ADDIS ABABA UNIVERSITY (AAU)

ADDIS ABABA INSTITUTE OF TECHNOLOGY (AAiT)

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Design Analysis of Web Caching Solution

By

Mershaye Bekele

Advisor

Dr. Yalemzewd Negash

A Thesis Submitted to the School of Electrical and Computer Engineering of Addis Ababa University in Partial Fulfillment of the Requirements for the Degree of Master of Science in Electrical Engineering

April 2016

Addis Ababa, Ethiopia

ADDIS ABABA UNIVERSITY (AAU)

ADDIS ABABA INSTITUTE OF TECHNOLOGY (AAiT)

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Design Analysis of a Web Caching Solution

By

Mershaye Bekele

Approval by Board of Examiners

_____	_____
Chairman, Dept. of Graduate Committee	*signature*
Dr. Yalemzewd Negash	_____
Advisor	*Signature*
_____	_____
Internal Examiner	*Signature*
_____	_____
External Examiner	*Signature*

Declaration

I, the undersigned, certify that the research work titled "*Design Analysis of a Web Caching Solution*" is my own work. The work has not been presented elsewhere for assessment and when material has been used from other sources it has been properly acknowledged or referred.

Mershaye Bekele _____

Name *Signature*

Addis Ababa

Place

_____

Date of submission

This thesis has been submitted for examination with my approval as a university advisor.

Dr. Yalemzewd Negash _____

Advisor's name *Signature*

Abstract

Any service provider has to make sure that its end-users get the best experience from network services in terms of quality of service, responsiveness, reliabilities, and geographical coverage. It also needs to ensure that there is a balance between the cost of delivering the Internet services and end user experience.

For better end user experience and customer satisfaction; accelerated delivery of web contents, low bandwidth usage, and decreased server loads are very important. This is where web caching comes into play. Web caching is a technique used to temporarily store a copy of information which has been requested by a user. Thus, subsequent requests of the same information will be served from the stored copy rather than the original location.

The thesis analyzes the design of deploying a web caching solution to save the link bandwidth which will be vital for exponentially increasing demand of Internet services. This leads to improved Internet traffic carrying capacity of the backbone network for the service providers while at the same time increasing the quality of services to the end user.

In this paper different caching architectures are explored and their performances are discussed and analyzed using simulation and analytical model. By applying the analytical model, the web caching architectures are compared in terms of numerical parameters such as connection and transmission times. On the other hand, the simulator is used to compare the hit rate, byte-hit rate, reduced packets and reduced latency of a web cache under different cache replacement techniques for sample web client traces.

The analytical result shows that hierarchical caching gives shorter connection times than distributed caching while distributed caching gives shorter transmission times than hierarchical caching. Besides from the simulation it is observed that the performance of cache replacement techniques depends on the web client request distributions.

Keywords: Web Caching, Web Caching Architecture, Hit Ratio, Latency, Replacement Policy, Cache Coherency.

Acknowledgment

First and foremost, I would like to express my deep gratitude to my advisor Dr. Yalemzewd Negash for his continuous guidance and support during the course of this thesis. His advice and support helped me achieve my thesis goals. Thus, I am very grateful to him for his patience and willingness to read my work and provide very useful and constructive suggestions to this thesis.

I must extend my gratitude to thank all the professors who taught me graduate courses. They all are hard-working professors and I believe their academic achievements will continue to enhance for the benefit of the university as well as for our country.

During the period of four years as a graduate student at AAU, many friends were ready to lend me a hand both in academic and in real life. Thus, I would like to thank all my colleagues of postgraduate communication 2004 E.C. entry extension program.

Last but not least, I owe more than thanks to my family members and work colleagues for their continuous support and encouragement throughout my postgraduate study. Without their support, it would have been more challenging for me to finish my postgraduate education in time.

Table of Contents

List of Figures

List of Tables

Acronyms

ASCII	American Standard Code for Information Interchange
CGMP	Cache Group Management Protocol
CPU	Central Processing Unit
CRP	Content Routing Protocol
FIFO	First In First Out
FTP	File Transfer Protocol
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ISP	Internet Service Provider
LFF	Largest File First
LFU	Least Frequently Used
LRU	Least Recently Used
LRV	Lowest Relative Value
MB	Mega Byte
MIME	Multipurpose Internet Mail Extensions
ms	Millisecond
NTE	Next to Expire
QoS	Quality of Service
TCP	Transmission Control Protocol
URL	Uniform Resource Locator
WAN	Wide Area Network
WCCP	Web Cache Communication Protocol
WWW	World Wide Web

Definition of Terms

Coherency [Cache]: The consistency of items in cache with those stored on the origin servers.

Freshness: Describes whether an item within a cache is still a candidate to serve to a client.

Hit Ratio: The ratio of the requests that can be retrieved from a cache to the total requests.

Invalidation: Removing content from a cache before its specified expiration date.

Latency: Describes how long after requesting a desired item the cache can return that item.

Miss [Cache]: Failure to find the required item within the cache.

Origin Server: The original location of the content.

Replacement Policy [Cache]: Used to choose items to discard to make room for the new ones.

Stale Content: An expired content within a cache.

Validation: Checking in with the origin server to see if the cached content is still fresh.

CHAPTER ONE

1 Introduction

These days imagining the world without Internet services is very difficult. In the past few decades, Internet has become one of the basic needs in people‟s lives. Its low cost, efficiency and easy to use characteristics make it a very popular means of communication, data sharing, and more importantly conducting online businesses. All these lead to a very rapid growth of World Wide Web. However, it is not realizable to cope up with this growth just by adding the capacity of servers and the links. One of the methods to reduce the loads at the web servers and at the same time reduce the latency to the clients is by deploying web caches.

A web cache is a storage device that is used to temporarily keep web pages and contents to be served for subsequent requests of the same item [56]. When a client tries to access an item or content, the request is analyzed and decision is made whether to forward it to the web cache or to the origin sever by network according to certain protocols. In networks with a proper web cache design, the request is not directly forwarded to the intended or origin server instead it is directed to the local web cache. Using different look up mechanisms, the cache tries to fetch the content from its own local storage and delivers it to the client. If the cache does not have the requested web page, it will initiate its own web request to the origin server on behalf of the client which requested the content. The server then send the content to cache, which in turn delivers to the client while leaving the copy of the document stored on the local storage device. Thus, when the same or another user requests the same content, the request is fulfilled locally from the web cache. This will avoid unnecessary use of the expensive international link while at the same time accelerates the delivery of web contents to end-users.

Some of the main benefits of web caches include [1]:

- **Cost savings due to WAN bandwidth reduction:** the web cache drastically reduces bandwidth consumption by caching content in region and closer to the user. This translates into a rapid return on investment and significant long-term cost savings for service providers on International bandwidth, as well as reducing backhaul traffic on domestic links. Web cache can be positioned to serve more users, deliver next-generation services and better leverage existing network investments by conserving and optimizing bandwidth with caching.

- **Improved productivity for end users:** The response of a local web cache is often three times faster than the download time for the same content over the WAN. End users see dramatic improvements in response times, and the implementation is completely transparent to them.

- **Operational logging:** Network administrators can learn which URLs receive hits, how many requests per second the cache is serving, what percentage of URLs are served from the cache, and other related operational statistics.

- **Increased performance on the same hardware:** For the server where the content originated, more performance can be squeezed from the same hardware by allowing aggressive caching. The content owner can leverage the powerful servers along the delivery path to take the brunt of certain content loads.

- **Availability of content during network interruptions:** With certain policies, caching can be used to serve content to end users even when it may be unavailable for short periods of time from the origin servers.

1.1 Problem Statement

Most web browsers have a very simple approach in accessing web contents on the network. Given a URL, containing a host name and an item on that host, they make a TCP connection to the named host and retrieve the specified item. If the host cannot be reached or the item does not exist, the user will receive an error message instead of the page requested. Since browsers, and their users, are independent there is a huge amount of replication in the information carried over the network: every user gets its own copy of every page it requests and users access the same content over and over. Thus, popular sites may have quite a lot of simultaneous connections transmitting identical copies of a single item over the same trunk routes which lead to high bandwidth usage, network congestion, server overloading and high user-perceived latency.

Web caching is a technique designed to deal with these problems. By implementing highly effective web caching technology, it is possible to avoid redundant and multiple requests from travelling all the way to the origin server. Thus, it benefits the service provider to generate additional revenue by saving the cost incurring on the international link or by adding greater number of Internet users without the need to enlarge the capacity of the existing network.

1.2 Thesis Objective

1.2.1 General Objective

The main objective of this research paper is to recommend web caching schemes applicable for a particular network scenario and content request pattern that result in an optimum performance from both service provider and end-user perspective.

1.2.2 Specific Objectives

The specific objectives to be met during the course of this thesis include:

- Get the basic understanding of principles and architectures for keeping copy of web pages and contents on storage devices that are physically or logically close to the user.
- Explore the advantages and disadvantages of the common web caching architectures.
- Identify the numerical cache parameters to be used for comparing the performance of different web cache schemes.
- Examine the performance of different large scale cache cooperation schemes or architectures using analytical and simulation methods.

1.3 Research Questions (RQs)

The thesis typically covers the answers of the following questions:

RQ1: What are the main issues associated with web caching techniques?

The paper addresses the issues and challenges in using the web caching solution. The main points include issues related to freshness, validity, administrative concerns and the like. The issues associated with web caching are covered in section 2.2 under Web Caching Parameters.

RQ2: What are the different deployments scenarios of web caching techniques?

With the context of the thesis it is important to discuss different web cache deployment techniques that include hierarchical, distributed and also a hybrid caching scheme where a number of caches cooperate at every level of a caching hierarchy using distributed caching. The web caching deployment scenarios are discussed in section 2.6 under Web Caching Architectures.

RQ 3: How to evaluate the performance of web caching technologies for different network deployment scenarios and which of these web caching solutions have better performance for specific network scenarios?

The main part of this thesis paper is to identify the cache parameters, and use them in comparing the performance of the different web architectures. The numerical parameters considered in this study include hit rate, byte-hit rate, reduced packets, connection time and transmission time. Different deployment techniques are compared using theoretical, analytical and simulation so as to suggest which type of deployment is suitable for a certain network. The results of both analytical model and simulation are presented in Chapter Four under Results and Analysis.

1.4 Literature Review

As an input to this study various literatures, white papers and books related to web caching are reviewed. The referred materials are those that focus on the concepts of single cache, cooperative cache, web caching architectures, performance analysis and cache placement and replacement policies. The following are just a few of these literatures which are relevant for the development of this study.

Jacobson [2] suggested that caching technology should rise comparable to the exponential growth of the Internet. He proposed that data should be available on sources that are adjacent to users rather than their place of origin.

Shim et al. [3] introduced a single cache algorithm that was relatively simple with only one proxy cache. The algorithm considered cache replacement and consistency maintenance for web proxies. Fan et al. [4] has described a protocol for multiple caching called "summary cache". In this protocol, every proxy keeps a summary of the cache directory of each participating cache. It then checks the summary before sending data for any queries of potential hits. To handle misses and for reducing total traffic, these caches cooperate with each other. Fan et al. recognized that to fully benefit from caching, caches should cooperate and serve each other"s misses to reduce the total traffic through the bottleneck.

Kelly [5] introduced techniques for optimizing performance and cost-effectiveness in designing new Web cache hierarchies and optimizing the existing ones. The paper quantified the impact of content-naming practices on cache performance; presented techniques for variable-quality-of-service cache management; described how a decentralized algorithm can compute economically-optimal cache sizes in a branching two-level cache hierarchy; and introduced a new protocol

extension that eliminates redundant data transfers and allows "dynamic" content to be cached consistently.

Integrating existing solutions to address performance and scalability issues was presented by Suresha [6]. Specifically, it aimed at achieving reduced bandwidth consumption from web infrastructure perspective, and reduced page construction times from user perspective. To address performance and scalability issues, various dynamic content caching approaches have been proposed in the literature.

Yao et al. [7] proposed a cooperative caching strategy for supporting a hybrid scheme based on a virtual hierarchy of caches. The virtual hierarchy is defined by hashing on the object and cache names, so that a distinct virtual hierarchy is created for each object. Unlike static hierarchies, where the root can become a bottleneck, their approach creates a different hierarchy and root for each object, so that loads are evenly balanced. Clients can determine the hierarchy for each object locally using a hierarchy of hashing functions.

In their paper, Che et al. [8] developed analytical modeling technique to characterize an uncooperative two level hierarchical caching system where the least recently used (LRU) algorithm is locally run at each cache. With this modeling technique, they were able to identify a characteristic time for each cache, which is used to understand the caching processes. In particular, a cache can be viewed roughly as a low-pass filter with its cutoff frequency equal to the inverse of the characteristic time. Documents with access frequencies lower than this cutoff frequency have good chances to pass through the cache without cache hits. This viewpoint enabled them to take any branch of the cache tree as a tandem of low-pass filters at different cutoff frequencies, which further results in the finding of two fundamental design principles. Finally, to demonstrate how to use the principles to guide the caching algorithm design, they proposed a cooperative hierarchical web caching architecture based on these principles. Both model-based and real trace simulation studies show that the proposed cooperative architecture results in more than 50% memory saving and substantial central processing unit (CPU) power saving for the management and update of cache entries compared with the traditional uncooperative hierarchical caching architecture.

Analytical model was used by Hurley et al. [9] to compare the mean response time of distributed and hierarchical Web caching architectures. The analytical model uses simplifying assumptions for distributed and hierarchical caching systems such that they can be defined using a birth-death model. The analytical results showed that the mean response time in the distributed caching

system tends to be less than that of the hierarchical caching system when the arrival rate of the two caching systems is equal as well as the hit-rate of the individual caches. The analytical comparison was done for an initial condition with one client and the limiting situations where the number of clients gets large. They also used simulation to validate their conclusions.

The paper by Tewari et al. [10] described the design and implementation of an integrated architecture for cache systems that scale to hundreds or thousands of caches with thousands to millions of users. Rather than simply try to maximize hit rates, they took an end-to-end approach to improving response time by also considering hit times and miss times. They derived three core design principles for large scale distributed caches: (1) minimize the number of hops to locate and access data on both hits and misses, (2) share data among many users and scale to many caches, and (3) cache data close to clients.

Lee Breslau et al. [11] addressed in their paper that the distribution of web requests from a fixed user community does not follow Zipf's law [12] precisely, but instead it follows a Zipf-like distribution with the exponent varying from trace to trace. Furthermore, they found that there is only a weak correlation between the access frequency of a web page and its size and its rate of change.

1.5 Methodology

To conduct this study in accordance to the aim and objective of this research, two types of research methods are used. First, the hierarchical caching is compared against the distributed caching in terms of connection time and transmission time using analytical method. Then by means of a simple simulator, the performance of different web cache replacement techniques including Least Recently Used (LRU), Lowest Relative Value (LRV), SIZE, and Hybrid are compared using numerical parameters such as hit rate, byte-hit rate, reduced packets and reduced latency for sample web client traces.

1.6 Scope of the Thesis

This work focuses on gathering relevant information about web caching solution from different reliable resources so as to have a good knowledge and understanding of the working principles of this technique. It then examines the information through analytical and simulation research methods. The methods used in this study include theoretical, analytical and using simple simulator. Finally, the paper proposes better performing web caching architectures that best suit a specific network. The author of this paper strongly believes that this study can be a cornerstone for the deployment of web caching solution in any service provider network. The results in the

paper can also be extended and used to confirm the viability of this solution for the benefit of the company as well as the end-users.

1.7 Organization of the Study

This study is organized in five chapters and several sections under these chapters. The first chapter is a brief introduction of web caching technology along with related literature reviews, research questions to be covered in the study, thesis objectives, and the research methodologies used to meet these objectives.

The rest of this paper is organized as follows. The next chapter focuses on the detail analysis of web caching solution that includes design parameters, cache positions and deployment techniques. Research methodologies including analytical model and simulator tool are addressed in Chapter Three while the results of these methods are analyzed in Chapter Four. On this chapter, total latency that combines the values of connection and transmission times is used in the comparison of hierarchical, distributed and hybrid caching schemes. In Chapter Five, limitations that encounter during the course of this study and conclusions based on the results obtained in the previous sections are discussed. Also at the end of the chapter, the future work related to this study is included.

CHAPTER TWO

2 Web Caching Solution

2.1 Overview

Fast response time is one of the vital factors for the success of the World Wide Web. Web caching can be used to reduce latency to the users and minimize the bandwidth usage to the service providers. Thus, web caching can significantly enhance end-user‟s web browsing experience and, at the same time, save bandwidth for service providers. In brief, a web caching is a technique that is used to temporarily store a copy of the data content requested by users from the Internet. After an original request for data has been successfully fulfilled from the origin server, the copy of that data will be stored in the cache. Thus, subsequent requests of the same information (e.g., HTML pages, images) results in the information being returned from the cache, if certain conditions are met, rather than the original location. Generally, cached information is stored on a physical storage device which is closer to the user and capable of fast delivery than the original server. The following flow diagram illustrates how caching works from an end-user perspective [13].

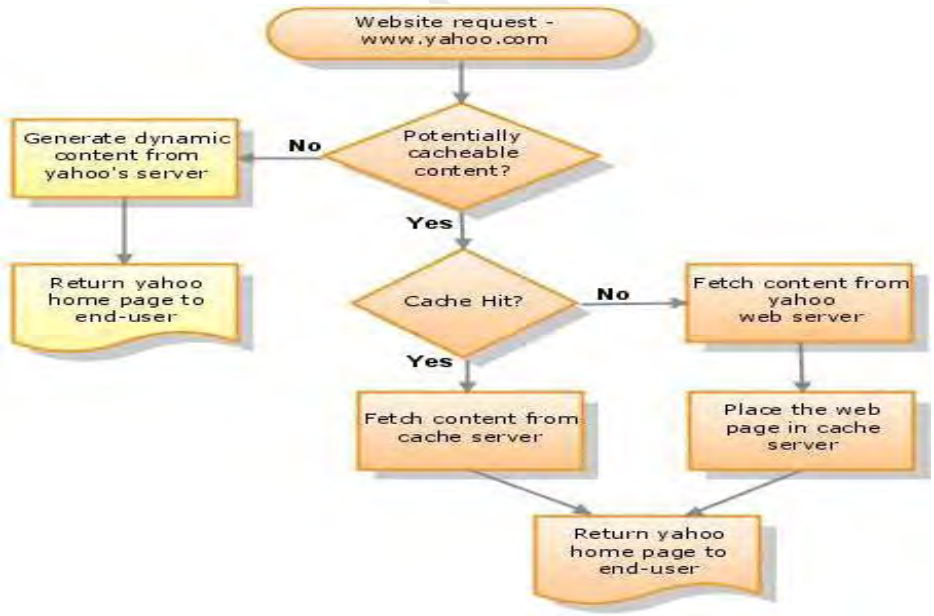


Figure 2-1: Flow Diagram of How Web Caching Works

2.2 Web Caching Parameters

Web caching solutions should be properly and carefully designed according to the network requirement. Misconfigured designs may lead to undesired results like high connection and transmission times, increased bandwidth usage, network loops, hot spots or bottlenecks and high

server loads. The following are some of the caching properties and parameters that should be considered during caching design.

Cache Hits: The state in which data requested by a web cache client is found and retrieved from a cache, as the cache already contains the copy of the requested data. This situation is known as a cache hit. The percentage of accesses that result in cache hits is known as the hit rate or hit ratio of the cache.

Cache Misses: In contrast to a cache hit, a cache miss is a failure to find the required data within the cache. In such case, the item is obtained from the original web servers. Usually cache misses fall into four categories [15]:

- **Compulsory misses:** These misses correspond to the first access to an object. The two key strategies for reducing compulsory misses are increasing the number of clients sharing a cache system and prefetching.
- **Capacity misses:** These misses occur when the system references an object that it has previously discarded from the cache to make space for another object. For shared caches, capacity misses are a relatively minor problem that can be adequately addressed by building cache nodes with a reasonable amount of disk space.
- **Communication or consistency misses:** These misses occur when a cache holds a stale copy of data that has been modified since it was read into the cache.
- **Uncacheable or error misses:** Objects are marked "Uncacheable" or encounter errors for a number of reasons, some of which might be addressed by more sophisticated cache protocols that support better cache consistency, caching dynamically generated results, dynamically replicating servers, negative result caching, and caching programs along with data.

Cacheable Objects: The wide variety of Internet information systems lead to a number of cases where objects should not be cached. For example, objects that are password protected are not cached. Rather, the cache acts as an application gateway and discards the retrieved object as soon as it has been delivered. It is possible to limit the size of the largest cacheable object, so that a few large FTP objects do not purge ten thousand smaller objects from the cache [27].

Unique Object Naming: A URL does not name an object uniquely; the URL plus the MIME header issued with the request uniquely identify an object.

Cache Coherency Mechanisms: As with caching in any system, maintaining cache consistency is one of the main issues that a Web caching architecture needs to address [16]. As more of the data on the Web is dynamically assembled, personalized, and constantly changing, an efficient

consistency management becomes a challenge. To prevent stale information from being served to clients, an intermediary cache must ensure that the locally cached data is consistent with that stored on the origin servers. There are three different ways of implementing cache coherency mechanisms [17]:

- **Directory Based Coherency:** In this method, the data being shared are kept in a common directory. The directory maintains the coherency between caches by ensuring that all caches have a similar version of the requested item. If any changes are made to an entry then the directory either updates it or invalidates it in all other caches with that entry.
- **Snooping Based Coherency:** In this method, each cache monitors the address lines so that to gain access to main memory which they have cached. Any activity on cache line will trigger message, which will be broadcast to all the caches to update the cache line with the activity.
- **Snarfing Coherency:** In this method, caches monitor both the address and the data in order to update its cache line from the main memory when another cache tries to update that cache line.

The exact cache consistency mechanism and the degree of consistency employed by an intermediary cache depend on the nature of the cached data; not all types of data need the same level of consistency guarantees. In general, the degrees of consistency that an intermediary cache can support fall into the following four categories [18].

- **Strong Consistency**: A cache consistency level that always returns the results of the latest (committed) write at the server is said to be strongly consistent. Due to the unbounded message delays on the Internet, no cache consistency mechanism can be strongly consistent in this idealized sense.
- **Delta Consistency:** A consistency level that returns data that is never outdated by more than δ time units, where δ is a configurable parameter, with the last committed write at the server is said to be delta consistent. In practice, the value of delta should be larger than *t* which is the network delay between the server and the intermediary at that instant, i.e. $t<\delta\leq\infty$.
- **Weak Consistency:** For this level of consistency, a read at the intermediary does not necessarily reflect the last committed write at the server but some correct previous value.
- **Mutual Consistency:** A consistency guarantee in which a group of objects are mutually consistent with respect to each other. In this case, some objects in the group cannot be more current than the others. Mutual consistency can co-exist with the other levels of consistency.

Cache Awareness: While no Web client is completely cache-aware, most support access through IP firewalls. Clients send all their requests to their proxy-server and the proxy-server decide how best to resolve it. There are advantages and disadvantages to the cache aware and cache-unaware approaches. Cache-unaware clients are simpler to configure; just set the proxy bindings that users already understand and which are needed to provide Web service through firewalls. On the other hand, cache aware clients would permit load balancing, avoid the single point of failure caused by proxy caching, and allow a wider range of security and end-to-end error checking [17].

Security and Privacy: Some sites require an authentication to access the web resources at the original servers. When the cache tries to access these resources, the server passes an "Unauthorized Message" to the cache; the cache then forwards it back to the client and purges the URL from the cache. The client browser, using the desired security model, prompts for a username and password, and reissues the request with the authentication and authorization encoded in the MIME header. The cache detects the authorization-related MIME header, treats it as any other kind of non-cacheable object, returns the retrieved document to the client, but otherwise purges all records of the object [27].

2.3 Web Cache Positions

Shared network caches can be located in one of the two positions in the network. A forward proxy acts on behalf of a specific group of content users. A reverse proxy, also called a server accelerator, acts on behalf of the origin server and helps a specific group of servers deliver content [20].

2.3.1 Proxy Caching

A proxy cache server receives HTTP requests from clients for a web object and if finds the requested object in its cache it returns the object to the user without disturbing the upstream network connection or destination server. If it is not available in the cache the proxy attempts to fetch the object directly from the objects home server. Finally, the proxy gets the object from the originating server, deposits the object‟s copy in its cache and returns the object to the user on behalf of object‟s home server. The benefits of proxy caching are supposed to reduce network trace and reduce average latency. Proxy caches are often located near network gateways to reduce the bandwidth required over expensive dedicated Internet connections. When shared with other users the proxies serve many clients with cached objects from many servers [21].

One disadvantage to this design is that the cache represents single point of failure in the network. When the cache is unavailable the network also appears unavailable to users. Furthermore, another drawback is that all user web browsers are manually configured to use the appropriate proxy cache. So, if the server is unavailable all of the users must reconfigure their browsers in order to use a different cache. The other issue with this solution is the lack of scalability. As demand rises one cache must continue to handle all requests. It is difficult to dynamically add more caches into the network as is possible with transparent proxy caching.

One variation to the proxy cache approach is the notion of Reverse Proxy Caching [19] in which caches deployed near the servers instead of near the clients. This is an attractive solution for servers that expect a high number of requests and want to assure a high-level of quality of service (QoS). Reverse proxy caching is also a useful mechanism when supporting web hosting farms (virtual domains mapped to a single physical site), an increasingly common service for many Internet service providers (ISPs).

2.3.2 Transparent Caching

One of the main drawbacks of the proxy server approach is the requirement to configure web browsers. The architecture of transparent caching eliminates this handicap. Transparent caches work by intercepting HTTP requests and redirecting them to web cache servers or clusters. There are two ways to deploy transparent proxy caching at the switch level and at the router level.

Router based transparent proxy caching uses policy based routing to direct requests to the appropriate cache or caches. In switch based transparent proxy caching the switch acts as a dedicated load balancer. This approach is attractive because it reduces the overhead normally incurred by policy based routing. Although it adds extra cost to the deployment, switches are generally less expensive than routers [36].

In transparent web caching deployment, the Internet traffic that arrives at the gateway should be redirected based on the type of traffic to the cache server (rather than to the intended origin server). This can be accomplished using TCP port number in most cases. The gateway router should also determine web cache server availability, and redirect requests to new cache servers as they are added to the network. If any of the cache servers fail, the gateway router should redistribute the failed cache server's load evenly among the remaining cache servers. The overall caching system should be able to continue operation using one less cache server, but operation must be unaffected creating a fully redundant caching system.

If an entire caching system fails, the gateway router automatically should stop redirecting traffic to the respective cache servers, sending clients' web requests to the actual destination web site in the traditional fashion. This loss of the entire cache system should appear to users as an increase in download time of web content, but must not have other significant effects. This failsafe response can be made possible by designing the caching system not to be directly in line with clients' other network traffic, or more simply by transparent deployment [39].Figure 2-2 shows the deployment of a transparent proxy caching at router level.

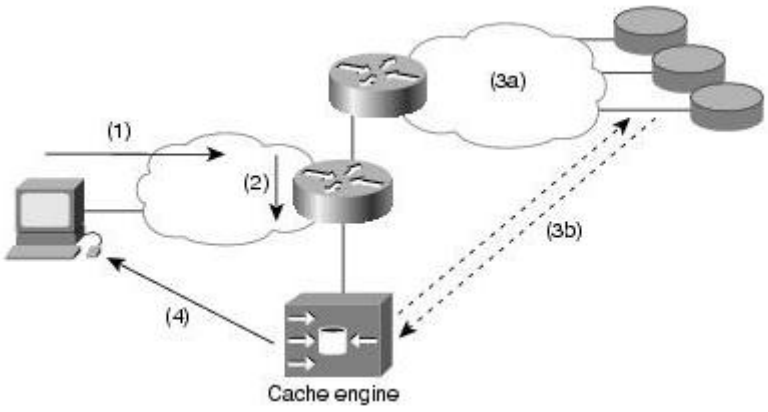


Figure 2-2: Transparent Network Caching

A cache server transparently caches as follows [14]:

1. A user requests a Web page using a browser.
2. The Web Cache Communication Protocol (WCCP) – enabled router analyzes the request, and based on TCP port number, determines if it should transparently redirect it to a cache server.
3. If a cache server does not have the requested content, it sets up a separate TCP connection to the end server to retrieve the content. The content returns to, and is stored on, the cache server.
4. The cache server sends the content to the client. Upon subsequent requests for the same content, the cache server transparently fulfills the requests from its local storage.

2.4 Smart Web Caching

2.4.1 Adaptive Web Caching

Adaptive caching consists of multiple distributed caches which dynamically join and leave cache groups based on content demand [40] [42]. The general architecture of the envisioned adaptive

web caching system would be composed of many cache servers that self-organize themselves into a tight mesh of overlapping multicast groups and adapt themselves as necessary to changing conditions. This mesh of overlapping groups forms a scalable, implicit hierarchy that is used to efficiently diffuse popular web content towards the demand.

Adaptive caching uses the Cache Group Management Protocol (CGMP) and the Content Routing Protocol (CRP). CGMP specifies how meshes are formed and how individual caches join and leave those meshes. CRP is used to locate cached content within the existing meshes [22].

2.4.2 Push Caching

The key idea behind this architecture is to keep cached data close to those clients requesting that information. Data is dynamically mirrored as the originating server identifies where requests originate. One main assumption of push cashing is the ability to launch caches that may cross administrative boundaries. Finally, push caching is targeted mostly at content providers, which will most likely control the potential sites at which the caches will be deployed [23].

2.4.3 Active Caching

An active cache scheme is proposed to support caching of dynamic contents at Web proxies. The growth of the Internet and the World Wide Web has significantly increased the amount of online information and services available to the general population of the society. The Active Cache is a scheme which migrates parts of server processing on each user request to the caching proxy a flexible on demand fashion via cache applets. A cache applet is a server supplied code that is attached with a URL or a collection of URLs [24].

2.5 Inter-Cache Cooperation and Routing

A Web caching system based on a single cache poses many limitations [25]. It does not scale well and the single proxy cache can easily become the performance and communication bottleneck. On the contrary, cooperative proxy caching systems [41], which consist of a set of cache proxy servers serving a group of users, can overcome these limitations. The effectiveness of cooperative caching is largely determined by the cooperative and routing strategy employed to coordinate inter-cache cooperation. The four main types of cooperation and routing strategies are: broadcast queries, hierarchical caching, URL hashing, and directory-based routing table [26].

Broadcast Queries Policy: When a cache proxy receives a client request, it first checks whether the requested object can be served from its local cache. If a valid cached copy is not found in the

local cache, this cache proxy will broadcast the request to all participating proxies asking for their help [27]. The main advantage of this policy is its flexibility that the effects of a proxy joining or departing the cooperative caching network are localized to its immediate neighbors. Two main drawbacks of this policy are: (a) a cache proxy has to wait for the last response from its neighbors before concluding that none of them has the requested documents and sending the request to the next level of the caching hierarchy; (b) broadcast queries result in extra network traffic and impose computational overhead on participating cache proxies.

Hierarchical Caching Policy: In this policy, when a local miss occurs, the cache proxy simply forwards the missed request to its parent in the caching hierarchy without attempting to query sibling cache proxies. Besides obvious savings in communication overhead compared with the Broadcast Queries Policy, this policy has the additional benefit of allowing different organizations to use a common high-level parent proxy without sharing cached contents at the lower levels. The main disadvantages of the hierarchical caching policy are: (a) cache proxies closing to the root need to store a large number of objects and can become performance bottlenecks; and (b) the entire hierarchy has to be traversed before it can be concluded that a requested object has to be fetched directly from the original Web server [28].

Directory-Based Cache Cooperation: In this approach, the location of cached objects is explicitly maintained by a directory server [4]. When a miss occurs at a certain cache proxy, it will query the directory server to find out which proxy can provide the requested object. Upon receiving a response from the directory server, this proxy will either contact another cache or visit the original Web server directly. To ensure that the directory server always provides fresh location (meta) information, any cache proxy that is caching new objects or dropping old contents needs to send updates to the directory server. Since the communications between the directory server and caches do not contain the actual content, messaging overhead associated with this approach is not significant. Another advantage of this approach is that it promotes loose and flexible connections between cache proxies. Individual proxies can be added to and removed from the system without the knowledge of other cache proxies. The main disadvantage of this policy is that the directory server may become a single point of failure.

Hashing Function-Based Cache Routing: In a hashing function-based approach, all Web clients store a common hash function that can be used to map any given URL to a hash space [29]. The hash space is partitioned and each set in the partition is associated with one of the sibling caches. When a client needs to access a Web object, it first hashes the URL of the object and then requests it from the sibling cache whose set contains the hash value. If this cache cannot satisfy

the request, it retrieves the object from the original server, places a copy in its cache, and forwards the object to the client. The advantages of hashing-based approaches include: First, they are scalable with respect to the number of user requests. More cache proxies can be easily added and the hash space can be repartitioned. Second, their computational and communication overheads are relatively low. Third, they lead to efficient use of cache storage space because no duplicated copies of the same document will be made on different cache proxies. The main disadvantage of these hashing-based approaches is that the same cache proxy must process all requests for a given URL, no matter where the requests come from. This can lead to potential performance degradation.

2.6 Web Caching Architectures

To increase the efficiency (i.e. the hit rate) of a web cache, large scale caching structure in which caches cooperate with each other is used rather than a single cache structure. There are generally two common approaches to implement a large scale cache cooperating schemes [56]. These are hierarchical and distributed caching.

In hierarchical caching, caches are carefully placed at strategic locations at different network levels [50], in a parent-child scenario. This scheme works as follows, at the bottom level of the hierarchy there are the client caches (e.g. browsers, etc.). When a request is not satisfied by the client cache, the request is redirected to the child cache. If the document is not found at the child level the request is then forwarded to the parent cache. If the document is not found at any cache level, the parent cache contacts directly the origin server. When the document is found, either at any cache or at the origin server, it travels down the hierarchy, leaving a copy at each of the intermediate caches. Further requests for the same document travel up the caching hierarchy until the document is hit at any cache level. If a hierarchical structure is arranged properly, the hit ratio can be increased significantly [49].

Some of the problems associated with hierarchical caching include:

- Every hierarchy introduces additional delays [32],
- Higher level caches may become bottlenecks and experience long queuing delays[56],
- Several copies of the same document are stored at different cache levels [31],
- Creates complex networks for management, maintenance and troubleshooting [49],
- To set up such a hierarchy, cache servers need to be placed at key access points in the network. This often requires significant coordination among participating cache servers [43],
- Needs high investment [49].

A completely different approach to implement large-scale cache is a distributed caching. In distributed caching, no intermediate caches are setup and only caches at the edge of the network cooperate to serve each other‟s misses. Distributed caching became common with the emerging of new applications that allow distributions of web pages, images, and music. This is because distributed caching has lower transmission times than hierarchical caching due to the fact that most traffic flows through less congested lower network levels [31]. Moreover, distributed cache creates reasonable share of loads within the network and does not generate hot spots or bottlenecks with high load which are some of the downsides of hierarchical caching.

Recently, several researchers have suggested the setup of totally distributed caching scheme, where there are only caches at the bottom level of the network which cooperate [49]. In order to decide from which cache to retrieve a miss document, caches keep metadata information about the content of every other cooperating cache. To make the distribution of the metadata information more efficient and scalable, a hierarchical distribution can be used [33, 10]. However, the hierarchy is only used to distribute information about the location of the documents and not to store document copies.

With distributed caching most of the traffic flows through low network levels, which are less congested and no additional disk space is required at intermediate network levels. However, a large scale deployment of distributed caching encounters several problems like longer connection times, higher bandwidth usage, and administrative issues. In a smaller scale, where close access caches are interconnected through fast links and bandwidth is plenty, distributed caching has very good performance with no additional intermediate cache levels.

Rodriguez et al. [31] proposed that a hybrid scheme with an optimal number of cooperating caches at every level could increase the performance of hierarchical and distributed caching with reduced latency, load and bandwidth cost. In hybrid scenario, caches cooperate at the same or at a higher level using the notion of distributed caching in order to serve each other‟s misses [30].

In hybrid caching when an item cannot be found within the cache, the cache first checks if any of the cooperating caches within the same network level has the copy of the item requested using the notion of distributed caching. If multiple caches happen to have the item copy, the cache with the lowest latency is selected. However, if the item is not found in any of the cooperating caches, the request is forwarded to the cache in the higher network level, using hierarchical caching.

Table 2-1 summarizes the web caching architectures discussed in this section along with their respective advantages and disadvantages.

Table 2-1: Comparison of Caching Architectures

Caching Architecture	Description	Advantage	Disadvantage
Hierarchical Caching	• Caches are placed at multiple levels of the network [50].	• Bandwidth efficient– especially when cache servers are slow • Efficiently diffuse popular web pages towards the demand. • Has shorter connection time. [8, 27, 31, 49]	• Cache server needs to be placed at key access points of the network, requires coordination among caches. • Each level adds a delay. • High levels are bottlenecks. • Multiple copies at different cache levels. • Keeping the freshness of contents is difficult. [8, 31, 32, 43, 56]
Distributed Caching	• Caches at the bottom level only. • No other intermediate caching levels. • Each cache server contains metadata on the data stored on other servers. • Hierarchy used only for distributing information about location of the copy. • No copying of actual documents. [10, 33, 49]	• Traffic flows through low network levels which are less congested. • No additional disk space required for intermediate network levels. • Better load sharing. • More fault tolerant. • Has shorter transmission time. [10, 31, 49]	• High connection times for large scale deployment • Higher bandwidth usage at low network levels. • Administrative issues. [10, 31, 49]
Hybrid Caching	• Caches cooperate with other caches at the same level or at a higher level using distributed caching [30].	• Reduced connection time • Reduced transmission time. [30, 55]	

CHAPTER THREE

3 Research Methodology

3.1 Analytical Model

3.1.1 Network Model

The network connecting the origin servers to the clients can be modeled as shown in Figure 3-1. The left part of the network topology represents the service provider network which is assumed to contain all the clients, while the one at the right represents a service provider network consisting of all the origin servers.

It is logical to simplify and model any service provider network with a three level hierarchy: access, distribution, and core network levels. All of the clients are connected to the access networks; the access networks are connected to the distribution networks; the distribution networks are connected to the core network. The core network in turn is connected to the other service providers via transoceanic or satellite links. For the sake of simplicity, the model comprises of just two service providers with one of the service providers containing all of the clients and the other service provider containing all the origin servers.

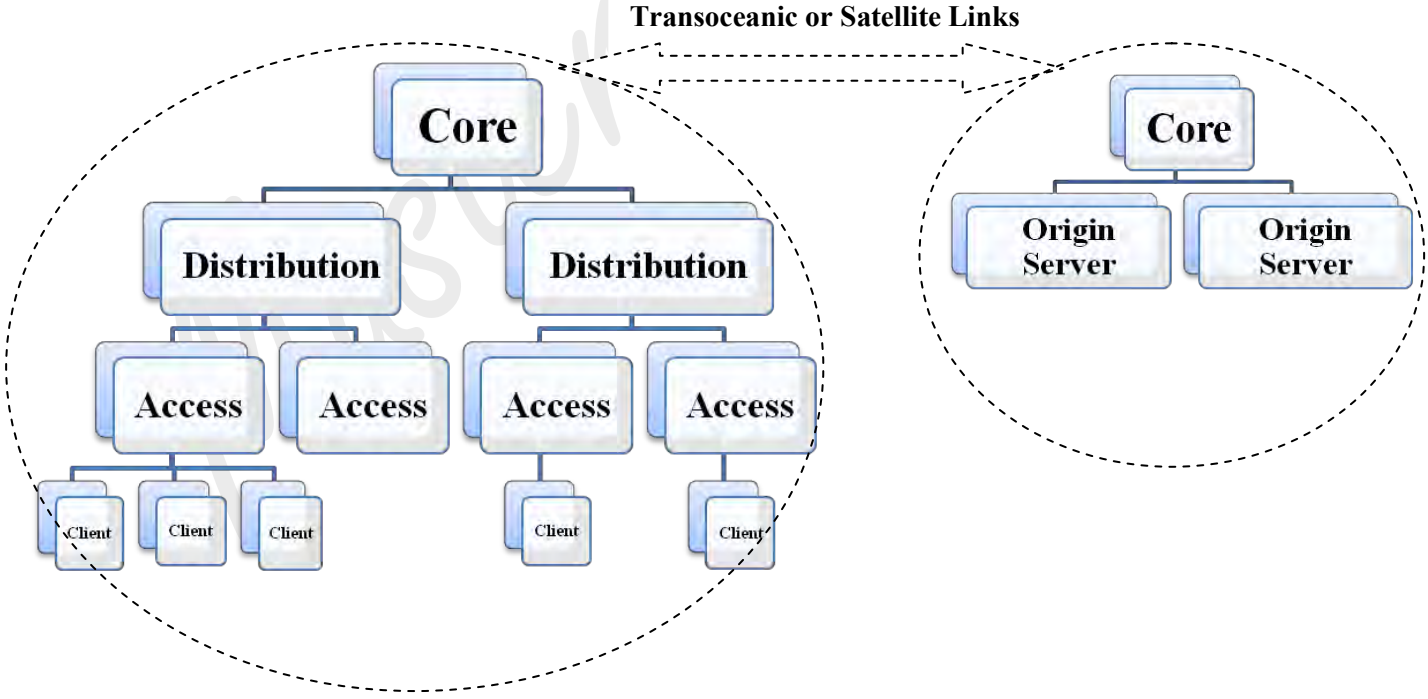


Figure 3-1: Network Topology

The network will be modeled as an *M*-ary tree [54], as shown in Figure 3-2. Let *M* be the nodal outdegree of the tree. Let *D* be the number of network links between the root node of a core network and the root node of a distribution network, and between the root node of a distribution

network and the root node of an access network. Let x be the number of links between an origin server and the local core node. Let l be the level of the tree within the range $0 \leq l \leq 2D+x$, where $l = 0$ is the access caches and $l = 2D+x$ is the origin server.

Let C_A, C_D, and C_C be the bandwidth capacity of the links at the access, distribution, and core networks. Let C be the minimum bandwidth on the International path. The clients are only on the leaves of the tree and not on the intermediate nodes.

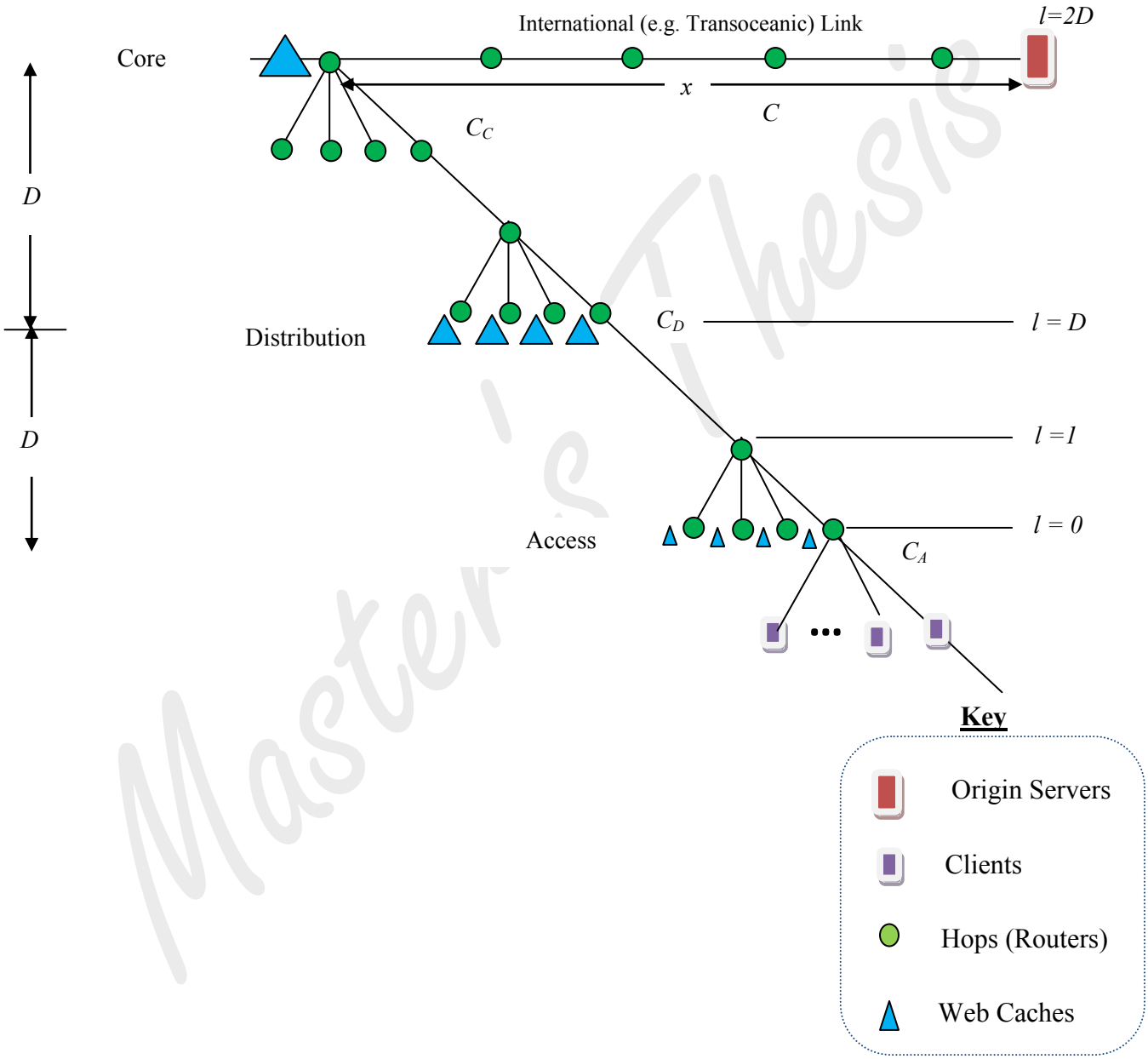


Figure 3-2: The Tree Model Caches Placement.

3.1.2 Document Model

Let N denote the total number of documents in the World Wide Web (WWW). Let S be the average document size. Assuming all the documents within the cache are updated or removed from the caches every Δ seconds. The requests for a single document i, $1 \leq i \leq N$, in the access

cache during an update period Δ are Poisson distributed [37] with an average request rate of λ_A. Assuming that requests are uniformly distributed between all M^{2D}accesscaches, there will be $\lambda_{tot} = \lambda_A * M^{2D}$ total requests for the document i. Let β_A be the request rate from an access network for all N documents, thus, β_A *is* given by $\beta_A = \sum_{i=1}^{N} \lambda_i$. Let $\beta_{tot} = \beta_A * M^{2D}$ be the total request rate from all the M^{2D}access caches for all documents. As it is written in different literatures, the request rate for the documents in the World Wide Web (WWW) is Zipf distributed [34] that is, if all the N documents are ranked in order of their popularity, the i^{th} most popular document has a request rate λ_{tot} given by:

$$\lambda_{tot} = \beta_{tot} \frac{\sigma}{i^{\alpha}}$$

Where α takes values between 0.6 and 0.8 [34], and σ is given by:

$$\sigma = \left(\sum_{i=1}^{N} \frac{1}{i^{\alpha}} \right)^{-1}$$

3.1.3 Hierarchical Caching

The most common scenario to place caches is usually at the access points so as to reduce the cost of traveling through the most congested and expensive upper layer or International networks. As shown in Figure 3-2, the model consists of one core network with one core cache, a total of M^D distribution networks with each network consisting of one distribution cache and a total of M^{2D}access networks which contain one access cache for each network.

In the model, caches are placed at distance 0 (access caches), distance D (distribution caches) and distance $2D$ (core cache).It is assumed that the capacity of the access link at every level is equal to the network link capacity at that level, i.e., C_A, C_D, and C_C for access, distribution and core network levels respectively.

3.1.4 Distributed Caching

In this solution, web caches are only deployed at the access network level, at l=0 of the model in Figure 3-2. The caches at the intermediate levels ($l=D$ and $l=2D$) are replaced by directory servers, forming a metadata hierarchy for the distribution of location information. Thus, there are no intermediate copies stored in the network. The requests are directed to the closest neighboring cache which has the copy of the document using the location information stored at the intermediate levels [49].

3.1.5 Latency Analysis

The overall latency *T* to obtain a document either from a cache or origin server can be divided into two parts, the connection time (T_c) and the transmission time (T_t). T_c is the time elapsed from the time the document is requested to the time the first data byte is received at the client. T_t is the time to transmit the document which depends on the size of the requested document. The expected total latency is given by [31]:

$$E[T] = E[T_c] + E[T_t]$$

Note: *E[A]* is the expected value of a random variable „A". The expected value is also known as the average or mean value.

3.1.5.1 Connection Time

The first part of the total latency is the connection time, which depends on the number of network links from the client to the cache consisting of the desired document copy. To illustrate the differences between the connection time in a distributed caching scheme and in hierarchical caching, let"s consider a single distribution network, with M^D access caches. If hierarchical caching is used, the distribution cache keeps a copy of all documents requested by any of the M^D access caches. Thus, a client requesting a document previously fetched by any other client in the same distribution network will meet the document at the distribution cache at level *D*.

In distributed caching no intermediate copy of the document is placed at level *D*, and all copies are kept at the access caches. In the distributed caching scheme, a request needs to travel up the hierarchy to a network level *l* such that the tree rooted at level *l* is the smallest tree containing a document copy. After traveling up the network to level *l*, the request needs to travel down towards the access cache with the document copy. In the worst case, to access a document copy from other access cache in the same distribution network, the request needs to travel up to level *D*, and then down to the access level using *2D* links. Let *d* denote the per-hop propagation delay. Thus, the connection time in a caching hierarchy and in a distributed caching scheme given by [31]:

The expected connection time in hierarchical caching:

$$E\left[T_C^h\right] = 4d \sum_{l\epsilon\{0,D,2D,2D+x\}} P(L = l) * (l + 1)$$

The expected connection time in distributed caching:

$$E\left[T_C^d\right] = 4d \sum_{l=0}^{2D} P(L = l) * (2l + 1) + 4d * P(L = 2D + x) * (2D + x + 1)$$

In the calculation for the connection time, one more link is considered to account for the distance between the client and the access cache. In hierarchical caching L is the number of links that a request needs to travel to hit the document. In distributed caching L is the minimum tree level for which any of the M^L access caches flowing from that level contains a copy of the document. The distribution of L is the same for the hierarchical and the distributed scheme. The rationale for the *4d* term is due to the three-way handshake of a TCP connection that increases the number of links traversed before any data packet is sent. When the document is hit in the origin server, both hierarchical and distributed caching travel the same number of links *2D+x+1*.

In order to calculate the distribution of L for document *i,* it is first required to find $P\ (L = l)$. To obtain $P\ (L = l)$, the relation $P\ (L= l) = P\ (L \geq l) - P\ (L \geq l + 1)$ can be used. Note that $P\ (L \geq l)$ is the probability that the number of links traversed in the tree to meet the document is equal to l or higher. To calculate $P\ (L \geq l)$ let t denote the time into the interval $[0, \Delta]$ at which a request occurs. The random variable t is uniformly distributed over the interval, thus P $(L \geq l)$ is given by:

$$P\ (L\ \geq\ l) = \frac{1}{\Delta}\int_0^{\Delta} P\ (L\ \geq\ l\ |t = \tau\ )\ d\tau \quad (1)$$

Where$P\ (L\ \geq\ l\ |t = \tau\ )$ is the probability that no request from the M^l access caches arrives for document *i* in the interval $[0, \tau]$

$$P\ (L\ \geq\ l\ |t = \tau\ ) = e^{-M^l \lambda_A \cdot\ \tau} \quad (2)$$

From (1) and (2), the following equation can be obtained:

$$P\ (L\ \geq\ l) = \frac{1}{M^l . \lambda_A . \Delta}\left(1 -\ e^{-M^l \lambda_A \cdot\ \Delta}\right)$$

3.1.5.2 Transmission Time

In this section the transmission time to send a document in a caching hierarchy and in distributed caching is calculated.

The transmission time depends on L, the closest level with a copy of the document [31]:

The expected transmission time in hierarchical caching:

$$E[T_t^h] = \sum_{l\epsilon\{0,D,2D,2D+x\}} E[T_t^h \,|L = l].P(L = l)$$

The expected transmission time in distributed caching:

$$E[T_t^d] = \sum_{l=0}^{2D+x} E[T_t^d \,|L = l].\ P(L = l)$$

Where $E[T_t^h \,|L = l]$ and $E[T_t^d \,|L = l]$ are the expected transmission times at a certain network level for hierarchical and distributed caching, respectively and is given by:

$$E[T_t^h \big| l] = \frac{S}{C_l - \beta_l^h.S} .(1 - \frac{\beta_l^h.S}{2C_l})$$

for the delay at every network level in a caching hierarchy, and

$$E[T_t^d \big| l] = \frac{S}{C_l - \beta_l^d.S} .(1 - \frac{\beta_l^d.S}{2C_l})$$

for the delay at every network level in a distributed caching scheme.

Where $\beta_l^d$and $\beta_l^h$are the aggregate request arrival rate at every network level l; C_l is the service rate (or link capacity) at every network level; and S is the average document size.

The hit rates at every network level can be calculated using the popularity distribution of the different documents, (i.e. Zipf) and the distribution of L.

$$hit_l = \sum_{i=1}^{N} \frac{\lambda_{tot}}{\beta_{tot}}.P(L \leq l)$$

3.2 Cache Efficiency – Hit Ratio Variables

The ratio of objects served from a cache to the total number of requests is called the hit ratio. Hit ratio is based on several factors, including the number of objects available on the Internet, the size of the cache object store, the average size of an object, the expiration time of an object, the fraction of objects that can be safely cached and the popularity distribution of objects on the Internet [35].

Cache Storage Size: The larger the storage space the cache has, the more objects it can store and the more likely it is that a request will result in a hit. Storage size is measured in Gigabytes (GB).
Object Size: The smaller an object is, the more objects can be stored in a given cache. Object size is measured in kilobytes (KB). The average object size on the Internet is not accurately known, but is generally considered to be in the range of 3 to 10 KB [46].

Replacement Policy: Eventually the cache object store will be filled and a request will occur for a new object for which there is not room in the cache. Once the cache is filled, a decision has to be made on how to replace the cache contents with objects that are more likely to be useful. The common replacement policies include [35]:

- **Least Recently Used (LRU):** Replace the object that is stored in cache without a request for the longest time. Under this policy the cache will eventually fill with the most recently requested objects.
- **First In, First Out (FIFO):** Replace the oldest object, based on when it was first stored in the object cache. The cache will eventually fill with the most recently refreshed objects.
- **Least Frequently Used (LFU):** Replace the object that has had the fewest requests, or lowest request rate, since it was first stored. The cache will eventually fill with the most frequently requested objects.
- **Next to Expire (NTE):** Replace the object that is forecast to expire soon. The cache will eventually fill up with the most stable (infrequently changed) objects.
- **Lowest Relative Value (LRV):** Replace the document with the lowest value where the calculation of the value takes into account the locality, cost and size of the document [52].
- **Hybrid:** Replace the document with the smallest function value which is computed for each document. The function considers the time to connect with server, the bandwidth to server, the number of times the document has been requested, and the size of the document [53].
- **SIZE (**also called Largest File First)**:** Replace the largest object, hoping to free up the most space for new objects. Often both size and age are considered in choosing the object to be replaced. The cache will tend to fill with the smallest objects [44].

Expiration Time: It is important that a cache deliver up to date information, even though objects on the Internet change. Some object, like stock quotes, news headlines and weather reports, change often and are not suitable for caching. Others, like daily news letters, quote of the day, and status reports change approximately once per day and can be cached if attention is paid to their expiration times. Many other objects, such as images and published reports, do not change, even over period of years. These are excellent candidates for caching.

The Hyper Text Transfer Protocol (HTTP) provides at least two mechanisms that help a cache to determine the likely expiration time for an object. The first is the "expires" field of the header. The server can provide a specific date and time after which this object is considered stale and needs to be refreshed from the origin server. The second is the "last modified time" field. This is the time when the object was either created or was lasted changed. If the object has been unchanged for a long time, it is reasonable to assume it will not be changing in the very near future.

Size of the Internet: The precise number of objects available on the Internet is not precisely known. The estimates of the size of the internet range from 100 million to more than a billion. This number is very important for estimating hit ratios if a uniform popularity model is assumed, but becomes relatively unimportant if a Zipf distribution popularity model is assumed.

Cacheability Percentage: Internet protocols provide a variety of mechanisms to allow objects to be marked as "not to be cached". These include immediate expiration times, use of the cache-control header options, use of "cookies" and dynamic pages [20].

Estimates of the fraction of Internet objects that are non-cacheable vary. If a non-cacheable object is requested, it cannot result in a cache hit. The non-cacheable percentage directly affects the cache hit ratio observed according to the following formula [35].

$$p(hit) = p(object\ is\ cacheable) \times p(object\ is\ in\ the\ cache)$$

Popularity distribution: The total number of objects in Internet is very large and not accurately known. In analyzing cache performance what is important is the size of the interest set. This is the total collection of objects that the user community will ever request. This is much smaller than the total Internet.

Some Internet objects are requested much more often than others. Attempts to model this popularity distribution have focused on two probability distributions [47]. The uniform distribution assumes that each object is equally likely to be requested. This simple assumption is not accurate. Another distribution, called the Zipf distribution can provide a more realistic model. A Zipf distribution is one where the probability of selecting the i^{th} most popular item is proportional to $1/_i$ [48].

Assuming a uniform distribution popularity model, any object is equally likely to be read. If a cache can hold 6 million objects, the probability of a requested page being in the cache for various assumptions about the number of objects in the Internet, is as follows. The assumptions

include: the total number of objects in the internet is fixed, the average object size is known, the objects are equally likely to be read (i.e. uniform popularity distribution), all requested objects are cacheable, and the effects of traffic to revalidate objects and refresh expired objects are ignored.

Table 3-1: The Probability of being in the Cache for Uniform distribution

Internet Size	Probability of Being in cache
10,000,000	60.00%
100,000,000	6.00%
1,000,000,000	0.60%

As can be seen in the above table, for a large Internet (e.g. more than 100 million objects) the maximum achievable cache hit ratio is quite low, below 6%.

The uniform model is generally considered an inaccurate representation of Internet request popularity because it provides for too broad a locality of reference[20].

Assuming a Zipf distribution, the probability of an object being requested is proportional to the rank of that object. If a cache can store 6 million objects the probability of a requested page being in the cache is simply the cumulative distribution function value evaluated at 6 million [35]. For such a cache, the probability of a requested page being in the cache for various assumptions about the number of objects in the Internet is as follows:

Table 3-2: The Probability of being in the Cache for Zipf distribution

Internet Size	Probability of being in Cache
10,000,000	96.8%
100,000,000	84.7%
1,000,000,000	75.3%

Note that the size of the Internet has a relatively small effect on the value of the hit ratio.

3.3 Estimating Hit Ratios

The actual hit ratio of a Web cache depends on several variables of the workload and of the cache design itself. As listed in the preceding section, the factors include the number of objects available on the Internet, the size of the cache object store, the average size of an object, the expiration time of an object, the fraction of objects that can be accurately cached (cacheability), and the popularity distribution of objects on the Internet. But if all these variables are used for the estimation of hit ratio, it will be very complicated. Thus, reasonable assumptions will be used to reduce hit ratio estimation to a relatively simple calculation[20].

The class of Zipf-like distribution models the frequency of occurrence of some event, *P*, as a function of the rank *i*, as a power-law function $Pi \sim 1/i^{\alpha}$ with the exponent α close to unity. Studies suggest that the popularity distribution of the Internet can be modeled by a Zipf-like distribution with the exponent α in the range of 0.5 to 0.7 [11]. An approximation of the sum of the first *n* elements of the distribution can be shown to be:

$$\sum_{i=1}^{n} 1/i^{\alpha} \approx \int_{1}^{n} 1/x^{\alpha}\, dx = \frac{x^{(1-\alpha)}}{(1-\alpha)} = \frac{n^{(1-\alpha)}}{(1-\alpha)} \; for\; \alpha < 1$$

Taking the ratio of this term to the total number of objects in the interest set gives the probability that an object selected will be one of the *n* most popular. Note that this collapses to the uniform distribution when α= 0.

This provides a simple formula for computing Web cache hit ratio. The probability of selecting any one of the *k* objects from population of *n* objects, representing a cache storing *k* items from an Internet of *n* objects is simply

$$k^{(1-\alpha)}/n^{(1-\alpha)} = (k/n)^{(1-\alpha)} \; for\; \alpha < 1$$

This formula is relatively insensitive to assumptions about the total number of objects making up the Internet. Figure 3-3 shows the probability of a cache hit as a function of cache object size for several choices of α [20], assuming that the Internet has one hundred million objects in the interest set, the average object size is 5 KB, and all objects are cacheable. Higher values of α, perhaps 0.7 or 0.8,correspond to users with common interests. Workgroups or people at a particular enterprise location may have this popularity distribution. Lower numbers of α, perhaps 0.5 or 0.6, correspond to less common interest. This may be accurate for ISP users served by an

interception Web cache. In any case, the result read from the graph has to be reduced by the cacheability percentage.

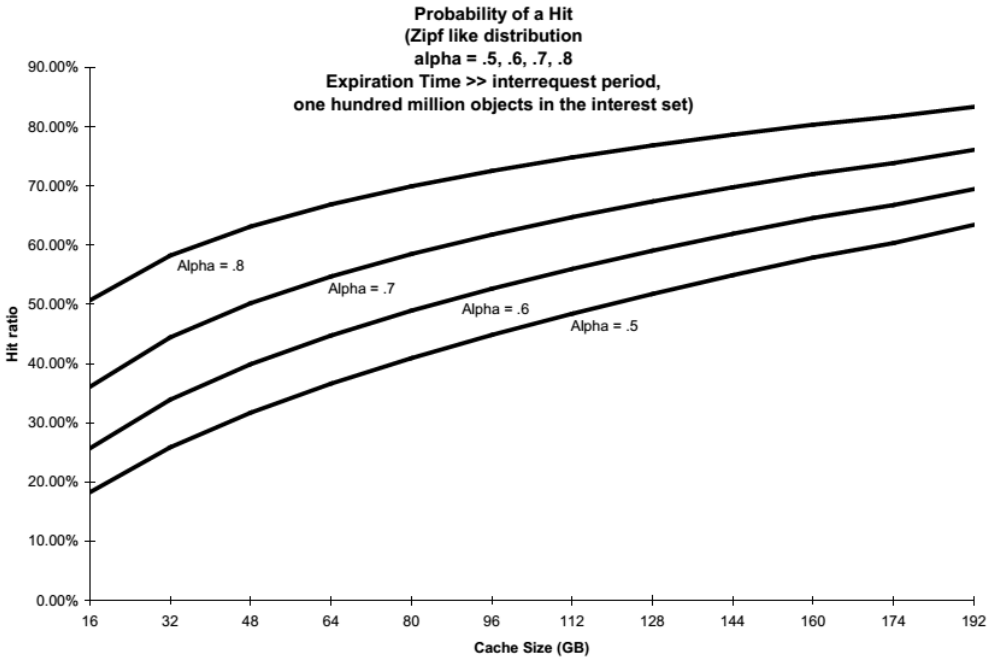


Figure 3-3: The probability of a cache hit as a function of cache object size

3.4 Simulation Tool

The simulator [51]selected for this study, named simply web cache-simulator, is written in C programming language and can be used to simulate non-uniform size web document caches. The simulator can simulate different cache replacement algorithms including LRU, SIZE, LRV, Hybrid and variations of GreedyDual algorithms [45].It takes input a text file describing each HTTP requests, calculates the hit ratio and byte hit ratio under an infinite-sized cache, and then calculates the hit ratio and byte hit ratio for each algorithm, under cache sizes being various percentages of the total data set size. The command line for the simulator is:

uniform request_count inputfile outputfile

Table 3-3: Description of Items in the Command Line

Item	Definition
uniform	is the name of the "main" in the c source file, i.e. uniform.c
request_count	is the number of requests in the input file
inputfile	is the name of the input text file
outputfile	is where the simulator will write results to

The request_count doesn't have to be the exact number of requests in the inputfile --- the simulator will read up to "request_count" number of requests from the input file. The argument is used by the simulator to pre-allocate a number of data structures used in the simulation.

The input file should be an ASCII text, each line describing an HTTP request.

Table 3-4: Format of the Input File to the Simulator

1st Column	2nd Column	3rd Column	4th Column	5th Column	6th Column
webserver_uid	urlpath_uid	file_size	latency	last_modified_time	access_sec

Table 3-5: Description of Items in the Input File

Item	Definition
webserver_uid	Unique IDs that specifies the Web server address of the request
urlpath_uid	Unique IDs that specifies the path of the URL of the request
file_size	The size of the accessed document, in bytes
latency	The duration of the HTTP request, in microseconds
last_modified_time	The last modified timestamp in the HTTP reply header.
access_sec	The timestamp in seconds of the HTTP request.

Here is an example of the lines in the input file:

1 1 250 180280 0 785431250

2 516 475 184105 0 785431765

The output contains lines showing the performance of each cache replacing algorithm under various metrics, for example:

```
Cache Size = 199625.000000 5.000000%
```

`5.00 LRU:Pure-Hit Rate=0.337222 0.758750`(Where the Pure-Hit Rate is the number of requests fulfilled by the cache as the percentage of the total requests from the users)

`5.00 LRU:Byte-Hit Rate=0.278333 0.626250`(Where the Byte-Hit Rate is the number of bytes fulfilled by the cache as the percentage of the total number of bytes requested by users)

show that when the cache size is 5% of the size of total data set size (i.e. the cache size needed to avoid capacity misses), LRU cache replacing algorithm yields a hit ratio of `0.337222`, which is `75.8750%` of the hit ratio of the infinite cache, and it yields a hit ratio of `0.278333`, which is `62.6250%` of the hit ratio under an infinite cache.

In addition, the beginning of the output file lists statistics like total request count, total byte count, total unique requests and total dataset size. It then lists hit ratio, byte hit ratio and reduced latency under infinite cache. After that it lists the performance of each cache replacing algorithm under various cache sizes. The performance metrics tracked by the simulator include: hit ratio, byte hit ratio, reduced latency, reduced network packets, reduced hops and reduced weighted hops.

Here is how the program works: "main" is in uniform.c. It first parses command arguments, then calls InfCache to read all requests into an array called "Requests", and to calculate the performance metrics under the infinite-sized cache. It then sets cache size to certain percentages of the data set size (calculated in InfCache), and calls the routine for each cache replacing algorithm.

Each cache replacing algorithm has three routines, for example LRU has LRU(), LRU_Add(), and LRU_Evict() routines. The LRU() is the main routine which reads the requests from the "Requests" array, checks each request to see whether the copy of the requested item is within the cache or not, if the item is already stored in the cache and can be served to the user, LRU() updates the hit rate, byte hit rate and related statistics for this particular cache replacing algorithm. However, if the item is not found within the cache and the cache has free spaces to accommodate additional items, LRU_Add() is called to add the copy of the new item into the cache, and at the same time the miss statistics related to this algorithm is updated. But if the cache doesn"t have the copy of the item and couldn"t add any more requests because of space limitation; the replacement routine, LRU_Evict() is called so as to evict the item without a request for the longest time while simultaneously updating the miss statistics.

This simulator tool and the analytical model basically have comparable computations for the hit-rate. They both result in similar results when the web cache simulator is computed under an infinite cache size. The detail description of each numerical parameter (pure-hit rate, byte hit rate…) can be found in section 4.3, under Simulator Results.

CHAPTER FOUR

4 Results and Analysis

In this chapter analytical and simulator methods are used to compare the performances of different web caching schemes. These results are then analyzed in order to meet the aim and objective of the study.

4.1 Numerical Comparisons

In this section the following typical values were picked for the different parameters in the model to acquire some quantitative results [54].

- The nodal outdegree of the network tree model: $M = 4$.
- The distance between caching levels: $D = 3$, (thus, there will be $64[M^D]$ distribution and $4096[M^{2D}]$ access caches)
- The distance from the top node of the core network to the origin server: $x = 10$.
- Number of Web documents: $N = 250$million (which are distributed following a Zipf distribution)
- The documents update time: $\Delta = 24h$.
- The total network traffic: $\beta_{tot} = 1000$ document req/s.
- The average document size of a Web document [38]: $S = 15Kb$.

4.1.1 Connection Time

The connection time is the first part of the perceived latency when retrieving a document and depends on the network distance to the document. Figure 4-1 shows the connection time for distributed and hierarchical caching for different document"s popularity computed using the expressions shown in Section 3.1.5.1.

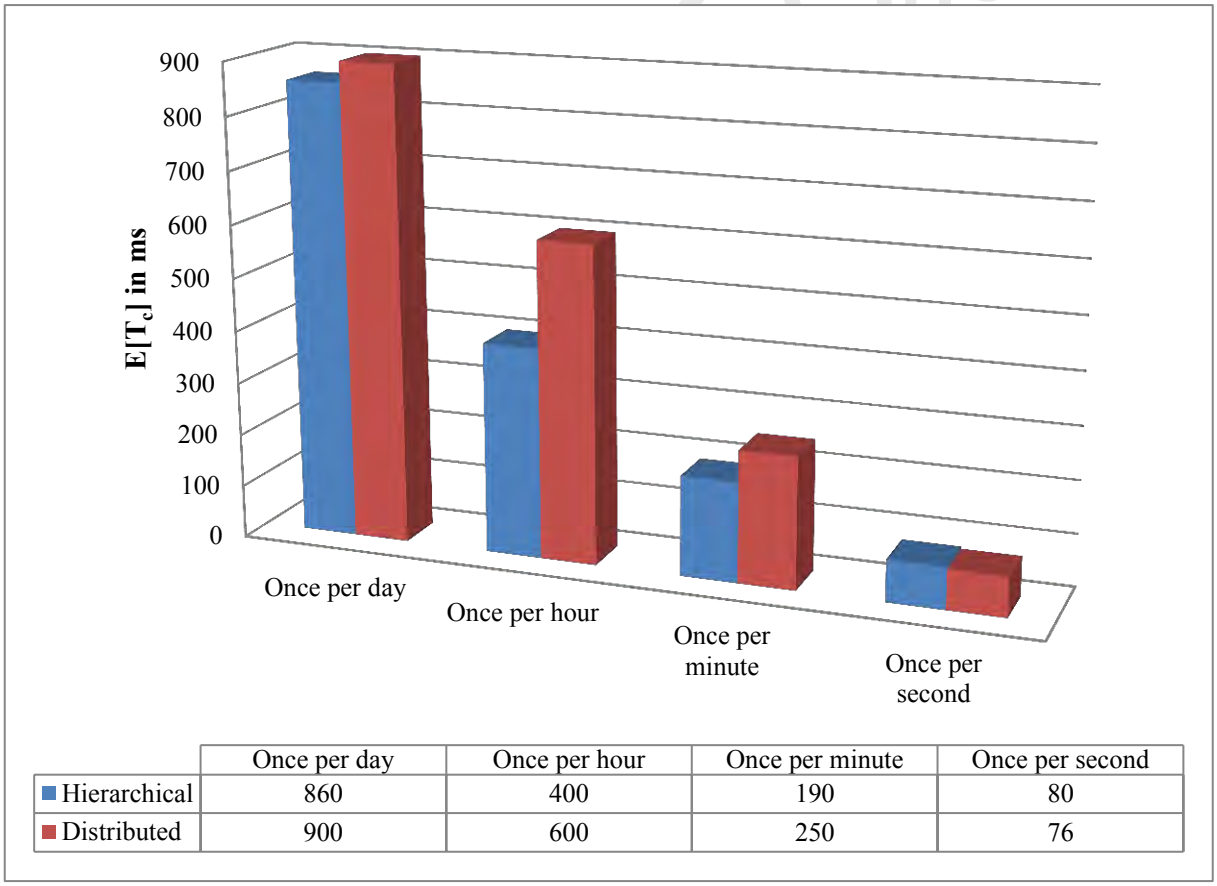
The expected connection time in hierarchical caching:

$$E\left[T_C^h\right] = 4d \sum_{l\epsilon\{0,D,2D,2D+x\}} P(L = l) * (l + 1)$$

The expected connection time in distributed caching:

$$E\left[T_C^d\right] = 4d \sum_{l=0}^{2D} P(L = l) * (2l + 1) + 4d * P(L = 2D + x) * (2D + x + 1)$$

As can be seen in the figure, for a very unpopular document (small λ_{tot}) both hierarchical and distributed caching experience high connection times (around 900ms) because the request needs to travel to the origin server. As the number of requests for a document increases, the average connection time decreases since there is a higher probability to hit a document at closer caches than the origin server. For all the documents which λ_{tot} ranges between one request per day and one request per minute, a hierarchical caching scheme gives shorter connection times than a distributed caching scheme. Document copies placed at the distribution and the core caches in a caching hierarchy reduce the expected network distance to hit a document. When the document is very popular a distributed caching scheme can benefit from close neighboring copies, reducing the connection time. In a hierarchical caching scheme, the document still has to be fetched from the distribution cache in case of a miss at the access cache [31].



	Once per day	Once per hour	Once per minute	Once per second
Hierarchical	860	400	190	80
Distributed	900	600	250	76

Figure 4-1: Connection Time as a function of the request rate.

4.1.2 Transmission Time

The second part of the overall latency is the time it takes to transmit the Web document. Using the expression obtained in Section 3.1.5.2, the transmission time of hierarchical caching can be compared against the distributed caching system.

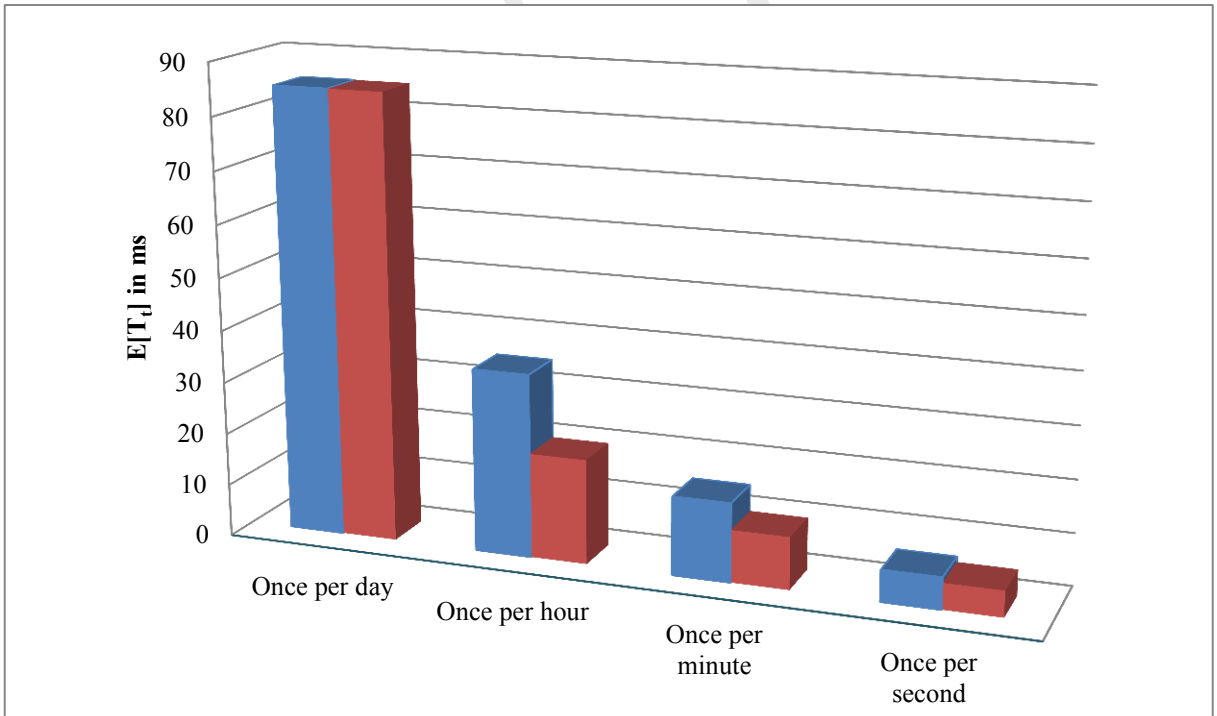
The expected transmission time in hierarchical caching is:

$$E[T_t^h] = \sum_{l\epsilon\{0,D,2D,2D+x\}} E[T_t^h \,|L = l].P(L = l)$$

The expected transmission time in distributed caching is:

$$E[T_t^d] = \sum_{l=0}^{2D+x} E[T_t^d \,|L = l].\ P(L = l)$$

In Figure 4-2, it can be observed that the increase in the transmission time is much higher (*i.e. 15ms, 5ms and 1ms* at request rates of *once per hour, once per minute and once per second,* respectively) for hierarchical caching than for distributed caching. Distributed caching gives shorter transmission times than hierarchical caching because requests travel through lower network levels. This is because many requests are satisfied and most of the traffic flows through the less congested lower network levels.

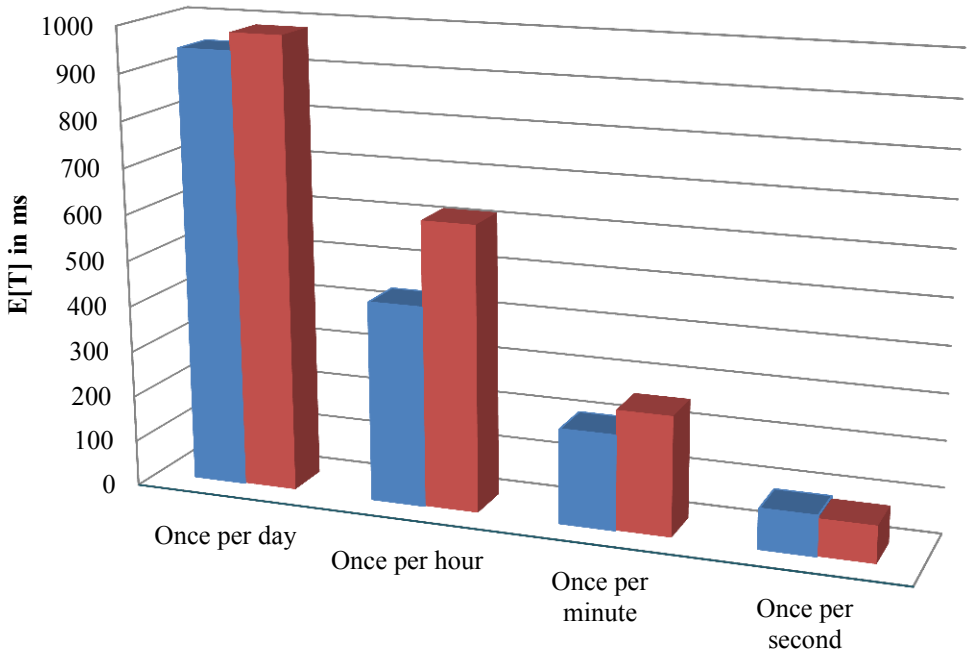


	Once per day	Once per hour	Once per minute	Once per second
Hierarchical	85	35	15	6
Distributed	85	20	10	5

Figure 4-2: Transmission Time as a function of the request rate.

4.1.3 Total Latency

The total latency is the sum of the connection and the transmission times described in Sections 4.1.1 and 4.1.2, respectively. For large document sizes, the transmission time is more relevant than the connection time. For small document sizes, the transmission time is very small and the connection time has a higher relevance. Distributed caching gives lower latencies for higher documents because distributed caching has lower transmission times than hierarchical caching [56]. The threshold for the document size depends on the degree of congestion in the core network. The higher the congestion, the lower is the size-threshold from which distributed caching has lower latencies than hierarchical caching. Distributed caching can decrease the retrieval latency of large documents and reduce the bandwidth usage at high network levels [31]. However, full deployment of distributed caching encounters several problems. Given that in a distributed caching scheme documents are retrieved from neighbor access caches, the experienced latency depends not only on the bandwidth of the requesting access cache, but also on the bandwidth of the neighbor cache that is contacted. In order to increase the local hit rate, institutes might want to increase the disk space of their cache.



	Once per day	Once per hour	Once per minute	Once per second
Hierarchical	945	435	205	86
Distributed	985	620	260	81

Figure 4-3: Total latency as a function of the request rate.

4.2 A Hybrid Caching Scheme

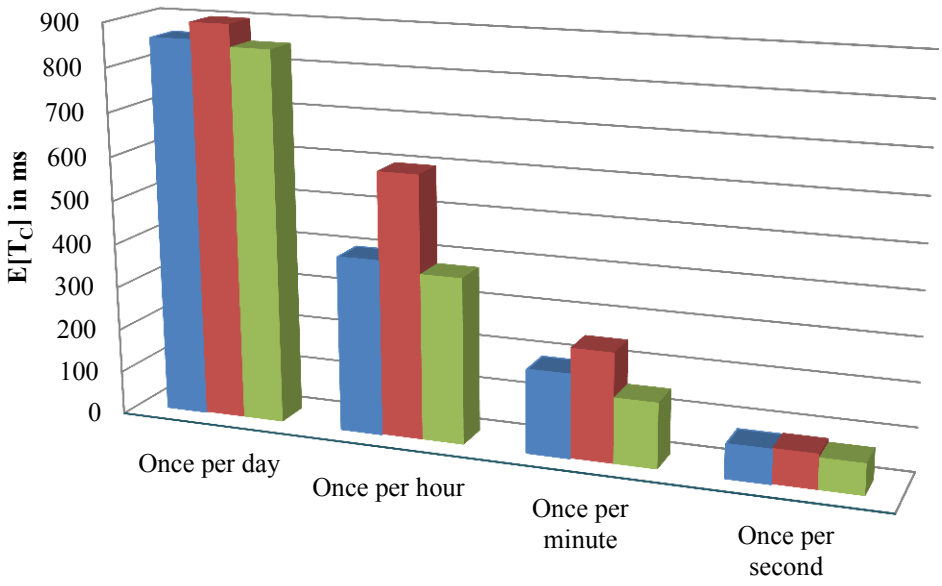
In previous sections it is observed that hierarchical caching gives shorter connection times than distributed caching and that distributed caching gives shorter transmission times than hierarchical caching. In this section the third web caching scheme, hybrid, is considered. In a hybrid caching scheme a certain number of caches cooperate at every network level of the caching hierarchy. Here, the influence of the number of cooperating caches at every level *k*, on the total retrieval latency is considered. It is found that there is an optimum number of cooperating caches at every cache level that minimizes the total latency [55]. This optimum number of cooperating caches depends on the network congestion, the parent cache congestion and the size of the document.

4.2.1 Connection Time

The connection time in a hybrid scheme depends on the number of cooperating caches at every cache level. The probability that a document is found among *k* cooperating access caches, which range from 1 (no cooperation) to $M^D = 64$ (all neighboring caches in the same cache level), is calculated using the formula in Section 3.2.5 by replacing the M^l term by k_c.

$$P\,(L \geq l) = \frac{1}{k_c.\,\lambda_A.\,\Delta}\left(1 - e^{-k_c.\lambda_A.\,\Delta}\right)$$

To find the value of k_c, the connection time is computed using this formula for the whole range of possible number of cooperating caches at the access network (i.e. 1 to 64). Then, the value that results the lowest connection time is chosen to be the optimal number of cooperating caches, k_c. Figure 4-3 presents the connection time for a hybrid scheme with the optimum number of cooperating caches k_c for distributed caching, and for hierarchical caching. The curve shows that a hybrid scheme with k_c cooperating caches at every cache level has always lower connection times than distributed caching and even lower connection times than hierarchical caching for a large range of documents‟ popularities. $k_c = 4$ is the number of cooperating caches in hybrid caching that gives the minimum connection time computed using the formula for the connection time provided in the previous section.

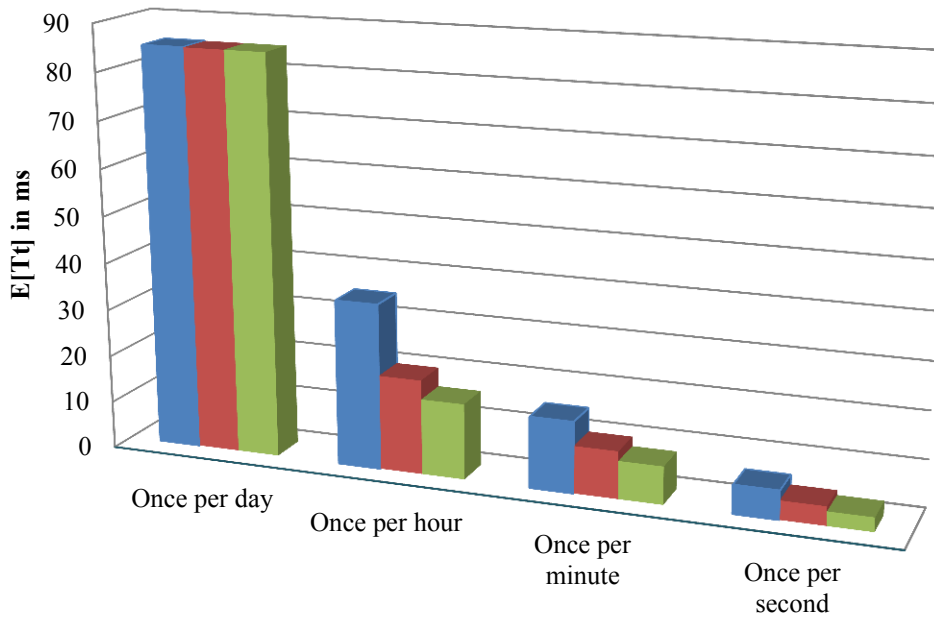


	Once per day	Once per hour	Once per minute	Once per second
Hierarchical	860	400	190	76
Distributed	900	600	250	80
Hybrid, k=4	850	380	150	72

Figure 4-4: Connection time in hybrid scheme (k_c=4)

4.2.2 Transmission Time

Similar to the connection time, the optimum number k_t of cooperating caches at every network level that minimize the transmission time can be found. In Figure 4-4, the transmission time for a hybrid scheme with the optimum number of cooperating caches k_t, for distributed caching, and for hierarchical caching is presented. It is observed that a hybrid scheme with k_t cooperating caches at every cache level has lower transmission times than hierarchical caching and even lower transmission times than distributed caching. k_t=16 is the number of cooperating caches in hybrid caching that yields the minimum transmission time computed using the formula for the transmission time provided in the previous section.

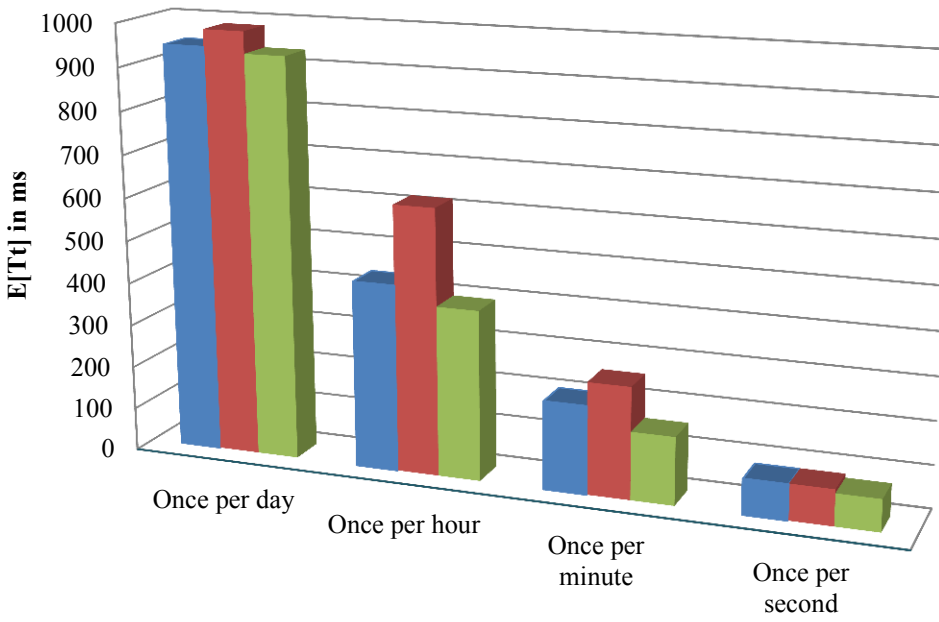


	Once per day	Once per hour	Once per minute	Once per second
Hierarchical	85	35	15	6
Distributed	85	20	10	4
Hybrid, k=16	85	16	8	3

Figure 4-5: Transmission time in hybrid scheme (k_t=16)

4.2.3 Total Latency

Depending on the document size, the connection time or the transmission time are more relevant on the total retrieval latency. For small document sizes the connection time is more important than the transmission time. For large document sizes, the connection time is almost negligible and the transmission time accounts for a big percentage of the whole latency. Thus, there is an optimum number of caches that should cooperate at every cache level to minimize the total latency, which depends on the document size. For small documents the optimum number of cooperating caches is close to k_c, since choosing k_c cooperating caches minimizes the connection time [21]. For large documents the optimum number of cooperating caches is close to k_t, since choosing k_t cooperating caches minimizes the transmission time [21]. For any document size, the optimum number of cooperating caches k_{opt} that minimizes the total retrieval latency is such that $k_c \leq k_{opt} \leq k_t$.



	Once per day	Once per hour	Once per minute	Once per second
Hierarchical	945	435	205	82
Distributed	985	620	260	84
Hybrid, k=16	935	396	158	75

Figure 4-6: Total latency in hybrid scheme

4.3 Simulator Result

The detail output of the simulator for the four sample request distributions can be found at the end of this document in Appendix section.

Table 4-1: Summary of the Sample Requests

Parameters	Sample One	Sample Two	Sample Three	Sample Four
Total Requests	1800	2000	1400	1400
Total Unique Requests	1000	1000	1000	1000
Total Bytes (in MB)	5.91	6.13	5.16	8.48
Total Unique Bytes (in MB)	3.99	3.99	3.99	7.69
Total Packets	13778	14547	11771	17967
Total Unique Packets	8950	8950	8950	15860

Parameters	Sample One	Sample Two	Sample Three	Sample Four
Average Connection Time	195317	195496	194254	180622
Number of Small Pages (Percentage of Small Pages)	737 (40.9%)	939 (47.0%)	429 (30.6%)	300 (21.4%)
Total Hops	1800	2000	1400	1400
Total Unique Hops	1000	1000	1000	1001
Total Weighted Hops	13778	14547	11771	17967
Total Unique Weighted Hops	8950	8950	8950	15860
Hit Rate	0.444444	0.500000	0.285714	0.284286
Byte Hit Rate	0.324563	0.349358	0.227383	0.092455
Reduced Packets	0.350414	0.384753	0.239657	0.117271
Reduced Latency (in ms)	0.415580	0.465568	0.270386	0.277090
Reduced Hops	0.444444	0.5	0.285714	0.284286
Reduced Weighted Hops	0.350414	0.384753	0.239657	0.117271

4.3.1 Pure-Hit Rate

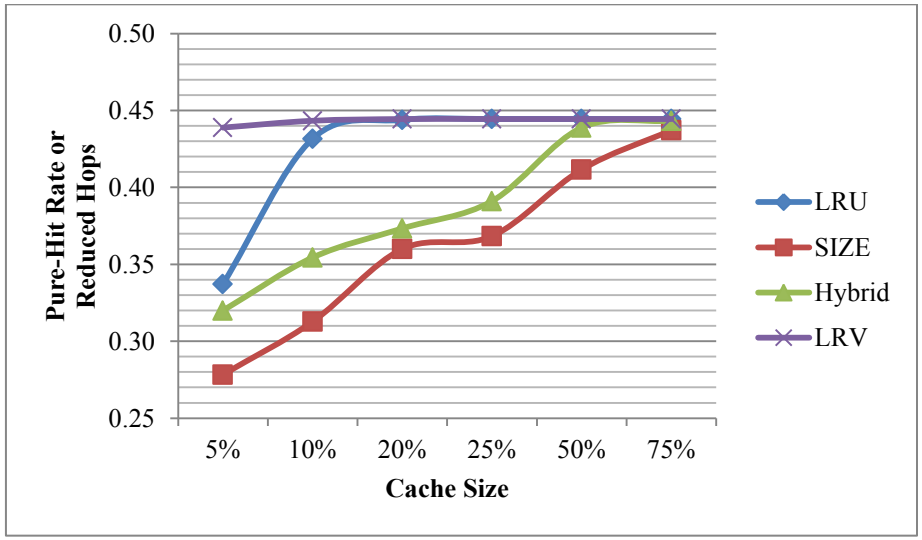


Figure 4-7: Hit Rate for Sample One

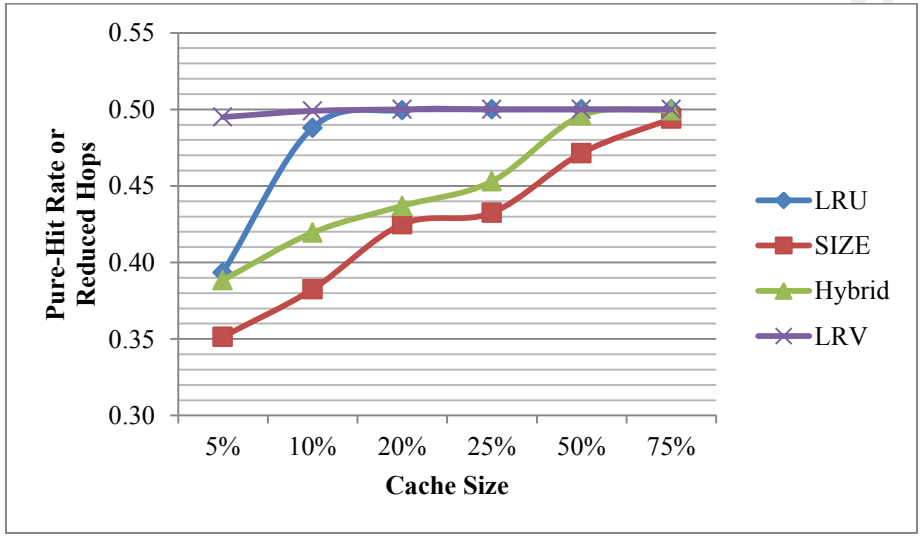


Figure 4-8: Hit Rate for Sample Two

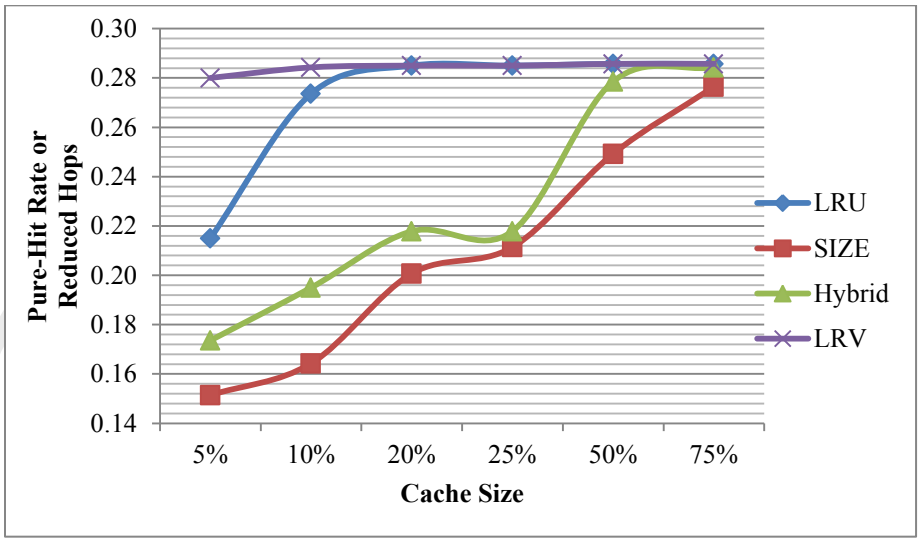


Figure 4-9: Hit Rate for Sample Three

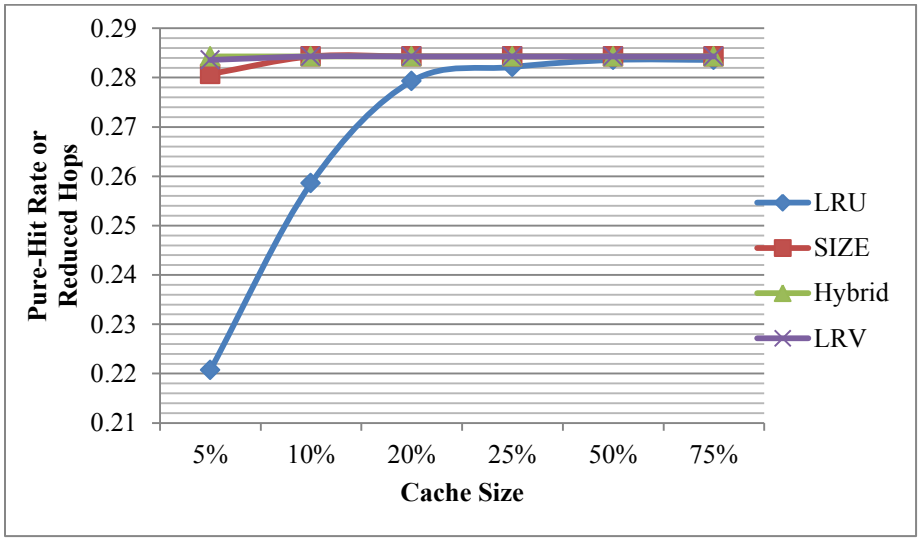


Figure 4-10: Hit Rate for Sample Four

4.3.2 Byte-Hit Rate

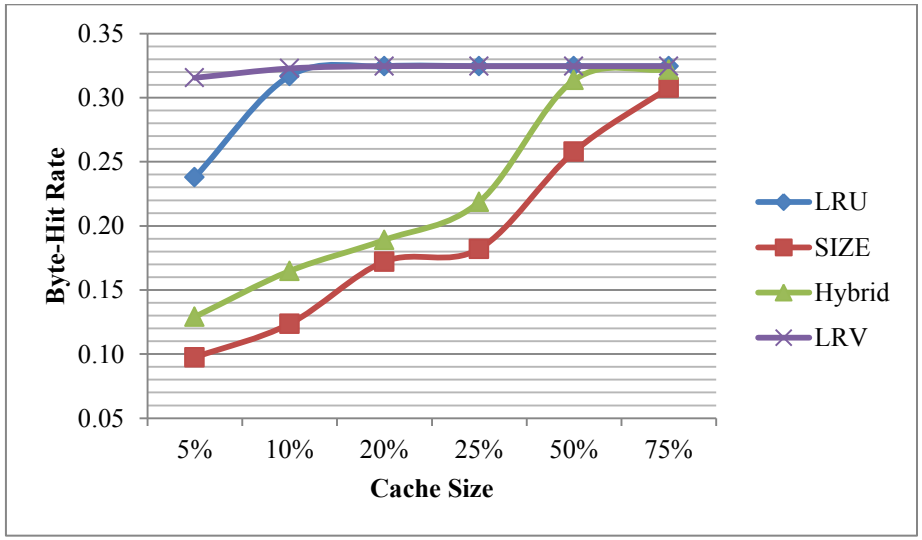


Figure 4-11: Byte-Hit Rate for Sample One

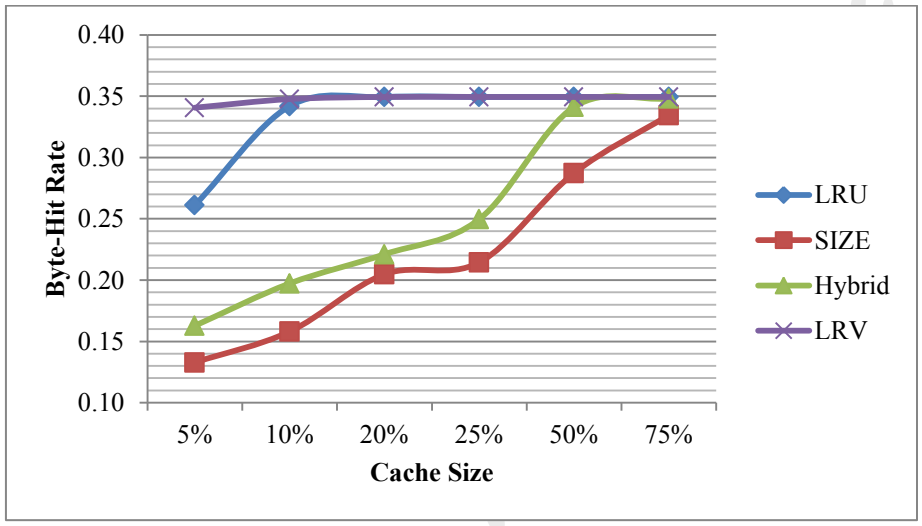


Figure 4-12: Byte-Hit Rate for Sample Two

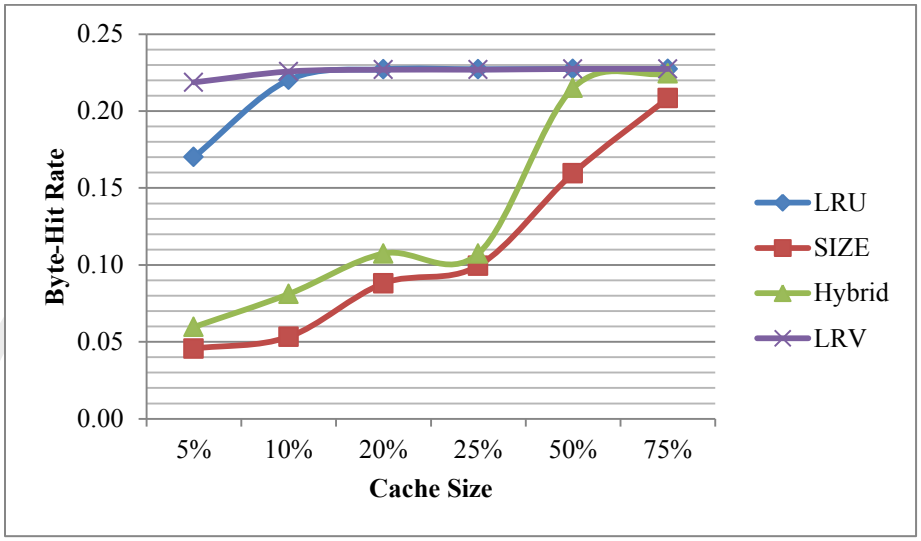


Figure 4-13: Byte-Hit Rate for Sample Three

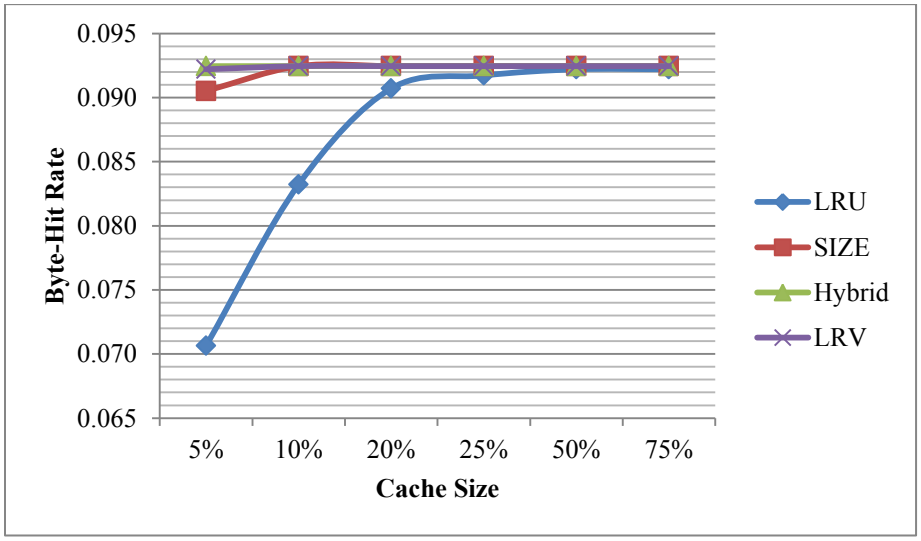


Figure 4-14: Byte-Hit Rate for Sample Three

Figure 4-8 to Figure 4-10 show the Pure-Hit Rate as function of Cache Size while Figure 4-11 to Figure 4-14 are Byte-Hit Rates as function of the percentages of the total data set size for the four web client request samples. The cache sizes being various percentages (from 5% to 75%) of the total data set size needed to avoid capacity misses. The simulator computes the Pure-Hit Rate as the ratio of the number of requests fulfilled by the cache to the total number of requests and the Byte-Hit Rate as the ratio of the number of bytes fulfilled by the cache to the total number of bytes requested by users. The simulator assumes that if there is a hit within the cache, the number of hops is reduced with the same amount, resulting equal values for the hit rate and reduced hops.

The cache replacing algorithms used in the study include LRU, SIZE, Hybrid and LRV. The LRU, which evicts objects without a request for the longest time, reaches the hit rate and byte-hit rate values of the respective web request samples under infinite cache size when the percentage of the total data set size is one-tenth and one-fifth for the first three samples and for the last sample of web client requests, respectively.

The SIZE algorithm which replaces the largest object when the cache is full, show similar increase in the hit rate and byte-hit rate as the cache size increases for the three sample web requests. For the last sample web request, however, SIZE yields higher and stable hit rate and byte-hit rate values closer to the respective values under an infinite cache size. This is because the fourth web request sample has more percentage of larger objects that can be removed to free up spaces for new requests as compared to the other web request samples.

The other replacing algorithm Hybrid, considers the time to connect with server, the bandwidth to server, the number of times the document has been requested, and the size of the document. However, the simulator simplifies these parameters and only number of request and size of the document are considered. Thus, Hybrid shows similar trend with the SIZE replacing algorithm for all sample web requests. The additional parameter, number of request used in Hybrid, made this algorithm to yield better hit rate and byte-hit rate than its SIZE counterpart.

The last replacing algorithm LRV shows superior performance over the other replacing algorithms for the whole range of cache sizes and for all sample web requests. This is because it takes into account the locality, cost and size of the document in determining which object is to be removed within the cache when the cache is full.

4.3.3 Reduced Packet

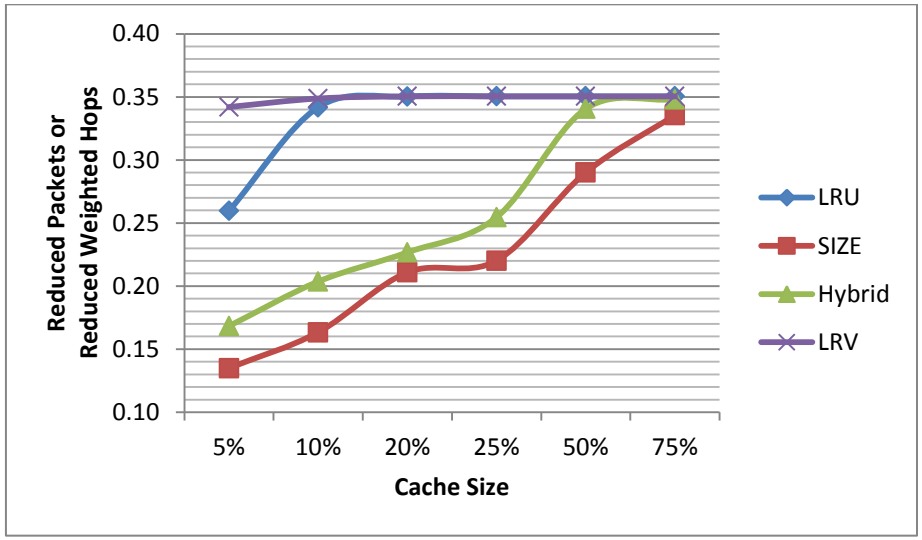


Figure 4-15: Reduced Packet for Sample One

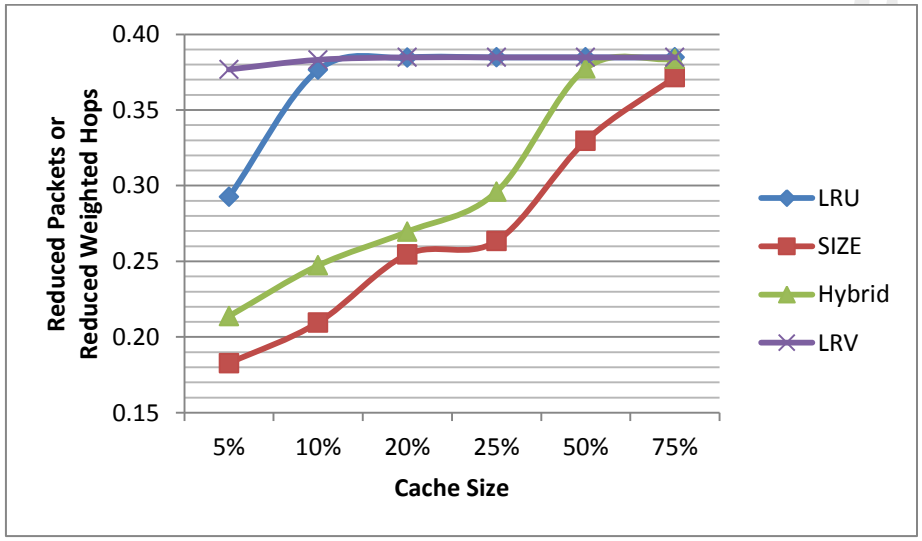


Figure 4-16: Reduced Packet for Sample Two

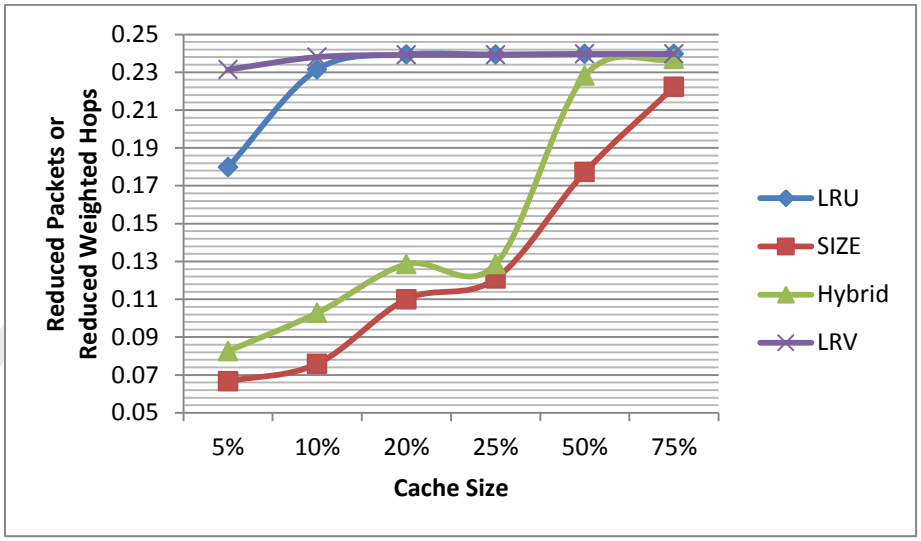


Figure 4-17: Reduced Packet for Sample Three

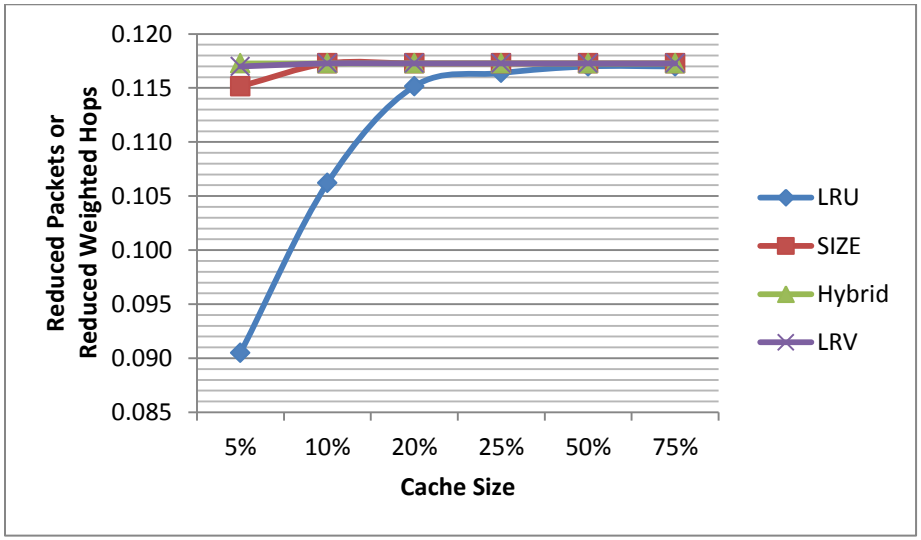


Figure 4-18: Reduced Packet for Sample Four

4.3.4 Reduced Latency

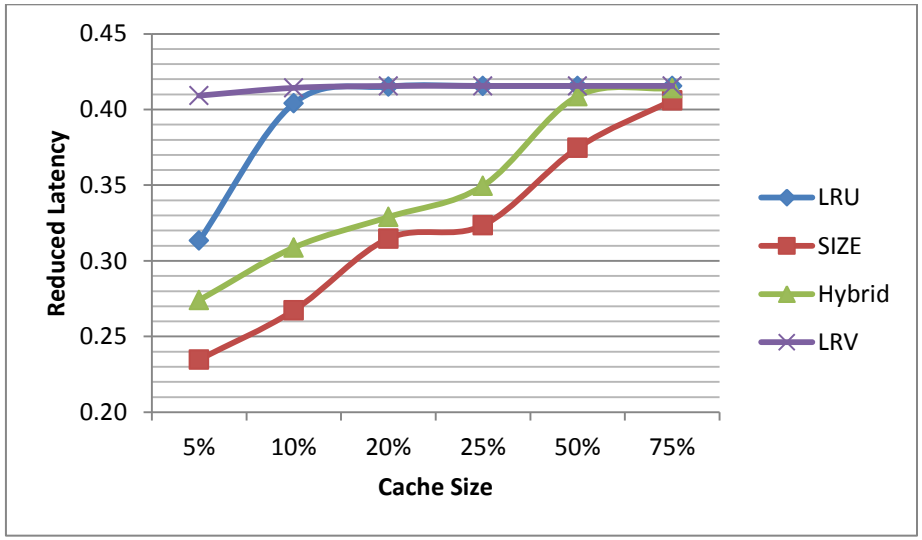


Figure 4-19: Reduced Latency for Sample One

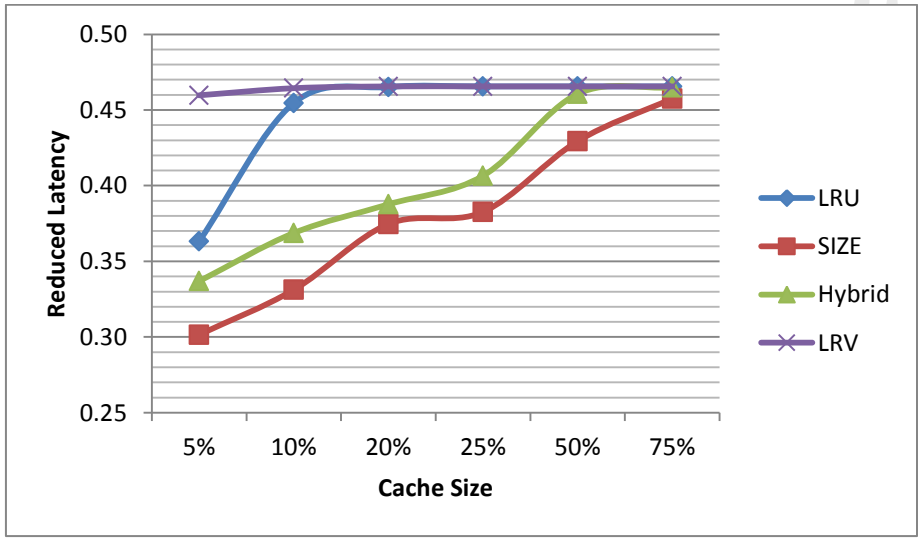


Figure 4-20: Reduced Latency for Sample Two

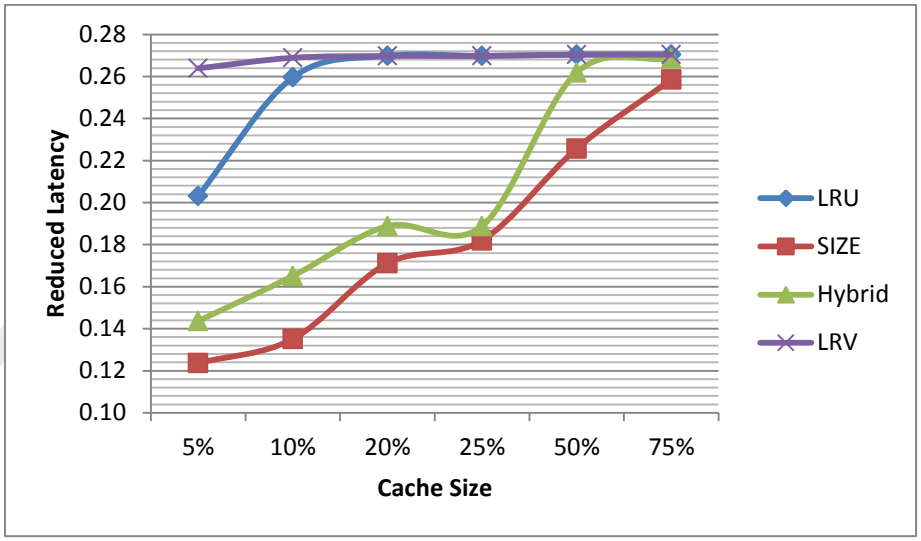


Figure 4-21: Reduced Latency for Sample Three

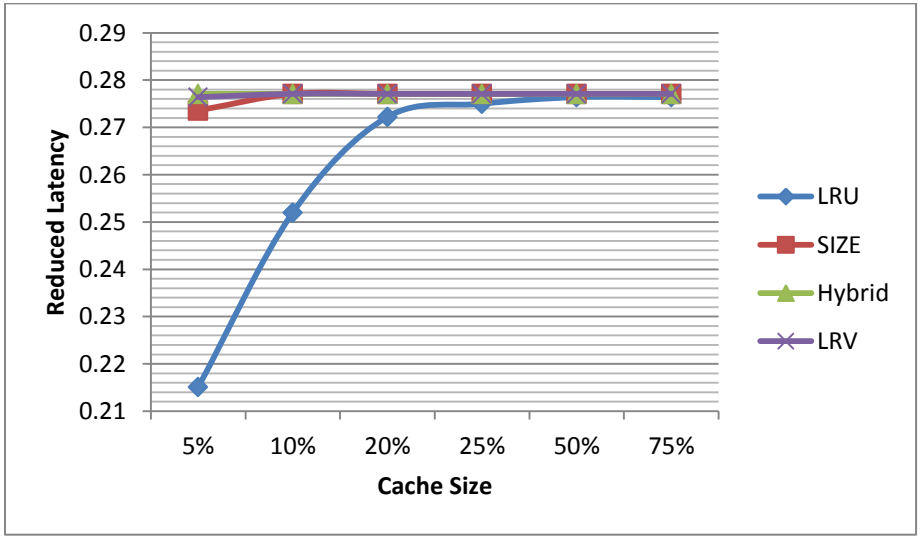


Figure 4-22: Reduced Latency for Sample Four

Figure 4-15 to Figure 4-22 show the Reduced Packet and Reduced Latency as function of the percentages of the total data set size for the four web client request samples. The Reduced Packet is calculated as the ratio of the number of redundant packets to the total number of packets within the request. Whereas, Reduced Latency is the reduction in total latency from cache hits by assuming a cache hit results in zero or no latency.

As the cache size increases both reduced packet and reduced latency increases. This is because there is enough space for the new requests within the cache so as to avoid capacity misses. This increases the probability of a document to be a hit in the cache, avoiding the request from traveling to the origin server which reduces the latency.

The same cache replacing algorithms used in hit rate and byte-hit rate are applied to compare the reduced packets and reduced latency. The output of the simulator show similar trends or as was observed in hit rate and byte-hit rate except for different values of the respective numerical parameters. Here as well the LRV replacing algorithm show high Reduced Packet and Reduced Latency as compared to the other the other replacing algorithms for all sample web requests.

4.3.5 Discussion

To summarize the results of the simulator obtained in this section, there is a high correlation between the performance of a cache replacing technique and the distribution of the web client request. Thus, different cache replacing algorithms yield different level of performances when they are applied for different web client requests. In addition pure-hit rate, byte-hit rate, reduced packets and reduced latency are observed to increase in a similar fashion as the cache size increases.

As can be observed from the curves, the outputs of the first three samples show similar trends for all numerical parameters considered in this paper. The output for the fourth sample, however, shows relatively different curve as compared to the respective parameter of the other samples. This is because the number of small size objects (i.e. less than 1024 bytes) is lesser in this sample. This is intentionally made to see how the size of the accessed item affects the performance of cache replacing algorithm. The results indicated that it is mandatory to closely examine the characteristics of the web request before choosing the type of web replacing algorithm.

CHAPTER FIVE

5 Conclusion and Future Work

5.1 Conclusion

In this study, analytical model is used to analyze the performance of hierarchical and distributed and caching systems in terms of transmission and connection times. The analytical model result shows that distributed caching gives longer connection times than hierarchical caching. This is expected because the cached documents are usually far from the clients which increase the average number of hops to satisfy the clients‟ requests and longer distance means longer time to travel, increasing the connection time. On the other hand, the results also showed that hierarchical caching gives shorter connection times. This benefit can be attributed due to the fact that document copies placed at distribution and core caches in a caching hierarchy reduce the expected network distance to hit a document.

Also, the results from the analytical model show that distributed caching gives shorter transmission times than hierarchical caching. This advantage of distributed caching over its hierarchical counterpart is because in the former scenario, caches are only deployed at the edge of the network. Thus most requests travel through lower network levels which are less congested as compared to the higher network. But in hierarchical caching, requests need to travel through the highly congested higher network levels to hit the document copy.

Combining the results of the above connection and transmission times, hierarchical caching scheme performs better in terms of total latency to fetch a document when quantitatively compared against the distributed caching. Thus, this scheme is recommended for small network deployment. For large scale deployment, however, a hybrid scheme with a certain number of caches cooperating at the same or higher network level of a caching hierarchy using distributed caching gives better cache efficiency. Similar results are obtained as in [55] where hybrid scheme gives as low transmission time as distributed caching and as low connection time as hierarchical caching. Thus, it combines the advantages of both hierarchical and distributed caching yielding reduced total latency.

In addition, a series of simulations were performed for different web client request distributions in terms of hit rate, byte hit rate, reduced packets, reduced latency, and reduced hops using various web cache replacing algorithms. The results show that irrespective of the type of web cache

replacing technique used, the efficiency or the hit ratio increases as the cache size of the single cache server increases.

From the various web cache replacement algorithms tested in the study, LRV algorithm which takes the locality, cost and size of the document into account outperforms the other replacement algorithms in many performance aspects. The algorithm shows high performance in hit rate, byte hit rate, reduced packets, reduced latency, and reduced hops for all of the web client request distributions.

5.2 Future Work

Although this research was carefully prepared and tried to meet its aims and objectives, the following two main limitations make a room for future work. First, finding the right simulator was a challenge. Thus, a simple simulator written in C programming language is used. However, the output of the simulator doesn‟t show sharp differences in the curves of the hit rate, byte-hit rate, reduced packets and reduced latency as function of cache sizes. All these graphs seemed to have a similar curve except for the different values for the respective sample web client traces used in the study. In addition, the simulator cannot be used to compare the different caching architectures because it is designed to simulate a single web cache rather than cooperating caches. Second, getting the actual web client traces to be used as an input to the simulator is difficult. So, just for the purpose of this study, sample traces are generated instead of actual web client traces. This impacts the output of the simulator as the performances of the web caches tend to depend on the web client request distributions from users.

Thus based on this study future works can be done to compare the different cooperative caching systems. First, simulation could be performed using better simulator tools for more than one network model using real web client traces from different institutes with different web request behaviors. The simulator should be capable of analyzing the performance of large scale cache cooperating schemes including hierarchical, distributed and hybrid caching systems in terms of the hit ratio, number of hops, traffic, bandwidth utilization and latency. Second, performance of hybrid caching should be analyzed when caches cooperate according to their geographical location as well as when caches are organized according to users interest. Finally, the load balancing among the cooperative caches should be considered. This helps to avoid cases where some caches are forced to discard very popular items due to lack of space while other caches contain less popular items. Thus, the cache efficiency can be increased by fairly distributing popular documents to different caches.

References

[1] M. H. Abu-Amara, "Network Caching Technologies", King Fahd University of Petroleum and Minerals, Saudi Arabia, Chapter 53.

[2] V. Jacobson, "How to kill the Internet", presented at SIGCOMM'95 Middleware Workshop, Available: ftp://ftp.ee.lhl.gov/talks/vj-webflame.ps.Z., 1995.

[3]J. Shim, P. Scheuermann, and R. Vingralek, "Proxy Cache Algorithms: Design, Implementation", and Performance, IEEE Transactions on Knowledge and Data Engineering, v.11 n.4, p.549-562, 1999.

[4] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area Web caching sharing protocol", ACM/IEEE Trans. Networking, 2000.

[5]T. P. Kelly, "Optimization in Web Caching: Cache Management, Capacity Planning, and Content Naming", A dissertation for the degree of Doctor of Philosophy, Computer Science and Engineering, The University of Michigan, 2002.

[6] Suresha, "Caching Techniques for Dynamic Web Servers", A thesis for Doctor of Philosophy, Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560 012 India, June 2007.

[7]Z. Yao, C. V. Ravi Shankar, and S. K. Tripathi, "Hash-Based Virtual Hierarchies for Caching in Hybrid Content-Delivery Networks", Department of Computer Science and Engineering, The University of California, Riverside 92507, May 13, 2001.

[8]H. Che, Y. Tung, and Z. Wang, "Hierarchical Web Caching Systems: Modeling, Design and Experimental Results", IEEE Journal on selected areas in communications, Vol. 20, No. 7, September 2002.

[9] R. T. Hurley, W. Feng, and B. Y. Li, "An Analytical Comparison of Distributed and Hierarchical Web-Caching Architectures", Computer Science/Studies Program, Trent University, Peterborough, Canada.

[10] R. Tewari, M. Dahlin, H. M. Vin, and J. S. Kay, "Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet", Department of Computer Sciences, The University of Texas at Austin, February 1998.

[11] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like distributions: Evidence and Implications", Technical report, University of Wisconsin Madison, Department of Computer Science, 1210 West Dayton Street, July 1998.

[12] G. K. Zipf, "Relative frequency as a determinant of phonetic change", Reprinted from the Harvard Studies in Classical Philology, Volume XL, 1929.

[13]ViSOLVE Open Source Solutions, "Optimized Bandwidth + Secured access = Accelerated Data Delivery", ViSolve White Paper, ViSolve Inc, March 2009.

[14]Cisco Systems, "Network Caching Technologies", http://docwiki.cisco.com/

[15] R. Tewari, M. Dahlin, H. M. Vin, and J. S. Kay, "Design Considerations for Distributed Caching on the Internet", IBM T.J. Watson Research Center, University of Texas at Austin, Cephalopod Proliferationists, Inc.

[16] D.A. Patterson and J.L. Hennessy, "Computer Organization and Design (4th ed.)", Morgan Kaufmann, 2009.

[17] S. Dey and M. S. Nair, "Design and Implementation of a Simple Cache Simulator in Java to Investigate MESI and MOESI Coherency Protocols", International Journal of Computer Applications (0975 – 8887), School of Computer Science, The University of Manchester, Manchester, United Kingdom, Volume 87 – No.11, February 2014.

[18] A. K. Iyengar, E. M. Nahum, A. A. Shaikh, and R. Tewari, "Web Caching, Consistency, and Content distribution", IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598, RC22845 (W0307-113), July 15, 2003.

[19]K. S. Candan, W. S. Li, Q. Luo, W. P. Hsiung, and D. Agrawal, "Enabling dynamic content caching for database-driven web sites", in Proceedings of ACM SIGMOD"01, May 2001.

[20] M. Hofmann and L. R. Beaumont, "Content Networking: Architectures, Protocols and Practice", The Morgan Kaufmann Series in Networking, 2005.

[21] A. I. Vakaliand G. E. Pallis, "A Study on Web Caching Architectures and Performance", Department of Informatics, Aristotle University, Thessaloniki, Greece.

[22]S. Michael, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson, "Adaptive Web Caching: towards a new global caching architecture", In Proceedings of the Symposium on Internet Technologies and Systems, 1997.

[23] J. Gwertzaman and M. Seltzer, "The case for geographical push cashing", in Proceedings of the Fifth Annual Workshop on Hot Operating Systems, pp. 51-55, Orcas Island, AW, May 1995.

[24] P. Cao, J. Zhang, and K. Beach, "Active cache: Caching dynamic contents on the web", In Proceedings of the 3rdInter-Gateway Conference on Web Caching, 1998.

[25]P. Krishnan, D. Raz, and Y. Shavitt, "Transparent enroute cache location for regular networks", In DIMACS Workshop on Robust Communication Networks: Interconnection and Survivability, DIMACS Book series, New Brunswick, NJ, USA, November 1998.

[26]M. Rabinovich and O. Spatscheck, "Web Caching and Replication", First Edition, Addison Wesley Publishing Co., New York, December 2001.

[27]A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A hierarchical Internet object cache", In Proc. USENIX'96, January 1996.

[28] M. K. Liu, F. Y. Wang, and D. D. Zeng, "Web Caching: A Way to Improve Web QoS", Vol.19, No.2, pp.113-127, The Lab for Complex Systems and Intelligent Sciences, Institute of Automation, The Chinese Academy of Sciences, Beijing 100080, P.R. China, March 2004.

[29] D. Karger, A. Sherman, A. Bernheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi, "Web Caching with Consistent Hashing", In Eighth International World Wide Web Conference, May 1999.

[30]G. Haßlinger and O. Hohlfeld, "Efficiency of Caches for Content distribution on the Internet", Deutsche Telekom Netzproduktion, Technische Universität Berlin/DT Labs, Darmstadt and Berlin, Germany.

[31] P. Rodriguez, C. Spanner, and E.W. Biersack, "Analysis of web caching architectures: hierarchical and distributed caching", IEEE/ACM Transactions on Networking (TON), 2001.

[32] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A hierarchical Internet object cache", Computer Science Department, University of Southern California, University of Colorado – Boulder, 1996.

[33]D. Povey and J. Harrison, "A Distributed Internet Cache", School of Information Technology University of Queensland Brisbane, QLD, 4072.

[34]L.W. Dowdy and D.V. Foster, "Comparative models of the file assignment problem", ACM Comp. Surveys, 14, 1982.

[35]L. R. Beaumont, "Calculating Web Cache Hit Ratios", Content Networking, March 2000.

[36] G. Barish and K. Obraczka, "World Wide Web Caching - Trends and Techniques", UCS information Science Institute, Admiralty Way, Marina del Rey, CA 90292, USA.

[37] S. Gribble and E. Brewer, "System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace", In Proceedings of the USENIX Symposium on Internet Technologies and Systems, December 1997.

[38] E. A. Brewer, P. Gauthier, and D. McEvoy, "The Long-Term Viability of Large-Scale Caching", 3rdInternational WWW Caching Workshop, Manchester, UK, June 1998.

[39] J. Xu, B. Li, and D. Lee, "Placement problems for transparent data replication proxy services", IEEE Journal on Selected Areas in Communication, Special Issue on Internet Proxy Services, vol. 20, no. 7, pp. 1383-1398, September 2002.

[40]L. Zhang, S. Floyd, and V. Jacobson, "Adaptive web caching," in Proceedings of the 1997 NLANR Web Cache Workshop, June 1997.

[41] M. R. Korupolu and M. Dahlin, "Coordinated placement and replacement for large-scale distributed caches", Proceedings of the IEEE Workshop on Internet Applications, July1999.

[42]S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson, "Adaptive web caching towards a new global caching architecture", Computer Networks & ISDN Systems, 1998.

[43] Z. Liang, "Transparent Web Caching with Load Balancing", Thesis for the degree of Master of Science, Queen's University, Kingston, Ontario, Canada, February 2001.

[44] S. Williams, M. Abrams, C. Standridge, G. Abdulla, and E. Fox, "Removal Policies in Network Caches for World-Wide Web Documents", *Proceedings on ACM Sigcomm '96*, Stanford University, CA, August 1996.

[45] P. Cao and S. Irani, „„Cost-Aware WWW Proxy Caching Algorithms"", Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS), Monterey, December 1997.

[46] P. Barford, A. Bestavros, A. Bradley, and M. E. Crovella, "Changes in Web Client access Patterns: Characteristics and Caching Implications," in World Wide Web, Special Issue on characterization and Performance Evaluation, Vol. 2, pp. 15-28, 1999.

[47] V. Almeida, A. Bestavros, M. Crovella, and A. Oliveira, "Characterizing Reference Locality in the WWW", Boston University Computer Science Department, TR-96-11, Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems (PDIS '96), December 1996.

[48] D. E. Knuth, "The Art of Computer Programming: Sorting and Searching", Volume 3, First Edition, 1973.

[49] J. Wang, "A Survey of Web Caching Schemes for the Internet", Computer Communication Review, vol.29, October 1999.

[50] M. S. Oneis, H. Barada, M. J. Zemerly, "Towards An Efficient Web Caching Hybrid Architecture", Computer Engineering Department, Khalifa University of Science, Technology, and Research, P.O. Box 573, Sharjah, United Arab Emirates.

[51] Pei Cao, "Non-Uniform Size Web Document Cache Simulator", in C programming language, University of Wisconsin Copyright 1997.

[52] P. Lorenzetti, L. Rizzo, and L. Vicisano, "Replacement Policies for a Proxy Cache".

[53] R. Wooster and M. Abrams, "Proxy Caching the Estimates Page Load Delays", In the 6th International World Wide Web Conference, Santa Clara, CA. April 1997.

[54] X. Hu and A.Zincir-Heywood, "Understanding the Performance of the Cooperative Web Caching Systems with Analysis Models and Simulations", Faculty of Computer Science, Dalhousie University, Halifax, NS, Canada.

[55] B. M. Ahmed, T. Helaly, and S. Rahman, "Prioritizing Documents and Applying Hybrid Caching Strategy for Network Latency Reduction", Department of Computer Science, North Dakota State University Fargo, North Dakota 58105, USA.

[56] D. Manuel, "Web Caching: a Memory-based Architecture", Departamento de Informática – Universidade do Minho, 4710 - 057 Braga, Portugal

Appendix

The following are the detail output of the simulator for the four samples of web client request distributions used in the study. The summary of these outputs can be found in Simulator Result in section 4.3.

A.1 Sample One

```
Total Requests = 1800
Total Unique Requests = 1000
Total Bytes = 5.91099e+06
Total Unique Bytes = 3.9925e+06

Total Packets = 13778
Total Unique Packets = 8950

Average Connection Time = 195317
Number of Small Pages = 737

Total Hops = 1800
Total Unique Hops = 1000

Total Weighted Hops = 13778
Total Unique Weighted Hops = 8950

Infinite Cache Size:
Hit Rate = 0.444444
Byte Hit Rate = 0.324563
Reduced Packets = 0.350414
Reduced Latency = 0.415580

Reduced Hops = 0.444444

Reduced Weighted Hops = 0.350414

----------------------------------------

Cache Size = 199625.000000  5.000000%

5.00  LRU: Pure-Hit Rate = 0.337222   0.758750
5.00  LRU: Byte-Hit Rate = 0.238002   0.733301
5.00  LRU: Reduced Packets = 0.259617 0.740887
5.00  LRU: Reduced Latency = 0.313333 0.753965
5.00  LRU: Reduced Hops = 0.337222    0.758750
5.00  LRU: Reduced Weighted-Hops = 0.259617 0.740887
5.00  SIZE: Pure-Hit Rate = 0.278333  0.626250
5.00  SIZE: Byte-Hit Rate = 0.097466  0.300299
5.00  SIZE: Reduced Packets = 0.134998 0.385253
5.00  SIZE: Reduced Latency = 0.234786 0.564959
5.00  SIZE: Reduced Hops = 0.278333   0.626250
5.00  SIZE: Reduced Weighted-Hops = 0.134998     0.385253
```

```
5.00  Hybrid: Pure-Hit Rate = 0.3200000.720000
5.00  Hybrid: Byte-Hit Rate = 0.1290700.397672
5.00  Hybrid: Reduced Packets = 0.168312    0.480323
5.00  Hybrid: Reduced Latency = 0.274030    0.659390
5.00  Hybrid: Reduced Hops = 0.320000 0.720000
5.00  Hybrid: Reduced Weighted-Hops = 0.168312   0.480323
5.00  LRV: Pure-Hit Rate = 0.438889   0.987500
5.00  LRV: Byte-Hit Rate = 0.315487   0.972035
5.00  LRV: Reduced Packets = 0.341994 0.975973
5.00  LRV: Reduced Latency = 0.409177 0.984592
5.00  LRV: Reduced Hops = 0.438889    0.987500
5.00  LRV: Reduced Weighted-Hops = 0.341994 0.975973
5.00  LRU: Pure-Hit Rate = 0.337222   0.758750
5.00  LRU: Byte-Hit Rate = 0.238002   0.733301
5.00  LRU: Reduced Packets = 0.259617 0.740887
5.00  LRU: Reduced Latency = 0.313333 0.753965
5.00  LRU: Reduced Hops = 0.337222    0.758750
5.00  LRU: Reduced Weighted-Hops = 0.259617 0.740887
----------------------------------------

Cache Size = 399250.000000  10.000000%

10.00  LRU: Pure-Hit Rate = 0.431667  0.971250
10.00  LRU: Byte-Hit Rate = 0.316904  0.976401
10.00  LRU: Reduced Packets = 0.3417770.975352
10.00  LRU: Reduced Latency = 0.4040350.972219
10.00  LRU: Reduced Hops = 0.431667   0.971250
10.00  LRU: Reduced Weighted-Hops = 0.341777     0.975352
10.00  SIZE: Pure-Hit Rate = 0.312778 0.703750
10.00  SIZE: Byte-Hit Rate = 0.123605 0.380833
10.00  SIZE: Reduced Packets = 0.163376     0.466239
10.00  SIZE: Reduced Latency = 0.267230     0.643029
10.00  SIZE: Reduced Hops = 0.312778  0.703750
10.00  SIZE: Reduced Weighted-Hops = 0.163376    0.466239
10.00  Hybrid: Pure-Hit Rate = 0.354444     0.797500
10.00  Hybrid: Byte-Hit Rate = 0.164780     0.507699
10.00  Hybrid: Reduced Packets = 0.203585   0.580986
10.00  Hybrid: Reduced Latency = 0.308779   0.743006
10.00  Hybrid: Reduced Hops = 0.3544440.797500
10.00  Hybrid: Reduced Weighted-Hops = 0.203585  0.580986
10.00  LRV: Pure-Hit Rate = 0.443333  0.997500
10.00  LRV: Byte-Hit Rate = 0.322796  0.994556
10.00  LRV: Reduced Packets = 0.3487440.995236
10.00  LRV: Reduced Latency = 0.4143110.996946
10.00  LRV: Reduced Hops = 0.443333   0.997500
10.00  LRV: Reduced Weighted-Hops = 0.348744     0.995236
10.00  LRU: Pure-Hit Rate = 0.431667  0.971250
10.00  LRU: Byte-Hit Rate = 0.316904  0.976401
10.00  LRU: Reduced Packets = 0.3417770.975352
10.00  LRU: Reduced Latency = 0.4040350.972219
10.00  LRU: Reduced Hops = 0.431667   0.971250
10.00  LRU: Reduced Weighted-Hops = 0.341777     0.975352
----------------------------------------
```

```
Cache Size = 798500.000000  20.000000%

20.00  LRU: Pure-Hit Rate = 0.443889  0.998750
20.00  LRU: Byte-Hit Rate = 0.324511  0.999838
20.00  LRU: Reduced Packets = 0.3502690.999586
20.00  LRU: Reduced Latency = 0.4151460.998955
20.00  LRU: Reduced Hops = 0.443889   0.998750
20.00  LRU: Reduced Weighted-Hops = 0.350269     0.999586
20.00  SIZE: Pure-Hit Rate = 0.360000 0.810000
20.00  SIZE: Byte-Hit Rate = 0.172179 0.530495
20.00  SIZE: Reduced Packets = 0.210916     0.601906
20.00  SIZE: Reduced Latency = 0.314778     0.757442
20.00  SIZE: Reduced Hops = 0.360000  0.810000
20.00  SIZE: Reduced Weighted-Hops = 0.210916    0.601906
20.00  Hybrid: Pure-Hit Rate = 0.373333     0.840000
20.00  Hybrid: Byte-Hit Rate = 0.189039     0.582442
20.00  Hybrid: Reduced Packets = 0.226738  0.647059
20.00  Hybrid: Reduced Latency = 0.328961  0.791569
20.00  Hybrid: Reduced Hops = 0.3733330.840000
20.00  Hybrid: Reduced Weighted-Hops = 0.226738  0.647059
20.00  LRV: Pure-Hit Rate = 0.444444  1.000000
20.00  LRV: Byte-Hit Rate = 0.324563  1.000000
20.00  LRV: Reduced Packets = 0.3504141.000000
20.00  LRV: Reduced Latency = 0.4155801.000000
20.00  LRV: Reduced Hops = 0.444444   1.000000
20.00  LRV: Reduced Weighted-Hops = 0.350414     1.000000
20.00  LRU: Pure-Hit Rate = 0.443889  0.998750
20.00  LRU: Byte-Hit Rate = 0.324511  0.999838
20.00  LRU: Reduced Packets = 0.3502690.999586
20.00  LRU: Reduced Latency = 0.4151460.998955
20.00  LRU: Reduced Hops = 0.443889   0.998750
20.00  LRU: Reduced Weighted-Hops = 0.350269     0.999586
---------------------------------------

Cache Size = 998125.000000  25.000000%

25.00  LRU: Pure-Hit Rate = 0.444444  1.000000
25.00  LRU: Byte-Hit Rate = 0.324563  1.000000
25.00  LRU: Reduced Packets = 0.3504141.000000
25.00  LRU: Reduced Latency = 0.4155801.000000
25.00  LRU: Reduced Hops = 0.444444   1.000000
25.00  LRU: Reduced Weighted-Hops = 0.350414     1.000000
25.00  SIZE: Pure-Hit Rate = 0.368333 0.828750
25.00  SIZE: Byte-Hit Rate = 0.182173 0.561285
25.00  SIZE: Reduced Packets = 0.220206     0.628418
25.00  SIZE: Reduced Latency = 0.323511     0.778456
25.00  SIZE: Reduced Hops = 0.368333  0.828750
25.00  SIZE: Reduced Weighted-Hops = 0.220206    0.628418
25.00  Hybrid: Pure-Hit Rate = 0.391111     0.880000
25.00  Hybrid: Byte-Hit Rate = 0.218733     0.673931
25.00  Hybrid: Reduced Packets = 0.254609   0.726595
25.00  Hybrid: Reduced Latency = 0.349607   0.841251
25.00  Hybrid: Reduced Hops = 0.3911110.880000
25.00  Hybrid: Reduced Weighted-Hops = 0.254609  0.726595
```

```
25.00  LRV: Pure-Hit Rate = 0.444444  1.000000
25.00  LRV: Byte-Hit Rate = 0.324563  1.000000
25.00  LRV: Reduced Packets = 0.3504141.000000
25.00  LRV: Reduced Latency = 0.4155801.000000
25.00  LRV: Reduced Hops = 0.444444   1.000000
25.00  LRV: Reduced Weighted-Hops = 0.350414     1.000000
25.00  LRU: Pure-Hit Rate = 0.444444  1.000000
25.00  LRU: Byte-Hit Rate = 0.324563  1.000000
25.00  LRU: Reduced Packets = 0.3504141.000000
25.00  LRU: Reduced Latency = 0.4155801.000000
25.00  LRU: Reduced Hops = 0.444444   1.000000
25.00  LRU: Reduced Weighted-Hops = 0.350414     1.000000
----------------------------------------

Cache Size = 1996250.000000  50.000000%

50.00  LRU: Pure-Hit Rate = 0.444444  1.000000
50.00  LRU: Byte-Hit Rate = 0.324563  1.000000
50.00  LRU: Reduced Packets = 0.3504141.000000
50.00  LRU: Reduced Latency = 0.4155801.000000
50.00  LRU: Reduced Hops = 0.444444   1.000000
50.00  LRU: Reduced Weighted-Hops = 0.350414     1.000000
50.00  SIZE: Pure-Hit Rate = 0.411667 0.926250
50.00  SIZE: Byte-Hit Rate = 0.257878 0.794539
50.00  SIZE: Reduced Packets = 0.290100     0.827879
50.00  SIZE: Reduced Latency = 0.374639     0.901483
50.00  SIZE: Reduced Hops = 0.411667  0.926250
50.00  SIZE: Reduced Weighted-Hops = 0.290100    0.827879
50.00  Hybrid: Pure-Hit Rate = 0.438889     0.987500
50.00  Hybrid: Byte-Hit Rate = 0.313751     0.966687
50.00  Hybrid: Reduced Packets = 0.340543   0.971831
50.00  Hybrid: Reduced Latency = 0.408759   0.983586
50.00  Hybrid: Reduced Hops = 0.4388890.987500
50.00  Hybrid: Reduced Weighted-Hops = 0.340543  0.971831
50.00  LRV: Pure-Hit Rate = 0.444444  1.000000
50.00  LRV: Byte-Hit Rate = 0.324563  1.000000
50.00  LRV: Reduced Packets = 0.3504141.000000
50.00  LRV: Reduced Latency = 0.4155801.000000
50.00  LRV: Reduced Hops = 0.444444   1.000000
50.00  LRV: Reduced Weighted-Hops = 0.350414     1.000000
50.00  LRU: Pure-Hit Rate = 0.444444  1.000000
50.00  LRU: Byte-Hit Rate = 0.324563  1.000000
50.00  LRU: Reduced Packets = 0.3504141.000000
50.00  LRU: Reduced Latency = 0.4155801.000000
50.00  LRU: Reduced Hops = 0.444444   1.000000
50.00  LRU: Reduced Weighted-Hops = 0.350414     1.000000
----------------------------------------

Cache Size = 2994375.000000      75.000000%

75.00  LRU: Pure-Hit Rate = 0.444444  1.000000
75.00  LRU: Byte-Hit Rate = 0.324563  1.000000
75.00  LRU: Reduced Packets = 0.3504141.000000
75.00  LRU: Reduced Latency = 0.4155801.000000
```

```
75.00  LRU: Reduced Hops = 0.444444   1.000000
75.00  LRU: Reduced Weighted-Hops = 0.350414     1.000000
75.00  SIZE: Pure-Hit Rate = 0.437222 0.983750
75.00  SIZE: Byte-Hit Rate = 0.307932 0.948759
75.00  SIZE: Reduced Packets = 0.335535     0.957539
75.00  SIZE: Reduced Latency = 0.406093     0.977170
75.00  SIZE: Reduced Hops = 0.437222  0.983750
75.00  SIZE: Reduced Weighted-Hops = 0.335535    0.957539
75.00  Hybrid: Pure-Hit Rate = 0.443333     0.997500
75.00  Hybrid: Byte-Hit Rate = 0.321946     0.991936
75.00  Hybrid: Reduced Packets = 0.348091   0.993372
75.00  Hybrid: Reduced Latency = 0.414107   0.996454
75.00  Hybrid: Reduced Hops = 0.4433330.997500
75.00  Hybrid: Reduced Weighted-Hops = 0.348091  0.993372
75.00  LRV: Pure-Hit Rate = 0.444444  1.000000
75.00  LRV: Byte-Hit Rate = 0.324563  1.000000
75.00  LRV: Reduced Packets = 0.3504141.000000
75.00  LRV: Reduced Latency = 0.4155801.000000
75.00  LRV: Reduced Hops = 0.444444   1.000000
75.00  LRV: Reduced Weighted-Hops = 0.350414    1.000000
75.00  LRU: Pure-Hit Rate = 0.444444  1.000000
75.00  LRU: Byte-Hit Rate = 0.324563  1.000000
75.00  LRU: Reduced Packets = 0.3504141.000000
75.00  LRU: Reduced Latency = 0.4155801.000000
75.00  LRU: Reduced Hops = 0.444444   1.000000
75.00  LRU: Reduced Weighted-Hops = 0.350414     1.000000
```

A.2 Sample Two

```
Total Requests = 2000
Total Unique Requests = 1000
Total Bytes = 6.13625e+06
Total Unique Bytes = 3.9925e+06

Total Packets = 14547
Total Unique Packets = 8950

Average Connection Time = 195496
Number of Small Pages = 939

Total Hops = 2000
Total Unique Hops = 1000

Total Weighted Hops = 14547
Total Unique Weighted Hops = 8950

Infinite Cache Size:
Hit Rate = 0.500000
Byte Hit Rate = 0.349358
Reduced Packets = 0.384753
Reduced Latency = 0.465568

Reduced Hops = 0.5
```

```
Reduced Weighted Hops = 0.384753

---------------------------------------

Cache Size = 199625.000000  5.000000%

5.00  LRU: Pure-Hit Rate = 0.393500   0.787000
5.00  LRU: Byte-Hit Rate = 0.261102   0.747377
5.00  LRU: Reduced Packets = 0.292569 0.760407
5.00  LRU: Reduced Latency = 0.363238 0.780204
5.00  LRU: Reduced Hops = 0.393500    0.787000
5.00  LRU: Reduced Weighted-Hops = 0.292569 0.760407
5.00  SIZE: Pure-Hit Rate = 0.351500  0.703000
5.00  SIZE: Byte-Hit Rate = 0.132845  0.380254
5.00  SIZE: Reduced Packets = 0.182718 0.474897
5.00  SIZE: Reduced Latency = 0.301522 0.647644
5.00  SIZE: Reduced Hops = 0.351500   0.703000
5.00  SIZE: Reduced Weighted-Hops = 0.182718     0.474897
5.00  Hybrid: Pure-Hit Rate = 0.388500 0.777000
5.00  Hybrid: Byte-Hit Rate = 0.162786 0.465957
5.00  Hybrid: Reduced Packets = 0.213790    0.555655
5.00  Hybrid: Reduced Latency = 0.336908    0.723651
5.00  Hybrid: Reduced Hops = 0.388500 0.777000
5.00  Hybrid: Reduced Weighted-Hops = 0.213790   0.555655
5.00  LRV: Pure-Hit Rate = 0.495000   0.990000
5.00  LRV: Byte-Hit Rate = 0.340615   0.974974
5.00  LRV: Reduced Packets = 0.376779 0.979275
5.00  LRV: Reduced Latency = 0.459712 0.987423
5.00  LRV: Reduced Hops = 0.495000    0.990000
5.00  LRV: Reduced Weighted-Hops = 0.376779 0.979275
5.00  LRU: Pure-Hit Rate = 0.393500   0.787000
5.00  LRU: Byte-Hit Rate = 0.261102   0.747377
5.00  LRU: Reduced Packets = 0.292569 0.760407
5.00  LRU: Reduced Latency = 0.363238 0.780204
5.00  LRU: Reduced Hops = 0.393500    0.787000
5.00  LRU: Reduced Weighted-Hops = 0.292569 0.760407
---------------------------------------

Cache Size = 399250.000000  10.000000%

10.00  LRU: Pure-Hit Rate = 0.488000  0.976000
10.00  LRU: Byte-Hit Rate = 0.341930  0.978736
10.00  LRU: Reduced Packets = 0.376435 0.978381
10.00  LRU: Reduced Latency = 0.454612 0.976469
10.00  LRU: Reduced Hops = 0.488000   0.976000
10.00  LRU: Reduced Weighted-Hops = 0.376435     0.978381
10.00  SIZE: Pure-Hit Rate = 0.382500 0.765000
10.00  SIZE: Byte-Hit Rate = 0.158024 0.452327
10.00  SIZE: Reduced Packets = 0.209596     0.544756
10.00  SIZE: Reduced Latency = 0.331192     0.711372
10.00  SIZE: Reduced Hops = 0.382500  0.765000
10.00  SIZE: Reduced Weighted-Hops = 0.209596    0.544756
10.00  Hybrid: Pure-Hit Rate = 0.419500     0.839000
10.00  Hybrid: Byte-Hit Rate = 0.197283     0.564702
```

```
10.00  Hybrid: Reduced Packets = 0.247267   0.642666
10.00  Hybrid: Reduced Latency = 0.368708   0.791953
10.00  Hybrid: Reduced Hops = 0.4195000.839000
10.00  Hybrid: Reduced Weighted-Hops = 0.247267  0.642666
10.00  LRV: Pure-Hit Rate = 0.499000  0.998000
10.00  LRV: Byte-Hit Rate = 0.347656  0.995128
10.00  LRV: Reduced Packets = 0.3831720.995891
10.00  LRV: Reduced Latency = 0.4644070.997507
10.00  LRV: Reduced Hops = 0.499000   0.998000
10.00  LRV: Reduced Weighted-Hops = 0.383172     0.995891
10.00  LRU: Pure-Hit Rate = 0.488000  0.976000
10.00  LRU: Byte-Hit Rate = 0.341930  0.978736
10.00  LRU: Reduced Packets = 0.3764350.978381
10.00  LRU: Reduced Latency = 0.4546120.976469
10.00  LRU: Reduced Hops = 0.488000   0.976000
10.00  LRU: Reduced Weighted-Hops = 0.376435     0.978381
---------------------------------------

Cache Size = 798500.000000  20.000000%

20.00  LRU: Pure-Hit Rate = 0.499500  0.999000
20.00  LRU: Byte-Hit Rate = 0.349308  0.999855
20.00  LRU: Reduced Packets = 0.3846150.999643
20.00  LRU: Reduced Latency = 0.4651700.999147
20.00  LRU: Reduced Hops = 0.499500   0.999000
20.00  LRU: Reduced Weighted-Hops = 0.384615     0.999643
20.00  SIZE: Pure-Hit Rate = 0.425000 0.850000
20.00  SIZE: Byte-Hit Rate = 0.204816 0.586262
20.00  SIZE: Reduced Packets = 0.254623     0.661783
20.00  SIZE: Reduced Latency = 0.374672     0.804765
20.00  SIZE: Reduced Hops = 0.425000  0.850000
20.00  SIZE: Reduced Weighted-Hops = 0.254623    0.661783
20.00  Hybrid: Pure-Hit Rate = 0.437000     0.874000
20.00  Hybrid: Byte-Hit Rate = 0.221057     0.632751
20.00  Hybrid: Reduced Packets = 0.269609   0.700733
20.00  Hybrid: Reduced Latency = 0.387642   0.832622
20.00  Hybrid: Reduced Hops = 0.4370000.874000
20.00  Hybrid: Reduced Weighted-Hops = 0.269609  0.700733
20.00  LRV: Pure-Hit Rate = 0.500000  1.000000
20.00  LRV: Byte-Hit Rate = 0.349358  1.000000
20.00  LRV: Reduced Packets = 0.3847531.000000
20.00  LRV: Reduced Latency = 0.4655681.000000
20.00  LRV: Reduced Hops = 0.500000   1.000000
20.00  LRV: Reduced Weighted-Hops = 0.384753     1.000000
20.00  LRU: Pure-Hit Rate = 0.499500  0.999000
20.00  LRU: Byte-Hit Rate = 0.349308  0.999855
20.00  LRU: Reduced Packets = 0.3846150.999643
20.00  LRU: Reduced Latency = 0.4651700.999147
20.00  LRU: Reduced Hops = 0.499500   0.999000
20.00  LRU: Reduced Weighted-Hops = 0.384615     0.999643
---------------------------------------

Cache Size = 998125.000000  25.000000%
```

```
25.00  LRU: Pure-Hit Rate = 0.500000  1.000000
25.00  LRU: Byte-Hit Rate = 0.349358  1.000000
25.00  LRU: Reduced Packets = 0.3847531.000000
25.00  LRU: Reduced Latency = 0.4655681.000000
25.00  LRU: Reduced Hops = 0.500000   1.000000
25.00  LRU: Reduced Weighted-Hops = 0.384753     1.000000
25.00  SIZE: Pure-Hit Rate = 0.432500 0.865000
25.00  SIZE: Byte-Hit Rate = 0.214442 0.613817
25.00  SIZE: Reduced Packets = 0.263422     0.684653
25.00  SIZE: Reduced Latency = 0.382658     0.821918
25.00  SIZE: Reduced Hops = 0.432500  0.865000
25.00  SIZE: Reduced Weighted-Hops = 0.263422    0.684653
25.00  Hybrid: Pure-Hit Rate = 0.453000     0.906000
25.00  Hybrid: Byte-Hit Rate = 0.249661     0.714626
25.00  Hybrid: Reduced Packets = 0.296006   0.769341
25.00  Hybrid: Reduced Latency = 0.406522   0.873176
25.00  Hybrid: Reduced Hops = 0.4530000.906000
25.00  Hybrid: Reduced Weighted-Hops = 0.296006  0.769341
25.00  LRV: Pure-Hit Rate = 0.500000  1.000000
25.00  LRV: Byte-Hit Rate = 0.349358  1.000000
25.00  LRV: Reduced Packets = 0.3847531.000000
25.00  LRV: Reduced Latency = 0.4655681.000000
25.00  LRV: Reduced Hops = 0.500000   1.000000
25.00  LRV: Reduced Weighted-Hops = 0.384753     1.000000
25.00  LRU: Pure-Hit Rate = 0.500000  1.000000
25.00  LRU: Byte-Hit Rate = 0.349358  1.000000
25.00  LRU: Reduced Packets = 0.3847531.000000
25.00  LRU: Reduced Latency = 0.4655681.000000
25.00  LRU: Reduced Hops = 0.500000   1.000000
25.00  LRU: Reduced Weighted-Hops = 0.384753     1.000000
----------------------------------------

Cache Size = 1996250.000000 50.000000%

50.00  LRU: Pure-Hit Rate = 0.500000  1.000000
50.00  LRU: Byte-Hit Rate = 0.349358  1.000000
50.00  LRU: Reduced Packets = 0.3847531.000000
50.00  LRU: Reduced Latency = 0.4655681.000000
50.00  LRU: Reduced Hops = 0.500000   1.000000
50.00  LRU: Reduced Weighted-Hops = 0.384753     1.000000
50.00  SIZE: Pure-Hit Rate = 0.471500 0.943000
50.00  SIZE: Byte-Hit Rate = 0.287369 0.822561
50.00  SIZE: Reduced Packets = 0.329621     0.856709
50.00  SIZE: Reduced Latency = 0.429413     0.922343
50.00  SIZE: Reduced Hops = 0.471500  0.943000
50.00  SIZE: Reduced Weighted-Hops = 0.329621    0.856709
50.00  Hybrid: Pure-Hit Rate = 0.496000     0.992000
50.00  Hybrid: Byte-Hit Rate = 0.341190     0.976620
50.00  Hybrid: Reduced Packets = 0.377397   0.980883
50.00  Hybrid: Reduced Latency = 0.460615   0.989362
50.00  Hybrid: Reduced Hops = 0.4960000.992000
50.00  Hybrid: Reduced Weighted-Hops = 0.377397  0.980883
50.00  LRV: Pure-Hit Rate = 0.500000  1.000000
50.00  LRV: Byte-Hit Rate = 0.349358  1.000000
```

```
50.00  LRV: Reduced Packets = 0.3847531.000000
50.00  LRV: Reduced Latency = 0.4655681.000000
50.00  LRV: Reduced Hops = 0.500000   1.000000
50.00  LRV: Reduced Weighted-Hops = 0.384753     1.000000
50.00  LRU: Pure-Hit Rate = 0.500000  1.000000
50.00  LRU: Byte-Hit Rate = 0.349358  1.000000
50.00  LRU: Reduced Packets = 0.3847531.000000
50.00  LRU: Reduced Latency = 0.4655681.000000
50.00  LRU: Reduced Hops = 0.500000   1.000000
50.00  LRU: Reduced Weighted-Hops = 0.384753     1.000000
---------------------------------------

Cache Size = 2994375.000000  75.000000%

75.00  LRU: Pure-Hit Rate = 0.500000  1.000000
75.00  LRU: Byte-Hit Rate = 0.349358  1.000000
75.00  LRU: Reduced Packets = 0.3847531.000000
75.00  LRU: Reduced Latency = 0.4655681.000000
75.00  LRU: Reduced Hops = 0.500000   1.000000
75.00  LRU: Reduced Weighted-Hops = 0.384753    1.000000
75.00  SIZE: Pure-Hit Rate = 0.494000 0.988000
75.00  SIZE: Byte-Hit Rate = 0.334598 0.957752
75.00  SIZE: Reduced Packets = 0.371760    0.966232
75.00  SIZE: Reduced Latency = 0.457565    0.982812
75.00  SIZE: Reduced Hops = 0.494000  0.988000
75.00  SIZE: Reduced Weighted-Hops = 0.371760    0.966232
75.00  Hybrid: Pure-Hit Rate = 0.499500     0.999000
75.00  Hybrid: Byte-Hit Rate = 0.348098     0.996392
75.00  Hybrid: Reduced Packets = 0.383653   0.997141
75.00  Hybrid: Reduced Latency = 0.464894   0.998553
75.00  Hybrid: Reduced Hops = 0.4995000.999000
75.00  Hybrid: Reduced Weighted-Hops = 0.383653  0.997141
75.00  LRV: Pure-Hit Rate = 0.500000  1.000000
75.00  LRV: Byte-Hit Rate = 0.349358  1.000000
75.00  LRV: Reduced Packets = 0.3847531.000000
75.00  LRV: Reduced Latency = 0.4655681.000000
75.00  LRV: Reduced Hops = 0.500000   1.000000
75.00  LRV: Reduced Weighted-Hops = 0.384753     1.000000
75.00  LRU: Pure-Hit Rate = 0.500000  1.000000
75.00  LRU: Byte-Hit Rate = 0.349358  1.000000
75.00  LRU: Reduced Packets = 0.3847531.000000
75.00  LRU: Reduced Latency = 0.4655681.000000
75.00  LRU: Reduced Hops = 0.500000   1.000000
75.00  LRU: Reduced Weighted-Hops = 0.384753     1.000000
```

A.3 Sample Three

```
Total Requests = 1400
Total Unique Requests = 1000
Total Bytes = 5.1675e+06
Total Unique Bytes = 3.9925e+06

Total Packets = 11771
```

```
Total Unique Packets = 8950

Average Connection Time = 194254.
Number of Small Pages = 429.

Total Hops = 1400
Total Unique Hops = 1000

Total Weighted Hops = 11771
Total Unique Weighted Hops = 8950

Infinite Cache Size:
Hit Rate = 0.285714
Byte Hit Rate = 0.227383
Reduced Packets = 0.239657
Reduced Latency = 0.270386

Reduced Hops = 0.285714

Reduced Weighted Hops = 0.239657

----------------------------------------

Cache Size = 199625.000000  5.000000%

5.00  LRU: Pure-Hit Rate = 0.215000   0.752500
5.00  LRU: Byte-Hit Rate = 0.170098   0.748065
5.00  LRU: Reduced Packets = 0.179849 0.750443
5.00  LRU: Reduced Latency = 0.203200 0.751520
5.00  LRU: Reduced Hops = 0.215000    0.752500
5.00  LRU: Reduced Weighted-Hops = 0.179849 0.750443
5.00  SIZE: Pure-Hit Rate = 0.151429  0.530000
5.00  SIZE: Byte-Hit Rate = 0.045661  0.200812
5.00  SIZE: Reduced Packets = 0.0666890.278270
5.00  SIZE: Reduced Latency = 0.1236340.457252
5.00  SIZE: Reduced Hops = 0.151429   0.530000
5.00  SIZE: Reduced Weighted-Hops = 0.066689     0.278270
5.00  Hybrid: Pure-Hit Rate = 0.1735710.607500
5.00  Hybrid: Byte-Hit Rate = 0.0596690.262416
5.00  Hybrid: Reduced Packets = 0.082576    0.344559
5.00  Hybrid: Reduced Latency = 0.143639    0.531239
5.00  Hybrid: Reduced Hops = 0.173571 0.607500
5.00  Hybrid: Reduced Weighted-Hops = 0.082576   0.344559
5.00  LRV: Pure-Hit Rate = 0.280000   0.980000
5.00  LRV: Byte-Hit Rate = 0.218660   0.961634
5.00  LRV: Reduced Packets = 0.231416 0.965615
5.00  LRV: Reduced Latency = 0.263880 0.975941
5.00  LRV: Reduced Hops = 0.280000    0.980000
5.00  LRV: Reduced Weighted-Hops = 0.231416 0.965615
5.00  LRU: Pure-Hit Rate = 0.215000   0.752500
5.00  LRU: Byte-Hit Rate = 0.170098   0.748065
5.00  LRU: Reduced Packets = 0.179849 0.750443
5.00  LRU: Reduced Latency = 0.203200 0.751520
5.00  LRU: Reduced Hops = 0.215000    0.752500
```

```
5.00  LRU: Reduced Weighted-Hops = 0.179849 0.750443
----------------------------------------

Cache Size = 399250.000000  10.000000%

10.00  LRU: Pure-Hit Rate = 0.273571  0.957500
10.00  LRU: Byte-Hit Rate = 0.220218  0.968489
10.00  LRU: Reduced Packets = 0.2314160.965615
10.00  LRU: Reduced Latency = 0.2595510.959928
10.00  LRU: Reduced Hops = 0.273571   0.957500
10.00  LRU: Reduced Weighted-Hops = 0.231416     0.965615
10.00  SIZE: Pure-Hit Rate = 0.164286 0.575000
10.00  SIZE: Byte-Hit Rate = 0.053272 0.234284
10.00  SIZE: Reduced Packets = 0.075864     0.316554
10.00  SIZE: Reduced Latency = 0.135113     0.499705
10.00  SIZE: Reduced Hops = 0.164286  0.575000
10.00  SIZE: Reduced Weighted-Hops = 0.075864    0.316554
10.00  Hybrid: Pure-Hit Rate = 0.195000    0.682500
10.00  Hybrid: Byte-Hit Rate = 0.081083    0.356590
10.00  Hybrid: Reduced Packets = 0.102795  0.428926
10.00  Hybrid: Reduced Latency = 0.165064  0.610477
10.00  Hybrid: Reduced Hops = 0.1950000.682500
10.00  Hybrid: Reduced Weighted-Hops = 0.102795  0.428926
10.00  LRV: Pure-Hit Rate = 0.284286  0.995000
10.00  LRV: Byte-Hit Rate = 0.225731  0.992732
10.00  LRV: Reduced Packets = 0.2380430.993265
10.00  LRV: Reduced Latency = 0.2688980.994499
10.00  LRV: Reduced Hops = 0.284286   0.995000
10.00  LRV: Reduced Weighted-Hops = 0.238043     0.993265
10.00  LRU: Pure-Hit Rate = 0.273571  0.957500
10.00  LRU: Byte-Hit Rate = 0.220218  0.968489
10.00  LRU: Reduced Packets = 0.2314160.965615
10.00  LRU: Reduced Latency = 0.2595510.959928
10.00  LRU: Reduced Hops = 0.273571   0.957500
10.00  LRU: Reduced Weighted-Hops = 0.231416     0.965615
----------------------------------------

Cache Size = 798500.000000  20.000000%

20.00  LRU: Pure-Hit Rate = 0.285000  0.997500
20.00  LRU: Byte-Hit Rate = 0.227207  0.999226
20.00  LRU: Reduced Packets = 0.2394020.998937
20.00  LRU: Reduced Latency = 0.2698130.997881
20.00  LRU: Reduced Hops = 0.285000   0.997500
20.00  LRU: Reduced Weighted-Hops = 0.239402     0.998937
20.00  SIZE: Pure-Hit Rate = 0.200714 0.702500
20.00  SIZE: Byte-Hit Rate = 0.088102 0.387462
20.00  SIZE: Reduced Packets = 0.110016     0.459057
20.00  SIZE: Reduced Latency = 0.171121     0.632879
20.00  SIZE: Reduced Hops = 0.200714  0.702500
20.00  SIZE: Reduced Weighted-Hops = 0.110016    0.459057
20.00  Hybrid: Pure-Hit Rate = 0.217857     0.762500
20.00  Hybrid: Byte-Hit Rate = 0.107388     0.472279
20.00  Hybrid: Reduced Packets = 0.128536   0.536335
```

```
20.00  Hybrid: Reduced Latency = 0.188827   0.698364
20.00  Hybrid: Reduced Hops = 0.2178570.762500
20.00  Hybrid: Reduced Weighted-Hops = 0.128536  0.536335
20.00  LRV: Pure-Hit Rate = 0.285000  0.997500
20.00  LRV: Byte-Hit Rate = 0.226903  0.997885
20.00  LRV: Reduced Packets = 0.2391470.997873
20.00  LRV: Reduced Latency = 0.2697330.997585
20.00  LRV: Reduced Hops = 0.285000   0.997500
20.00  LRV: Reduced Weighted-Hops = 0.239147     0.997873
20.00  LRU: Pure-Hit Rate = 0.285000  0.997500
20.00  LRU: Byte-Hit Rate = 0.227207  0.999226
20.00  LRU: Reduced Packets = 0.2394020.998937
20.00  LRU: Reduced Latency = 0.2698130.997881
20.00  LRU: Reduced Hops = 0.285000   0.997500
20.00  LRU: Reduced Weighted-Hops = 0.239402     0.998937
----------------------------------------

Cache Size = 998125.000000  25.000000%

25.00  LRU: Pure-Hit Rate = 0.285000  0.997500
25.00  LRU: Byte-Hit Rate = 0.227207  0.999226
25.00  LRU: Reduced Packets = 0.2394020.998937
25.00  LRU: Reduced Latency = 0.2698130.997881
25.00  LRU: Reduced Hops = 0.285000   0.997500
25.00  LRU: Reduced Weighted-Hops = 0.239402    0.998937
25.00  SIZE: Pure-Hit Rate = 0.211429 0.740000
25.00  SIZE: Byte-Hit Rate = 0.099534 0.437734
25.00  SIZE: Reduced Packets = 0.120890     0.504431
25.00  SIZE: Reduced Latency = 0.182024     0.673202
25.00  SIZE: Reduced Hops = 0.211429  0.740000
25.00  SIZE: Reduced Weighted-Hops = 0.120890    0.504431
25.00  Hybrid: Pure-Hit Rate = 0.217857     0.762500
25.00  Hybrid: Byte-Hit Rate = 0.107388     0.472279
25.00  Hybrid: Reduced Packets = 0.128536   0.536335
25.00  Hybrid: Reduced Latency = 0.188827   0.698364
25.00  Hybrid: Reduced Hops = 0.2178570.762500
25.00  Hybrid: Reduced Weighted-Hops = 0.128536  0.536335
25.00  LRV: Pure-Hit Rate = 0.285000  0.997500
25.00  LRV: Byte-Hit Rate = 0.226903  0.997885
25.00  LRV: Reduced Packets = 0.2391470.997873
25.00  LRV: Reduced Latency = 0.2697330.997585
25.00  LRV: Reduced Hops = 0.285000   0.997500
25.00  LRV: Reduced Weighted-Hops = 0.239147     0.997873
25.00  LRU: Pure-Hit Rate = 0.285000  0.997500
25.00  LRU: Byte-Hit Rate = 0.227207  0.999226
25.00  LRU: Reduced Packets = 0.2394020.998937
25.00  LRU: Reduced Latency = 0.2698130.997881
25.00  LRU: Reduced Hops = 0.285000   0.997500
25.00  LRU: Reduced Weighted-Hops = 0.239402     0.998937
----------------------------------------

Cache Size = 1996250.000000  50.000000%

50.00  LRU: Pure-Hit Rate = 0.285714  1.000000
```

```
50.00  LRU: Byte-Hit Rate = 0.227383  1.000000
50.00  LRU: Reduced Packets = 0.2396571.000000
50.00  LRU: Reduced Latency = 0.2703861.000000
50.00  LRU: Reduced Hops = 0.285714   1.000000
50.00  LRU: Reduced Weighted-Hops = 0.239657     1.000000
50.00  SIZE: Pure-Hit Rate = 0.249286 0.872500
50.00  SIZE: Byte-Hit Rate = 0.159595 0.701878
50.00  SIZE: Reduced Packets = 0.177215     0.739454
50.00  SIZE: Reduced Latency = 0.225716     0.834794
50.00  SIZE: Reduced Hops = 0.249286  0.872500
50.00  SIZE: Reduced Weighted-Hops = 0.177215    0.739454
50.00  Hybrid: Pure-Hit Rate = 0.278571     0.975000
50.00  Hybrid: Byte-Hit Rate = 0.215016     0.945609
50.00  Hybrid: Reduced Packets = 0.228103   0.951790
50.00  Hybrid: Reduced Latency = 0.261870   0.968505
50.00  Hybrid: Reduced Hops = 0.2785710.975000
50.00  Hybrid: Reduced Weighted-Hops = 0.228103  0.951790
50.00  LRV: Pure-Hit Rate = 0.285714  1.000000
50.00  LRV: Byte-Hit Rate = 0.227383  1.000000
50.00  LRV: Reduced Packets = 0.2396571.000000
50.00  LRV: Reduced Latency = 0.2703861.000000
50.00  LRV: Reduced Hops = 0.285714   1.000000
50.00  LRV: Reduced Weighted-Hops = 0.239657     1.000000
50.00  LRU: Pure-Hit Rate = 0.285714  1.000000
50.00  LRU: Byte-Hit Rate = 0.227383  1.000000
50.00  LRU: Reduced Packets = 0.2396571.000000
50.00  LRU: Reduced Latency = 0.2703861.000000
50.00  LRU: Reduced Hops = 0.285714   1.000000
50.00  LRU: Reduced Weighted-Hops = 0.239657     1.000000
----------------------------------------

Cache Size = 2994375.000000    75.000000%

75.00  LRU: Pure-Hit Rate = 0.285714  1.000000
75.00  LRU: Byte-Hit Rate = 0.227383  1.000000
75.00  LRU: Reduced Packets = 0.2396571.000000
75.00  LRU: Reduced Latency = 0.2703861.000000
75.00  LRU: Reduced Hops = 0.285714   1.000000
75.00  LRU: Reduced Weighted-Hops = 0.239657     1.000000
75.00  SIZE: Pure-Hit Rate = 0.276429 0.967500
75.00  SIZE: Byte-Hit Rate = 0.208360 0.916336
75.00  SIZE: Reduced Packets = 0.222241     0.927331
75.00  SIZE: Reduced Latency = 0.258541     0.956193
75.00  SIZE: Reduced Hops = 0.276429  0.967500
75.00  SIZE: Reduced Weighted-Hops = 0.222241    0.927331
75.00  Hybrid: Pure-Hit Rate = 0.284286     0.995000
75.00  Hybrid: Byte-Hit Rate = 0.224390     0.986834
75.00  Hybrid: Reduced Packets = 0.236938   0.988657
75.00  Hybrid: Reduced Latency = 0.268546   0.993195
75.00  Hybrid: Reduced Hops = 0.2842860.995000
75.00  Hybrid: Reduced Weighted-Hops = 0.236938  0.988657
75.00  LRV: Pure-Hit Rate = 0.285714  1.000000
75.00  LRV: Byte-Hit Rate = 0.227383  1.000000
75.00  LRV: Reduced Packets = 0.2396571.000000
```

```
75.00  LRV: Reduced Latency = 0.2703861.000000
75.00  LRV: Reduced Hops = 0.285714   1.000000
75.00  LRV: Reduced Weighted-Hops = 0.239657     1.000000
75.00  LRU: Pure-Hit Rate = 0.285714  1.000000
75.00  LRU: Byte-Hit Rate = 0.227383  1.000000
75.00  LRU: Reduced Packets = 0.2396571.000000
75.00  LRU: Reduced Latency = 0.2703861.000000
75.00  LRU: Reduced Hops = 0.285714   1.000000
75.00  LRU: Reduced Weighted-Hops = 0.239657     1.000000
```

A.4 Sample Four

```
Total Requests = 1400

Total Unique Requests = 1000
Total Bytes = 8.48068e+06
Total Unique Bytes = 7.6966e+06

Total Packets = 17967
Total Unique Packets = 15860

Average Connection Time = 180622
Number of Small Pages = 300

Total Hops = 1400
Total Unique Hops = 1000

Total Weighted Hops = 17967
Total Unique Weighted Hops = 15860

Infinite Cache Size:
Hit Rate = 0.284286
Byte Hit Rate = 0.092455
Reduced Packets = 0.117271
Reduced Latency = 0.277090

Reduced Hops = 0.284286

Reduced Weighted Hops = 0.117271

----------------------------------------

Cache Size = 384829.750000  5.000000%

5.00  LRU: Pure-Hit Rate = 0.220714   0.776382
5.00  LRU: Byte-Hit Rate = 0.070648   0.764138
5.00  LRU: Reduced Packets = 0.090499 0.771713
5.00  LRU: Reduced Latency = 0.215090 0.776248
5.00  LRU: Reduced Hops = 0.220714    0.776382
5.00  LRU: Reduced Weighted-Hops = 0.090499 0.771713
5.00  SIZE: Pure-Hit Rate = 0.280714  0.987437
5.00  SIZE: Byte-Hit Rate = 0.090518  0.979052
5.00  SIZE: Reduced Packets = 0.1151560.981965
5.00  SIZE: Reduced Latency = 0.2735830.987346
5.00  SIZE: Reduced Hops = 0.280714   0.987437
```

```
5.00  SIZE: Reduced Weighted-Hops = 0.115156     0.981965
5.00  Hybrid: Pure-Hit Rate = 0.2842861.000000
5.00  Hybrid: Byte-Hit Rate = 0.0924551.000000
5.00  Hybrid: Reduced Packets = 0.117271    1.000000
5.00  Hybrid: Reduced Latency = 0.277090    1.000000
5.00  Hybrid: Reduced Hops = 0.284286 1.000000
5.00  Hybrid: Reduced Weighted-Hops = 0.117271   1.000000
5.00  LRV: Pure-Hit Rate = 0.283571   0.997487
5.00  LRV: Byte-Hit Rate = 0.092209   0.997341
5.00  LRV: Reduced Packets = 0.116992 0.997627
5.00  LRV: Reduced Latency = 0.276393 0.997486
5.00  LRV: Reduced Hops = 0.283571    0.997487
5.00  LRV: Reduced Weighted-Hops = 0.116992 0.997627
5.00  LRU: Pure-Hit Rate = 0.220714   0.776382
5.00  LRU: Byte-Hit Rate = 0.070648   0.764138
5.00  LRU: Reduced Packets = 0.090499 0.771713
5.00  LRU: Reduced Latency = 0.215090 0.776248
5.00  LRU: Reduced Hops = 0.220714    0.776382
5.00  LRU: Reduced Weighted-Hops = 0.090499 0.771713
----------------------------------------

Cache Size = 769659.500000  10.000000%

10.00  LRU: Pure-Hit Rate = 0.258571  0.909548
10.00  LRU: Byte-Hit Rate = 0.083235  0.900272
10.00  LRU: Reduced Packets = 0.1061950.905553
10.00  LRU: Reduced Latency = 0.2519980.909447
10.00  LRU: Reduced Hops = 0.258571   0.909548
10.00  LRU: Reduced Weighted-Hops = 0.106195     0.905553
10.00  SIZE: Pure-Hit Rate = 0.284286 1.000000
10.00  SIZE: Byte-Hit Rate = 0.092455 1.000000
10.00  SIZE: Reduced Packets = 0.117271     1.000000
10.00  SIZE: Reduced Latency = 0.277090     1.000000
10.00  SIZE: Reduced Hops = 0.284286  1.000000
10.00  SIZE: Reduced Weighted-Hops = 0.117271    1.000000
10.00  Hybrid: Pure-Hit Rate = 0.284286     1.000000
10.00  Hybrid: Byte-Hit Rate = 0.092455     1.000000
10.00  Hybrid: Reduced Packets = 0.117271   1.000000
10.00  Hybrid: Reduced Latency = 0.277090   1.000000
10.00  Hybrid: Reduced Hops = 0.2842861.000000
10.00  Hybrid: Reduced Weighted-Hops = 0.117271  1.000000
10.00  LRV: Pure-Hit Rate = 0.284286  1.000000
10.00  LRV: Byte-Hit Rate = 0.092455  1.000000
10.00  LRV: Reduced Packets = 0.1172711.000000
10.00  LRV: Reduced Latency = 0.2770901.000000
10.00  LRV: Reduced Hops = 0.284286   1.000000
10.00  LRV: Reduced Weighted-Hops = 0.117271     1.000000
10.00  LRU: Pure-Hit Rate = 0.258571  0.909548
10.00  LRU: Byte-Hit Rate = 0.083235  0.900272
10.00  LRU: Reduced Packets = 0.1061950.905553
10.00  LRU: Reduced Latency = 0.2519980.909447
10.00  LRU: Reduced Hops = 0.258571   0.909548
10.00  LRU: Reduced Weighted-Hops = 0.106195     0.905553
----------------------------------------
```

```
Cache Size = 1539319.000000      20.000000%

20.00  LRU: Pure-Hit Rate = 0.279286  0.982412
20.00  LRU: Byte-Hit Rate = 0.090714  0.981175
20.00  LRU: Reduced Packets = 0.1151560.981965
20.00  LRU: Reduced Latency = 0.2722120.982398
20.00  LRU: Reduced Hops = 0.279286   0.982412
20.00  LRU: Reduced Weighted-Hops = 0.115156     0.981965
20.00  SIZE: Pure-Hit Rate = 0.284286 1.000000
20.00  SIZE: Byte-Hit Rate = 0.092455 1.000000
20.00  SIZE: Reduced Packets = 0.117271     1.000000
20.00  SIZE: Reduced Latency = 0.277090     1.000000
20.00  SIZE: Reduced Hops = 0.284286  1.000000
20.00  SIZE: Reduced Weighted-Hops = 0.117271    1.000000
20.00  Hybrid: Pure-Hit Rate = 0.284286     1.000000
20.00  Hybrid: Byte-Hit Rate = 0.092455    1.000000
20.00  Hybrid: Reduced Packets = 0.117271   1.000000
20.00  Hybrid: Reduced Latency = 0.277090   1.000000
20.00  Hybrid: Reduced Hops = 0.2842861.000000
20.00  Hybrid: Reduced Weighted-Hops = 0.117271  1.000000
20.00  LRV: Pure-Hit Rate = 0.284286  1.000000
20.00  LRV: Byte-Hit Rate = 0.092455  1.000000
20.00  LRV: Reduced Packets = 0.1172711.000000
20.00  LRV: Reduced Latency = 0.2770901.000000
20.00  LRV: Reduced Hops = 0.284286   1.000000
20.00  LRV: Reduced Weighted-Hops = 0.117271     1.000000
20.00  LRU: Pure-Hit Rate = 0.279286  0.982412
20.00  LRU: Byte-Hit Rate = 0.090714  0.981175
20.00  LRU: Reduced Packets = 0.1151560.981965
20.00  LRU: Reduced Latency = 0.2722120.982398
20.00  LRU: Reduced Hops = 0.279286   0.982412
20.00  LRU: Reduced Weighted-Hops = 0.115156     0.981965
---------------------------------------

Cache Size = 1924148.750000      25.000000%

25.00  LRU: Pure-Hit Rate = 0.282143  0.992462
25.00  LRU: Byte-Hit Rate = 0.091728  0.992137
25.00  LRU: Reduced Packets = 0.1163800.992406
25.00  LRU: Reduced Latency = 0.2750000.992459
25.00  LRU: Reduced Hops = 0.282143   0.992462
25.00  LRU: Reduced Weighted-Hops = 0.116380     0.992406
25.00  SIZE: Pure-Hit Rate = 0.284286 1.000000
25.00  SIZE: Byte-Hit Rate = 0.092455 1.000000
25.00  SIZE: Reduced Packets = 0.117271     1.000000
25.00  SIZE: Reduced Latency = 0.277090     1.000000
25.00  SIZE: Reduced Hops = 0.284286  1.000000
25.00  SIZE: Reduced Weighted-Hops = 0.117271    1.000000
25.00  Hybrid: Pure-Hit Rate = 0.284286     1.000000
25.00  Hybrid: Byte-Hit Rate = 0.092455    1.000000
25.00  Hybrid: Reduced Packets = 0.117271   1.000000
25.00  Hybrid: Reduced Latency = 0.277090   1.000000
25.00  Hybrid: Reduced Hops = 0.2842861.000000
```

```
25.00  Hybrid: Reduced Weighted-Hops = 0.117271  1.000000
25.00  LRV: Pure-Hit Rate = 0.284286  1.000000
25.00  LRV: Byte-Hit Rate = 0.092455  1.000000
25.00  LRV: Reduced Packets = 0.1172711.000000
25.00  LRV: Reduced Latency = 0.2770901.000000
25.00  LRV: Reduced Hops = 0.284286   1.000000
25.00  LRV: Reduced Weighted-Hops = 0.117271     1.000000
25.00  LRU: Pure-Hit Rate = 0.282143  0.992462
25.00  LRU: Byte-Hit Rate = 0.091728  0.992137
25.00  LRU: Reduced Packets = 0.1163800.992406
25.00  LRU: Reduced Latency = 0.2750000.992459
25.00  LRU: Reduced Hops = 0.282143   0.992462
25.00  LRU: Reduced Weighted-Hops = 0.116380     0.992406
----------------------------------------

Cache Size = 3848297.500000      50.000000%

50.00  LRU: Pure-Hit Rate = 0.283571  0.997487
50.00  LRU: Byte-Hit Rate = 0.092209  0.997341
50.00  LRU: Reduced Packets = 0.1169920.997627
50.00  LRU: Reduced Latency = 0.2763930.997486
50.00  LRU: Reduced Hops = 0.283571   0.997487
50.00  LRU: Reduced Weighted-Hops = 0.116992     0.997627
50.00  SIZE: Pure-Hit Rate = 0.284286 1.000000
50.00  SIZE: Byte-Hit Rate = 0.092455 1.000000
50.00  SIZE: Reduced Packets = 0.117271     1.000000
50.00  SIZE: Reduced Latency = 0.277090     1.000000
50.00  SIZE: Reduced Hops = 0.284286  1.000000
50.00  SIZE: Reduced Weighted-Hops = 0.117271    1.000000
50.00  Hybrid: Pure-Hit Rate = 0.284286     1.000000
50.00  Hybrid: Byte-Hit Rate = 0.092455     1.000000
50.00  Hybrid: Reduced Packets = 0.117271   1.000000
50.00  Hybrid: Reduced Latency = 0.277090   1.000000
50.00  Hybrid: Reduced Hops = 0.2842861.000000
50.00  Hybrid: Reduced Weighted-Hops = 0.117271  1.000000
50.00  LRV: Pure-Hit Rate = 0.284286  1.000000
50.00  LRV: Byte-Hit Rate = 0.092455  1.000000
50.00  LRV: Reduced Packets = 0.1172711.000000
50.00  LRV: Reduced Latency = 0.2770901.000000
50.00  LRV: Reduced Hops = 0.284286   1.000000
50.00  LRV: Reduced Weighted-Hops = 0.117271     1.000000
50.00  LRU: Pure-Hit Rate = 0.283571  0.997487
50.00  LRU: Byte-Hit Rate = 0.092209  0.997341
50.00  LRU: Reduced Packets = 0.1169920.997627
50.00  LRU: Reduced Latency = 0.2763930.997486
50.00  LRU: Reduced Hops = 0.283571   0.997487
50.00  LRU: Reduced Weighted-Hops = 0.116992     0.997627
----------------------------------------

Cache Size = 5772446.000000  75.000000%

75.00  LRU: Pure-Hit Rate = 0.283571  0.997487
75.00  LRU: Byte-Hit Rate = 0.092209  0.997341
75.00  LRU: Reduced Packets = 0.1169920.997627
```

```
75.00  LRU: Reduced Latency = 0.2763930.997486
75.00  LRU: Reduced Hops = 0.283571   0.997487
75.00  LRU: Reduced Weighted-Hops = 0.116992     0.997627
75.00  SIZE: Pure-Hit Rate = 0.284286 1.000000
75.00  SIZE: Byte-Hit Rate = 0.092455 1.000000
75.00  SIZE: Reduced Packets = 0.117271     1.000000
75.00  SIZE: Reduced Latency = 0.277090     1.000000
75.00  SIZE: Reduced Hops = 0.284286  1.000000
75.00  SIZE: Reduced Weighted-Hops = 0.117271    1.000000
75.00  Hybrid: Pure-Hit Rate = 0.284286     1.000000
75.00  Hybrid: Byte-Hit Rate = 0.092455     1.000000
75.00  Hybrid: Reduced Packets = 0.117271   1.000000
75.00  Hybrid: Reduced Latency = 0.277090   1.000000
75.00  Hybrid: Reduced Hops = 0.2842861.000000
75.00  Hybrid: Reduced Weighted-Hops = 0.117271  1.000000
75.00  LRV: Pure-Hit Rate = 0.284286  1.000000
75.00  LRV: Byte-Hit Rate = 0.092455  1.000000
75.00  LRV: Reduced Packets = 0.1172711.000000
75.00  LRV: Reduced Latency = 0.2770901.000000
75.00  LRV: Reduced Hops = 0.284286   1.000000
75.00  LRV: Reduced Weighted-Hops = 0.117271     1.000000
75.00  LRU: Pure-Hit Rate = 0.283571  0.997487
75.00  LRU: Byte-Hit Rate = 0.092209  0.997341
75.00  LRU: Reduced Packets = 0.1169920.997627
75.00  LRU: Reduced Latency = 0.2763930.997486
75.00  LRU: Reduced Hops = 0.283571   0.997487
75.00  LRU: Reduced Weighted-Hops = 0.116992     0.997627
```