LB_L5, adapts better to higher request intensities than the Content and RTT. For examples, at a latency factor of 30 milliseconds, MRT outperforms Content, Workload, RTT and LB_L5 by an average of 19%, 7%, 14% and 7%, respectively when the HTTP request intensity is 100%, as opposed to 47%, 7%, 38% and 7%, respectively when the HTTP request intensity is 250%. MRT also adapts better to large network latency. For example at HTTP request intensity of 100%, MRT outperforms Content and RTT by 14% and 15.5%, respectively and has similar performance as Workload and LB_L5 when the network latency factor is 5 milliseconds, as opposed to 35%, 44%, 36% and 12%, respectively when the network latency factor is 125 milliseconds. It is also shown that the average response time increase of MRT is more controlled than that of Content, Workload and LB_L5 when the expiration time decreases. The experiments conducted for investigating MRT cache server workload balancing show that the workload of cooperating cache server are well balanced when the network latency is small. For large network latencies, MRT balances server workload better than RTT and LB_L5.

Chapter 5

Conclusion

The objective of this research is to optimize the performance of transparent distributed Web caching systems in terms of request response time. The proposed Minimum Response Time (MRT) scheme distinguishes non-cacheable requests from cacheable requests based on HTTP request header. It intelligently redirects cacheable requests to the cache server with the minimum HTTP request response time. MRT estimates the HTTP request response time based on cache server content, cache server workload, Web server workload and network latency. MRT uses four extended ICP messages and one extended switch routing table to track the most recent update of the above information.

5.1 Contributions

In general, our work has accomplished the following goals:

- We studied different switching-based transparent Web caching schemes. We investigated how they combine to support cache cooperation in distributed Web caching systems. We also analyzed the advantages and disadvantages of different server selection algorithms.

- We proposed a heuristic solution to optimize the performance of the distributed L5 switching-based transparent Web caching system, namely the Minimum Response Time (MRT) scheme. MRT improves upon the performance of other schemes.

- A detailed simulation model was developed to study the performance of the proposed MRT Web caching scheme. We compared the performance of MRT with Content, Workload, RTT and LB_L5 Web caching schemes. Performance results show that MRT outperforms all other schemes in terms of HTTP request response time. MRT outperforms Content, RTT and LB_L5 in terms of workload of cache servers. MRT is also shown to adapt better not only to high HTTP request intensity and unbalance request intensity but also to large network latency.

5.2 Future Work

There are a number of aspects of our work that need further investigation:

- **Object size:** In our research, when we estimate the request response time, we do not consider the object size. The object size affects the time spent on server delay, which is incurred retrieving the object from the disk, and the time spent on the transmission latency. These times may have a significant effect on the request response time if most of the requested objects are large objects, such as Video or Audio documents. Request response times cannot be compared directly if objects are of different size. How to map the object size to time components remains open for future research.
- **Huge network latency**: The balancing between network latency and server workloads has an inherent problem. In MRT, when the network latency factor is

huge, all client requests are directed to the Web client's local cache server. The cooperation among cache servers is lost and the workload of cache servers cannot be effectively balanced. MRT is suitable to use among a cluster of cache servers within reasonably large network latency. The cooperation among cache server clusters with huge network latency need to be investigated further.

- **Request Priority**: The Layer 5 switches used in MRT can distinguish different types of the requested objects, such as multimedia objects or image objects. Layer 5 switches can assign priorities to requests for different types of objects so that switches can service high priority requests before low priority requests. This kind of prioritized transparent Web caching should be investigated further.

Bibliography

[1] K. Claffy and D.Wessels, "ICP and the Squid Web Cache". 1997

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

[3] M.Abrams, C.R. Standridge, G. Abdulla, S.Williams, and E.A.Fox. Caching Proxies: Limitations and Potentials. In *Proceedings of the 4th International World-Wide Web Conference.* 1995

[4] A.Bestavros. Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic, and Service Time for Distributed Information Systems. In *Proceedings of the 1996 International Conference on Data Dissemination.* 1996

[5] 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.

[6] 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.

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

[8] Cieslak, M., and Foster, D. "Web Cache Coordination Protocol V 1.0". *Internet Draft of IETF, draft-ietf-wrec-web-pro-00.txt*, Jun 1999.

[9] D.Wessels. "Configuring Hierarchical Squid Caches", 1997

[10] A. Chantkuthod, P. B. Danzig, C.Neerdaels, M.F. Schwartz and K.J. Worrell. "A Hierarchical Internet Object Cache". Technical Report 95-611-University of Southern California, Boulder, California, U.S.A. 1996

[11] D. Wessels and K. Claffy, "Internet Cache Protocol (ICP), version2". RFC 2186, Sep 1997

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

[13] S. G. Dykes, C.L.Jeffery, and S.Das, "Taxonomy and design analysis for distributed web caching". In *Proceedings of the IEEE Hawaii Int'l. Conference Con System Sciences (HICSS'99)*, Jan. 1999

[14] D.Povey and J. Harrison. "A distributed internet cache". In *20th Australian Computer Science Conference*, Feb 1997

[15] R. Tewari, M.Dahlin, H.M.Vin, and J.S. Kay, "Design considerations for distributed caching on the Internet". In *Proceedings of the International Conference on Distributed Computing Systems (ICDS'99)*, 1999.

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

[17] B. Williams, "Transparent web Caching Solutions", Director of Strategic Business Planning, Alteon Networks. White Paper. In *The third International WWW Caching Workshop*, 1998.

[18] ArrowPoint Communications, "Content Smart Cache Switching". White paper.

[19] ACEdirector "Load Balancing - Technical Specifications". Product Overview, 2000

[20] Z. Liang, H.Hassanein and P.Martin "Transparent Distributed Web Caching". In *Proceedings of the IEEE Local Computer Network Conference*, Nov 2001, pp.225-233.

[21] Extreme Networks Server Load Balancing. TechBrief, 2000

[22] Cisco CSS-11150 - Content Services Switch. Cisco Product Catalog, Jan 2002

[23] M.E.Crovella and R.L.Carter, "Dynamic server selection in the internet". In *Proceedings Of the Third IEEE Workshop on the Architecture and Implementation of High Pref.Comm. Subsystems* (HPCS'95), Aug.1995, pp.158-162

[24] J.D.Guyton, M.F.Schwartz. "Locating Nearby Copies of Replicated Internet Services". In *SIGCOMM'95*, Cambridge, MA, USA, pp.288-298, 1995

[25] R.L. Cater and M.E.Crovella, "Server selection using dynamic path characterization in wide-area networks". In *Proceedings Of IEEE Infocom'97*, Apr.1997

[26] M.Sayal, Y.Breitbart, P.Scheuermann, and R. Vingralek, "Selection algorithms for replicated web servers". In *Performance Evaluation Review*, vol.26, no.3, pp.44-50, Dec.1998.

[27] S.G.Dykes, K.a. Robbins and C.L.Jeffery, "An Empirical Evaluation of Client-side Server Selection Algorithms". In *Proceedings Of IEEE Infocom'00*, Vol 3, pp.1361-1370, Mar, 2000

[28] D. Andresen, T.Yang, V.Holmedahl, and O,H.Ibarra, "SWEB: Towards a scalable World Wide Web server on multicomputers". In *Proceedings Of the 10th Int'l, Parallel Processing Symp. (IPPS'96)*, Apr 1996.

[29] D. M.Dias, W.Kish, R.Mukherjee, and R.Tewari, "A scalable and highly available web server". In *Proceedings Of the 41st IEEE Computer Society Int'l, Conference*, Feb. 1996.

[30] Internet Control Message Protocol. RFC 792, Sep 1981

[31] 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

[32] Hypertext Transfer Protocol -- HTTP/1.1. RFC 2068, Jan 1997

[33] SSH Protocol. *Internet Draft of IETF, draft-ietf-secsh-connect-14.txt*, Nov 2001

[34] C.Yoshikawa, B.Chun, P.Eastham, A.Vahdat, T. Anderson, and D.Culler, "Using smart clients to build scalable services". In *Proceedings Of the 1997 USENIX Annual Technical Conference (USENIX'97)*,1997, pp.105-117

[35] L.Fan, P.Cao, J.Almeida, and A.Z.Broder," Summary cache: A scalable wide-area web cache sharing protocol". Technique Report, pp.1461, Department of Computer Science, University of Wisconsin-Madison, Feb 1998.

[36] C. Faloutsos and S. Christodoulakis, "Design of a Signature File Method that Accounts for Non-Uniform Occurrence and Query Frequencies". In Proceedings of *11th International Conference on VLDB*, pp. 165-170, Stockholm, Sweden, August 1985.

[37] 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), pp. 272-273, Madison, W1, June 1998

[38] M. Rabinovich, J. Chase, 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, Nov 1998

[39] J. Almeida and P.Cao, "Measuring Proxy Performance with the Wisconsin Prxoy Benchmark". In *Proceedings of the Third International Caching Workshop*, June 1998

[40] P. Barford and M. Crovella, "Generating Representative Wb Workloads for Network and Server Performance Evaluation", In *Proceedings of the 1998 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, July 1998

[41] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance". *ITTT/ACM Transactions on Networking*, August 1993

[42] Y.P., Recursive Estimation and Time-Series Analysis, *Springer-Verlag*, 1984, pp. 60-65

[43] CERN Httpd. Available at: http://www.w3.org/pub/WWW/Daemon

[44] Netscape Proxy Cache Server. Available at: http://www.netscape.com

[45] Glassman, S. "A Caching Relay for the World Wide Web". In *Proceedings of the First International World Wide Web Conference*, 1994

[46] Xp. Chen and P. Mohapatra, "Lifetime Behavior and its Impact on Web Caching", Department of Electrical and Computer Engineering, 1998

Appendix A

Bloom Filter

To analyze the relationship between the optimum number of hash functions W for a Bloom Filter with the Bloom Filter size F and number of objects D in the cache server, we formalize the problem using an approach similar to that in [36] and derive the result for the Bloom Filter from the weighted Bloom Filter. The difference between a weighted Bloom Filter and a Bloom Filter is that in a weighted Bloom Filter the objects with high frequency are represented by more bits while the objects with low frequency are represented by fewer bits. In a regular Bloom Filter, all objects in a cache server are represented by the same number of bits.

In a weighted Bloom Filter, we assume that, according to some conditions such as frequency the set S of all objects in a cache can be 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

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

We define the following variables:

D_i: The number of objects in subset S_i, $D = D_1 + D_2 + \ldots + Dn$ is the total number of objects in the cache.

P_i: The access probability for objects in S_i. It is the possibility that any object in subset S_i will be accessed.

W_i: The weight for subset S_i. It is the number of hash functions for subset S_i

F: The Bloom Filter size

In a weighted Bloom Filter representing D objects, the probability that a particular bit is 0 is:

$$R_0 = (\frac{F-1}{F})^{W1*D1 + W2*D2 + \ldots + Wn*Dn} = (1-\frac{1}{F})^{W1*D1 + W2*D2 + \ldots + Wn*Dn} \qquad (A.1)$$

We know that $(1 - \frac{1}{x})^b = e^{-b/x}$ when $x \to \infty$

Equation (1) can be approximated as:

$$R_0 \approx e^{-(W1*D1 + W2*D2 + \ldots W3*D3)/F}, \quad \text{when } F \to \infty \qquad (A.2)$$

Hence the probability that a particular bit is 1 is:

$$R_1 = 1 - R_0 \approx 1 - e^{-(W1*D1 + W2*D2 + \ldots Wn*Dn)/F} \qquad (A.3)$$

The false prediction probability is:

$$Fp = P_1*R_1^{W1} + P_2*R_1^{W2} + \ldots + Pn*R1^{Wn} \qquad (A.4)$$

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:

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

$$\frac{\partial F_p}{\partial W_i} = \frac{R}{1-R}\frac{F}{D_i} P_i R^{W_i} \ln R + \sum_{j=1}^{n} P_j W_j R^{W_j} = 0, \quad 1 \le i \le n \tag{A.5}$$

Equation above is equivalent to:

$$\frac{P_1 R^{W_1}}{D_1} = \frac{P_1 R^{W_1}}{D_1} = \ldots = \frac{P_1 R^{W_1}}{D_1} = \frac{F_p}{D} = K \tag{A.6}$$

K is a constant independent of i.

Substituting equation (A.6) into (A.5), then

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

or

$$\frac{R}{1-R} = \frac{\sum_{j=1}^{n} W_j D_j}{F \ln R} \tag{A.7}$$

From equations (A.6) and (A.7),

$$\frac{R}{1-R} = \frac{\ln(1-R)}{\ln R} \text{ or } R = \frac{1}{2} \tag{A.8}$$

Substituting equation (A.8) into (A.7), (A.8)

$$\sum_{j=1}^{n} W_j D_j = F \ln 2 \tag{A.9}$$

Substituting equation (A.8) to (A.6),

$$W_i = \frac{1}{\ln 2}\left[\ln \frac{p_i}{D_i} - \ln K\right] \tag{A.10}$$

Substituting equation (A.10) to (A.9),

$$\ln K = \frac{-F(\ln 2)^2 + \sum_{i=1}^{n} D_i \ln \frac{P_i}{D_i}}{D} \tag{A.11}$$

Substituting equation (A.11) to (A.10), the optimum values for W_i is:

$$W_i = \frac{F \ln 2}{D} + \frac{1}{\ln 2}\left[\ln \frac{P_i}{D_i} - \frac{\sum_{j=1}^{n} D_j \ln \frac{P_i}{D_i}}{D}\right], 1 \le i \le n \tag{A.12}$$

For a Bloom Filter n = 1, $W_1 = W_2 = \ldots W_n = W$, $D_1 + D_2 + \ldots + D_n = D$, $P_1 = \ldots = P_n = 1$, substituting all values to equation (A.12), we can get the optimum value for W:

$$W = \frac{F \ln 2}{D} \tag{A.13}$$

Appendix B

Implementation Pseudo Codes

The Pseudo Code for Layer 5 switches in MRT

A pseudo code description of the functions of a L5 Switch is shown in Figure B.1. Lines 1 to 9 show how a L5 Switch deals with different TCP messages. When a L5 Switch receives a TCP connection request from a Client, it accepts the connection by sending back a TCP_ACK (lines 3 - 4). When a L5 Switch receives a TCP ACK from a proxy cache server, it means a TCP connection has been established between the L5 switch and the cache. Then the L5 Switch will relay a HTTP request from the client to that cache (lines 5 - 6). When it receives a TCP_FIN (TCP connection finished signal) from a proxy cache or the web server it relays the signal to the client to tear down the TCP connection between them (lines 7 - 8)

Lines 10 to 20 indicate how a L5 Switch deals with the HTTP requests and HTTP responses. When a L5 Switch receives a HTTP request from a client, it will make the routing decision and find out which server it should go to and make a TCP connection request to that server (lines 12 - 16). The details of how the L5 Switch makes the routing

decision are presented from Lines 63 to 78 on page 96. When a L5 Switch receives the requested object from the cache server or the Web server it will relay it to the client who makes the request (lines17 - 20).

Lines 21 to lines 48 describe how a L5 Switch deals with different extended ICP messages. If an in coming ICP message is ICP_UPDATE_CONTENT, the L5 Switch will find out which cache server's contents need be updated according to the ICP message's SenderAddress field. If the ICP message's time stamp is greater than that cache server's time stamp for the last content update, which means the ICP message is valid, the L5 Switch will update that cache server's content and number of objects in its CacheArrayTable and send back the ICP_UPDATE_ACK message to that cache server. If a coming ICP message is ICP_UPDATE_WORKLOAD and the sender is a cache server, the L5 Switch will update the workload, the time stamp and the query response time for the sender cache server in its CacheArrayTable. The elapse time, that is the time between the L5 Switch sending the ICP_QUERY_WORKLOAD and the L5 Switch receiving the ICP_UPDATE_WORKLOAD, will be recorded in its CacheArrayTable as the current network latency between the L5 Switch and the sender cache server. A L5 Switch also updates the Web server's workload when it receives the updated workload sent by the Web server.

As described in lines 51 to 56, a L5 Switch queries the workload of cache servers by periodically sending ICP_QUERY_WORKLOAD to each of the cache servers and tracks the query time in its CacheArrayTable. If there is a timeout before next query message and the L5 switch finds out that the workload of some caches is not updated, then the L5

switch will set the workload of those caches as infinity (lines 57 - 61). In this way, a L5 Switch can avoid redirecting requests to a non-responsive cache server.

```
// On receipt of TCP Messages
   1. Procedure onReceiveTCPMessages(msg:TCP)
   2. {
   3.     if (msg.type ==TCP_SYN)
   4.         sendTCP_ACK(clientAddress);

          // relays http request to a cache when receives a TCP_ACK from that cache
   5.     if (msg.type ==TCP_ACK)
   6.           sendHTTP_Request(cacheAddress);

          // relays tcp_fin to client when receives a tcp_fin from proxy or web server to
          //terminate the TCP connection
   7.     if (msg.type == TCP_FIN)
   8.         sendTCP_FIN(clientAddress).;
   9.}

// On receipt of HTTP Messages
   10 Procedure onReceiveHTTPMessage (msg: HTTP)
   11 {
           // Redirects http_request to different cache server or web server based on the
             routing decision, first send a tcp connection request to that destination
   12       if (msg.type == HTTP_REQUEST)
   13       {
   14           destinationAddress=makeRoutingDecision();
   15           sendTCP_SYN( destinationAddress)
   16.      }

           // http_response from proxy/web server relay http_response to the client
   17.     if (msg.type == HTTP_RESPONSE)
   18      {
   19.         sendHTTP_Response(requestedObject, clientAddress)
           }
   20.}
```

```
// receives different ICP messages

21. Procedure onReceiveICPMessage(msg:ICPMsg)
22. {

        //if the ICP message is update content
23.     if (msg.OPCode = =ICP_UPDATE_CONTENT)
24.       for i:=1 to NumOfCacheServers
25.         if (CacheArrayTable[i].SenderAddress = = msg.SenderAddress) &&
26.         if (msg.TS > CacheArrayTable[i].Last_Content_UpdateMsg_TS)
27.         {
28.           CacheArrayTable[i].Content:= msg.Content
29.           CacheArrayTable[i].Count = msg.Num
30.           CacheArrayTable[i].Last_Content_UpdateMsg_TS= msg.TS;
31.           sendMsg(ICP_UPDATE_CONTENT_ACK, mySwitch.IPAddress,
32.           msg.TS,  CacheArrayTable [i]. SenderAddress)
33.         }

        //if the ICP message is update workload
34.     if (msg.OPCode = =ICP_UPDATE_WORKLOAD)
35.       if  (msg.SenderAddress = = WebServer.IPAddress)
36.           Webserver.Workload = msg.Workload
37.       else
          {
38.        for i:=1 to NumOfCacheServers
39.          if CacheArrayTable[i].SenderAddress = = msg.SenderAddress) &&
40.           (msg.TS > CacheArrayTable[i].Last_Workload_UpdateMsg_TS)
             {
41.            CacheArrayTable [i].Workload:= msg.Workload
42.            CacheArrayTable[i].Last_Workload_UpdateMsg_TS= msg.TS;
43.            CacheArrayTable[I].Workload_QueryRes_Time = getCurrentTime();

               //roundtrip time is calculated as the latency
44.            if(msg.TS == CacheArrayTable[i].Workload_Query_TS)
45.              CacheArrayTable[i].networklatency =
                          CacheArrayTable[i].Workload_QueryRes_Time -
                          CacheArrayTable[i].Workload_Query_Time;
46.          }
47.       }
48.}
```

```
// a Layer 5 switch query workload of caches periodically
    49. Procedure Query_Workload (query_Workload_Interval)
    50. {
          //broadcast workload query message to all cache servers
    51.   for i:=1 to NumOfCacheServers
    52.   { CacheArrayTable[i].Workload_Query_TS+=1;
    53.     CacheArrayTable[i].Workload_QueryTime = getCurrentTime();
    54.     sendMsg (ICP_QUERY_WORKLOAD, mySwitch.IPAddress,
    55.    CacheArrayTable[i]. Workload_Query_TS, CacheArrayTable[i].IPAddress)
    56.   }
          //checkif the query workload message lost or no response
    57.   wait until (getCurrentTime()>sendTime+Timeout)
    58.   for i:=1 to NumOfCacheServers
            if (CacheArrayTable[i].Workload_QueryResponseTime<
                  CacheArrayTable[i].Workload_QueryTime)
    59.     {
    60.         CacheArrayTable [i].Workload= INFINITY;
    61.     }
    62. }

// Layer 5 switch makes routing decision
    63 Procedure makeRoutingDecision(req:HTTPRequest)
    64 {
    65      if (req.isCacheable == false)      //if request is non-cacheable
    66        destinationAddress = WebServer.IPAddress
    67      else {                              //if request is cacheable
    68          for i:=1 to NumOfCacheServers
              {
    69          if (CacheArrayTable[i].Content == req.Content)    Fp = calFp();
    70          else   Fp = 1;
    71          ResponseTimeArray [i].ResTime =
                Fp* ( CacheArrayTable[i].networklatency
                      +processTime(CacheArrayTable[i].Workload)
                      + 2*RTTcs_ws +processTime(WebServer.Workload)) +
                   (1-Fp) * (CacheArrayTable[i].networklatency +
                        processTime(CacheArrayTable[i].Workload))
    72        }
              //pick the cache server with the minimum response time
    73        for i:=1 to NumOfCacheServers
    74          if (ResponseTimeArray[i].ResTime<MinResTime)
    75          {     MinResTime = ResponseTimeArray[i].ResTime;
    76                destinationAddress = ResponseTimeArray[i].IPAddress;
    77.         }
    78. }
```

Figure B.1 Pseudo Code For Layer 5 Switch in MRT

The Pseudo Code for Proxy Cache Server in MRT

A pseudo code description of the added functions (distribute content and deal with extended ICP messages) of a Proxy Cache Server in MRT is shown in Figure B.2. To cooperate with L5 Switches, the cache servers must support the extended ICP messages. Lines 1 to 17 describe how a cache server sends its updated content to each L5 switch with ICP_UPDATE_CONTENT. A proxy cache periodically publishes its content information and the number of objects in it to all the L5 Switches. If there is a timeout before the next ICP_UPDATE_CONTENT and the cache server find out that some L5 Switches do not acknowledge the ICP_UPDATE_CONTENT, then the cache server will send the same ICP_UPDATE_CONTENT again.

Lines 18 to 31 present how a cache server deals with different ICP messages. If the coming ICP message is ICP_UPDATE_CONTENT_ACK, the cache will update the timestamp for it in its SwitchArray table. If the coming ICP message is ICP_QUERY_WORKLOAD, the cache server will send its workload to any L5 switch that queries the workload.

```
   //send the content information to switch periodically
1   Procedure distributeContent (content:BLOOM_FILTER, num:Int)
2   {
3      for i :=1 to NumOfSwitches
4      {
5            SwitchArray[i].Content_Update_TS +:=1;
6            SendMsg  (ICP_UPDATE_CONTENT, myCache. IPAddress,
                       content, num, SwitchArray [i]. Content_Update_TS,
                       switchArray[i].IPAddress);
7      }
8.     sendTime = getCurrentTime();
9.     wait until (getCurrentTime()>sendTime+Timeout)
10     for i :=1 to NumOfSwitches do
11     {
12       if(SwitchArray[i].Content_Update_TS!=
13         SwitchArray[i].Content_Update_ACK_TS)
14         SendMsg (ICP_UPDATE_CONTENT, myCache. IPAddress, content, num,
15                  SwitchArray [i].Content_Update_TS,  switchArray[i].IPAddress);
16     }
17 }


18 Procedure onReceiveMessage(msg:ICPMEssage)
19 {
20     switch (msg.OPCode)
21     {
22         case ICP_UPDATE_CONTENT_ACK:
23            for i :=1 to NumOfSwitches
24               if  (SwitchArray[i].IPAddress == msg.SenderAddress)
25                  SwitchArray[i].Content_Update_Ack_TS := msg.TS;

26         case ICP_QUERY_WORKLOAD)
27            for i :=1 to NumOfSwitches)
28               if  (SwitchArray[i].IPAddress == msg.SenderAddress)
29                  sendMsg(ICP_UPDATE_WORKLOAD,myCache.IPAddress,
                              myWorkload, msg.TS, SwitchArray[i].IPAddress
30     }
31 }
```

Figure B.2 Pseudo Code For Proxy Cache Server In MRT

The Pseudo Code for Web Server in MRT

A pseudo code description of the added update workload function of a Web Server in MRT is shown in Figure B.3. To cooperate with L5 Switches, the Web server supports the extended ICP_UPDATE_WORKLOAD message. Lines 1 to 7 describe how a Web server periodically sends its updated workload to L5 switches.

```
1. Procedure OnUpdateWorkloadTimeout ()
2. {
3.       myWorkload = getWorkload();
4.       for i :=1 to NumOfSwitches
5.          if (SwitchArray[i].IPAddress == msg.SenderAddress)
6.             sendMsg(ICP_UPDATE_WORKLOAD, myIPAddress, myWorkload,
                      SwitchArray[i].IPAddress)
7. }
```

Figure B.3 Pseudo Code For Web Server In MRT

Appendix C

Validation Checking Pseudo Code

The following pseudo code C.1 shows how W3C httpd 3.0 maintains cache consistency (it is derived from study of httpd's source code).

```
Procedure GET (P:page, S: date)
// Answer a HTTP GET request.  S is the date in the request's If-Modified-Since: header.
// If there was no If-Modified-Since: header, S = -infinity.
// Note that the Date: and Last-Modified: headers are stored with every cached page, and
//are transmitted unchanged whenever a cached page is returned.
{
      if (P in Cache & current_time < cached_P.expiration_time)
                        // in cache, not expired
      {
            if (cached_P.last_modified <= S)
            {
                  return "Not Modified";
                  Date = cached_P.date;
            }
            else
                  return cached_P
      }
```

```
if (P in cache & current_time >= cached_P.expiration_time)  // in cache, expired
{
        send If-Modified-Since(cached_P.last_modified) to higher level cache
        if (response="Not Modified")
        {
             cached_P.expiration_time= new_expire_time(P);
             return "Not Modified";
            // with headers, including Date:, as we just received them from higher-
            //level cache
         }

        else
        {
           cached_p = received_P;
           cached_P.expiration_time = new_expire_time(P);
           return cached_P;
        }
}

if (P not in cache)
{
      send If-Modified-Since(S) to Higher level cache
      if (whole document was returned)
      {
         cached_P = response
         cached_P.expiration_time = new_expire_time(P)
      }
      return response to client, including headers
}
}
```

Figure C.1 Expiration Pseudo Code

Appendix D

The Simulator Structure

The simulator used in our study is a discrete event-driven simulation. It simulates the Content, Workload, RTT, LB_L5 and MRT Web caching schemes. The major classes in the simulation are as follows:

Class Sim

It is the main class of the simulator. It initializes the Web clients, cache servers and the layer 5 switches. It varies the request intensity factors to generate various controlled traces. It runs the four simulated Web caching schemes.

Class ClientCluster

It simulates a client cluster by reading proxy traces to generate HTTP requests. It handles the TCP messages and HTTP request and response messages.

Class L5Switch_ContentSwitch

It simulates a layer 5 switch used in a Content-based transparent Web caching scheme. It redirects the requests in a round robin fashion or randomly.

Class L5Switch_Workload

It simulates a layer 5 switch used in a Workload-based transparent Web caching scheme. It redirects the requests based on request content and each cache server's workload.

Class L5Switch_RTT

It simulates a layer 5 switch used in a Round trip time-based transparent Web caching scheme. It redirects the requests based on the request content and round trip time between the switch and each cache server.

Class L5Switch_LB_L5

It simulates a layer 5 switch used in the LB_L5 Web caching scheme.

Class L5Switch_MRT

It simulates a layer 5 switch used in our MRT Web caching scheme. It redirects the requests based on the MRT selection algorithm.

Class ProxyCacheL5Content

It simulates a proxy cache server used in a Content-based transparent Web caching scheme. It supports LRU replacement algorithm and expiration time validation checking mechanism.

Class ProxyCacheL5Workload

It simulates a proxy cache server used in a Workload-based transparent Web caching scheme. It supports the LRU replacement algorithm and the expiration time validation checking mechanism.

Class ProxyCacheL5RTT

It simulates a proxy cache server used in a Round trip time-based transparent Web caching scheme. It supports the LRU replacement algorithm and the expiration time validation checking mechanism.

Class ProxyCacheL5LB_L5

It simulates a proxy cache server used in the LB_L5 Web caching scheme. It supports the LRU replacement algorithm and the expiration time validation checking mechanism.

Class ProxyCacheL5MRT

It simulates a proxy cache server used in our MRT Web caching scheme. It supports the LRU replacement algorithm and the expiration time validation checking mechanism.

Class WebServer

It simulates a Web server. It accepts HTTP requests and sends back HTTP responses. In our simulation, the Web server also supports the ICP messages. It periodically calculates its current workload and sends its workload to switches with the ICP_UPDATE_WORKLOAD message.

Appendix E

Confidence Intervals

Normally, confidence intervals placed on the mean values of simulation results can be used to describe the accuracy of the simulation results. Consider the results of N statistically independent simulation runs for the same experiment: X_1, X_2, ..., X_N. The sample mean, $\overline{X}$ is given as:

$$\overline{X} = \frac{\sum_{i=1}^{N} X_i}{N}$$

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

$$S_x^2 = \frac{\sum_{i=1}^{N} (X_i - \overline{X})^2}{N-1}$$

The standard derivation of the sample mean is given by: $\frac{S_x}{\sqrt{N}}$.

Under the assumption of independence and normality, the sample mean is distributed in accordance to the T-Distribution, which means the sample mean of the simulation runs

fall in the interval $\pm\varepsilon$ within the actual mean with a certain probability drawn from the T-Distribution.

$$\varepsilon = \frac{S_x t_{\alpha/2,N-1}}{\sqrt{N}}$$

where $t_{\alpha/2,N-1}$ is the value of the T-distribution with N-1 degrees of freedom with probability $\alpha/2$.

The upper and lower limits of the confidence interval regarding the simulation results are:

$$\text{Lower Limit} = \overline{X} - \frac{S_x t_{\alpha/2,N-1}}{\sqrt{N}}$$

$$\text{Upper Limit} = \overline{X} + \frac{S_x t_{\alpha/2,N-1}}{\sqrt{N}}$$