Web Cache Location and Provisioning for a Regional Internet Service Provider

BY

SWATHY REDDY MOGATALA

A Thesis submitted to
the Faculty of Graduate Studies
in partial fulfillment of requirements for the degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION

Web Cache Location and Provisioning for a Regional Internet Service Provider

BY

Swathy Reddy Mogatala

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirement of the degree

Of

MASTER OF SCIENCE

Abstract

The rapid growth in web usage has led to dramatically increased loads on individual web servers and on the network infrastructure. To improve performance with this increasing web usage, studies have been carried out on web and web caching architectures. Web caching improves the network performance by reducing bandwidth consumption, load on the original servers and latency. However, one of the common problems in a network is to find an optimal location to place these caches. The goal of our project is to reduce the latency of accessing a web object by efficiently managing caches and placing them in the suitable locations within Regional Internet Service Provider's network.

In this thesis, we solve the cache location problem for a Regional Internet Service Provider using a mathematical model. An optimization problem was solved for two cases, fixed cache capacity model and variable cache capacity model. The aim of these models is to reduce the overall average response time for client requests, by efficiently placing caches inside the network, in accordance with the available budget. The goal of our model is to provide the service providers with a clear view of the performance of their network. This view is very important in real-world applications, where utmost importance is focused on attaining maximum performance while reducing the amount spent in purchasing the caches. Optimization is performed using NEOS servers' - Mixed Integer Non-Linear Programming solver.

The second part of this thesis is validation of optimization models using a simulation tool. We analyzed trace files to find various parameters, such as average file size, elapsed time, cache size, cache hit and miss percentages and the maximum traffic load for all major links. A simulation tool Network Simulator-2 (NS-2) was used to simulate the existing network. The elapsed time obtained from the simulation models will be noted for two cases fixed cache capacities and variable cache capacities. Finally the results that have been obtained using optimization models are validated

using the simulation model, and the optimal budget required for the network is reported.

Acknowledgements

I would like to thank Dr. Jose Rueda for his support and inspiration towards my thesis. I am very grateful to Dr. Jeff Diamond for his technical support through out my masters program. My sincere thanks go to my co-advisors Dr. Ellen Liu for her encouragement and guidance during my thesis. I would like to thank Dr. Miroslaw Pawlak for his valuable reviews and suggestion on this thesis. I am thankful to my husband Mogatala Harshavardhan Reddy for helping and inspiring me to complete masters' degree.

I would like to thank Sasktel Telecommunications, Saskatoon for the great collaboration in this thesis. Special thanks go to Mr. Adeniyi Adebowale Oke from Sasktel for providing us with required data.

I would like to thank Peer Mohamed Shajahan, Computer Sciences, University of Manitoba for helping me in the initial formulation of the optimization problem.

I am very thankful to my parents Ravindranatha Reddy M. and Krishna Kumari, my in-laws Nagi Reddy M. and Soudhamini M., my sister Anitha Reddy S. and my brother Venkateshwar Reddy for their moral support and for encouraging me in pursuing my dreams.

I am thankful to my friends Vijay, Mpho, Alice, Daglenia, Manju and all at TRLabs, Winnipeg who made my student life more enjoyable and adventurous.

I would like to acknowledge TRLabs, Winnipeg for providing financial support and research facilities. Finally I would like to thank University of Manitoba for providing me an opportunity to pursue my masters.

Table of Contents

Abstract I

Acknowledgements III

Table of Contents IV

List of Figures VII

List of Tables VIII

Acronyms IX

1 Introduction 2

1.1 Motivation 2
1.2 Objective 3
1.3 Problem Statement 5
1.4 Organization of the thesis 6

2 Background Study 8

2.1 Caching Architectures 10
2.2 Cache Prefetching 14
2.3 Content Replacement Techniques, Coherence/Consistency and Protocols 15
2.4 Cache Location/Placement and Replication 21
2.5 Cache Scalability and Cacheability 23

3 Related Work 28

3.1 Advantages of Cache Location 28
3.2 Optimization Models 29
3.3 Mixed Integer Programming Models 33
3.4 Discussions 34

4 Analysis of Web Cache Performance at the Internet Service Providers 37

4.1 Motivation 37
4.2 Analysis Based on Trace Files Collected from Regional ISP 37
4.2.1 Web Cache Performance 38
4.2.2 Web Content Type 38
4.2.3 Co-operation Among Caches 39
4.2.4 Percentage of Traffic Based on the Document Sizes 41
4.2.5 Topology and Peak Total Down Stream Traffic 42
4.2.6 Location of Web Caches in the Regional ISP 43

4.3 Summary 43

5 Optimization Model 45

5.1 Introduction 45
5.2 Network Overview and Assumptions 46
5.2.1 Routing 46
5.2.2 Full Dependency 47
5.2.3 Process Sharing Queue and Balanced Fairness 47
5.2.4 Calculation of Cache Cost Based on Cache Capacity 49
5.2.5 Internet Node 51
5.3 Optimization Model for Cache Location Problem 51
5.3.1 Objective Function 52
5.3.2 Variable Cache Capacity Model 57
5.3.3 Fixed Cache Capacity Model 64
5.4 Summary 69

6 Simulation Model 71

6.1 Introduction 71
6.2 Network Model Components 72
6.2.1 Caches 72
6.2.2 Servers 72
6.2.3 Clients 72
6.2.4 Topology 73
6.2.5 Routing 73
6.3 Traffic Modeling 74
6.4 Assumptions 77
6.5 Trace Driven Simulations 78
6.6 Data Flow 79
6.7 Summary 81

7 Optimization and Simulation Results 83

7.1 Performance Measures 83
7.2 Results from Optimization Models 83
7.3 Results from Simulation Models 88
7.4 Error Estimation 91
7.5 Discussions 93

8 Conclusions 95

8.1 Contributions 95
8.2 Recommendation for Future Work 96

A. Optimization Languages and Software 99

A.1 NEOS Server 99

A.2 AMPL 99
A.3 MINLP 99

B. Simulation tool and Trace File Formats 102

B.1 NS-2 Overview 102
B.2 Modifications to NS-2 102
B.3 Input and Output Trace File Formats 102
B.3.1 Cache File 103
B.3.2 Topology File 103
B.3.3 Trace File 103
B.3.4 Output File Format 104
B.3.5 Delay Calculation 105

C. Definitions 107

D. Optimization and Simulation Tables 109

D.1 Optimization Output Values for Variable Cache Capacity Model 109
D.2 Optimization Output Values for Fixed Cache Capacity Model 112
D.3 Simulation Output Values for Variable Cache Capacity Model 114
D.4 Simulation Output Values for Fixed Cache Capacity Model 114

References 116

List of Figures

Figure 1.1: Regional ISP Network Topology 4
Figure 1.2: Proposed model for saving cost ($) 5
Figure 2.1: Location of Web Cache in a Network 9
Figure 2.2: Web Cache Functionality 9
Figure 2.3: Hierarchical caching model with different levels 11
Figure 2.4: Distributed Caching model 13
Figure 4.1: Peak Downstream Traffic [DIAMO1] 43
Figure 5.1: Cost vs. Peak Throughput Capacity for Caching Appliances [DIAMO1] 51
Figure 5.3: Path taken from node j to reach node k 60
Figure 5.4: Workload on the link 61
Figure 5.5: Calculating available bandwidth on the bottleneck link 62
Figure 6.1: Web Cache Basic Simulation Flowchart [SAIL03] 80
Figure 7.1: Optimization Results for Fixed Cache Capacity Model 84
Figure 7.2: Optimization Results for Variable Cache Capacity Model 85
Figure 7.3: Optimization Results for Variable Cache Capacity Model (Re-run) 88
Figure 7.4: Simulation Results for Fixed Cache Capacity Model 89
Figure 7.5: Simulation Results for Variable Cache Capacity Model 90
Figure 7.6: Curve of Interest for Variable Cache Capacity Model 90
Figure 7.7: Error Bars for Variable Cache Capacity model 92
Figure 7.8: Error Bars for Fixed Cache Capacity model 92

List of Tables

Table 4.1: Web cache performance 38
Table 4.2: Hit percentages based on the content type 39
Table 4.3: Co-operation among web caches 40
Table 4.4: File size distribution in web caches 42
Table 5.1: Cost and capacity of caching appliances 50
Table 5.2: Parameters for the proposed optimization model 67
Table 5.3: Decision variable for the proposed optimization model 67
Table 5.4: Supplementary variables for the proposed model 68
Table B.1 Delay calculation based on distances 105
Table B.2: Place and corresponding node ID 105
Table D.1: Optimization values of variable cache capacity model 109
Table D.2: Location of the cache for variable cache capacity model 110
Table D.3: Capacity of the caches at locations described in Table D.2 112
Table D.4: Optimization values for fixed cache capacity model 112
Table D.5: Location of cache and number of caches for fixed cache capacity model ... 114
Table D.6: Simulation values of fixed cache capacity model 114
Table D.7: Simulation values for fixed cache capacity model 115

Acronyms

AMPL	A Mathematical Programming Language
CARP	Cache Array Routing Protocol
CLP	Cache Location Problem
DSS	Decision Support Systems
FTP	File Transfer Protocol
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transport Protocol
ICP	Internet Cache Protocol
ISP	Internet Service Providers
LFU	Least Frequently Used
LLF	Least Latency Frequency
LRU	Least Recently Used
PCV	Piggyback Cache Validation
PSI	Piggyback Server Invalidation
Mbps	Megabits per second
Bps	Bytes per second
MILP	Mixed Integer Linear Programming
MINLP	Mixed Integer Non-Linear Programming
NEOS	Network Enabled Optimization System
NS2	Network Simulator version 2
OS	Operating System
R_+	All positive real numbers
RFC	Request for Comments
SCSP	Server Cache Synchronization Protocol
TCP	Transmission Control Protocol
TERC	Transparent En-Route Caches
TTL	Time to Live
UDP	User Datagram Protocol
WCCP	Web Cache Communication Protocol
Z_+	All positive integers

CHAPTER ONE

1 Introduction

The Internet is considered as one of the fastest media for accessing information (such as scientific, sports, entertainment, education, stock market and shopping). Since the Internet is inexpensive and easy to access, the use of Internet is growing exponentially each year [WANG99]. This increase is resulting in network congestion and server overloading. To alleviate this problem, web caching has been introduced to reduce load on severs and reduce traffic flow over the Internet.

Web caching is defined as the temporary storage of frequently accessed web objects, such as web pages, graphics, media files, and text files. Several studies [DANZ93], [HEDD97], [KRIS98], [KRIS00] have been conducted to find the best location for placing a cache in the network. In recent years, placing caches inside the network has become more popular [DANZ93], [HEDD97] than placing them at the edges of the networks. This is due to the fact that a significant amount of traffic that traverses through peering links, which connect the Internet Service Providers (ISPs) to the internet, can be reduced. In other words, placing caches inside the network will decrease the overall traffic within the network. Hence cost ($) and time can be saved in upgrading the network. This is the most important factor from the ISPs' perspective. ISPs need web caching for two main reasons, to reduce traffic and to reduce access time (service perceived as better by users).

1.1 Motivation

In our study, we present a model for placing caches in single server tree network (i.e. for any request, the destination is always the internet). This thesis is focused on a real-world scenario, in which we intend to reduce the cost ($) of buying caches and at the same time we also propose to reduce the average client access latency (time taken to get the requested document). This is very important for any ISP for the following reasons:

- It may not be essential to add a cache to the network, if adding a cache in the network will not have an impact on reducing the average user access latency.

- Based on the given budget, even small scale ISPs can manage their networks efficiently, by placing caches inside their network.

Hence, in order to address the above discussed issues, we design an optimization model. This model will not only find the best location to place caches inside the network; but will also reduce the average access delay for the clients' requests, according to the available budget.

To solve the above mentioned problem, we proposed a mathematical model, which will place caches inside the network optimally based on the available budget. This model will reduce the cost of buying caches, in an effort to improve performance, after the maximum achievable performance is attained. There is also a possibility of the following question being raised; "What will happen if we cannot attain the maximum performance within the given budget?" In this case, our proposed model will try to attain the maximum performance (reduce latency) that can be achieved within the allocated budget.

1.2 Objective

The objective of this research is to optimally place web caches within a Regional Internet Service Providers (ISP's) network within an allocated budget. In order to plan for placing web caches within the regional ISP network, a good understanding of traffic flow within the network is essential. We study, analyze, optimize and validate models of the existing network to provide guidelines for future network cache location planning. The approach used in this thesis is:

1. To collect trace files from the existing web caches placed within a Regional ISP.
2. To analyze the trace files to get the current network traffic characteristics and the amount of traffic estimation for the future.
3. To estimate the current and future web cache requirement
4. Come up with a mathematical model, to solve cache location problem for a Regional ISP.

5. To build a network simulation model to validate the mathematical model for web cache location problem.
6. To evaluate the performance using the new optimal model for the regional ISP that connects the major cities in Saskatchewan.

While some researchers have been focusing on placing caches on the edge of the network in form of proxy or web caches, other researchers have been focusing on placing the web caches in front of the original server to reduce the load on the server [KRIS98]. The approach in this research focuses on placing web caches on any node/nodes within the SaskTel network, which minimizes the overall network traffic on the regional ISP and reduces average latency to the clients. Figure 1.1 shows the network topology used throughout this thesis.

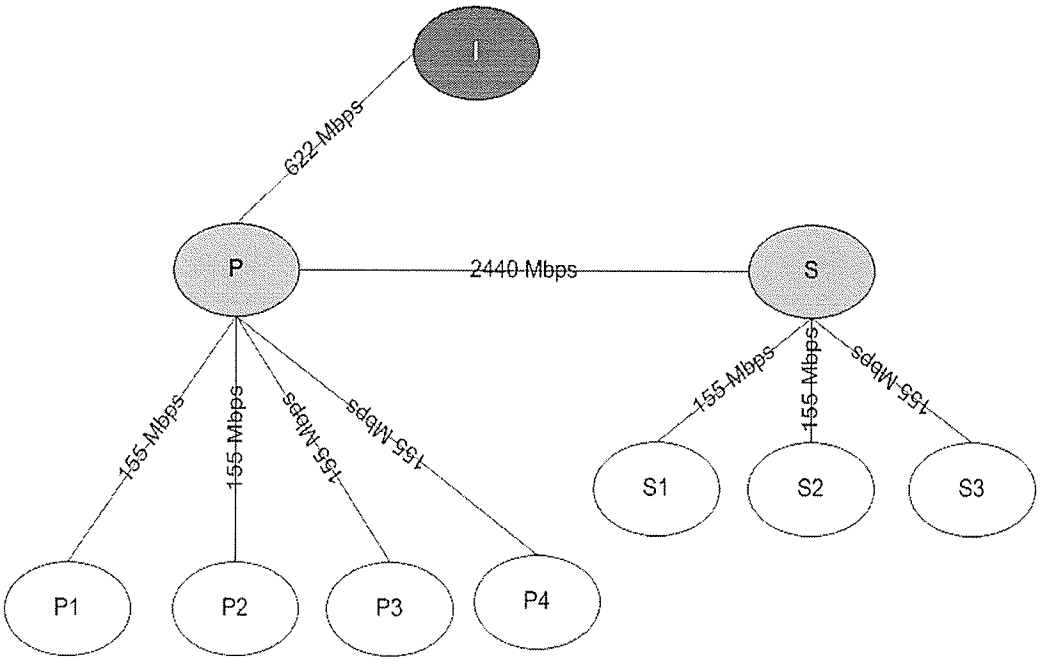

Figure 1.1: Regional ISP Network Topology

Hence the goal of our thesis is to place the web caches optimally for a regional ISP. Figure 1.2 explains the goal of this thesis in detail. The curve in the figure shows, as

we increase the budget (i.e. increasing number of web caches) latency of requests reduces. We aim at choosing a budget that would maximize the throughput or minimizes the latency and any increase in the budget would not greatly affect the system performance. From figure 1.2, we are interested in the "Budget needed" point on x-axis, as we see that any increase in the budget beyond "Budget needed" would not improve the latency (referring to y-axis).

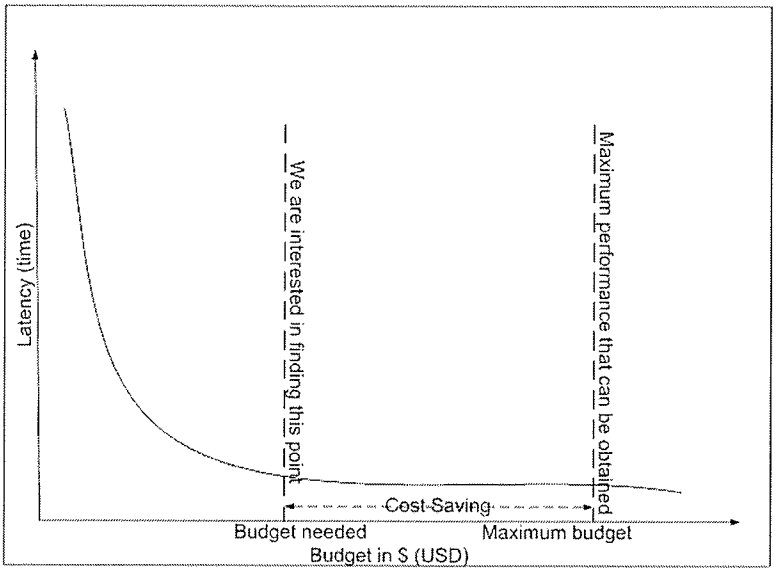

Figure 1.2: Proposed model for saving cost ($)

1.3 Problem Statement

One of the problems that Internet Service Providers (ISPs) are experiencing is higher bandwidth requirements due to growing internet use. Efficient placement of caches inside their network is one of the methods to solve the problem. In recent years, competition between ISPs has increased rapidly due to the fact that one ISP is superior to another in providing faster access or in providing the internet service at a cheaper price. In order to attract customers, ISPs are spending a lot of money on buying caches and placing them in the networks, in order to improve performance.

Hence, from the ISP's perspective, we can state the problem as; "how to minimize the load on all the links in the network, which in turn maximizes the performance of the network within a given budget".

1.4 Organization of the thesis

The rest of the thesis is organized as follows:

Chapter 2 discusses the background study of caches.

Chapter 3 discusses previous and current work that is related to this thesis.

Chapter 4 provides an overview of the Sasktel workload characteristics and analysis of the current cache requirements.

Chapter 5 describes the mathematical model and the assumptions used in the optimization model.

Chapter 6 describes the simulation model and assumptions used.

Chapter 7 provides the results of optimization and simulation models and a brief discussion.

Chapter 8 gives the summary and conclusion of the thesis and recommended future work.

Appendix A provides an introduction to optimization tools and concepts.

Appendix B provides an introduction to simulation tools used and trace formats used in simulation model.

Appendix C provides definitions of most frequently used terms in this thesis.

Appendix D provides results of optimization and simulation models.

CHAPTER TWO

2 Background

Web caching is the temporary storage of web objects such as HTML (Hyper Text Markup Language) documents. The advantages of using web caching include reduction in bandwidth consumption as fewer requests and responses need to go over the network. Web caching reduces the load on the servers by reducing the number of requests that are directly served by the server. Using web caches reduces latency for requests, as the caches are closer to the clients. Another advantage of using web caches occurs when the original server cannot be reached.

Web caching is a promising approach to the problem of rising Internet and Intranet traffic for three main reasons: quality of service, surge protection (Surges occur when a very large group of users wants access to the same small number of pages), and overall traffic reduction.

A cache can be configured as a proxy for browser users, or it can be transparent to browser users. Virtually all cache products can be configured to operate in either form. With some additional engineering by the vendor, caches can also be configured as server accelerators, or "reverse proxy" caches. Figure 2.1 shows the location of the cache in the network. Figure 2.2 shows basic steps when a client requests for a document. In step1 client makes a request for a document, step2 the request is passed to the cache engine, in step3 if the cache does not have the requested document cached, the request is forwarded to the origin server. The server responds with the requested document, the cache retains a copy of the document and, in step 4 the cache serves the request. If the document is in the cache, this is considered a cache hit. If the document is obtained from the origin server it is considered a miss.

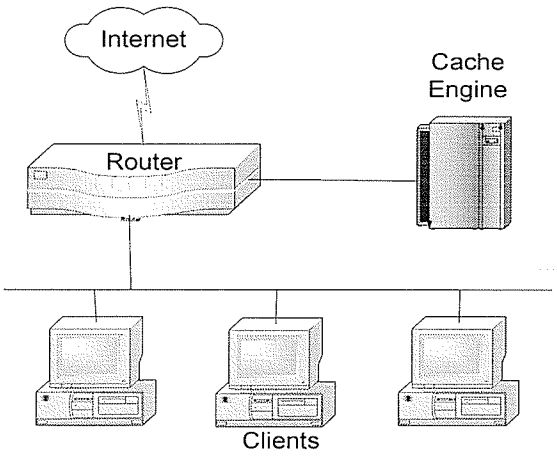

Figure 2.1: Location of Web Cache in a Network

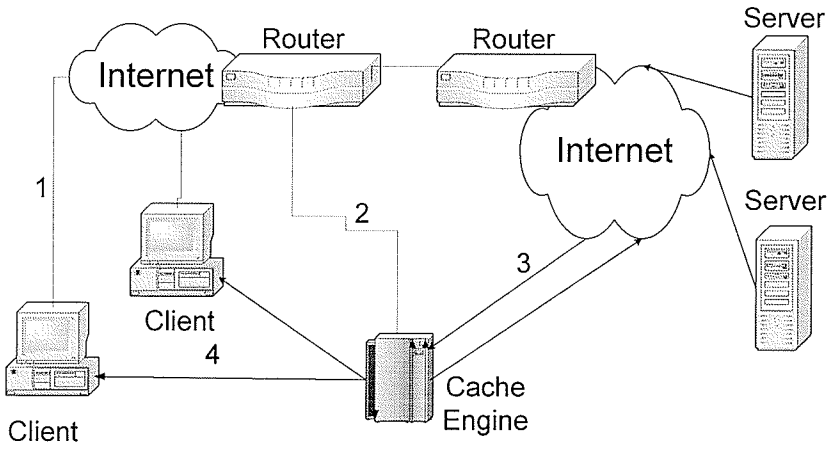

Figure 2.2: Web Cache Functionality

The basic issues that are important when dealing with any caches are listed below:

- Cache Architecture
- Cache Routing
- Cache Prefetching
- Cache Replacement
- Cache Coherence

- Cache Placement
- Cache Resource allocation
- Cache Scalability
- Cacheability of web objects

2.1 Caching Architectures

Hierarchical Caching Architecture

In a hierarchical caching architecture caches are placed at multiple levels within the network as shown in Figure 2.3. Client caches are placed at the lowest level of the hierarchy. If a client cache does not satisfy a request, the request is redirected to the institutional cache. If the document is not present at the institutional level, the request travels to the regional cache, which in turn forwards unsatisfied requests to the national cache. If the document is not present at any cache level, the national cache contacts the origin server directly. When the document is found, either at a 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 request finds the document [RODR99] [CM1].

Advantage of using hierarchical caching is that document can be retrieved at any level of the network. Problems with hierarchical caches are 1)cache servers need significant coordination between each other, 2)additional delay is added if the documents are not found at the lower level, 3)high-level caches causes bottleneck, and 4)there are multiple copies which are placed at every hierarchical level. Some of the projects that are using hierarchical caches are Harvest, Adaptive, and Access Driven.

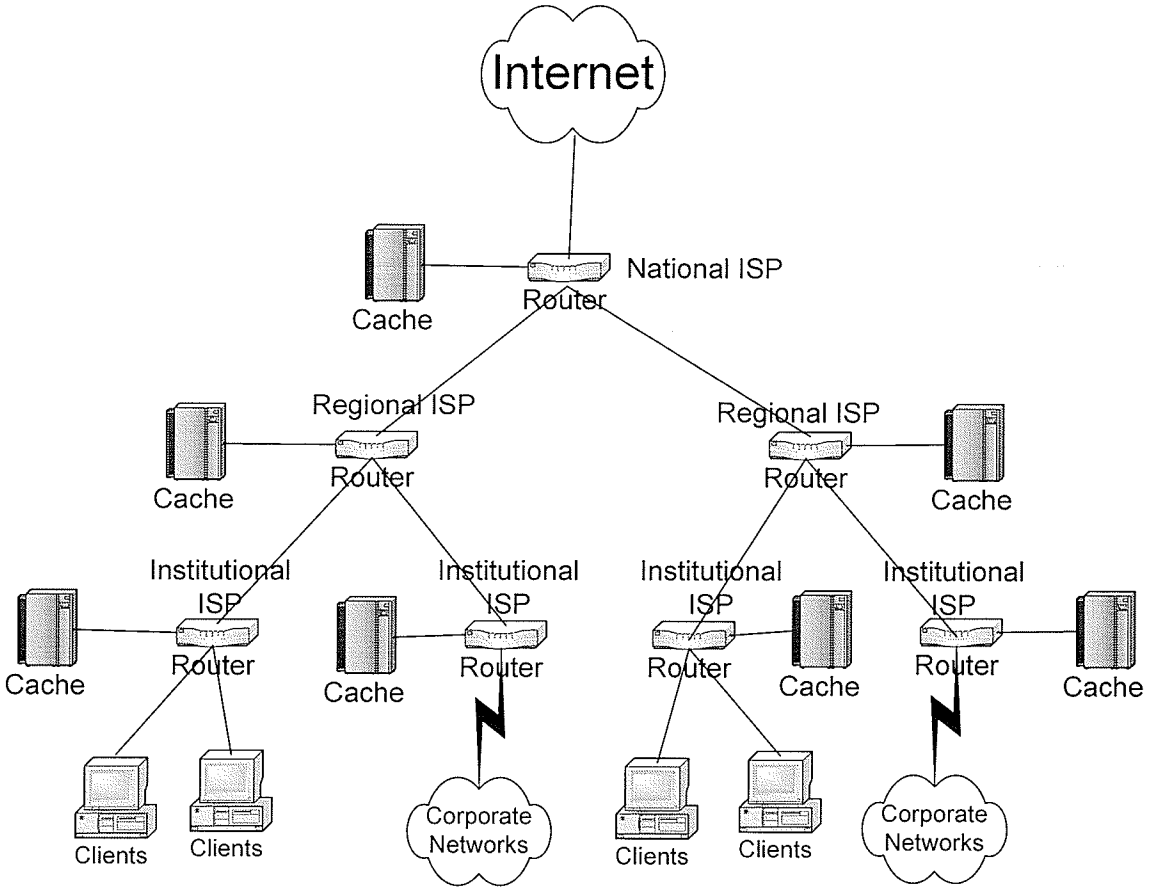

Figure 2.3: Hierarchical caching model with different levels

Distributed Caching Architecture

With distributed caching, no intermediate caches are set up, and there are only institutional caches at the edge of the network that cooperate to serve each other's misses. Since there are no intermediate caches that store and centralize all documents requested by lower level caches, institutional caches need other mechanisms to share the documents they contain. Figure 2.4 shows the distributed architecture [RODR99]. Some of these mechanisms are described as follows.

- Institutional caches can query the other cooperating institutional caches for documents from local misses (this is usually done using the Inter Cache Protocol ICP.) However, using a query-based approach may significantly increase the bandwidth consumption and the latency experienced by the client since a cache

needs to poll all cooperating caches and wait for the slowest one to answer [CM1].

- Institutional caches can keep a digest or summary of the content of the other cooperating caches, thus avoiding the need for queries/polls. Content digests/summaries are periodically exchanged among the institutional caches. To make the distribution of the digest/summary more efficient and scalable, a hierarchical infrastructure of intermediate nodes can be used. However, this hierarchical infrastructure only distributes information about the location of the documents but does not store document copies [CM1].
- Institutional caches can cooperate using a hash function that maps a client request into a certain cache. With this approach, there are no duplicate copies of the same document in different caches and there is no need for caches to know about each other's content. However, having only one single copy of a document among all cooperating caches limits this approach to local environments with well-interconnected caches. In short every cache keeps a meta-data table, hierarchical structure for meta-data and helper list in every other cache [CM1].

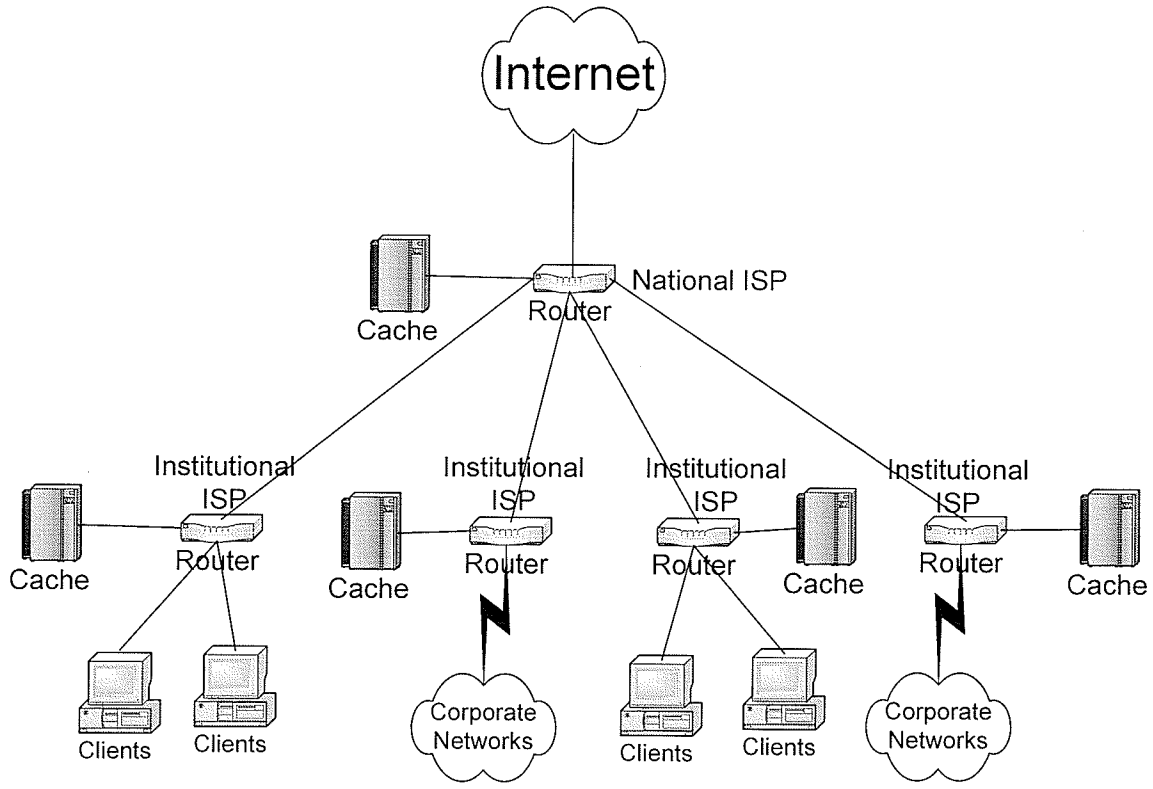

Figure 2.4: Distributed Caching model

Advantages of using distributed caching model are better load balancing among the caches, less congestion due to less traffic on the network, and no additional disk space is required. Disadvantages of using distributed caching model are high connection time (access delay), high bandwidth is required and administrating these caches is complex. Projects like ICP (Internet Cache Protocol), and CARP (Cache Array Routing Protocol) use distributed caching model.

Hybrid Caching Architecture

Hybrid caches cooperate at every network level of a caching hierarchy. Hybrid caching schemes combine advantages of both hierarchical and distributed caching. Hybrid caching model have many advantages; it reduces both connection and transmission time, minimizes total latency, lowers the bandwidth usage, doesn't have multiple copies in different levels of the hierarchy, reduces disk requirements for the

caches, reduces the load on the caches even in the top-level, reduces the load on the server. Projects like ICP use hybrid caching model [CM1].

2.2 Cache Prefetching

Prefetching is the speculative retrieval of resources into a cache in anticipation that it can be served from the cache in the future [DAVI01]. An advantage of using prefetching is it increases hit ratio. According to studies caches usually gain hit ratio <50%.

Some of the methods used for prefetching are:

- **Pull Caching:** The server sends out copies of the requested document to the caches that a user went through to get at the server
- **Push Caching:** The server puts data into caches without client input. Where the server puts the data usually depends on cache location. For example, geographical push caching will push data to the caches that are closest to its clients.
- **Server hint based caching**: Data from the server, such as frequency of access of certain documents and origin of document request, are used to determine what to pre-fetch and where.
- **Local based:** This method attempts to predict which document will be requested most often, and use strategic caching of these documents. The method relies on reference patterns to predict what to prefetch next.

Prefetching can be implemented between clients and server, between proxies and servers, and between clients and proxies [WANG99]. Between clients and servers the documents are predicted by using user's history, early studies show that prefetching from web server to individual clients can reduces the client's latency by 45% by doubling the cost of network traffic. Prefetching between proxies and servers reduces the client latency by up to 60% for high bandwidth clients. Push technology, where most popular documents are pushed to web proxies, which in turn push these documents to the clients, can anticipate 40% of the user's request. Prefetching between clients and proxies predict user's next reference, use networks idle time for

fetching the predicted documents, and it improves performance and reduces the client's latency by up to 23.4%.

Caching algorithms can be classified as passive or active algorithms. Passive algorithms do not use prefetching mechanism where as an active algorithms use some form of prefetching. The metrics used for evaluating caching algorithms include hit/miss rate, response/access time, and bandwidth utilization.

Caches prefetch pages based on the content of the pages being served, like CacheFlow (now known as blue coat) and Wcol. need to have access to the links within web pages, these links are not available from logs. Even if page contents were logged, caches that perform prefetching may prefetch objects that are not on the user request logs and thus have unknown characteristics of size and retrieval costs. Current support for prefetching in HTTP/1.1 is insufficient because prefetching with GET is not good. Davison argue that 1) GET is unsafe because in practice, GET requests may have (not insignificant) side-effects, and 2) GET can unfairly use server resources. As a result, Davison assert that prefetching with GET is not good. Existing prefetching implementations can cause problems with undesirable side effects and server abuse because the prefectcher may not know that the object requested is very large and may take long time to arrive, and the potential for these problems may thwart additional prefetching development and deployment [DAVI01].

2.3 Content Replacement Techniques, Coherence/Consistency and Protocols

Cache Content Replacement

After caching the various cacheable objects requested by clients, a cache will eventually become full. Thus, when a new document needs to be cached, something else that has already been cached should be evicted from the cache. This requires a replacement policy, which will select one or more documents to be evicted from the cache so as to make room for the new document. There have been many removal policies proposed, a few of them are outlined as follows. Replacement Algorithm can

be classified as Traditional policy, key based policy and cost based policy. Combination of these policies can be performed for different traffic patterns [CM1] [CAI99] [KRIS99].

Traditional Replacement Policies

- LRU (Least Recently Used): In this algorithm the document that was not used for along period of time is replaced by the new document.
- LFU (Least Frequently Used): Document that is accessed less frequently is replaced with the new document, if there is no empty space for the new document.
- Pitkow/Recker: Replaces the objects in the LRU order, but if the objects are accessed within the same day, the largest size document is replaced first.

Key-Based Replacement Policies

In key-based replacement policy, replacements of objects are done based on the key/keys metrics (parameters).

- Size: Evicts the largest document. The objects are removed in order of size, with the largest object replaced first.
- LRU-MIN (biased in favor of smaller objects): Suppose we have an object S that needs to be placed in the cache. If there is an object, which has size of at least S, then remove and replace with new object, else remove objects in LRU order of size of at least S/2. The process continues until the object fits in the cache.
- LRU-Threshold (never cache too large object): Is the same as LRU, except documents larger than a certain threshold size are never cached.
- LLF (Lowest Latency First): Tries to minimize average latency by removing the document with the lowest download latency first.
- Hyper-G: is a modification of LFU, with last access time and size considerations.

Cost –Based Replacement Policies

Replacement policy is based on the cost which depends on factors like access time of the object from the original server, entry time of the object into the cache, expiration time of the object, transfer time of the object to the requested client and so on.

- **GreedyDual-Size (evict lowest cost/size):** GreedyDual-Size algorithm incorporates locality with cost and size to decide which document to evict. This replacement policy can enhance hit ratio. Greedy Dual-Size algorithm takes into consideration factors such as locality, size and latency/cost.
- **Server-assisted scheme:** Here the object of least value is evicted first. The factors that affect the least value are cost of fetching the object, size of the object, price of the cache between the requests for a document.
- **Hybrid**: This scheme aimed at reducing the total latency. A function is computed for each document, which is designed to capture the utility of retaining a given document in the cache. The document with the smallest function value is then evicted. The function for a document p located at server s depends on the following parameters: c_s, the time to connect with server s, b_s the bandwidth to server s, n_p the number of times p has been requested since it was brought into the cache, and z_p, the size (in bytes) of document p. The function is defined as [WOOS97]:

$$\frac{(c_s + \frac{W_b}{b_s})(n_p)^{W_n}}{Z_p}$$

Where W_b and W_n are constants. Estimates for c_s and b_s are based on the times to fetch documents from servers in the recent past.

Cache Coherency

It is necessary to ensure that a cached object accessed by some client is the same as the object at the originating server at the instant it is being requested. If a stale object is returned to the client, it could lead to incorrect results or improper decisions.

To ensure cache coherence we need to estimate when a particular object that is cached becomes stale. If object is stale, we update the copy from the server, or simply discard the object so that next time if the object is requested, the current up-to-date copy of the object is retrieved. Periodic checks could be made to compare the timestamp of the cached object with the timestamp of the object at the originating server to see if the cached object is still up-to-date. Information about the validity of the object could be included in the headers when the server sends an object to the

proxy server. Using information provided by the server in the headers, the proxy could decide when the particular object will become stale.

Cache Coherency Approaches

HTTP defines several headers to support caching which are listed below

1. If-Modified-Since: A date can be appended to a GET message to obtain a conditional GET. It is being replied to by returning the requested page only if it has been modified since that date [GRACIA].
2. Pragma: A no-cache pragma can be attached to a GET message to indicate that the page must be retrieved from the remote server irrespective of whether it is in some cache on the way up, or whether it was modified or not.
3. Last-Modified: A date is returned with every message requested by a GET to indicate the last time the page was modified [GRACIA].
4. Date: A date is returned with every requested page to indicate the last time the document was considered to be fresh. This date, in combination with the `Last-Modified` date gives the user an idea of how stale the data is [GRACIA].
5. Expires: A date may be returned with a document to indicate for how long the document will remain unmodified.

Strong Cache Consistency

1. Client Validation or Polling-every-time: In client polling whenever an object is fetched from the server by the client, a timestamp is also included. If a client requests for an object the cache checks to see if the time stamp in the cache is same as that in the server, if it vary it gets the new object from the server.
2. Server Invalidation: In the case of server invalidation, the server keeps track of all the caches that are caching its objects and whenever the objects are modified it send the caches with modified objects. Callbacks improve cache consistency as well as save network bandwidth by not requiring clients to poll servers periodically but there are clear scalability issues and privacy/security concerns regarding this approach, since servers need to keep track of all the caches caching its objects.

Weak Cache Consistency

1. Adaptive TTL (Adaptive Time to Leave also called as Alex protocol): Adaptive TTL uses an update threshold mechanism to determine how often to perform polling to the server. Adaptive TTL handles the problem by adjusting a document's time-to-live based on observations of its lifetime. Adaptive TTL takes advantage of the fact that file lifetime distribution tends to be bimodal; if a file has not been modified for a long time, it tends to stay unchanged. Thus, the time-to-live attribute to a document is assigned to be a percentage of the document's current "age", which is the current time minus the last modified time of the document [WANG99].

2. Piggyback Invalidation: There are three invalidation mechanisms that are proposed. Piggyback Cache Validation (PCV), Piggyback Server Invalidation (PSI) and hybrid approach which is the combination of PCV and PSI. In PCV, whenever cache needs to communicate with server, the cache piggybacks a list of cached request that are stale for validation from server. In PSI server piggyback on reply to a proxy, the list of resources that have changed since the last access to the server by the client is sent. A proxy client could also combine the PSI mechanism with PCV to create a hybrid approach where the choice of the mechanism depends on the time since the proxy last requested invalidations number of cached resources [WANG99]. For smaller time gaps PSI mechanism is used, for larger time gaps PCV is used.

Caching Protocols

- **ICP – Internet Cache Protocol (RFC 2186, RFC 2187 for Applications) ICP ver. 2:**

Internet Cache Protocol developed by Peter Danzig, et al. at the University of Southern California is a message format used for communicating between Web caches. Web caches use HTTP to transfer data object, but caches benefit from a simpler, lighter communication protocol. ICP is mainly used to locate a Web object in

its adjacent cache. If a request is made to a cache in a mesh and if the cache does not have the requested object, it sends a query to its neighboring cache using ICP request, the neighboring cache replies with a HIT or a MISS" [WCP] [CM1] [WCRPS].

- **Simple Hit-Metering and Usage-Limiting for HTTP (RFC 2227):**

Most of the origin servers collect the demographic information or control the number of times the cached object was used. The origin server uses techniques like cache-busting to get demographic information or control the number of times the cached object was used, this technique uses network resources, so to minimize the usage of the network resources Simple Hit-Metering and Usage-Limiting for HTTP was proposed in which new features like Hit-metering. Usage-limiting features were added to HTTP. Hit-metering allows an origin server to get the counts of the number of clients using a resource instance via a proxy cache, or a hierarchy of proxy caches and Usage-limiting allows an origin server to control the number of times a cached object may be used by a proxy client, proxy cache or a hierarchy of caches, before it is modified by the origin server.

- **Server Cache Synchronization Protocol (SCSP) (RFC 2334)**

Server Cache Synchronization Protocol attempts to solve the problem of generalized cache synchronization/cache-replication for distributed protocol entities. SCSP synchronizes caches of a set of server that are in the same server group. Server groups are identified by group ID's also called as SGID.

- **Web Cache Communication Protocol (WCCP)**

Web Cache Communication Protocol (WCCP) developed by Cisco system establishes and maintains transparent redirection of the requested type of traffic flowing through a cluster of routers. The protocol specifies interaction between the routers or between the layer3 switches. The traffic was redirected to the cluster of routers to improve the client's response time and also to improve the network bandwidth requirements [WCCP].

- **Cache Digest**

Cache digest is primarily used to share summaries of the URLs that are cached by a given server. Cache digest is both a data format and a protocol for fetching cached objects over the network [CACHED]. Cache digests provide peering between cache servers without exchanging a sequence of request/responses, which causes the problems of latency, and congestion related issues as in some protocols like ICP and HTCP.

- **Cache Array Routing Protocol (CARP)**

CARP provides scalability and efficiency of an arrayed network of servers. It uses hash-based routing to provide a deterministic path for request resolution through an array of proxies. The request resolution path, based upon a hashing of proxy array member identities and URLs, means that for any given URL request, the browser or downstream proxy server will know exactly where in the proxy array the information will be stored, whether already cached from a previous request, or making a first Internet hit for delivery and caching. CARP provides 2 powerful benefits, one is the deterministic request resolution path and the other is that CARP eliminates the duplication of content in an array of caches. The advantages of these are faster response time for the requests and efficient use of resources [CARP].

- **Hypertext Caching Protocol (RFC 2756)**

The Hypertext Caching Protocol is employed within Internets to identify Hypertext Transport Protocol (HTTP) proxies used to cache, typically web data. HTCP allows request and response headers to be used to manage the caches and also to expand the domain of managing the cache which includes remote monitoring of cache's addition, deletions and also hints about web objects, such as the third party locations or the unavailability of web objects.

2.4 Cache Location/Placement and Replication

Placement can affect performance of caches [KRIS00]. It is always recommended to place the caches close to place where the traffic flow is maximum and close to place where network bottlenecks occur often. The location of web caches should be a function of traffic flow and network topology.

The criteria for placing caches are:

- Place caches on the users side of a bottleneck
- Place Web cache servers close to the flow of traffic
- Place a local Web cache server close to users Internet connection
- Place Web cache servers where more networks join together

Caches can be placed at different levels in the network. Generally the levels are first level web caches and upper level web caches. The first level web caches are placed as close as possible to the placers external internet connection, close to the users and if there are several first level caches, the better place to place the cache is where the network come together. Typically the first level caches are placed inside the campus, where the institutions/organization network meets. Upper level web caches are placed where network joins, on or close to a national Internet exchange point and on or close to an international Internet exchange point. Basically the upper level caches are placed closer to the national Internet exchange point. It is recommended that more than 3 levels of caches should be avoided, as it may be difficult to maintain. Chapter 3 discusses cache location/placement in detail.

Replication: The owners of the data place copies of the data at multiple locations, preferably close to the intended users. This is known as replication (cached copies).

Resource Allocation Policy in Web Caching

Resource allocation mainly deals with the allocation of resources between the caches. If there is more than one cache there should be a means to interoperate between caches, so that the workload is distributed among the caches. The resources of the storage server namely cache, memory and CPU are shared in an adhoc manner among the clients. These resources play an important role in determining the overall throughput and latency of data-access. Ercetin and Tassiulas [ERCE01] proposed a

market based allocation policy for a two-stage server-proxy model. Their model represents a real word situation, where the servers and users will not collaborate to achieve the system optimal solution. Instead every competing agent competes for the maximum benefits without consideration of others and this competition among proxy caches leads to a solution that is better than the solution provided by conventional caching methods. A price-directed market-based distributed algorithm was proposed to solve the two-stage server-proxy cache resource allocation problems. In this algorithm initial set of prices are announced to the servers. The servers determine their resource (cache) demands according to these prices as well as the request rates, and the observed delays from the proxies. The servers request these resources from the proxies. Prices are then iteratively changed to accommodate the demands for resources until the total demand equals to the total amount of resource available [ERCE01]. A mechanism was proposed for caching resource allocation of multimedia objects, one of the reasons to cache multimedia object is that the cost of the hard drive is reducing drastically during recent years and cost to maintain these multimedia object which are large in size may not be reduced. The other reason to cache multimedia object is to reduce traffic load on the network. The mechanism that was proposed was a parent-child hierarchy to mimic the routing techniques in the Internet. Multimedia object that was considered is assigned to a home node. The client can query on the path from client to parent, if the object is found it is retrieved else obtained from the home node [KOZUCH].

2.5 Cache Scalability and Cacheability

Scalability

Scalability is the ability to add users without adversely affecting system performance for any one user. The challenges in designing and implementing any scalable caches are proper location of the cache, large cache routing table and out of date caching tables. Designers of web infrastructure are concerned about the scalability and deployment of many web caches as they are spread all over the Internet. Some of the common approaches to improve scalability in caches are to grow a distributed tree

away from popular server towards high demands sources and perform cache resolution by either using routing tables or hash functions.

Cache routing table: Malpani et al. [MALP95] found that if a group of caches interact with each other they would use the resources effectively. To make the system robust multiple servers cooperate to form large distributed cache. All servers were made functionally equivalent so that any server could handle the request of any client. Use of multicasting provided efficient use of network bandwidth. Provey et al. [PROV97] manually constructed a hierarchy that must be traversed by all requests. The main idea of the technique is to reduce the load on the top-level caches.

Hash functions: Some protocols like CARP allow query-less distributed caches using hash functions. The exact cache location of the object is obtained from the membership list and the URL. Whenever the URL is removed or added the hash tables are updated by adding new hash function and these are distributed among the caches. In Summary cache [FAN98], each proxy keeps a summary of the URLs of cached documents at each participating proxy and checks these summaries for potential hits before sending any queries. Two factors contribute to the protocol's low overhead: the summaries are updated only periodically, and the directory representations are very economical, as low as 8 bits per entry. Karger et al. [DAVI97] used a special hashing function called consistent hashing to construct per-server distribution tree with load balancing properties. A consistent hash function only needs minimal changes, when the range of the function changes. Consistent hashing technique is designed to relieving hot spots on the web.

Cacheability of User Requested Web Objects

Studies on the cache logs have shown that cached objects can be reused. Some objects like the embedded images do not change very often as the content of the document itself. So a means to cache the embedded objects as long as they do not change, need to be incorporated into existing caches. Studies have shown that most of the html documents do not change very often, but they do not have the last modified

date and so need to be retrieved from the original server. Also inclusion of cookies does make the content uncachable. Recent studies have shown that 30 % of the current documents have cookies, these documents are customized and it is inappropriate to cache them. Craig et al. [WILL99] observed that there is potential to reuse the cached objects that are currently being not used due to inaccurate and non-existent directives.

Dynamic Data Caching

The amount of traffic that can be cached on the Internet currently is limited [DYNAMI]. Only some fraction of the Internet data can be cached. The important issue is how to make more data cacheable and reduce the latency of accessing non-cacheable documents and improve the network performance. Use of accelerators or active caches can help to some extent.

Accelerators are placed infront of the web servers to speed user's access time. The accelerator assumes the IP address of the web server. The web server accelerator receives all requests addressed to the web server and handles those requests as if it were the web server. The web server takes a new IP address known only to the web server accelerator. The accelerator becomes the web server's only user, filling its cache with content requested by the user community in the intranet or Internet [TAWSM]. The accelerators allow both dynamic and static data to be cached.

Active caches support caching of dynamic documents of web proxies by allowing the server to supply applet through a Java application, called as "cache applet", the proxies that require documents will invoke cache applets without contacting the actual server.The applets do the required proceesing and return back to the clients. The role of the cache applet is to be invoked when there is a cache hit to the document. Active cache scheme can save the usage of bandwidth significantly at the expence of the CPU cost [CAO98].

Existing Caching Products

Operating System (OS) vendors and server hardware vendors like Sun, Novell and Microsoft provide their own caching products, there are some third parties that offer caching products that also support these operating systems. Most caching products can run on any platforms like Windows, Solaris and are also easy to be integrated with management tools that run on the same operating system.

CHAPTER THREE

3 Related Work

Significant research has been carried out related to optimization of cache performance [CHAN96], [WILL96], replication of servers for network load balancing [HEDD97], cache hierarchies [CHAN96] and co-operation between caches [FAN00]. All these studies suggest that one of the major factors that contributes to the performance (minimizes latency) of the network is the location of the cache/caches in the network. Studies have shown [DANZ93], [HEDD97] that placing caches inside the network where networks meet rather than placing them on edges, will improve the performance of the network. Most existing ISPs have started placing caches inside their network, resulting in the reduction of response time for client requests.

3.1 Importance of Cache Location

A study by Danzig et al [DANZ93] suggested that placing caches in the network's External Nodal Switching System (ENSS) would improve hit rate (the fraction of the requests that can be served from a cache). However, the performance depends on the dataflow of the backbone link. Danzig et al [DANZ93] also suggests that hierarchical internet caching outperforms other popular Internet cache implementations. The experimental results show that hierarchy does not measurably increase access latency. In this thesis, we use a tree topology.

Deploying caches within a private network of an organization has also become popular in recent years. This technique has the capability of reducing incoming traffic and providing faster access to popular web objects. Avella et al [AVEL03] argue that careful placement of caches inside any organization will have an impact on the overall performance of the network. Improvement in performance can be attained only if each link has enough bandwidth (i.e. every request that is requested is served either by a cache or a server).

Zhang et al. [ZHAN97] [ZHAN98] proposed an adaptive web caching structure using multicast for data dissemination to the caches. To assist caches making web query forwarding decisions, a URL routing framework was designed. For fast search within each cache group, neighbor caches share content information. This paper focus on reducing the waiting time and overhead associated with multicast queries [ZHAN97]. The paper also considers how a smooth transition from the current cache infrastructure to a new infrastructure can be carried out. Methods presented in our thesis could be used to optimally place adaptive caches.

A dynamic programming method presented by Tamir [TAMI96], could be used for solving the non-transparent cache location problem, in cases where requests are routed to the nearest web cache. The method presented in this paper would work only for a tree network having only one source node and all the routes were know to the program. Our thesis considers more than one source node.

3.2 Optimization Models

A study by Ercetin and Tassiulas [ERCE01] determined that the performance (average hit ratio) of a network can be maximized by deriving optimal strategies. They proposed a strategy, based on the idea that when a system reaches equilibrium, adding a cache to it will not improve the performance. This strategy helps in minimizing the number of caches that need to be placed in the network for optimal performance. The main objective of our thesis is related to this work (This paper provided the definition and our thesis has the implementation of the proposed strategy), where we set a particular budget. From the results of our thesis, we observe that any increase in the number of caches after a certain budget does not improve the performance of the network.

The location problem was addresses in [CUNH97] by Cunha for push servers and network load balancing. Cunha presents simple heuristics for load balancing which

can reduce traffic in the network significantly. This paper provided an introduction to location problem, but optimal solutions were not presented.

[BEST96] [BRES99] assume full dependency in a network of caches. I.e. if a particular page is found in a cache, then they assume that the page is found in every other cache in the network. Krishnan et al. [KRIS00] also used a similar model. This model simplifies the cache location problem by assuming that a document found in one cache, is placed on all other web cache in the network. This assumption is quite practical, because pages that are most popular among all the pages are stored in all the caches. The advantage of using the full dependency model is it reduces the complexity of the optimization problem, by considering a fixed set of documents (most popular documents) that can be cached in all caches.

Krishnan et al [KRIS99] present a location problem similar to the one presented by Cunha. In this paper the cache location problem was formulated for both general and transparent caches. In this study, optimization algorithms for line, tree and ring networks were also suggested. However, tree structured networks are considered to be more scalable than line and ring networks because tree networks allow easy addition of nodes to the network. The authors also suggest that the cache location problem for single server and multiple server cases can be solved using a greedy algorithm. The greedy algorithm places caches on the tree iteratively in a greedy fashion. It checks each and every node in the tree in order to determine where to place the first cache. Then, this algorithm places the cache in the node that minimizes the costs. Once the first cache is placed, the algorithm looks for an appropriate location for the next cache. Krishnan et al assume that if a request is cached on any cache, the request is cached in all other caches of the network. In this paper the authors wish to minimize the average access delay using an efficient dynamic programming algorithm for a single web server but we focus on multiple caches distributed in the network.

Another study by Krishnan et al [KRIS00] describes an optimal solution for placing caches in ring, line and tree networks. They consider the cache location problem for line, ring and tree topologies; have polynomial solutions in the case of a single server. They present an optimal dynamic programming solution for the cache location problem having a tree topology for a single server similar to [TAMI96]. Among all network topologies, the tree network topology is widely used to design a network. This is because real-world networks can be easily represented in the form of a tree (hierarchies). Krishnan et al also provide an optimal dynamic programming solution, which solves the tree network in $O(nhk)$ time, where n is the number of network nodes, h is the height of the tree and k is the number of caches placed in the network. However, this study illustrates how different topologies can be used and how they are related to the real world networks. The authors refer to most similar problems as NP-hard problems in general, but, in some cases, the problem can be solved in polynomial time, when the network topology has specific structure. The paper also presents a greedy heuristic which performs comparably to the optimal algorithm and requires less time for the experiments.

In contrast to the above paper by Krishnan et al, Qiu et al [QIU02], considered a different application of the same problem, involving the placement of web server replicas on the Internet. They evaluated the performance of a number of different heuristic solutions and developed several placement algorithms that use workload information, such as client latency and request rates, to make informed placement decisions. They evaluated the placement algorithms using both synthetic and real network topologies, as well as web server traces. Results show that placement of web replicas is crucial for network performance. The results also showed that for the general network case, a greedy algorithm that places replicas based on a distance metric as well as load information performed the best. This model considers the problem from a relatively narrow point of view. They consider a fixed number of caches. They do not consider the cost of caches, the capacity of caches or the cost of moving caches within the network.

A recent study by Li and Shen [LI04] on the proxy placement problem for tree networks and autonomous systems presented a novel mathematical model. They proposed dynamic programming-based algorithms and showed that the algorithms converge theoretically to the optimal solutions. Finding the optimal positions for placing a fixed number of web proxies in the network is a challenging task. This paper considers the problem of computing the optimal locations to place a number of web proxies for tree networks such that the overall access gain (time to access the requested documents during certain period of time) is minimized. The optimal placement is obtained using a dynamic programming-based algorithm. The author describe that the time complexity of the algorithm is $O(n^2)$, where n is the number of nodes in the network. They also extend the model to autonomous systems and present a dynamic programming based algorithm, whose time complexity is $O(n^2k)$. The model proposed in this paper is not suitable for all types of network topologies and also it does not consider reducing the traffic on the bottleneck links. This work is related to [KRIS99], where k caches are placed optimally in a tree network.

Another study by Diamond [DIAMO1] suggests that different types of caches (these caches vary in peak throughput) can be purchased and placed in the network depending on the amount of load at each node in the network. In this model a web caching infrastructure for a period of over five years was planned, using an integer programming model. The model determines the caches are needed in each year and where exactly they have to be placed in the network. This paper discusses the requirement planning of network caches for a period of 5 years based on the network traffic estimation. In our model we are interested in placing the caches in the network which will reduce mean time to transfer one unit of data from client node to the cache/original server node based on a budget that can be invested for a current year. The models presented in [KRIS99] and [DIAMO1] do not include costs for maintaining caches of variable sizes. In our model we solve the optimization problem for variable cache capacities. Most of the papers we have studies have not expressed the aspect of the cost of network traffic on the backbone links. The cost of network traffic on backbone links could be used, in an indirect way, to model a desire to move

caches closer to clients in order to improve performance from the perspective of the clients and also reduce the network traffic.

3.3 Mixed Integer Programming Models

Mixed Integer programming models have been widely used to solve cache location problems. Koskela et al [KOSKEL] propose that mixed non-linear integer programming models perform better compared to linear models for web cache optimization problems [KOSKEL]. In their paper on web cache optimization, they propose that the problem in web cache optimization is to decide which strategy to use in replacement of cache objects. While commonly used policies use heuristic rules, the proposed model estimates the value of each web object by using features collected from the HTTP responses and from the HTML structure of the pages. A case study was performed using both a linear model and a non-linear mixed integer model to classify about five thousand web pages according to their popularity. Results show that the linear model does not find any correlation between the features and document popularity, but the non-linear mixed integer model gives better results. In this paper the main focus was on replacement policies of web pages, and results have shown that using a non-linear mixed integer model would yield better solutions for solving the cache replacement strategies.

Wierzbicki [WIER02] suggests that Mixed Integer Linear Programming (MILP) can be used to solve the cache location problem. The paper discusses models of the Cache Location Problem from Decision Support Systems (DSS- System that support decision-making activities) perceptive. The approach leads to the design of models that allow the user the flexibility to specify his/her preferences, and are precise in the search for solutions that are closest to preferred solution. The models designed used many constraints and criteria and are specified using Mixed Integer Linear Programming. The paper suggests that medium sized problems can be solved optimally with state-of-the-art MILP solvers in reasonable time. The author presents a general discussion of designing a DSS for placing caches on a network. The paper

includes a number of different criteria, which may be used to decide on the location of caches. The mean delay to transfer data, the cost of the caches, the number of existing caches which needed to be moved, and a measure of fairness for the service obtained by clients at the various nodes in the network are considered. The paper also discusses about the variation of hit rates as a function of the number of clients assigned to a cache. The author focuses on different parameter settings for solving an optimization problem for a cache location problem. Results of two different approaches for a small example problem were conducted and compared. The paper focuses on how to extend existing heuristics to solve larger problems using the newly developed models.

Models proposed in [WIER02], [XUEY03] and [KOSKEL] focus on the cache location problem for Decision Support Systems and on page replacement policy respectively. In all these papers we observe that Mixed Integer Linear/non-Linear Programming would be a best bet to solve a complex problem. In our thesis we focus on solving the cache location problem using integer programming with added complexity of a non-linear objective function.

T. Koskela et al [KOSK03] designed a LP (Location Problem) algorithm for optimization of cache location. They considered 2 stages in the optimization problem. In the first stage, the objective function for the location problem is chosen. In the second stage, the performance metrics which describe the location are chosen. Performance metrics are chosen according to a particular objective set [KOSK03]. This book was helpful in defining the various stages involved in defining an optimization model. The details of implementation of the cache location problem are not explained in this book.

3.4 Discussions

Optimizing the location of web caches is an effective method to save network bandwidth, alleviate server load, and reduce latency experienced by end users. The

most common problem with web cache placement is to compute the optimal locations for placing web cache proxies in a network such that the objective function is minimized or maximized. In this thesis, we address the cache location problem for a regional ISP network. We propose a mathematical model to solve the problem for both fixed and variable cache capacity models. We aim at maximizing the overall performance of the network, specifically the mean time to transfer one unit of data within a network.

Having looked at the cache location problem under various scenarios, we observe that, most studies in the literature provide solutions for the single server case, or for placing a fixed number of caches within a network or for placing caches optimally in a given network topology. Many factors, such as delays caused by bottleneck links, cost of cache capacity, demand on the links, maximum link capacity were not considered. In our thesis we address all these parameters, along with the maximum budget that can be invested by the ISP provider.

CHAPTER FOUR

4 Analysis of Web Cache Performance at Internet Service Providers

The objective of this chapter is to provide an analysis of the ISP's network traffic, web caching performance and a forecast to estimate future requirements for 7 small cities and the 2 large cities that connect to the ISP's network. This chapter describes an analysis of trace files from web caches on the ISP's network. Planning for the network needs a good understanding of traffic flow characteristics and network topologies.

4.1 Motivation

Web caching has been widely employed to improve network performance by reducing traffic, reducing load on servers and reducing user latency. Internet Service Providers have started deploying web caches within their networks to reduce latency and utilize the resources in an efficient way. Decreasing the traffic on the links connecting to the Internet will save cost to the ISP's. The trace files collected from the ISP enables us to plan for web caches required in the future based on the current traffic flow. Understanding trace files will help in studying the nature of the traffic flow and help in defining performance measures and interpreting the optimization and simulation results.

4.2 Analysis Based on Trace Files Collected from Regional ISP

The goal of the trace files assessment was to gather information on the nature of the web traffic and performance of the web caches. The assessment will be used to predict the amount of traffic that can be cached. The assessment was to be carried out considering following factors that affect the performance of the network traffic:

- Web cache performance
- Type of web content cached
- Co-operation between web caches
- Sizes of the web documents

- Topology and peak total down stream traffic
- Location of web caches

4.2.1 Web Cache Performance

The trace files (cache logs) collected from the SaskTel network were used to test the performance of the web caches. Table 4.1 shows that hit percentage ratio (number of documents that are served from the web caches rather than the original server) of 7 trace files collected from ISP's network is around 55%-65%. The byte hit ratio (total size of hit documents/ total size of all pages) was around 29- 35%. From analysis the hit ratio (hit documents/ total number of documents) was good, so performance of the web caches is acceptable with respect to document hit ratio and byte hit ratio.

A long time analysis performed on the ISP's network traffic showed that approximately 30% of HTTP (byte hit ratio of HTTP traffic) traffic is served from the caches and rest is served from the origin server. This hit percentage of the HTTP traffic is used as the value of the parameter α (byte hit ratio) in chapter 5.

Description	% Hits	Byte hit ratio
Test1	53.13%	32.22%
Test2	55.45%	34.22%
Test3	60.5%	31.48%
Test4	63.70%	34.66%
Test5	62.93%	32.26%
Test6	62.23%	33.33%
Test7	61.02%	29.89%

Table 4.1: Web cache performance

4.2.2 Web Content Types

We performed test to study web content types, Table 4.2 describes the details of all the content type and the corresponding hit percentages of HTTP traffic (number of

pages that were found in the caches and were hits). These results were obtained by analyzing 10 trace files collected from the ISP's network. This study was performed to study the content types in the trace files.

Content	Total percentage	Hit percentage of the content
Dynamic data	20-25%	5-7%
Not-Proper-format	30-35%	75-80%
Image	30-35%	70-75%
Html	5-6%	18-30%
Others	> 5%	20-25%

Table 4.2: Hit percentages based on the content type

4.2.3 Co-operation Among Caches

We performed a test to give an estimation of request for data that is common on 2 different traces files collected from 2 caches that were placed in the SaskTel network. This test was carried out to examine the inter-operation between two caches and how this would impact the performance of the caches. Table 4.3 explains in detail the traffic due to different caches namely cache 3 and cache 4 which were placed at node P3 and node S in Figure 1.1.

Description	
Traffic due to common files that were hits on both caches. i.e. Request that were common in both the caches 3 and 4 and were marked as served by cache in cache logs	21.79%

Traffic due to common files that were misses on both caches. i.e. Request that were common in both the caches 3 and 4 and were marked as served by server in cache logs.	52.23%	
Total traffic that was created due to common files in both the caches (i.e. cache 3 and cache 4). (Percentage of traffic that were existing in both the caches logs)	37.01%	
Description	**Cache3**	**Cache4**
Traffic due to hits (Number of documents that were served by cache)	37.38%	60.45%
Traffic due to miss (Number of documents that were not served by caches)	60.45%	39.54%
% of documents that are not common on both caches (Documents that were not common on both the caches logs)	92.6%	87.6%
% of documents size that were same on the 2 caches for hits misses (Documents that were common on both cache logs and marked as served for cache)	4%	2.6%
% of documents size that were same on the 2 caches for misses (Documents that were common on both cache logs and marked as served for server)	2.3%	1.7%
% of storage in both caches (Documents that were common on both caches logs, which includes documents server by cache and server)	6.3%	4.3%

Table 4.3: Co-operation among web caches

From Table 4.3 we can see that the percentage of data that was common on both caches was around 6.3% and 4.3% for cache 3 and cache 4 respectively. This ratio may slightly differ with respect to trace files collected at different time periods. But from the studies the percentage of hit rate by peering with other caches is generally around 5%. Hence we conclude that applying some co-operative protocols may enhance the performance of the network and also help in efficient use of network bandwidth within ISP's.

4.2.4 Percentage of Traffic Based on Document Size

Web caches handle documents of varied sizes. The number of cacheable objects could vary based on the size of the documents and also depends on the location of the cache. The size of documents can range from hundreds of bytes to megabytes. The time that a document remains popular on the web is highly variable; it does not follow a Zipf's law precisely, but instead follows a Zipf-like distribution with the exponent varying from trace to trace [BRES99]. A situation may occur where we need to place a large unpopular document in the web cache. A replacement policy may have to evict thousands of frequently accessed small documents that are more likely to satisfy future cache requests. Such a situation contributes to reduced performance of web caches. Hence document size plays a major role in designing a web cache.

An analysis was performed on 14 trace files collected from 4 caches from the Sasktel network. Each trace file was about 1GB and the traces were collected over peak traffic periods of the day. Table 4.4 shows the file size distribution in percentages. We can conclude that majority of files i.e. around 60% of documents were less than few kilobytes in size. These results will be used in the simulation model in chapter 6 to estimate latency.

File size in kilo bytes	% of the files(RANGE)
Less than 1K	57.0% - 62.0%
1K+1B-2K	7.8% - 8.9%

File size in kilo bytes	% of the files(RANGE)
2K+1B-4K	9.5% - 12.16%
4K+1B-8K	7.0% - 9.0%
8K+1B-16K	5.5% - 6.5%
16K+1B-32K	3.48% - 4.21%
32K+1B-64K	1.6% - 1.92%
64K+1B-128K	0.50% - 0.56%
128K+1B-256K	0.13% - 0.19%
256K+1B-512K	0.06% - 0.08%
512K+1B-1M	0.03% - 0.04%
1M+1B-2M	0.02% - 0.03%
2M+1B-4M	0.01% - 0.02%
4M+1B-8M	0.005% - 0.007%
8M+1B-16M	0.0022% - 0.003%
16M+1B-32M	0.001% - 0.002%
32M+1B-64M	0.0001% - 0.0005%
64M+1B-128M	0.0001% - 0.0005%
Greater than 128M	0.00001% - 0.0003%

Table 4.4: File size distribution in web caches

4.2.5 Topology and Peak Total Down Stream Traffic

Figure 4.1 represents the network topology and the downstream demand on each link used for simulation and optimization models [SASKT1]. The topology takes into account the 7 small cities and 2 larger cities that are connected to the regional ISP. The traffic demand on each link includes the total traffic on the link. This data was obtained from [DIAMO1], [SASKT2]. Based on the analysis on the content type (sec 4.2.1 and 4.2.2) and web trace analysis, we have estimated that around 30% of the total HTTP traffic was hit traffic.

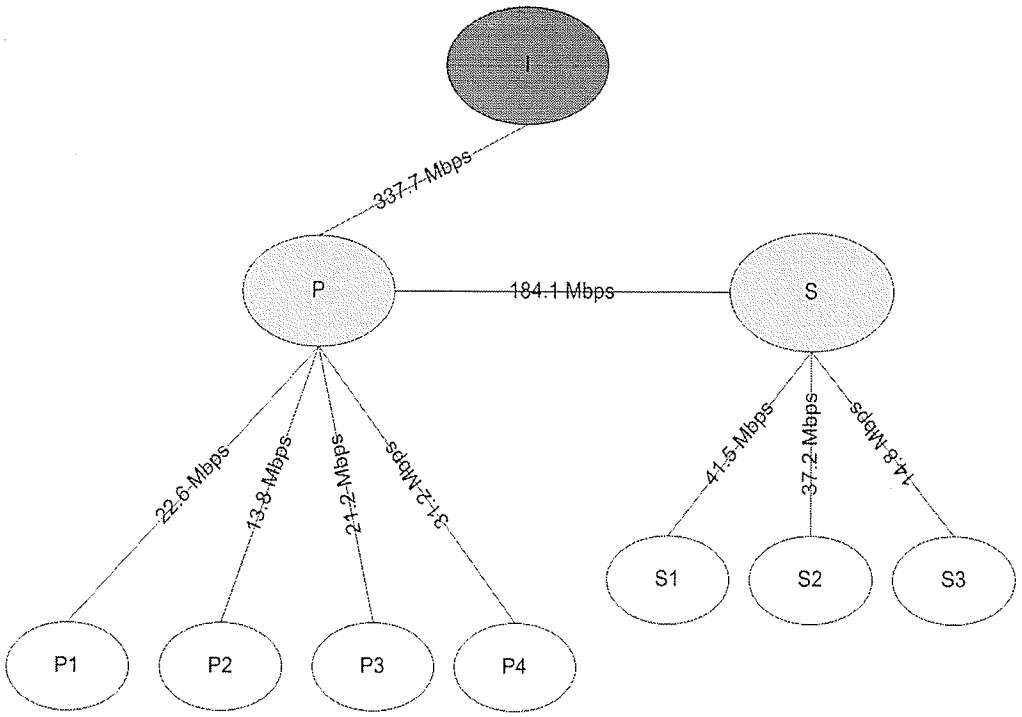

Figure 4.1: Peak Downstream Traffic [DIAMO1]

4.2.6 Location of Web Caches in the Regional ISP

There are 6 caches in the regional ISP; they are located at node *P* and *S* in figure 4.1. 4 caches are placed at node *P* and 2 caches are places at node *S*. Each web cache cost around $4000 USD and can handle around 30Mbps [DIAMO1].

4.3 Summary

Understanding and characterizing the regional ISP network traffic is a key factor in the performance study of a network. In this chapter we have performed an analysis of hit rate, web cache capacities, demand on each link and the overall HTTP traffic. We use these values in our next 2 chapters, chapter 5 Optimization model and on the chapter 6 Simulation model. The next chapter describes the optimization model and the assumptions made in solving the cache location problem for a regional ISP.

CHAPTER FIVE

5 Optimization Model

One of the objectives of this thesis is to design an optimization model to solve the cache location problem for a regional ISP as described in chapter 1. A series of optimizations are performed for different values of total budget. The results from these optimizations provide us with the best locations to place the caches in the network for a given budget. This chapter describes some of the issues addressed with the optimization model and the modeling methods used. It also describes the assumptions made. Section 5.1 provides an introduction to optimization model. Section 5.2 provides a network overview and assumptions. Section 5.3 provides the mathematical models. Finally the chapter is concluded in section 5.4.

5.1 Introduction

The cache location optimization problem involves designing the caching architecture such that certain objectives are achieved as much as possible. Generally, the objective of web cache location problem includes reducing the time to transfer one unit of data from source to destination, reducing the traffic going to the original server or passing through the inter provincial or international links, alleviating server bottlenecks, minimizing user access latency, and maintaining transparency to the end user. In our objective function we aim at reducing the mean time to transfer one unit of data from source to destination. A conventional way of designing the CLP (Cache Location Problem) optimization algorithm is described next. It can be called a heuristic approach, since the CLP algorithms used are typically heuristic rules, whose variables are assigned values to minimize or maximize the objective function. It can also be referred as an empirical approach, since the optimization is made based on the measured or simulated web cache access log traces [KOSK03].

The design of a CLP algorithm for optimization of cache location can be considered as follows: In the first stage, the objective function for the cache location problem is chosen. The objective function includes minimizing the time to transfer one unit of data or minimizing the traffic volume. In the second stage, the performance metrics

which describe the cache location are chosen. Typically used metrics are hit rate which represents what percentage of the requests are served from the cache, without passing through the outgoing network links, the load on each link - representing the traffic on the each link on the path to the original server, slack of bandwidth on each link and slack of cache capacities for hit and miss traffic. The performance metrics are chosen according to the particular objectives set [KOSK03]. In the last stage, experiments are performed based on the CLP algorithms formulated, and the one which maximizes/minimizes chosen performance metrics (constrains) are selected, as described in the next section 5.2. Results obtained from the optimization model are then tested using simulations using real measurements, or using simulated data. Another possibility is running experiments in a real environment. This procedure is too expensive and must be repeated if there are changes in the objectives, network or server resources, traffic patterns or when new caches are purchased.

The modeling was performed for 2 cases. In the first case we used a fixed cache capacity and in the second case we used a varible cache capacity model. The optimization models formulated have convex non–linear objective functions with integer, binary and real variables, hence we use a Mixed Integer Non-linear program (MINLP) solver for soving our optimization problem. MINLP (Section A.3) solver provided by NEOS server (Section A.1) was used to solve the problem. It implements a branch-and-bound algorithm. The input format to the solver was AMPL (Section A.2), details about the solver and the server are expalined in Appendix A.

5.2 Network Overview and Assumptions

This section gives an overview of the components that are used and the assumptions that are made in designing the model.

5.2.1 Routing

In this thesis, we describe an optimization model that places caches inside a tree network. In our model, Transparent En-Route Caches (TRECs) are used to route the

requests from the source to the destination. In TRECs, caches are placed only along the path from the source (clients) to the destination (server). An en-route cache normally inspects each request that is sent from the client to the server. If the request is cached, (i.e. if the requested document is found in the cache) the en-route cache will send the requested document to the client. If the requested document is not found in the cache, the en-route cache will forward the request to the web server (internet) along the standard routing path. These en-route caches are transparent to both clients and servers. Since TRECs are easy to introduce into existing networks and are easy to manage [KRIS00], TRECs are widely deployed among most existing networks. In our optimization model we assume requests flow upstream to the root of the tree only. Studies by Krishnan at el [KRIS00] have shown that 93% of the routes are stable, so we used a fixed routing scheme for both optimization and simulation model as shown in figure 4.1.

5.2.2 Full Dependency

For simplicity, we assume that our network follows the "full dependency" model. In other words, if a particular document is found in a cache then the same document is assumed to be found in all other caches. This assumption is considered quite practical in the real-world, because studies [BEST96],[BRES99] show that only a fraction of the pages are considered the most popular among all the pages that are stored in the server, and these pages are most likely requested by the clients from all the nodes. Thus, "full dependency" model will allow us to store the most popular pages in all caches.

5.2.3 Process Sharing Queue and Balanced Fairness

Process sharing queues have become an important tool in flow level modeling for the internet. For example, consider a single server process sharing queue having a capacity C. This capacity is shared equally by all the customers in the system. If there are X customers in the system, then each customer will receive service from the system at the rate of X/C. In the process sharing queue, customers will receive service

from the system as soon as they arrive. Generally, HTTP traffic running on Transmission Control Protocol (TCP) shares the resources of the network between the flows (i.e. transfers of web objects). Since in the real network model all the requests share the same links, we propose to use process sharing queues to model our system.

At any given time in a network the sum of allocated bandwidth for each flow that passes through a link should not exceed the link capacity. i.e. let s be the file size of a request at a time t, and let each path r (A path is a sequence of consecutive nodes in a topology graph and the length of the path is the number of edges traversed.), is allocated some bandwidth $\phi_r(t)$, such that the sum of bandwidth allocated to each flow (web object) does not exceed the link l capacity C_l:

$$\text{i.e. } \sum_{\{r|l\in r\}} \phi_r(t) \le C_l \quad \forall l \in L \qquad \text{Equation 5.2.3.1}$$

Balanced fairness is an allocation policy that can be considered as the most efficient insensitive allocation [TIMO03]. Network performance depends on a number of complex factors like the nature of requests arrival process at various nodes, bandwidth sharing among requests traversing a common link, distribution of request size. When bandwidth allocation is based on balanced fairness and request arrival follows a Poisson process, the time required in transferring a request and the expected throughput depends on the average traffic load on each path and the file size of the requests. This is referred as insensitivity property and is satisfied by a class of network models know as Whittle queuing networks [DIAM03].

In obtaining the upper and lower bound of our objective function, the throughput of the flows in the network is based on the bandwidth allocation according to balanced fairness scheme explained above. Let C_l and A_l represent the capacity of a link l and demand on the link l respectively. Let $r = \{l_1, l_2, l_3 \ldots l_{|r|}\}$ represents a path in a network. Let $T_r(s)$ represent the mean time to transfer a request of size s on a path r under balanced fairness. Then bounds derived are given by [DIAM03]:

$$\max_{l \in r} \frac{s}{C_l - A_l} \leq T_r(s) \leq \sum_{l \in r} \frac{s}{C_l - A_l} \qquad \text{Equation 5.2.3.2}$$

Lower bound, first part of equation 5.2.3.2 is the mean time to transfer one unit of data through a bottleneck link. $C_l - A_l$ is defined as the minimum bandwidth or minimum cache capacity that is available on any link between source and destination nodes. This is the link bottleneck approximation for our mathematical model.

Upper bound, second part of equation 5.2.3.2 is the mean time to transfer one unit of data through all the links along the path from source node to destination node. This is the store and forward approximation in our mathematical model.

Usage of equation 5.2.3.2 in our optimization model will be explained in detail in section 5.3.1.

Our optimization problems aim at minimizing the value of a linear combination of the two terms in equation 5.2.3.2, which form the objective function of our optimization model. In our objective function, the time required in transferring a request and the expected throughput depends on the average traffic load on each path and the file size of the request. Hence the bandwidth allocation in our model is similar to the balanced fairness allocation policy explained above.

5.2.4 Calculation of Cache Cost Based on Cache Capacity

From [DIAMO1], we obtained the table which gives the cost of some caching appliances and their corresponding cache capacities in Mbps as shown in table 5.1.

Product	Cost US$	Cache Capacity (Mbps)
AratechC3100	14900	83
BBGcache	7595	38
Chamomile	2500	16
iCache400	3250	44

iMimic	9895	60
iCache2500	26950	144
Kotetu	2000	11
Pyramid	4950	69
StratacacheE-55	3895	27
StratacacheF-120	8695	65
Swell1000	2549	7
TNclabs	9000	67

Table 5.1: Cost and capacity of caching appliances

From the above table we calculated the cost of caching appliances for 1 Mbps of cache capacity using linear least squares with non-negativity constraints for first degree polynomial (using Matlab's lsqnonneg command). We obtained the value of the slope to be 159.14.

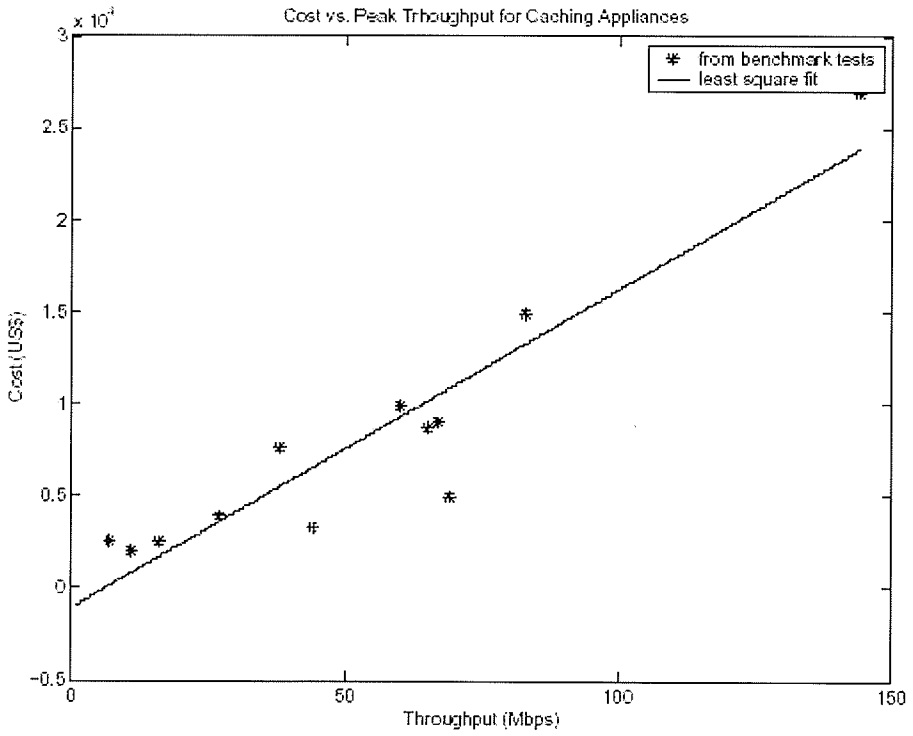

Figure 5.1: Cost vs. Peak Throughput Capacity for Caching Appliances [DIAMO1]

Figure 5.1 represents the relation between cost and peak throughput capacity of caching appliances. We observe that cost to peak throughput capacity of caching appliances is close to linear. Hence, we assume the cost of 1 Mbps of cache capacity is approximately equal to $159.14 for both fixed and variable cache capacity models.

5.2.5 Internet Node

In our model we assume that the root node represents the network outside the ISP's network, basically the Internet. All the requests that are not served by the caches are passed to this node. To consider a network outside the ISP's network would make the system complicated and difficult to optimize, so to simplify the model we make this assumption. Node *I* in figure 4.1 represents the Internet node.

5.3 Optimization Model for Cache Location Problem

The network is modeled by a connected graph *G (J, E)*, where *J* is the set of nodes and *E* is the set of edges representing the links. The input parameters for the model are summarized in table 5.2. Similarly the decision variables are summarized in table 5.3 and the supplementary variables whose values are completely dependent on the decision variable are summarized in table 5.4.

In this thesis, we construct an optimization model to reduce the mean time taken to transfer one unit of data from the source to the destination (for the entire network). The first model considers caches with variable cache capacities and the second model considers caches of fixed cache capacities. We obtain results for both cases.

5.3.1 Objective Function

The objective function for fixed cache capacity and variable cache capacity models is the same. The objective function is a non-linear function and all the constraints are linear. The objective function has two terms weighted by a factor of Q for upper bound and *(1-Q)* for lower bound (Upper and lower bounds are explained in detail in section 5.2.3). The first term represents the lower bound (explained in detail in Section 5.2.3) on the mean time to transfer one unit of a data on the shared links on its path from source to destination. The second term represents an upper bound on the mean time to transfer one unit of data on the shared links on its path from source to destination.

Referring to equation 5.2.3.2, the first part of the equation defines the lower bound of the objective function. The term $C_l - A_l$ defines the slack (available bandwidth) on the link l and $\min_{l \in r}\{C_l - A_l\}$ defines the minimum slack on the path r. Hence we define the bottleneck link on the path r as link l where $C_l - A_l$ is minimum for $l \in r$.

Let $cache_j$ be the node where the first cache is encountered along the path from node j to the root of the tree. Because of the full dependency assumption, all the requests from node j, which result in cache hits, will be served from a cache on this node.

Let $u_j = \min_{l \in h_j}\{C_l - A_l\}$, where h_j is the path from j to $cache_j$. Since the network is a tree, there is only one path to the cache node. u_j represents the slack on the bottleneck link along the path from node j to the first cache encountered along the path to the root of the tree.

Let D_j represents the demand at node j due to network traffic demand originating at j. Since D_j is the demand from node j and u_j is the slack on the bottleneck link, we can determine from equation 5.2.3.2 a lower bound on the mean time (τ) to transfer 1

Mb of data. Thus a lower bound is the total waiting time or sum of latencies for all hits to the destination (node j) is,

$$\tau = T\frac{D_j}{u_j} \qquad \text{(Equation 5.3.1)}$$

τ represents the mean time to transfer one Mb of data through link j. Where T *(time period)* is the duration of fixed time interval.

Let,

$$Z = \frac{T\sum_j \frac{D_j}{u_j}}{\sum_k D_k T}, \qquad \text{(Equation 5.3.2)}$$

Z in the above equation represents the sum of all mean latencies divided by the total number of bits transferred and thus the mean time to transfer one Mb within the network as a whole.

Let,

$$Y = \frac{\sum_j \frac{D_j}{u_j}}{\sum_k D_k} \qquad \text{(Equation 5.3.3)}$$

Y in above equation represents the sum of all mean latencies divided by the total number of bits transferred and thus represents the mean time required to transfer one Mb of data within the network as a whole for hits. Since we consider $\sum_k D_k$ as a constant (i.e. the demand is constant as it can be directly obtained from the log files when running the real experiment), we can also use in our objective function,

$$Y = \sum_j \frac{D_j}{u_j} \qquad \text{(Equation 5.3.4)}$$

Therefore the lower bound part of the objective function for hit traffic and the miss traffic can be defined as in Equation 5.3.6. We use s_j (equation 5.3.5) to represent the

slack of bandwidth for miss traffic (miss traffic is the traffic that is not served by the caches along its path to original server) along the path from source to the root node.
Let,

$$s_j = \min_{l \in m_j} \{C_l - A_l\}, \qquad \text{(Equation 5.3.5)}$$

In equation 5.3.5 m_j is the path from node j to the root node.

We represent both s_j and u_j separately, since miss traffic is not served by any caches on the path from source to destination where as the hit traffic is served by a cache on the path to the destination. α is the fraction of hit traffic.

Hence the lower bound of the objective function is defined as follows

$$L = \sum_j \left(\left(\frac{\alpha D_j}{u_j \sum_o D_o} + \frac{(1-\alpha) D_j}{s_j \sum_o D_o} \right) (1 - \delta_{jk}) \right) \qquad \text{(Equation 5.3.6)}$$

Where K, represents the Internet node and δ_{jK} is set to one if j and K are the same node, else set to 0. The term $(1-\delta_{jK})$ is added to both upper and lower bound terms, to calculate the mean time up to node K (Internet node).

The second term of the objective function is based on the upper bound on the mean time to transfer one Mb of data on the links on its path from source to destination as discussed in section 5.2.3.
Let,

$$\beta_{jrl} = \begin{cases} \text{Slack on link } rl & \text{If rl is on path } h_{jrl}, \text{ i.e. } h_{jrl} = 1 \text{ and } f_{jl} = 0 \\ M & \text{Otherwise} \end{cases} \qquad \text{(Equation 5.3.7)}$$

$$\lambda_{jl} = \begin{cases} \text{Slack at node } l & \text{If } l \text{ is on path } h_{jrl},\ i.e.\ h_{jrl} = 1,\ f_{jl} = 0 \text{ and } y_l = 0 \\ M & \text{Otherwise} \end{cases}$$

(Equation 5.3.8)

$$\sigma_{jrl} = \begin{cases} \text{Slack on link } rl & \text{If } rl \text{ is on path } h_{jrl}\ (h_{jrl} = 1) \\ M & \text{Otherwise} \end{cases}$$

(Equation 5.3.9)

In the above three equations *M (same as BigM in table 5.2)* is a very large number. The term h_{jrl} is set to one if the path between j and the Internet node passes through link rl else it is set to zero. The term f_{jl} is set to one if requests from node j pass through a cache before reaching the node l else it is set to zero. The term y_l is set to one if at least one cache is present on node l, else set it is to zero.

Equation 5.3.7, defines the slack of bandwidth for traffic from node j passing through link rl. The slack on link rl is defined as the difference between the bandwidth on the link rl and demand on the link rl (misses and hits) in Mbps which is incident on link rl from its children and not filtered out by caches lower in the tree.

In equation 5.3.8, the slack at node l is defined as the difference between the capacity of the cache on node l and the total demand (misses and hits) in Mbps which is incident on node l from its children and not filtered out by caches lower in the tree.

In equation 5.3.9, the slack on link rl is defined as the difference between the bandwidth on the link rl and demand on the link rl in Mbps which is incident on link rl from its children and not filtered out by any caches (i.e. miss traffic).

Let,

$$U = \sum_j \left(R + (1-\delta_{jk}) \left(\frac{\alpha\left(\sum_{\forall r,l \in J^3, h(j,r,l)=1} \left(\frac{1}{\beta_{jrl}}\right) + \sum_l \left(\frac{1}{\lambda_{jl}}\right) \right) D_j}{\sum_o D_o} + \frac{(1-\alpha)\left(\sum_{\forall r,l \in J^3, h(j,r,l)=1} \left(\frac{1}{\sigma_{jrl}}\right) \right) D_j}{\sum_o D_o} \right) \right)$$

(Equation 5.3.10)

Where: K represents the Internet node.

δ_{jK} is set to one if j and K are the same node, else set to 0. The term $(1-\delta_{jK})$ is added to both upper and lower bound terms to calculate the mean time up to node K (Internet node).

$R = [\alpha/(c + Mf_{jK}) + (1-\alpha)/c]$

R represents the mean time to transfer one Mb of data on the bottleneck link. c is the maximum throughput that can be achieved on the link connecting to the Internet node (i.e. link PI in figure 4.1). Hence $\alpha/(c + Mf_{jK}) + (1-\alpha)/c$ defines the mean time to transfer one Mb of data through a bottleneck link from node j to node K (internet node).

U defines the upper bound of the objective function. In the upper bound of the objective function the term $\frac{1}{\beta_{jrl}}$, is added if rl is along the path j to the cache node which serves its demands else we add $\frac{1}{M}$. The term $\frac{1}{\lambda_{jl}}$, is added if node l is along the path j to the cache node which serves its demands else we add $\frac{1}{M}$. Similarly the term $\frac{1}{\sigma_{jrl}}$, is added if link rl is along the path j to the original server node which serves its demands else we add $\frac{1}{M}$. A very small value $\frac{1}{M}$ is added to the mean time to transfer one Mb of data to avoid the divide by zero error. Hence for upper bound, the mean time to transfer one unit of data is the summation of mean time to transfer one unit of data on each link along the path from source node to destination node. Destination node is Internet node in case, where the traffic load is not filtered by a cache.

Hence the objective function is the overall mean time (for the entire network), taken to transfer one Mb from source to destination given as,

$$z = (1-Q)(L) + QU$$

-Or-

$$z = (1-Q)\left(\sum_j \left[\left(\frac{\alpha D_j}{u_j \sum_o D_o} + \frac{(1-\alpha) D_j}{s_j \sum_o D_o} \right)(1-\delta_{jK}) \right] \right) +$$

$$(Q)\sum_j \left(R + (1-\delta_{jk}) \left(\frac{\alpha \left(\sum_{\forall r,l \in J^3, h(j,r,l)=1} \left(\frac{1}{\beta_{jrl}}\right) + \sum_l \left(\frac{1}{\lambda_{jl}}\right) \right) D_j + (1-\alpha) \left(\sum_{\forall r,l \in J^3, h(j,r,l)=1} \left(\frac{1}{\sigma_{jrl}}\right) \right) D_j}{\sum_o D_o} \right) \right)$$

Where $R = [\alpha / (c + Mf_{jK}) + (1-\alpha)/c]$ (R explained in section 5.3.1) and the objective function has two terms weighted by a factor of Q for upper bound and $(1-Q)$ for lower bound.

5.3.2 Variable Cache Capacity Model

In the variable cache capacity case, we use the cost for 1 Mbps cache capacity as described in section 5.2.4. The cost of each cache is defined by a product of cache capacity and cost for 1 Mbps cache capacity. For different budgets, the optimizer outputs the cache capacity and the corresponding cache location of each node defined in the topology. Tables 5.2, 5.3 and 5.4 describe in detail parameters, variable and supplementary variables respectively. The following is the mathematical model defined.

Minimize

$$z = (1-Q)\left(\sum_j \left[\left(\frac{\alpha D_j}{u_j \sum_o D_o} + \frac{(1-\alpha) D_j}{s_j \sum_o D_o} \right)(1-\delta_{jK}) \right] \right) +$$

$$(Q)\sum_j \left(R + (1-\delta_{jk}) \left(\frac{\alpha \left(\sum_{\forall r,l \in J^3, h(j,r,l)=1} \left(\frac{1}{\beta_{jrl}}\right) + \sum_l \left(\frac{1}{\lambda_{jl}}\right) \right) D_j + (1-\alpha) \left(\sum_{\forall r,l \in J^3, h(j,r,l)=1} \left(\frac{1}{\sigma_{jrl}}\right) \right) D_j}{\sum_o D_o} \right) \right)$$

Where R=$[\alpha / (c + f_{jK}) + (1-\alpha)/c]$ (R explained in section 5.3.1)

Subject to:

y_j * minDemand <= CacheCapacity$_j$ $\forall j \in J$ (Constraint 5.3.2.1)

CacheCapacity$_j$ <= sumDemand * y_j $\forall j \in J$ (Constraint 5.3.2.2)

CacheCapacity$_K$ = 0 (Constraint 5.3.2.3)

$$f_{jk} \le y_j + \sum_{l \ne j,k} V_{jl} V_{lk} y_l (1-\delta_{lj})(1-\delta_{lk}) \quad \forall j,k \in J$$ (Constraint 5.3.2.4)

$$(1-\delta_{jk})(y_j + \sum_{l} (1-\delta_{lj})(1-\delta_{lk}) V_{jl} V_{lk} y_l) \le countnodes^* f_{jk} \quad \forall j,k \in J^2$$ (Constraint 5.3.2.5)

$f_{jj} = 0$ $\forall j \in J$ (Constraint 5.3.2.6)

$$z_j \le cacheCapacity_j + sumDemand(1-y_j) \quad \forall j \in J$$ (Constraint 5.3.2.7)

$$z_j = D_j + \sum_{k \ne j} D_k V_{kj} (1-f_{kj}) \quad \forall j \in J$$ (Constraint 5.3.2.8)

$$x_{rl} = V_{rl} sign(B_{rl})(D_r(1-\alpha y_r) + \sum_{t \ne r} V_{tr} D_t (1-\alpha f_{tl})) \quad \forall (r,l) \in J^2$$

(Constraint 5.3.2.9)

$x_{rl} \le B_{rl}$ $\forall (r,l) \in J^2$ (Constraint 5.3.2.10)

$$u_j \le B_{rl} - x_{rl} + BigM(y_j + y_r + f_{jr}) \quad \forall jrl \in J^3, h_{jrl} = 1$$ (Constraint 5.3.2.11)

$$u_j \le cacheCapacity_l - z_l + BigM(1-y_l V_{jl})(f_{jl}) \quad \forall j,l \in J^2$$ (Constraint 5.3.2.12)

$$s_j \le B_{rl} - x_{rl} \quad \forall jrl \in J^3, h_{jrl} = 1$$ (Constraint 5.3.2.13)

$u_j <= c + BigM * f_{j,K}$ $\forall j \in J$ (Constraint 5.3.2.14)

$\beta_{jrl} = B_{rl} - x_{rl} + BigM(f_{jl})$ $\forall jrl \in J^3, h_{jrl} = 1$ (Constraint 5.3.2.15)

$\lambda_{jl} = \text{CacheCapacity}_l - z_l + BigM((1-y_l)+(1-f_{jl}))$ $\forall jl \in J^2, Vj_l = 1$ (Constraint 5.3.2.16)

$\sigma_{jrl} = B_{rl} - x_{rl}$ $\forall jrl \in J^3, h_{jrl} = 1$ (Constraint 5.3.2.17)

$\text{cost}_j = slope * \text{CacheCapacity}_j$ $\forall j \in J$ (Constraint 5.3.2.18)

$totalCost = \sum_j \text{cost}_j$ $\forall j \in J$ (Constraint 5.3.2.19)

$totalCost <= T$ (Constraint 5.3.2.20)

$h_{jrl} = V_{jr}V_{rl}sign(B_{rl})$ $\forall jrl \in J^3$ (Constraint 5.3.2.21)

$BigM = \max(B_{jk}) * 10$ $\forall j,k \in J$ (Constraint 5.3.2.22)

$\text{minDemand} = \min(D_j)$ $\forall j \in J$ (Constraint 5.3.2.23)

$\text{sumDemand} = \sum_j D_j$ $\forall j \in J$ (Constraint 5.3.2.24)

$(c - s_j) \geq 0$ $\forall j \in J$ (Constraint 5.3.2.24a)

$y_j \in \{0,1\}$ $\forall j \in J$ (Constraint 5.3.2.25)

$f_{jk} \in \{0,1\}$ $\forall j,k \in J^2$ (Constraint 5.3.2.26)

$n_j \in Z_+$ $\forall j \in J$ (Constraint 5.3.2.27)

$z_j \in R_+$ $\forall j \in J$ (Constraint 5.3.2.28)

$x_{jk} \in R_+$ $\forall j,k \in J^2$ (Constraint 5.3.2.29)

$c, u_j, \beta_{jrl}, \lambda_{jl}, s_j, \sigma_{jrl} > 0$ $\forall j \in J$ (Constraint 5.3.2.30)

$cacheCapacity_j, \text{cost}_j, \text{totalcost} \in R_+$ $\forall j \in J$ (Constraint 5.3.2.31)

The objective function is described in section 5.3.1.

Constraint 5.3.2.1 and 5.3.2.2 define y_j= *sign(cacheCapacity)*. If *cacheCapacity* is positive value, then y_j is set to 1 else set to 0.

Constraint 5.3.2.3 states that no caches are placed at node *K*, because node *K* is set as the Internet node (described in section 5.2.5). (In our thesis we are trying to places caches within a regional ISP network, but not on external nodes. Node *K* in the considered as internet node in our network model)

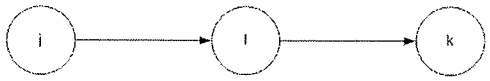

Figure 5.3: Path taken from node j to reach node k

Constraint 5.3.2.4 and 5.3.2.5 guarantee that f_{jk} to be 1 or 0, depending on whether there is a cache between *j* and *k* (figure 5.3), but not including *k*. If a cache is placed between node *j* and *k* but not on *k* then f_{jk} is set to 1, else set to 0.

Constraint 5.3.2.7 assures that the load on the cache at node *j* should not exceed the capacity of the caches at node *j* (if there is a cache). For example, let us consider that node *j* has a cache and the peak throughput capacity of that cache is 100 Mbps. Therefore, when these values are integrated into the constraint $z_j \leq cacheCapacity_j + sumDemand(1 - y_j)$, we will get $z_j \leq 100Mbps$ (since $y_j = 1$ i.e. cache is present at node *j*, $sumDemand(1 - y_j)$ will become zero). In other words, the maximum load that a cache can support is 100 Mbps. If there are no caches placed at node *j* then $y_j=0$ and we will take *sumDemand* into consideration. Since *sumDemand* is the maximum demand of entire network, the constraint becomes redundant.

Constraint 5.3.2.8 defines z_j, the load on a cache at node *j*. The load on z_j, can be defined as the sum of all traffic that is coming from different nodes $(k_1, k_2, \ldots, k_n)$ to node *j*, such that the traffic is not first filtered by a cache en-route. The factor V_{kj} guarantees that only traffic from leaf nodes which is routed through *j* contributes to the load. The factor $(1-f_{kj})$ guarantees that only traffic which is not filtered en-route contributes to the load.

Constraint 5.3.2.9 defines the workload on link rl. The workload on link rl can be obtained by calculating the total traffic at node r (this includes traffic that is generated at node r and traffic that is generated by neighboring nodes, which pass through node r to reach the destination) that passes through node l in order to reach the server. Constraint 5.3.2.8 defines an overview of how workload is calculated on link rl. For example, let us consider figure 5.4, in which traffic to node r is coming from node $t1$ and node $t2$. However, a fraction α of the traffic that is generated from nodes $t1$, $t2$ and node r, is filtered at node r, if a cache is placed at node r. If a cache is not placed at node r then the entire traffic from node r will be directed towards node l. This shows that only the unfiltered traffic will flow from node r to node l. This will give the traffic on the link rl.

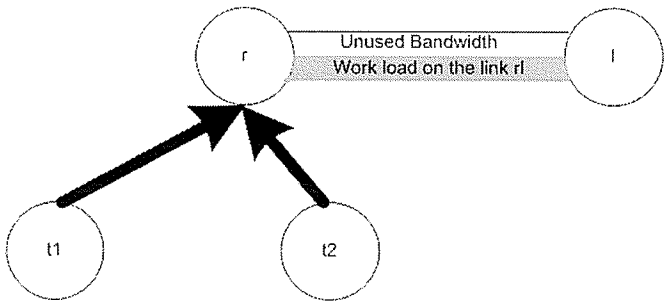

Figure 5.4: Workload on the link

In Constraint 5.3.2.9, V_{rl} determines if the requests from node r pass through node l, in order to reach the server. $sign(B_{rl})$ determines the existence or absence of a direct link between node r and node l. The variables used in constraint 5.3.2.9 are explained below;

- $sign(B_{rl}) = 1$ if there are no intermediate nodes present between the nodes r and l.
- $sign(B_{rl}) = 0$, if there are some intermediate nodes present between the nodes r and l.

- $(1-\alpha y_r)$=1, if no cache is placed at node r else if there is a cache at node r, then a fraction α of the traffic load is filtered at node r.
- $(1-\alpha f_{tl})$= 1, if there is no cache on the path from node t to l, but not including l. else a fraction α (the byte hit ratio), is filtered at node r.
- $V_{tr}D_t(1-\alpha f_{tl})$ gives the traffic load (which flow from node t) on link rl.
- $D_r(1-\alpha y_r)$ gives the demand on node r.

Constraint 5.3.2.10 guarantees that the workload on the link rl is always less than or equal to its total bandwidth.

Constraint 5.3.2.11 determines the minimum slack on the bottleneck link for hit traffic (hit traffic is the traffic load that is served by the cache).

In this case, we are interested in finding the bottleneck link, i.e. where (in which link) the slack is minimum. Figure 5.5 explains in detail how we calculate slack on the bottle neck link in equation 5.3.2.11. u_j represents the minimum slack on the path from node j to internet. The term $BigM(y_j+y_r+f_{jr})$ is used to verify that the traffic flowing through the path between node j and node r or any other node on the path from node j to r is not filtered by a cache.

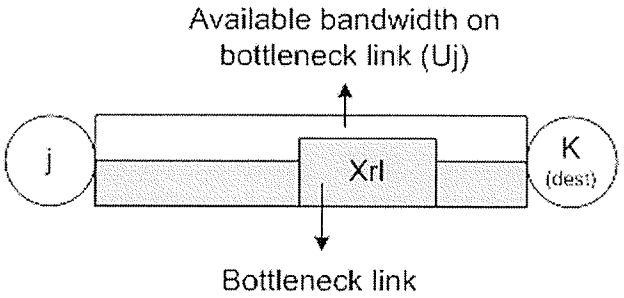

Figure 5.5: Calculating available bandwidth on the bottleneck link

Note: The objective function is a decreasing function of u_j, so that u_j will be as large as possible subjected to constrains 5.3.2.11, 5.3.2.12, 5.3.2.14.

Constraint 5.3.2.12 guarantees that the bottleneck slack for requests from node j must also be smaller than the slack in capacity of the cache, which might be serving the requests. $BigM(1-y_l V_{jl})(f_{jl})$ describes that constrains 5.3.2.12 becomes redundant, if there is a cache on a node on the path from node j to node l, so that demand is filtered en-route to l or if there is no cache on node l.

s_j represents the minimum available bandwidth along the path from node j to the root or Internet node. This load is not filtered by any caches on the path. We assume that a fraction of $(1-\alpha)$ of the total load on each link is not filtered by the caches, and only a fraction of α is served by the caches on the path from source to Internet node. Constraint 5.3.2.13 guarantees that s_j is the minimum slack (available bandwidth along the path from node j to Internet node) along the path from node j to Internet node.

Constraint 5.3.2.14 ensures that maximum throughput on the bottleneck link is less than a constant c (c is a mean throughput, calculated from the trace files collected from the Sasktel network). This is meant to model cases where the bottleneck is at the server or the DSL link to the client, and not within the network.

Constraint 5.3.2.15, 5.3.2.16, and 5.3.2.17 define β_{jrl}, λ_{jl} and σ_{jrl} (β_{jrl}, λ_{jl} and σ_{jrl} as defined under section 5.3.1) respectively.

Constraint 5.3.2.18 defines the cost of the cache at node j. The cost of the cache is defined as the slope or the cost of one Mbps of cache capacity times the cache capacity. Calculation of cost of cache capacity is described under section 5.2.4.

Constraint 5.3.2.19 defines the total cost of all the caches in the network.

Constraint 5.3.2.20 describes that the total cost of all the caches should be less than the available budget T.

Constraints 5.3.2.21 to 5.3.2.24 are defined in Table 5.2. Constraints 5.3.2.25 to 5.3.2.31 define the type of data that the variables take.

5.3.3 Fixed Cache Capacity Model

In the fixed cache capacity case, we use caches with capacities of fixed size W, The cost of each cache is given by E and the total budget is T. For different budgets, the optimizer outputs the number of caches to be placed on each node within the topology. Tables 5.2, 5.3 and 5.4 describe the parameters, decision variables and supplementary variables respectively in detail. The following is the mathematical model defined:

Minimize

$$z=(1-Q)\left(\sum_{j}\left[\left(\frac{\alpha D_j}{u_j\sum_{o}D_o}+\frac{(1-\alpha)D_j}{s_j\sum_{o}D_o}\right)(1-\delta_{jK})\right]\right)+$$

$$(Q)\sum_{j}\left(R+(1-\delta_{jk})\left(\frac{\alpha\left(\sum_{\forall r,l\in J^3,h(j,r,l)=1}\left(\frac{1}{\beta_{jrl}}\right)+\sum_{l}\left(\frac{1}{\lambda_{jl}}\right)\right)D_j}{\sum_{o}D_o}+\frac{(1-\alpha)\left(\sum_{\forall r,l\in J^3,h(j,r,l)=1}\left(\frac{1}{\sigma_{jrl}}\right)\right)D_j}{\sum_{o}D_o}\right)\right)$$

Where $R=\left[\alpha/(c+f_{jK})+(1-\alpha)/c\right]$

Subject to:

$$N+p=\sum_{j\in J}n_j \qquad \text{(Constraint 5.3.3.1)}$$

$$Ep\le T \qquad \text{(Constraint 5.3.3.2)}$$

$$y_j\le n_j \qquad \forall j\in J \quad \text{(Constraint 5.3.3.3)}$$

$$f_{jk}\le y_j+\sum_{l\neq j,k}V_{jl}V_{lk}y_l(1-\delta_{lj})(1-\delta_{lk}) \qquad \forall j,k\in J \quad \text{(Constraint 5.3.3.4)}$$

$$(1-\delta_{jk})(y_j+\sum_l (1-\delta_{lj})(1-\delta_{lk})V_{jl}V_{lk}y_l) \le countnodes \forall j,k \in J^2$$

(Constraint 5.3.3.5)

$f_{jj} = 0 \qquad \forall j \in J$ (Constraint 5.3.3.6)

$z_j \le K * n_j + sumDemand(1-y_j) \qquad \forall j \in J$ (Constraint 5.3.3.7)

$z_j = D_j + \sum_{k \ne j} D_k V_{kj}(1-f_{kj}) \qquad \forall j \in J$ (Constraint 5.3.3.8)

$$x_{rl} = V_{rl} sign(B_{rl})(D_r(1-\alpha y_r) + \sum_{t \ne r} V_{tr} D_t (1-\alpha f_{tl})) \qquad \forall (r,l) \in J^2$$

(Constraint 5.3.3.9)

$x_{rl} \le B_{rl} \qquad \forall (r,l) \in J^2$ (Constraint 5.3.3.10)

$u_j \le B_{rl} - x_{rl} + BigM(y_j + y_r + f_{jr}) \qquad \forall jrl \in J^3, h_{jrl} = 1$ (Constraint 5.3.3.11)

$u_j \le Wn_l - z_l + BigM(1 - y_l V_{jl} + (f_{jl}) \qquad \forall j,l \in J^2$ (Constraint 5.3.3.12)

$s_j \le B_{rl} - x_{rl} \qquad \forall jrl \in J^3, h_{jrl} = 1$ (Constraint 5.3.3.13)

$u_j <= c + BigM * f_{j,K} \qquad \forall j \in J$ (Constraint 5.3.3.14)

$n_j \le (N + T/E) y_j \qquad \forall j \in J$ (Constraint 5.3.3.15)

$c - s_j \ge 0 \qquad \forall j \in J$ (Constraint 5.3.3.16)

$\beta_{jrl} = B_{rl} - x_{rl} + BigM(f_{jl}) \qquad \forall jrl \in J^3, h_{jrl} = 1$ (Constraint 5.3.3.17)

$\lambda_{jl} = Kn_l - z_l + BigM((1-y_l) + (1-f_{jl})) \quad \forall jl \in J^2, Vj_l = 1$ (Constraint 5.3.3.18)

$\sigma_{jrl} = B_{rl} - x_{rl} \qquad \forall jrl \in J^3, h_{jrl} = 1$ (Constraint 5.3.3.19)

$h_{jrl} = V_{jr} V_{rl} sign(B_{rl}) \qquad \forall jrl \in J^3$ (Constraint 5.3.3.20)

$BigM = \max(B_{jk}) * 10 \qquad \forall j,k \in J$ (Constraint 5.3.3.21)

$\text{minDemand} = \min(D_j) \qquad \forall j \in J$ (Constraint 5.3.3.22)

$\text{sumDemand} = \sum_j D_j \qquad \forall j \in J$ (Constraint 5.3.3.23)

$n_{10} = 0$ (Constraint 5.3.3.24)

$y_j \in \{0,1\} \qquad \forall j \in J$ (Constraint 5.3.3.25)

$f_{jk} \in \{0,1\} \qquad \forall j,k \in J^2$ (Constraint 5.3.3.26)

$n_j \in Z_+ \qquad \forall j \in J$ (Constraint 5.3.3.27)

$z_j \in R_+$ $\forall j \in J$ (Constraint 5.3.3.28)

$x_{jk} \in R_+$ $\forall j,k \in J^2$ (Constraint 5.3.3.29)

$c, u_j, \beta_{jrl}, \lambda_{jl}, s_j, \sigma_{jrl} > 0$ $\forall j \in J$ (Constraint 5.3.3.30)

$p \in Z_+$ (Constraint 5.3.3.31)

The objective function is described under section 5.3.1.

Constraint 5.3.3.1 helps us determine the total number of caches that are purchased. More precisely, the parameter N represents the number of caches that are initially available in the network and p gives the number of caches that are purchased. The sum of N and p gives us the total number of caches that are currently placed in the network.

Constraint 5.3.3.2 guarantees the total amount ($) spent to purchase caches (number of caches purchased * cost of each cache) is always less than or equal to the allocated budget (T).

Constraint 5.3.3.3 and Constraint 5.3.3.15 guarantee that y_j is set to 1, if there is a cache on the node j and the number of caches on the node is greater than 0.

The rest of the constraints are the same as in section 5.3.2.

All the equations from (Equation 5.3.3.21) through (Equation 5.3.3.24) can be explained by the definitions in Table 5.2. Equation (Equation 5.3.3.25) through (Equation 5.3.3.31) defines the type of data that the variables take.

Parameters	*Description*
T	Total given budget
V_{jk}	1 If HTTP requests from node j pass through node k to reach the servers that are outside the network, in the

	absence of caches 0 Otherwise
α	Hit-ratio for all caches
D_j	Demand in Mbps for HTTP traffic from node j
B_{rl}	Bandwidth of the link from node r to node l
δ_{jk}	1 if j and k are the same 0 Otherwise
Slope	Represents the cost for 1Mbps link in USD
c	Represents the maximum throughput on the bottleneck link
Q	Represents the weight for lower and upper bound in the objective function
BigM	A big value
minDemand	Minimum node demand in the network
sumDemand	Sum of all the demands on all the links
h_{jrl}	1 if the path between j and the internet passes through link rl 0 Otherwise
N	Number of caches available initially
E	Cost of each cache
W	Peak throughput capacity in Mbps of a cache
Countnodes	Count of nodes

Table 5.2: Parameters for the proposed optimization model

Decision Variable	*Description*
n_j	Number of caches stationed at node j

Table 5.3: Decision variable for the proposed optimization model

Supplementary Variables	*Description*
y_j	1 If at least one cache is present on node j 0 Otherwise
f_{jk}	1 If requests from node j pass through a cache before reaching the node k 0 Otherwise
x_{rl}	HTTP traffic in Mbps traversing the link between nodes r and l
z_j	HTTP traffic load for caches placed on node j
p	Number of caches to be purchased
u_j	Minimum slack on bottleneck link for hit traffic.
s_j	Minimum slack on bandwidth on the path of miss traffic
β_{jrl}	Available slack of bandwidth on the link rl or a very large number
σ_{jrl}	Available slack of bandwidth on the link rl or a very large number for miss traffic
λ_{jl}	Available slack of cache capacity on the node r or a very large number
$cacheCapacity_j$	Cache capacity at node j
$cost_j$	Cost of cache at node j
totalcost	Cost of all the caches

Table 5.4: Supplementary variables for the proposed model

5.4 Summary

This chapter has described in detail the optimization models to solve a cache location problem for a regional ISP network model. The chapter gave an introduction to the optimization model, followed by the optimization models for 2 cases; fixed cache capacity and variable cache capacity. Optimization was performed using the NEOS server to solve the mixed integer non-linear problems. A Mathematic Programming Language (AMPL) was used as input format to the NEOS server. The results are presented in chapter 7 section 7.1.

CHAPTER SIX

6 Simulation Model

To validate the optimization model of the cache location problem described in chapter 5, we use a simulation model. One of the objectives of this thesis is to validate the mathematical model through simulations. A series of simulations are carried out to test the performance of the Sasktel network by placing the caches at nodes obtained from the mathematical model. This chapter discusses the simulation model and the traffic modeling techniques. It also outlines the network model used for performance analysis of the system and the assumptions made in the simulations. We designed and implemented a simulation model using Network Simulator version 2 (NS2) for validating the results obtained from optimization for the cache location problem for a regional ISP. Section 6.1 provides an introduction to the simulation models. Section 6.2 presents the components of the network model and simulation model used. Section 6.3 describes the traffic model. Section 6.4 discusses trace driven simulations. Section 6.5 discusses the assumptions made in designing a network model. Section 6.6 describes the steps used in simulation. We conclude the chapter with a summary in section 6.7.

6.1 Introduction

Designers of networks usually aim at gaining the highest network performance at the lowest cost. Modeling and simulation of networks is a good start for the design and engineering of real networks. Simulation is an imitation of real system by a computer program.

Simulation models allow a system to be modeled at any level of detail: from a direct translation of a queuing network model to capturing every aspect of the system's behavior. Simulation also supports the collection of any performance metric that can be defined and programmed [SIMU02]. We use the simulation model to validate the optimization model.

Model validation involves developing an acceptable level of confidence such that the performance measures predicted by the model are applicable, if applied to real system [LIU03]. Validation ensured that the assumptions used in developing a model are reasonable. If the model is correctly implemented, it would be close to the real systems. Most of the cache location problems are solved by optimization and validated by using a simulation model [KRIS00] [DIAMO2] [CZYZ97].

6.2 Network Model Components

In this section we describe the network components.

6.2.1 Caches

Web caches store a list of most popular web pages. Each cache has a pool of web pages, consisting of a page ID and page size. Each cache also contains a list of page IDs, which acts as a queue and allows for LRU page discarding (described in section 2.3.1.2) when the cache is full. When a client requests a page in the cache, the cache returns the requested page, if the cacheability of the requested page is one and if it has the page in its pool. Otherwise, the cache will mark which client/cache has made that request, and forwards the request to the next cache towards the destination server. Once the page is found either in a cache or a server, a copy of the requested page is placed on all the previously requested caches until they hit the requested client.

6.2.2 Servers

A server contains the original copy of the pages requested; it contains all the pages in the system. Every request that has been requested will eventually be served by a server, if there are no caches en-route to the server, or if no cache has the page cached or if the page is a non-cacheable page.

6.2.3 Clients

Clients request documents. Every request that has been requested will eventually be served by a server, if there are no caches en-route to the server, or if the cache does not have the page cached or if the page is a non-cacheable page. The server or cache

serve the request to the client and will save a copy in the caches which are on the path to the client.

6.2.4 Topology

The topology of the simulation model involves a network topology and definition of bandwidth, delay and buffer size for each link. The first step in topology creation is defining the base nodes [SAIL03]. The base nodes are created using the topology file (section B.3.2 defines the attributes in a topology file.). We create the base nodes first and give each node an ID called node ID, once the base nodes are generated, clients are attached to the base nodes and are given the same ID as node ID. Since it is difficult to simulate individual cache and server bandwidth, we create a new node for each server/cache which is attached to the base node with the equivalent node id. A cache or server is then attached to that node and added to either the cache or server array with index the same as node id, so that there is a link from the base node to the newly created node [SAIL03].

6.2.5 Routing

Web caches developed in recent years have the ability to route TCP and UDP data flows based on their URL and cacheability in the HTTP headers. Hence HTTP request can be assigned to various caches depending on the cacheability and popularity. This technology can be used to construct transparent caches as explained in section 5.2.1.

The routes between nodes are described in section 5.2.1. When a page is requested, the page has a route which accepts 2 parameters. The first parameter is the current node ID, and the second parameter is the destination node ID. The route method returns the id of the next node on the path from the node with the current id to the node with the destination ID. This routing is done only for the base nodes, i.e. those described in the topology file, not the nodes added for caches/servers. Since each cache/server has the same array index as the base node it is connected to, we can

effectively access these, ignoring the id of the node to which the cache/server is attached.

6.3 Traffic Modeling

The data collected from the Sasktel web logs were used to generate traffic for the simulation model. The log files were collected for a period of 8 hours. For the purpose of simulation, we ran the simulation for a fixed number of requests, so they would not take too long to run. At random a set of 10000 requests from a trace file were collected and distributed among the nodes, based on the demand as indicated in figure 4.1. A trace file consists of time ordered list of request and their associated variables as described in Appendix B.3.3.

The trace file was modified to distribute the traffic based on the demand mentioned in figure 4.1. I.e. the fraction of traffic origination from a node *j* (say P_j) should be equal to the demand on the node *j* divide by the total demand of the network. From section 4.2.1 we observed that each link has around 30 % of traffic as hit traffic and around 70% of traffic as miss traffic. Hence the total traffic originating from each node should approximately be a fraction of P_j of the over all demand and 30% of the total traffic origination at any node *j* is served by a cache and 70% of the total traffic originating at node *j* is served by the server.

To generate traffic closer to the model used in solving the optimization problem, the following modifications were performed on the trace files.

The start time for each request was calculated using equation 6.3.1. The start time of each request was modified, such that the total demand of the network is consistent with the optimization model.

$$starttime\_\mathrm{mod}ified = starttime\_original * \frac{B}{D} \qquad \text{Equation 6.3.1}$$

Where: *starttime_modified* is the time of arrival of a request modified (described in Appendix B.3.3)

starttime_original is the time or arrival of a request from the original trace file.

D (Same as sumDemand in Table 5.2) is the total demand on the network in bytes/sec. This value is obtained by summing up all the demand of the links in the network (referring to figure 4.1).

B represents total number of bytes transferred for the 10,000 requests subtrace divided by the elapse time given as

$$B = \frac{\sum_j S_j}{T_n - T_1} \qquad \text{Equation 6.3.2}$$

Where S_j represents the file size of the j^{th} request of n requests, T_n is the start time of the last request and T_1 is start time of first request.

Traffic was distributed proportional to the demand (figure 4.1) originating at each node. Equation 6.3.3 represents the fraction of demand on each link.

$$P_j = \frac{D_j}{\sum_j D_j} \qquad \text{Equation 6.3.3}$$

P_j represents the fraction of demand from node j and each link has a fraction of α and 1- α for hit and miss traffic load respectively. Let cumulative fraction for each link j be c_j. Let hit_miss_{ij} represent cumulative fraction for hit and miss (i represent the cumulative fraction of hit and miss) request of link j. The above method was used to assign IP addresses to nodes, such that the total traffic is divided amongst the nodes according to the fraction P_j.

A random number "hitmiss" was chosen between 0 and 1 to represent whether the request will be a hit or a miss. A random number was chosen between 0 and 1, represented as X_j, where j represents the source nodes. For each request the following algorithm was repeated.

If $c_k(j) \leq X_j \leq c_{k+1}(j)$, then the requests with IP address j are attached to node k, if the fraction $p_j > 0$ (Where $p_j = P_j$ initially and then calculated using equation 6.3.4). If $p_j > 0$ then we assume the document to be hit or miss based on the hitmiss attribute. If $p_j = 0$, then the request j is assigned to next link, whose $p_j > 0$.

The cumulative sum c_j is calculated at the arrival of new request by using new fraction p_j obtained by using equation 6.3.4

$$p_j = (tw * P_j - sw_j)/(tw - \sum_j sw_j) \qquad \text{Equation 6.3.4}$$

Where: tw is the sum of file sizes of all the requests in the trace file

p_j is new fraction of demand on the link j

P_j is the target fraction of demand from node j

sw_j is a the sum of file size of all requests assigned to node j

The new cumulative fractions for each link are calculated using the following equation.

$$c_j = \sum_{k=1}^{j} p_j \qquad \text{Equation 6.3.5}$$

Equation 6.3.5 is used to calculate the new cumulative fraction for each link j.

We continue this process until all requests are assigned to the nodes. The fraction of load on each node should be approximately equal to the original fraction i.e P_j and the load on each should have approximately a fraction of α and $1-\alpha$ for hit and miss traffic load respectively. We assume that an IP address attached to a node is always attached to the same node on its next occurrence, unless the link's total traffic load fraction is greater than or equal to the required fraction of load on the link.

The requests in the trace files were sorted in descending order of size. The above algorithm was implemented in Perl script and was run for a couple of trace files. We observed that for each link around 30 % of traffic was hit traffic and around 70% of traffic was miss traffic. The total traffic originating from each link was approximately

a fraction of P_j of the over all demand. On execution of the above mentioned algorithm we could get approximately the same link demand as shown in figure 4.1 and the fraction of hit and miss requests were approximately 30% and 70% respectively (30% of traffic was assumed as byte hit ration based on the studies described under section 4.2.1). The output file obtained from this algorithm was used as input to our simulation model. The importance of this algorithm was, to generate a trace file with demand on the links and byte hit ratio on the links to be similar to the real ISP's network.

6.4 Assumptions

In our performance analysis study, we observed that around 30% of HTTP traffic is served from the caches and rest of the traffic is served from the original server. Hence we assume that around 30% of total traffic originating from each node (as in figure 4.1) is served from the cache which is on the path between the originating node and the destination node.

We assume that all miss traffic will be served by the root node, which represents the server for of all requests. Hence, in simulations all the requests have root node as the destination node.

In the simulation and optimization models, we assume that all the traffic is Hypertext Transfer Protocol (HTTP) traffic, which includes HTML, text, images, audio, video and dynamic data. Other traffic, other than HTTP is not considered in our simulations.

We assume that the propagation delay within a regional ISP are very small (This assumption was made after running a series of simulation with and without link delays in the network, not much difference was found between the two results. The reason for this is we used a small network to perform our simulations), hence we assume the propagation delay on the links (figure 4.1) to be zero in both optimization

and simulation models. We also assume that there is enough buffer space in the routers.

6.5 Trace Driven Simulations

There are 2 methods of network modeling, analytical modeling and discrete-event simulation. Analytical modeling is performed by a set of mathematical expression as described in chapter 5. Using the mathematical model or the optimization model used in chapter 5, we will not be able to simulate the dynamic nature of the network. Thus validation of the analytical model requires discrete-event simulation, which allows for more realistic modeling of real system.

In discrete event simulation, the representation of time is quantified, and the system state only changes when an event occurs [SIMU02]. Discrete event simulation of a computer system may be categorized as *event-driven* or *cycle-based.* In event-driven simulation models, a series of events occur asynchronously and at irregular intervals. Event-driven simulations can model a wide variety of systems. In cycle-based simulations, all changes in system state are synchronized to a single clock. The simulation is essentially a large state machine that changes state at each clock.

There are three basic techniques to generate workloads for simulation: stochastic, trace-driven, or execution-based [SIMU02]. Stochastic simulations describe the arrival patterns of requests by sampling from a probability distribution. Trace-driven simulation represents the workload as a sequence of operations or requests. The third method, execution-based simulation, is used to model systems in detail. The input to an execution based simulation is the same executable code that the real system will run. This method requires writing and validating a detailed simulation; it leverages existing compilers and permits a wide-variety of workloads to be modeled with little effort [SIMU02]. The disadvantages of using any simulation include the effort required to write and validate the simulation program and the fact that same simulation will run many times slower than the real system.

There are many simulation tools that are available to model a communication network. The simulation tool that we used in this thesis is NS-2 because of good support for the TCP/IP protocol stack and because it is a free tool, NS-2 is described in Appendix B.

6.6 Data Flow

Figure 6.1 shows a basic request flow diagram of the simulation model. On request of a page by the client, if the requested document is found, either at a cache or the origin server, new TCP and sink agents are created. The TCP agent is at the source, and the sink at the client, and they are connected. Next, a File Transfer Protocol (FTP) agent is created and the page is sent to the client. After all data has been transferred, TCP's close method is called (after all the data is acknowledged) and all the agents are scheduled to be destroyed.

On arrival of a request, a new session is started. The client requests a document using HTTP send-request. If there is a cache on the next node on the path from client to destination the cache checks for the document, if the document is present, then we have a cache-hit, else it is a cache-miss. If cache-hit is called, the cache transfers the requested document to the original client. If cache-miss method is called then the request is sent to the server which replies to the original client. Figure 6.1 shows the basic flow chart diagram.

Initially we followed a LRU replacement policy for replacing a page in the cache. The caches needed to be populated with the most popular pages before running the simulation. This process is time consuming, and we would run the simulations for months to populate the caches before running the required simulations.

We choose a fraction α of the trace file as hits traffic and a fraction $1-\alpha$ as miss traffic. Bernoulli distribution was used to decide whether a request was a hit or miss request based on the fraction. Hit and miss document distribution is detailed under section 6.3.

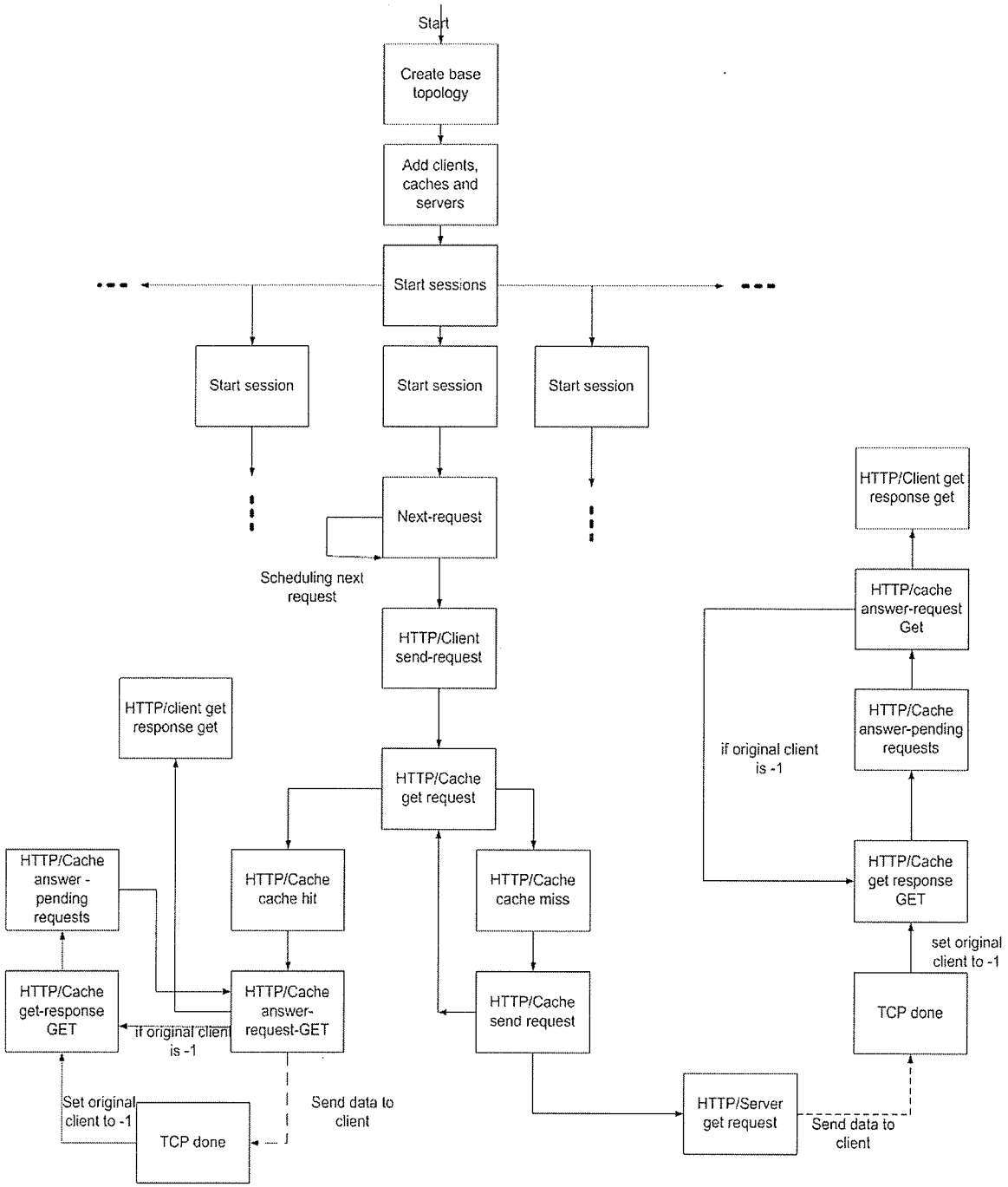

Figure 6.1: Web Cache Basic Simulation Flowchart [SAIL03]

We used "cacheability" field in the simulation input trace files (described in Section B.3.2). Cacheability field in the trace file was used to determine ahead of time whether a document is served from a cache or from the origin server. The cacheability field is either set to 0 or 2, if the cacheability field was set to 0 then we considered the request to be a miss and this request was destined to Internet node. By setting the cacheability to 2, we served the request from cache on the path to the Internet node.

6.7 Summary

This chapter has described in detail the network and traffic model used in the simulations. It gave an introduction to the network components, i.e. how clients, servers and the caches were generated and modeled. A detailed explanation of the traffic model and the assumption made are also described. The next chapter gives the results of optimization model and simulation model and a discussion of results.

CHAPTER SEVEN

7 Optimization and Simulation Results

In this chapter, we present the results for the optimization and simulation models. The results from the optimization are compared with the results from the simulation models to test the validity of the models. We attempt to determine the location of the caches in the regional ISP network and the target amount which would be reasonable to spend in improving the network performance. Section 7.1 describes the performance measure to be used. Section 7.2 presents the results of optimization model. Section 7.3 presents the results from simulation model. Section 7.4 discusses the uncertainty nature of our model. Section 7.5 discusses the results.

7.1 Performance Measures

In ISP's like Sasktel, problems like network congestion, higher bandwidth consumption, and increased server load may arise mainly due to increased number of web users and also due to transfer of huge files. To address such problems, we measure the performance of the system as the ratio of summation of elapse time of requests from source to destination(destination may be either a cache or the origin server) to the summation of file size of requests transferred, i.e. the mean time to transfer one Mb of data from source to destination.

7.2 Results from Optimization Models

A series of optimizations were run to determine the capacity and location of caches and the budget that would be suitable for ISP's network. All the optimizations were run using the MINLP solver provided by NEOS server; described in Appendix A. This section describes optimization results for both variable and fixed cache capacity models. Two hundred optimizations were run for both models. Optimizations and simulations were performed varying Q values (the upper and lower bounds of the objective function are weighted by the value of Q and $(1-Q)$ respectively. Increasing the value of Q increases the weight on upper bound and decreasing the value of Q increases the weight on the lower bound) and budget (as in Chapter 5 table 5.2). The

values of Q were ranging from 0 to 1 in steps of 0.1 and budget was chosen from \$0 to \$100000 in steps of \$5000.

Figure 7.1 represents the graph for fixed cache capacities, we used a fixed cache capacity of 30 Mbps each costing \$30*slope (Slope =159.14, described in Section 5.2.4). The x-axis of the figure represents the budget in US dollars and the value of the y-axis represents the mean time to transfer 1 Mb of data in the network. Table D.4 contains the values represented in figure 7.1. Table D.5 contains the cache locations (Table B.2 represents the node ID corresponding to the figure 4.1) and the number of caches at each location.

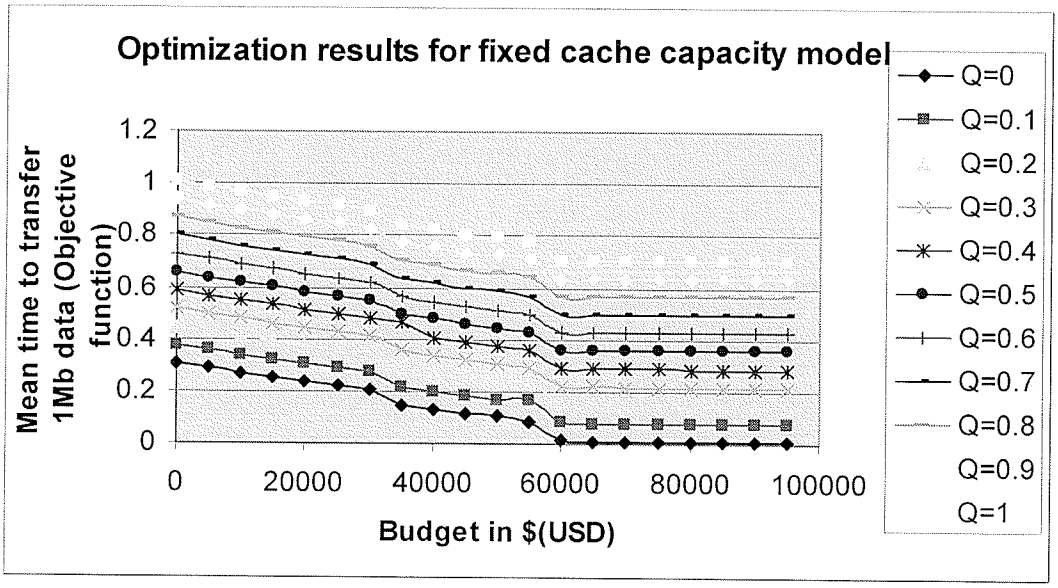

Figure 7.1: Optimization Results for Fixed Cache Capacity Model

From optimization results we observe the location of caches and the number of caches placed at each node of the network were same for Q value ranging from 0 to 1. In figure 7.1, we observe the curves have same pattern for Q values ranging from 0 to 1. We also observe that increasing the budget decreases the objective function value, however increasing the budget greater than \$60,000 has little effect on mean time to transfer one Mb of data. Hence we conclude that in fixed cache capacity model, the

weights (i.e. Q values) on the upper and lower bound effect mean time to transfers one Mb of data from source to destination but does not effect the location of the caches and number of caches placed at each node in the network. Maximum performance is obtained at the point where the curves are flattened or straight as seen in figure 7.1(The curves are flattened after it reaches a budget of around $60K).

Figure 7.2 represents the optimization results for the variable cache capacity model (explained in detail in chapter 5). X-axis in the graph represents the budget in USD and y-axis represents mean time to transfer one Mb of data from source to destination in a network. From curves in figure 7.2, we observe that increasing the budget decreases the objective function and the curves flatten as the budget is increased beyond $60K as in fixed cache capacity model. Some points on the curve, like at x-axis 45000 and Q value of 0.9, we observe that the objective value is greater than objective value at Q value of 1. This was because optimizer could not solve the objective function for minimal optimal solutions for Q value of 0.9 and budget of $45000.

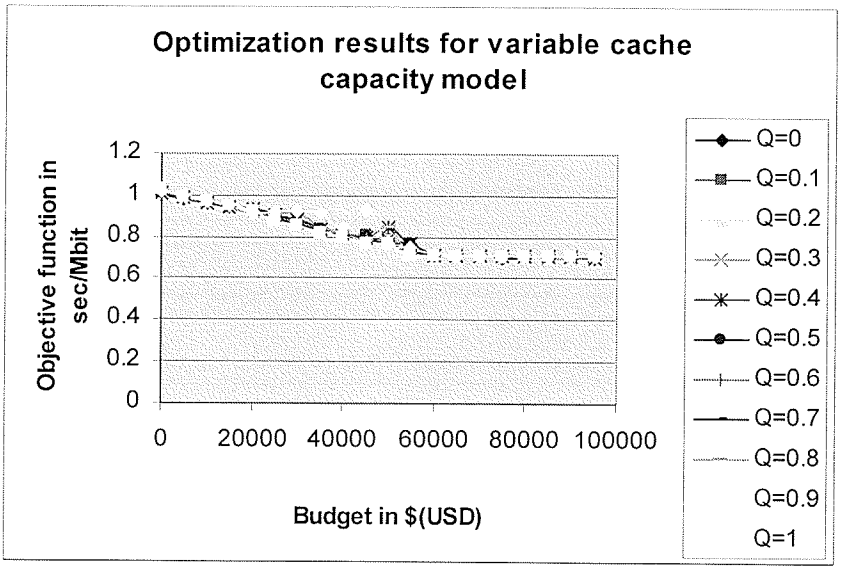

Figure 7.2: Optimization Results for Variable Cache Capacity Model

We performed a diagnostic test, to test if the solution obtained by the optimization solver was minimal optimal solution. Equations 7.1.1 through 7.1.4 were used to test if the solution obtained from the solver were optimal solutions. f_0 and f_1 represents the lower and upper bounds of the objective function respectively and x_0 and x_1 represents the values of the variables in the objective function for 2 different points on the same budget but having different Q values. All the equations will be satisfied if x_0 and x_1 are optimal with respect to f_0 and f_1.

$$f_0(x0) \leq f_0(x1) \qquad \text{Equation 7.1.1}$$

$$f_1(x1) \leq f_1(x0) \qquad \text{Equation 7.1.2}$$

$$f_0(x0) \leq f_1(x0) \qquad \text{Equation 7.1.3}$$

$$f_0(x1) \leq f_1(x1) \qquad \text{Equation 7.1.4}$$

We tested the results obtained by solver by solving for different outputs of lower and upper bound functions of the objective function separately. For example we ran the optimization for variable cache capacity model with Q=0.9 and Q=0.4 for a budget of $45000. Results of objective function value and values of all the variables in the optimization problem were collected. Values of the variables obtained for Q=0.9 and Q=0.4 for a budget of $45000 were substituted in equations 7.1.1 through equation 7.1.4.

Equation 7.1.1 represents that the lower bound function value(obtained by substituting the values of the lower bound function variables collected by solving the optimization problem for Q=0.4 and budget= $45000 into the lower bound function) of the first set should be less than or equal to lower bound function value(obtained by substituting the values of the lower bound function variables collected by solving the optimization problem for Q=0.9 and budget= $45000 into the lower bound function) of the second set.

Equation 7.1.2 represents that upper bound function value(obtained by substituting the values of the upper bound function variables collected by solving the optimization problem for Q=0.9 and budget= $45000 into the upper bound function) of the first set should be less than or equal to the upper bound function value(obtained by substituting the values of the lower bound variables collected by solving the optimization problem for Q=0.4 and budget= $45000 into the upper bound function) of the second set.

Equation 7.1.3 represents that lower bound function value(obtained by substituting the values of the lower bound function variables collected by solving the optimization problem for Q=0.4 and budget= $45000 into the lower bound function) of the first set should be less than or equal to the upper bound function value(obtained by substituting the values of the lower bound variables collected by solving the optimization problem for Q=0.4 and budget= $45000 into the lower bound function) of the second set.

Equation 7.1.4 represents that lower bound function value(obtained by substituting the values of the upper bound function variables collected by solving the optimization problem for Q=0.9 and budget= $45000 into the lower bound function) of the first set should be less than or equal to the upper bound function value(obtained by substituting the values of the upper bound variables collected by solving the optimization problem for Q=0.9 and budget= $45000 into the upper bound function) of the second set.

If all the above equations were satisfied, the solutions obtained were optimal solutions. We found that some points on the curve were not optimal solutions. The results obtained from the solver were tested for optimal solutions; we found that some of the solutions were not optimal, so we re-ran this optimization by changing the budget slightly. Example: For Q=0.9 and budget of $45000, the solution obtained from the optimizer was not optimal, so we changed the budget slight by $45000.01 and re-ran the optimization to solve the problem for optimal solution.

Figure 7.3 represent the optimization results for variable cache capacity model obtained by testing and re-running (re-ran all the optimization which did not have optimal solution by changing the budget slightly as mention above) some of the optimizations of figure 7.2. Table D.1 contains the values of figure 7.3. Table D.2 contains the cache locations (Table B.2 represents the node ID corresponding to the figure 4.1) and Table D.3 contains the cache capacity corresponding to the cache location in Table D.2

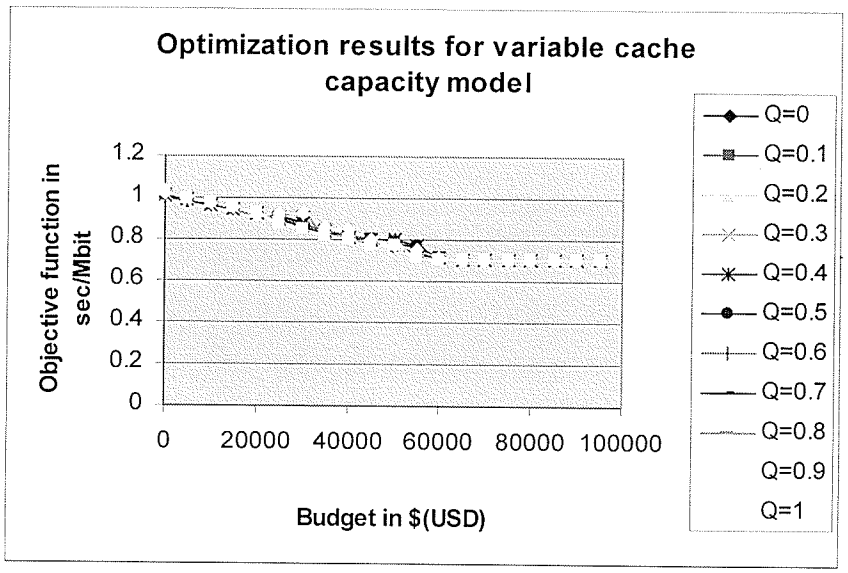

Figure 7.3: Optimization Results for Variable Cache Capacity Model (Re-run)

7.3 Results from Simulation Models

A series of simulations were run to obtain more accurate performance measures as a function of the location and capacity of the caches. All the simulations were run using NS-2, NS-2 is explained in details in Appendix B. This section describes simulation results for both variable and fixed cache capacity models. The simulation model takes as input the location of the caches and cache capacities from the optimization model. Twenty simulations were run for fixed and two hundred simulations were run for variable cache capacity models.

Figure 7.4 represents simulation results for fixed cache capacity. From optimization results we observed that for each value of the budget the location and number of caches were the same for each value of Q. Hence we ran only one simulation for each value of the budget as shown in figure 7.4. Table D.7 contains the data for figure 7.4.

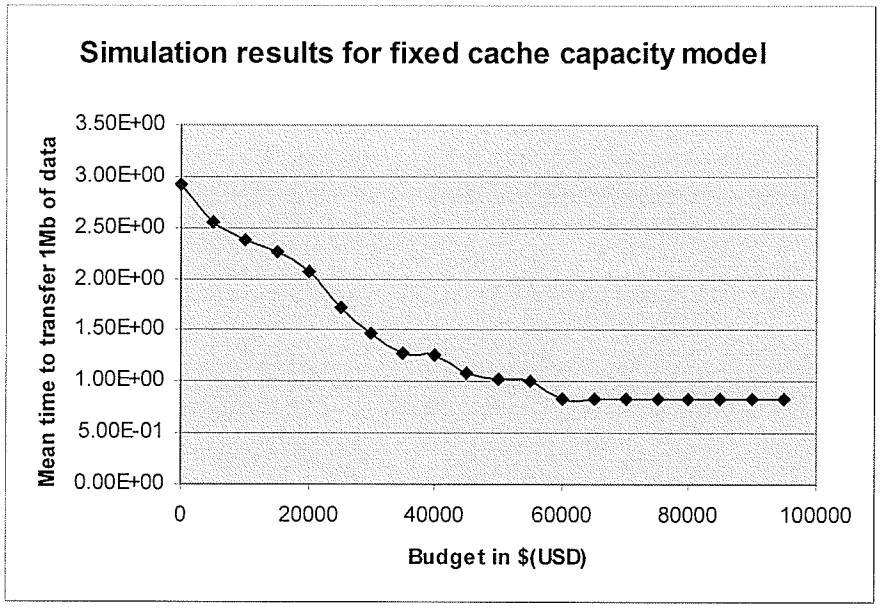

Figure 7.4: Simulation Results for Fixed Cache Capacity Model

From figure 7.4 we observe the curve starts flattening from around $60,000, any increase in the budget beyond $60,000 does not decrease the mean time to transfer one Mb of data from source to destination. By investing a budget of around $60,000, the network can reduce the mean time to transfer one Mb of data from 2.910316 seconds to 0.83 seconds.

Figure 7.5 represents simulation results for variable cache capacity model. We ran the simulation for different location and number of caches obtained from the optimization. Table D.6 represents values of figure 7.4.

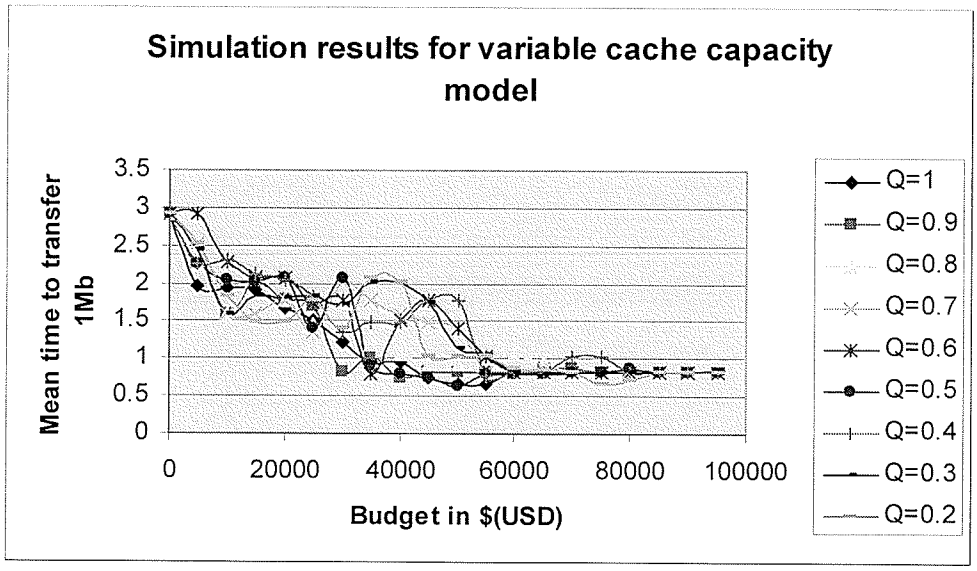

Figure 7.5: Simulation Results for Variable Cache Capacity Model

Figure 7.6 represents the best performance curve obtained by taking the lower curve (this curve is obtained by plotting a graph that passed through or closer to a maximum number of points in the figure 7.5.) of figure 7.5. We observe from figure 7.5 that the curves are not very smooth, this is because the optimization results were not minimal optimal in all cases due to its non-linearity nature. We considered only 10 values rather than 20 values as in all other figures for simplicity.

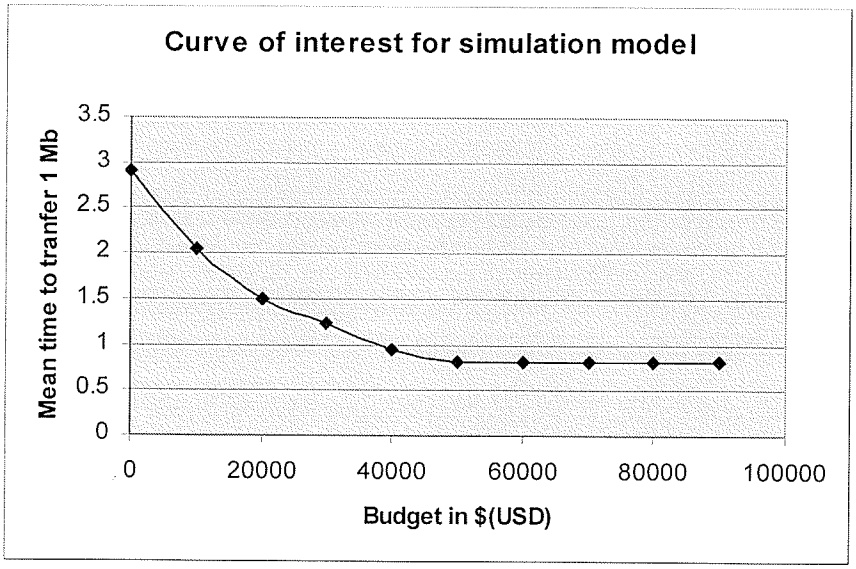

Figure 7.6: Curve of Interest for Variable Cache Capacity Model

From figure 7.6 we observe the curve starts flattening from around $60,000, any increase in the budget beyond $60,000 does not decrease the mean time to transfer one Mb of data from source to destination. By investing a budget of around $60,000, the network can reduce the mean time to transfer one Mb of data from 2.91 seconds to 0.83 seconds.

On observing that simulation results obtained from both the models, fixed cache capacity model and variable cache capacity model, we conclude that around $60K is a ideal amount to invest for an ISP's network.

The results from the optimization vary from that of the simulation models slightly. We observe that in simulation model the mean time to transfer one Mb of data reduces from 2.91 seconds to 0.83 seconds for a budget of $60,000, where as in the optimization model we observe that the mean time to transfer one Mb of data reduces from 1 second to 0.7 seconds for a budget of $60,000. This difference in magnitude is because simulation model can consider other details like delays in the network. In all the cases we observe that a good amount to invest in order to improve the performance in an ISP was around $60,000.

7.4 Error Estimation

In both fixed and variable cache capacity model we considered cache location and capacity values for budget ranging from 0 to 90000 in steps of 10000. Hence we have only 10 values for each trace file. In each case we collected 5 different traces, each having 10000 requests from the ISP's log files. We ran 5 instances of each simulation using five different traces in order to estimate the magnitude of the error due to variation in the trace.

Figure 7.7 represents the error bars for variable cache capacity. Each dot on the graph represents the average over the 5 traces of the mean time to transfer 1 Mb of data. The error bars correspond to an interval one standard deviation above and below the

average observed (standard deviation is +6sec above and -6sec below as in figure 7.7).

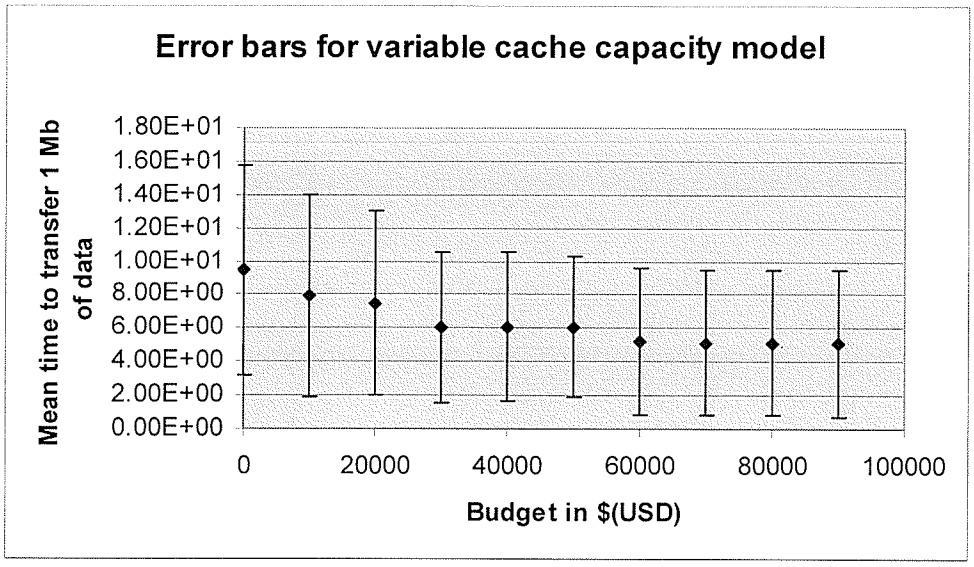

Figure 7.7: Error Bars for Variable Cache Capacity model

Figure 7.8 represents the results with error bars for fixed cache capacity. We observed that in both the models the standard deviation is around +6.0 above and -6.0 sec below the average to transfer 1 Mb of data.

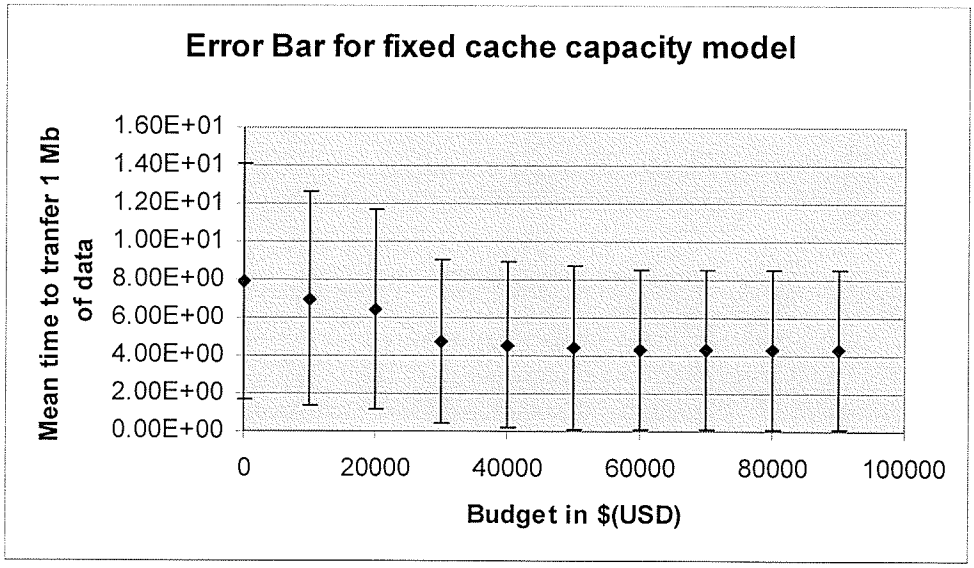

Figure 7.8: Error Bars for Fixed Cache Capacity model

We observe deviation is quite high in both cases; the reason for this variation is because the log files used to estimate error in our model were collected for a period of 8 hour, we considered 10,000 requests at random from the log files. Since the number of requests that we used in our simulations were relatively small compared to the complete log file, there may be variation in our estimation. The other reason may be due to varying network traffic during some hours of a day, for example the number of people accessing the internet may be less between 12 PM and 1 PM of a working day. Increasing the number of request may reduce the variation; since simulations are time consuming we considered only 10,000 requests.

The mean time to transfer 1 Mb of data would be a varying factor; it does not affect our budget estimations drastically as the mean time gets flatten after we reach a certain budget. We are more interested in the budget; an ISP provider would invest for their network then the mean time.

7.5 Discussions

From the optimization (figure 7.1 and 7.3) and simulation (figure 7.4 and 7.6) results we observe that for budgets greater than $60,000, time to transfer 1Mb of data almost remains the same and any increase in the cache capacities or the budget would not improve the performance of the system. Hence we conclude that the ideal amount to be invested for the exiting regional ISP is around $60000. Any increase in cache capacity may not improve the network performance.

All the results presented here are based on the cost of cache capacities. The cost estimation may vary depending on the changes in cache capacity cost, but all the other estimation such as size of the cache and the location of the caches are still valid and can be applied to ISP's. Our models can be applied to any other network, by providing network topology, budget, cost of one Mb of cache capacity and traffic on the link.

CHAPTER EIGHT

8 Conclusions

In this chapter, we summarize the contributions of this thesis and discuss possible future work. This thesis, presents a mathematical model and a simulation model for efficiently placing caches in the network within the available budget. This is one of the most important factors in the real-world scenario, where some ISPs may have limited budgets while others may have more. However, one of the goals of all ISPs is to reduce the overall average response time for clients' requests, within the allocated budget, which is the goal of our thesis.

Our models not only help in attaining the maximum performance within the given budget, but also help in predicting the need for upgrading the network. This thesis has solved the cache location problem for Sasktel telecommunication for a given budget. The results showed that a budget of around $60,000 is ideal investment for the existing network. However the models presented here are based on the cost of cache capacities estimations available. All the other estimation such as size of the cache and the location of the caches are still valid and can be applied to ISP.

Our models presented in this thesis can be applied to any other network, provided we have information on network topology, target budget, cost of one Mb of cache capacity and traffic on the links.

8.1 Contributions

The contributions of the thesis are as follows:

1. A survey of web caching technology was performed; we provided details of cache architecture, cache routing, cache prefetching, cache replacement, cache coherence, cache placement, cache resource allocation, cache scalability and cacheability of web objects.
2. The workload characteristics of an ISP's network traffic have been examined.

3. We performed analysis of trace files to get the current network traffic characteristics and traffic estimates for the future. We also performed studies to estimate the cost of web cache capacities.
4. We developed a mathematical model, to solve the cache location problem for a Regional ISP. The mathematical models were developed for fixed cache capacities and variables cache capacities. Both models developed, showed that under our current cost assumption the required budget for the existing Sasktel network was around $60,000(USD)
5. We developed a simulation model to validate the mathematical model, used to solve the cache location problem for a Regional ISP. The simulation models were developed for fixed cache capacities and variable cache capacities. The required budget estimate of $60K (USD) from the optimization was validated.
6. Most of the optimization problems related to cache locations are NP-hard (nondeterministic polynomial time). This is the reason that most of the algorithms developed to solve these problems are based on heuristics. In this thesis we have discussed how a combination of simulation and Mixed Integer Non-Linear Programming can be used to solve the Cache Location Problem. We considered a small network; hence we did not require heuristics.

8.2 Recommendation for Future Work

A primary objective in this line of study will be to obtain the maximum performance with the given budget. Caches are and will be future basis for better network performance at reasonable cost because they are effective way to reduce network workload.

The following issues can be studied for future:

1. This thesis is focused only on a regional ISP; in future we would like to extend to larger ISPs.
2. The optimization model defined in this thesis for both the cases fixed cache capacity and variable cache capacities, does not consider the TCP slow start process. In the future we would like to consider this issue.

3. The number of caches required and their location was estimated only for the existing year. We could extend the problem to solve for a period of over 5 or 10 years.
4. Other interesting directions for further research include the extension of the model to multicast traffic and wireless traffic.
5. We did not consider the movement of caches. At a later stage, we also plan on including some cost ($) function for moving caches from one node to another. Other factors such as interaction between caches and the effect of changing network bandwidth could also be addressed in order to achieve better performance. These factors are very important in real-world circumstances. We could extend our study to include various factor such as changing the network topology and parameters like bandwidth.
6. In this thesis we were interested in Static cache location, but techniques like the ones used in active networks and the continued process of memory cost reduction may lead to a scenario in which caches can be dynamically moved in the network. This will require local distributed techniques to deal with the dynamic optimal cache location problem. Dynamic optimal cache location refers to the problem of determining the set of web cache for the placement of a document. For the dynamic case, two sub-optimal algorithms can give results close to the optimal cache placement: a) Dynamic First Fit (DFF); b) Minimum Weight Perfect Match (MWPM) [GHOS04] [KRIS00].
7. In this thesis, we assumed propagation delay to be small, (This assumption was done after comparing the results of simulations with and without propagation delay) and hence was not considered in our simulation models. However propagation delays can reduce the throughput to some extent for larger networks. In our future experiments we would like to introduce propagation delays into our simulation models.

APPENDIX A

A. Optimization Languages and Software

A.1 NEOS Server

The Network Enabled optimization System (NEOS) Server introduces a novel approach for solving optimization problems. The NEOS project was launched by Argonne National Laboratory and Northwestern University in late 1994. Optimization solvers supported by NEOS represent the state-of-the-art in optimization software. Optimization problems are solved automatically with minimal input from the user. Users only need a definition of the optimization problem; all additional information required by the optimization solver is determined automatically [CZYZ97] [NEOS]. Users of the NEOS Server submit a problem and their choice of optimization solver over the Internet using a JAVA submission tool, e-mail submission or web interface [FERR98] [CZYZ97].

A.2 AMPL

A Mathematical Programming Language (AMPL) [FOUR02] is an algebraic modeling language. It was designed and implemented around 1985. AMPL is notable for the similarity of its arithmetic expression to customary algebraic notation, and for the generality and power of its set and subscripting expression; it is also extended to express structures such as network flow constraints and piecewise linearity's.

A.3 MINLP

Mixed Integer Nonlinear Programming (MINLP) refers to mathematical programming with continuous and discrete variables and nonlinearities in the objective function and constraints. The use of MINLP is a natural approach of formulating problems where it is necessary to simultaneously optimize the system structure (discrete) and parameters (continuous).

The general form of a MINLP is

Minimize $f(x, y)$

Subject to $g(x, y) <= 0$

$x \in X$

$y \in Y$

The function $f(x, y)$ is a nonlinear objective function and $g(x, y)$ a nonlinear constraint function. The variables x, y are the decision variables, where y is required to be integer value. X and Y are bounding-box-type restrictions on the variables. In our optimization problem we formulated the problem, objective function is non-linear and all constrains are linear [LEYF99].

The NEOS Server offers Mixed Integer Non-Linear Programming (MINLP) solver for the solution of mixed integer nonlinearly constrained optimization problems in AMPL format. MINLP solver for NEOS server was developed by Roger Fletcher, and Sven Leyffer [NEOS]. MINLP is suitable for large nonlinearly constrained problems with a modest number of degrees of freedom [LEYF99].

MINLP implements a branch-and-bound algorithm searching a tree whose nodes correspond to continuous nonlinearly constrained optimization problems. The continuous problems are solved using filterSQP, a Sequential Quadratic Programming solver which is suitable for solving large nonlinearly constrained problems.

APPENDIX B

B. Simulation tool and Trace File Formats

B.1 NS-2 Overview

University of Southern California Information Sciences institute developed Network Simulator (NS-2) in 1989. This software is freely available, with full source code written in C++ and accessed using OTCL (Object oriented Tool Command Language, an interpreter language.). The objects written in OTCL can instantiate objects and call them from C++ code. NS is a discrete event simulator targeted at networking research. Ns provides support for simulation of Transmission Control Protocol (TCP), User Datagram Protocol (UDP), IP (Internet Protocol) routing, and multicast protocols over wired and wireless (local and satellite) networks [NS2]. NS-2 has a network animator (nam), used to view network topology and animated movement of traffic flow for a period of time

B.2 Modifications to NS-2

NS-2 provides with flexibility of changing the functionality of the any layer in the layer model (TCP/IP architecture or the ISO-OSI architecture). NS-2 was attractive because of its well defined HTTP and web caches classes. But we needed to make some modifications to the actual methods because of its undesirable nature. The trace file HTTP format was modified from two file format to one file format (details can be see in [NS-2] NS-2 documentation on PagePool). Other modification that has been done was, we change the packets size in TCP Layer from 1500 to 1000 bytes.

B.3 Input and Output Trace File Formats

We used trace driven simulations model. The model needs three files to run the simulation, cache file, topology file and trace file as it inputs and output's a log file. This Section describes about the trace file formats used in the simulations.

B.3.1 Cache File

Cache file is used to specify which node in the topology has cache. The format for cache file is show below:

[Node number] [Cache size]

Where: Node number is the number of the node in the topology

Cache size is the size of the cache in bytes.

B.3.2 Topology File

Topology file is used to specify the topology of the network to be simulated i.e. the nodes that are connected and link bandwidth. The format for topology file is shown below:

[Source node] [Destination node] [Bandwidth] [Delay] [Buffer size]

Where: Source node is the node number of one of the nodes on the duplex link

Destination node is the node number of the node at other end of a duplex link

Bandwidth is the bandwidth in bytes per second of the link between the source and destination node.

Delay is the propagation delay incurred for a packet to traverse the link between the source and destination node.

Buffer size is the size of the buffer in bytes on the link between the source and destination node. If this buffer becomes full, packets will be dropped.

If the source and destination are the same on each row of the file, this defines the properties of the link to the server or cache server which is attached to the node [SAIL03].

B.3.3 Trace File

The trace file is used to define the list of all events that occur in a time period. The trace file is the input file to the simulation model. Basically is a traffic generation file. The format for trace file is shown below:

[Start time] [URL] [File size] [Source node] [Destination node] [Cacheability]

Where: Start time is the time a request is made by a client node for document in seconds

URL (Uniform Resource Locator) is the URL address of the document being requested the client node

File size is the size of the object requested in bytes.

Source node is the node number associated with the node where the document was requested.

Destination node is the node number associated with the node from where the document is requested. The request will be routed to this origin server or destination node initially, but will be served by any node along the way whose cache contains the requested object.

Cacheability specifies if the document that is requested is cacheable on any node, between the source node and destination node, provided there is a cache on the path.

B.3.4 Output File Format

The output for the simulation is sent store in the output file. Each row of output corresponds to the completion of one file transfer. The format of output file is shown below:

[Start time] [End time] [Document id] [Document size] [Server node] [Client node]

Where: Start time is the time in seconds when the request originates at a client node

End time is the time in seconds when the file transfer completes

Document id is the id of the document requested by the client node

Document size is the size in bytes of the requested document transferred

Server node is the number of the node from where the document was served; it may be either from a server or from a cache.

Client node is the number of the node where the request originated.

B.3.5 Delay Calculation

Table B.1 represents the link between two nodes (referring to figure 1.1) their distances in kilometers and propagation delay in seconds.

Places	Distance in Km	Delay in Sec
P-P1	227.99	$113.99*10^{-5}$
P-P2	374.325	$187.162*10^{-5}$
P-P3	462.553	$231.277*10^{-5}$
P-P4	331.499	$165.74*10^{-5}$
S-S1	141.841	$70.9*10^{-5}$
S-S2	139.587	$69.79*10^{-5}$
S-S3	273.056	$136.528*10^{-5}$
S-P	261.626	$130.8*10^{-5}$
P-I	-	$3*10^{-3}$

Table B.1 Delay calculation based on distances

Places	Node ID
R1	1
R2	2
R3	3
R4	4
S1	5
S2	6
S3	7
R	9
I	8

Table B.2: Place and Corresponding Node ID

APPENDIX C

C. Definitions

Document: Request / page
Note: In this report we used page, request, document interchangeably.

Cache Capacity: Maximum throughput that can be achieved by a cache in Mbps in optimization models and Mbps in simulation models

Hit percentage: Percentage of documents that are directly served from the web cache instead of going to original server.

Miss percentage: Percentage of documents that were not served from the web caches, but are served from original servers.

Optimization: Selection of a better or best alternative among a number of possible states or affairs, usually the maximization or minimization of a function that depends on the objective of the system the objective function.

Throughput: Number of bytes of data transferred for 1 sec (simulation model) or number of Mb (Mega bits) transferred in one sec from source to destination (optimization model)

APPENDIX D

D. Optimization and Simulation Tables

D.1 Optimization Output Values for Variable Cache Capacity Model

Cost vs. Q	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	1	1.0029	1.0054	1.0082	1.01	1.0125	1.015	1.0175	1.0191	1.0215	1.023
5000	0.9822	0.9838	0.9864	0.9881	0.9887	0.9904	0.992	0.9927	0.9927	0.9969	0.990
10000	0.9619	0.9631	0.9638	0.9641	0.9641	0.9656	0.9669	0.9656	0.9674	0.9677	0.97
15000	0.9391	0.9397	0.9432	0.9451	0.9433	0.9434	0.9425	0.9417	0.9416	0.9497	0.949
20000	0.9209	0.9212	0.9256	0.9193	0.927	0.9182	0.9197	0.9171	0.9157	0.9126	0.922
25000	0.9012	0.9006	0.9001	0.9026	0.8985	0.8987	0.8971	0.8947	0.9008	0.9226	0.877
30000	0.8728	0.862	0.8688	0.8662	0.8793	0.873	0.8742	0.8559	0.9199	0.9207	0.851
35000	0.8363	0.8367	0.8359	0.8443	0.8341	0.8359	0.8397	0.8308	0.8305	0.9101	0.81
40000	0.8174	0.8166	0.8226	0.8142	0.8137	0.8119	0.8169	0.8071	0.8117	0.8489	0.795
45000	0.7983	0.8021	0.7962	0.8009	0.7992	0.797	0.7915	0.7857	0.7837	0.9345	0.766
50000	0.7873	0.7918	0.7835	0.7953	0.8471	0.7632	0.772	0.7707	0.8479	0.768	0.766
55000	0.7637	0.7525	0.748	0.748	0.7689	0.7685	0.7279	0.7287	0.7195	0.75	0.710
60000	0.7121	0.7118	0.7134	0.7113	0.7111	0.7112	0.7106	0.7103	0.7102	0.719	0.708
65000	0.7082	0.7078	0.708	0.7081	0.7083	0.7085	0.7095	0.7089	0.7094	0.7094	0.708
70000	0.7059	0.7069	0.7072	0.7075	0.7076	0.7079	0.7082	0.7085	0.7088	0.7093	0.708
75000	0.7045	0.707	0.7059	0.7065	0.707	0.7074	0.7077	0.7082	0.7087	0.7092	0.708
80000	0.7036	0.7051	0.7051	0.7058	0.7065	0.7072	0.7075	0.7078	0.7082	0.7089	0.708
85000	0.7031	0.7038	0.7046	0.7053	0.706	0.7066	0.7074	0.7077	0.7082	0.7085	0.708
90000	0.7033	0.7034	0.7042	0.7054	0.7057	0.7064	0.7072	0.7076	0.708	0.7091	0.708
95000	0.7029	0.7044	0.7039	0.7047	0.7055	0.7064	0.7071	0.7074	0.7078	0.7091	0.708

Table D.1: Optimization values of variable cache capacity model

Cost vs. Q	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5000	3	1	3	7	1	1	2	3	1	1	4
10000	2,5	2,6	2,6	3,4	2,5	6,7	3,7	3,4	3,4	3,4	5,7
15000	5,6	1,2,6	1,6,7	1,6,7	1,6,7	4,5	5,6	2,4,6	5,6	4,5	1,2,3,4
20000	5,6,7	1,4,5	1,4,5	1,2,3,6	3,4,6,7	1,2,6	1,3,5,7	4,5,6	4,5,6	4,5,6	1,4,6,7
25000	3,4,5,6	1,2,4,5,7	1,2,7	1,2,3,4,6	1,2,3,4,7	1,2,3,4,6	4,5,6,7	1,3,4,5,7	1,3,5,6	3,4,5,6	1,3,4,5,6
30000	9	1,4,7	1,4,5	1,5,6	1,4,5	1,3,5	1,4,6	4,5	1,5,6	1,2,3,4	1,2,3,5
35000	1,3,4,5,6,7	2,3,5	4,5,6	4,5	4,5,6	2,9	1,2,3,4,5,6	4,5,6	3,9	1,2,4	2,3,4,5
40000	2,4,7	1,5	2,3,4,5,6,7	2,4,9	5	2,4,9	1,3,4,5,6	3,4	1,4	2,4,9	2,3,9

45000	1,2,4,9	2,3,4,9	4,5,7	4,5,6,7	4,7	1,3	4,5,6	1,4,5	3,5	1,4,6	2,5,6,7
50000	1,2,3,4,9	1,2,3,4,7,9	1,3,5	2,3,5,6,9	1,4,6	2,4,5	4,5,7	4,5,7	1,4,5,6	4,45	1,2,3,5,6
55000	7,8	5,6	4,5	3,8	1,2,3,4,5,9	4,5,7	1,8	2,8	4,8	2,8	1,4,9
60000	2,7,8	5	4,7,8	4,8	5,8	1,3,5	4,7,8	4,8	1,8	4,8	4,8
65000	5,8	4,5,7	1,7,8	7,8	4,8	2,4,8	4,8	6,8	1,2,8	6,8	4,8
70000	8,9	5	5	4,8	1,8	4,5,8	5,8	8,9	8,9	1,6,8	4,8
75000	8,9	5,8	1,4,6	5,8	5	1,3,4,5	4,6,8	5,8	4,5,8	4,5,8	4,8
80000	8,9	5	2,3,5,6	5,8	3,5,6	3,6,8	4,5,8	1,5,8	1,5,7,8	4,5,8	4,8
85000	8,9	4,8	4,5,8	8,9	1,4,8	5	5,6,8	4,6,8	2,3,6,8	8,9	4,8
90000	8,9	5	5,6,8	4,5,7,8	8,9	1,4,8	4,5,8	4,5,8	2,6	8,9	4,8
95000	3,7,8,9	5,6	1,3,4,5	1,3,5,8	8,9	1,5,8	3,5,8	2,3,5,8	4,5,8	8,9	4,8

Table D.2: Location of the cache for variable cache capacity model

Cost vs. Q	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5000	31.5	31.41	31.4	31.41	31.41	31.41	31.41	31.41	31.41	31.41	31.41
10000	16.55, 46.26	18.27, 44.54	18.27, 44.54	25.9,36.91	16.55, 46.26	43.83, 18.98	35.81, 27.01	25.91, 36.91	25.90, 36.91	25.90, 36.91	45.58, 17.23
15000	49.4,44.7	29.33, 19.06, 45.83	28.94, 45.34 4,19.93	28.94, 45.34, 19.93	28.94, 45.34, 19.93	41.19, 53.03	49.47, 44.75	16.70, 35.56, 41.96	49.47, 44.75	41.19, 53.03	24.06, 14.85, 22.51, 32.83
20000	54.13, 49.16, 22.3	31.22, 41.33, 53.18	31.20, 41.30, 53.16	30.27, 19.79, 28.63, 47.04	26.05, 37.09, 43.63, 18.85	39.6,27.08,59.02	28.78, 27.18, 49.87, 19.80	36.05, 47.10, 42.50	36.06, 47.11, 42.51	36.05, 47.09, 42.49	26.77, 36.12, 43.98, 18.75
25000	26.45, 37.57, 48.85, 44.16	29.09, 18.87, 38.83, 50.3,20.05	63.47, 45.73, 47.87	28.58, 18.47, 26.99, 38.23, 44.87	33.89, 22.62, 32.13, 44.46, 23.93	28.57, 18.47, 26.98, 38.22, 44.86	39.42, 50.98, 46.18, 20.46	27.45, 25.90, 36.90, 48.08, 18.72	30.11, 28.47, 51.67, 46.83	26.45, 37.57, 48.85, 44.16	23.13, 21.72, 31.83, 42.35, 38.0
30000	188.46	62.8,78.43,47.33	48.99, 62.21, 77.27	46.58, 74,67.97	48.99, 62.21, 77.26	53.65, 51.28, 83.59	50.78, 64.31, 73.36	84.95, 103.52	46.56, 73.99, 67.95	47.99, 33.64, 45.79, 61.03	42.40, 29.25, 43.93, 72.85

35000	30.39, 28.74, 40.35, 52.06, 47.19, 21.10	49.9,65.95,104.11	65.09, 80.59, 74.21	99.59, 120.38	65.08, 80.58, 74.20	18.53, 201.41	30.57, 20.03, 28.92, 40.57, 52.31, 47.43	65.09, 80.59, 74.211	24.88, 195.04	74.10, 54.04, 91.72	31.61, 47.42, 57.51, 83.30
40000	17.40, 36.61, 197.25	102.12,149.26	25.03, 35.12, 48.09, 60.98, 55.64, 26.43	17.41, 36.64, 197.33	251.29	17.41, 36.62, 197.31	39.47, 37.54, 51.03, 64.37, 58.85	111.07,140.23	113.41,137.91	17.36, 36.55, 197.36	15.47, 23.26, 212.54
45000	27.93, 17.96, 37.46, 199.322	18.19, 26.64, 37.80, 200.156	99.89, 120.72,62.11	71.38, 87.85, 81.08, 42.48	171.361,111.337	143.99,138.77	84.44, 102.91,95.34	75.68, 93.57, 113.44	112.9, 169.84	78.03, 96.32, 108.33	38.98, 100.14,93.93,49.62
50000	28.67, 18.54, 27.08, 38.34, 201.44	28.10, 18.10, 26.54, 37.67, 19.27, 184.4	91.45, 87.88, 134.8	25.08, 35.192,61.20,55.86,136.86	87.12, 107,119.98	67.52, 111.98,134.66	110.92,133.45,69.71	110.93,133.46,69.72	60.33, 75.54, 92.64, 85.62	143.28,170.819	51.42, 36.42, 49.2,95.95,81.08
55000	16.177,329.33	178.55,166.956	157.90,187.63	22.8599,322.76	31.87, 21.04, 30.17, 42.09, 54.24, 166.08	122.07,146.23,77.34	24.35, 321.159	15.27, 330.25	33.36, 312.2	15.557,329.96	29.50, 39.30, 276.69
60000	19.59, 20.8,336.52	337.7	39.47, 20.50, 316.97	40.74, 336.29	52.23, 324.69	110.36,106.2,160.43	39.51, 20.52, 316.88	40.84, 336.10	39.27, 337.7	41.22, 335.7	47.79, 329.12
65000	70.62, 337.7	144.08,171.67,92.54	40.98, 29.67, 337.7	70.74, 337.7	70.63, 337.7	24.06, 46.63, 337.7	70.63, 337.7	70.65, 337.7	41.84, 28.83, 337.7	70.63, 337.7	70.62, 337.7
70000	202.32,237.42	337.7	337.7	102.154,337.7	102.04,337.7	51.35, 64.74, 323.71	102.04,337.7	202.38,237.38	202.434,237.356	41.09, 60.94, 337.7	102.03,337.7
75000	217.305,253.843	133.44,337.7	132.54,160.38,178.25	133.563,337.7	337.7	101.45,97.57,123.84,148.35	62.29, 71.15, 337.3	52.05, 81.41, 337.7	59.65, 73.84, 337.7	59.40, 74.04, 337.7	133.44,337.7
80000	232.299,270.259	337.7	83.05, 107.037,161.59,150.90	164.97,337.7	129.32,192.812,180.432	67.06, 97.86, 337.7	73.99, 90.86, 337.7	65.40, 99.49, 337.7	49.77, 78.33, 36.79, 337,7	74.92, 89.9,337.7	164.85,337.7

85000	247.2 93,28 6.674	196.2 65,33 7.7	88.6,1 07.70 2,337. 7	247.3 6,286. 72	88.10, 108.1 6,337. 7	337.7	101.8 9,94.3 7,337. 7	92.33, 103.9 6,337. 7	45.77, 60.83, 89.70, 337.7	247.5 2.286. 45	196.2 6,337. 7
90000	267.7 7,297. 6	337.7	118.0 44,10 9.67,3 37.7	80.56, 98.48. 79,33 7.7	267.7 8,297. 6	102.5 8,125. 17,33 7.7	103.1 7,124. 51,33 7.7	103.1 8,124. 52,33 7.7	196.8 55,28 2.61	267.7 9,297. 6	227.6 7,337. 7
95000	57.38, 45.03, 222.8 2,271. 55	307.6 25, 289.1 6	129.3 7,124. 61,15 6.65,1 86.18 7	74.92, 71.87, 112.4 08,33 7.7	286.0, 310.8 1	105.4 2,153. 73,33 7.7	103.0 5,156. 03,33 7.7	59.75, 78.15, 121.1 98,33 7.7	117.7 7,141. 36,33 7.7	286.3 10.8	259.0 8,337. 7

Table D.3: Capacity of the caches at locations described in Table D.2

D.2 Optimization Output Values for Fixed Cache Capacity Model

Cost vs. Q	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	0.3042	0.3752	0.4463	0.5174	0.5876	0.6584	0.7293	0.8001	0.8701	0.9408	1.0116
5000	0.2868	0.3569	0.427	0.4971	0.5672	0.6374	0.7075	0.7776	0.8477	0.9178	0.9879
10000	0.2692	0.3399	0.4095	0.4797	0.5496	0.6194	0.6893	0.7592	0.8291	0.899	0.9689
15000	0.2518	0.3216	0.3915	0.4614	0.5313	0.6011	0.671	0.7409	0.8107	0.8806	0.9505
20000	0.235	0.3047	0.3743	0.444	0.5136	0.5832	0.6529	0.7225	0.7921	0.8618	0.9314
25000	0.2201	0.2898	0.3595	0.4291	0.4988	0.5685	0.6382	0.7079	0.7787	0.8473	0.9169
30000	0.2033	0.2728	0.3423	0.4117	0.4812	0.5506	0.6201	0.6895	0.759	0.8284	0.8979
35000	0.1468	0.2165	0.2862	0.3559	0.4685	0.4953	0.565	0.6348	0.7045	0.7742	0.8439
40000	0.1294	0.1988	0.2682	0.3376	0.407	0.4823	0.5459	0.6153	0.6847	0.7542	0.8236
45000	0.1126	0.1818	0.251	0.3202	0.3894	0.4586	0.5278	0.597	0.6661	0.7353	0.8045
50000	0.1097	0.1702	0.2393	0.3084	0.3775	0.4466	0.5157	0.5848	0.6539	0.723	0.7921
55000	0.0858	0.1672	0.2366	0.293	0.3621	0.4311	0.5002	0.5693	0.6445	0.7074	0.7765
60000	0.0173	0.0863	0.1553	0.2242	0.2932	0.3621	0.4311	0.5001	0.569	0.638	0.7069
65000	0.0096	0.0793	0.1491	0.2188	0.2885	0.3583	0.428	0.4977	0.5675	0.6372	0.7069
70000	0.0098	0.0766	0.1474	0.2173	0.2873	0.3572	0.4272	0.4971	0.5671	0.637	0.7069
75000	0.0065	0.0766	0.1466	0.2167	0.2875	0.3567	0.4276	0.4972	0.568	0.637	0.707
80000	0.0061	0.0762	0.1463	0.2164	0.2864	0.3565	0.4266	0.4967	0.5667	0.637	0.7068
85000	0.006	0.0761	0.1462	0.2163	0.2863	0.3565	0.4265	0.4967	0.5667	0.6368	0.7068
90000	0.0059	0.076	0.1461	0.2162	0.2863	0.3564	0.4265	0.4966	0.5667	0.6368	0.7066
95000	0.0058	0.0759	0.146	0.2161	0.2862	0.3564	0.4265	0.4966	0.5667	0.6367	0.7065

Table D.4: Optimization values for fixed cache capacity model

The first value in Table D.5 represents cache location and the second value represents number of caches at that location. For example 1-1, represents that 1 cache is placed at node 1.

Cost vs. Q	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5000	1-1,	1-1,	1-1,	1-1,	1-1,	1-1,	1-1,	1-1,	1-1,	1-1,	1-1,
10000	3-1,1-1	3-1,1-1	3-1,1-1	3-1,1-1	3-1,1-1	3-1,1-1	3-1,1-1	3-1,1-1	3-1,1-1	3-1,1-1	1-1, 1-3
15000	1-1,5-2	1-1,5-2	1-1,5-2	1-1,5-2	1-1,5-2	1-1,5-2	1-1,5-2	1-1,5-2	1-1,5-2	1-1,5-2	1-1,5-2
20000	1-1,3-1,5-2	1-1,3-1,5-2	1-1,3-1,5-2	1-1,3-1,5-2	1-1,3-1,5-2	1-1,3-1,5-2	1-1,3-1,5-2	1-1,3-1,5-2	1-1,3-1,5-2	1-1,3-1,5-2	1-1,3-1,5-2
25000	1-1,5-2,6-2	1-1,5-2,6-2	1-1,5-2,6-2	1-1,5-2,6-2	1-1,5-2,6-2	1-1,5-2,6-2	1-1,5-2,6-2	1-1,5-2,6-2	1-1,5-2,6-2	1-1,5-2,6-2	1-1,5-2,6-2
30000	1-1,3-1,5-2,6-2	1-1,3-1,5-2,6-2	1-1,3-1,5-2,6-2	1-1,3-1,5-2,6-2	1-1,3-1,5-2,6-2	1-1,3-1,5-2,6-2	1-1,3-1,5-2,6-2	1-1,3-1,5-2,6-2	1-1,3-1,5-2,6-2	1-1,3-1,5-2,6-2	1-1,3-1,5-2,6-2
35000	9-7,	9-7,	9-7,	9-7,	9-7,	9-7,	9-7,	9-7,	9-7,	9-7,	9-7,
40000	9-7,1-1	9-7,1-1	9-7,1-1	9-7,1-1	9-7,1-1	9-7,1-1	9-7,1-1	9-7,1-1	9-7,1-1	9-7,1-1	9-7,1-1
45000	9-7,1-1,3-1	9-7,1-1,3-1	9-7,1-1,3-1	9-7,1-1,3-1	9-7,1-1,3-1	9-7,1-1,3-1	9-7,1-1,3-1	9-7,1-1,3-1	9-7,1-1,3-1	9-7,1-1,3-1	9-7,1-1,3-1
50000	9-7,1-1,2-1,3-1	9-7,1-1,2-1,3-1	9-7,1-1,2-1,3-1	9-7,1-1,2-1,3-1	9-7,1-1,2-1,3-1	9-7,1-1,2-1,3-1	9-7,1-1,2-1,3-1	9-7,1-1,2-1,3-1	9-7,1-1,2-1,3-1	9-7,1-1,2-1,3-1	9-7,1-1,2-1,3-1
55000	1-1,3-1,4-2,9-7	1-1,3-1,4-2,9-7	1-1,3-1,4-2,9-7	1-1,3-1,4-2,9-7	1-1,3-1,4-2,9-7	1-1,3-1,4-2,9-7	1-1,3-1,4-2,9-7	1-1,3-1,4-2,9-7	1-1,3-1,4-2,9-7	1-1,3-1,4-2,9-7	1-1,3-1,4-2,9-7
60000	8-12,	8-12,	8-12,	8-12,	8-12,	8-12,	8-12,	8-12,	8-12,	8-12,	8-12,
65000	8-13,	8-13,	8-13,	8-13,	8-13,	8-13,	8-13,	8-13,	8-13,	8-13,	8-13,
70000	8-14,	8-14,	8-14,	8-14,	8-14,	8-14,	8-14,	8-14,	8-14,	8-14,	8-14,
75000	8-15,	8-15,	8-15,	8-15,	8-15,	8-15,	8-15,	8-15,	8-15,	2-1,8-14	2-1,8-14
80000	8-16,	8-16,	8-16,	8-16,	8-16,	8-16,	8-16,	8-16,	8-16,	6-2, 8-14	5-2, 8-14
85000	8-17,	8-17,	8-17,	8-17,	8-17,	8-17,	8-17,	8-17,	8-17,	8-17,	2-1,3-1,5-2,8-13
90000	8-18,	8-18,	8-18,	8-18,	8-18,	8-18,	8-18,	8-18,	8-18,	8-18,	1-1,5-2,6-2, 8-13

95000	8-19,	8-19,	8-19,	8-19,	8-19,	8-19,	8-19,	8-19,	8-19,	8-16,5-3	1-1,5-2,6-2,8-14

Table D.5: Location of cache and number of caches for fixed cache capacity model

D.3 Simulation Output Values for Variable Cache Capacity Model

Cost/Q	1	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2
0	2.9103	2.9103	2.9103	2.9103	2.9103	2.9103	2.9103	2.9103	2.9103
5000	1.9729	2.2578	2.2578	2.51	2.9092	2.2578	2.2578	2.4341	2.51
10000	1.9236	1.5977	1.5977	1.5977	2.2842	2.0386	2.3139	1.5977	1.8306
15000	1.9111	2.0026	2.0586	1.5977	2.0586	2.0026	2.1328	1.8306	1.4863
20000	1.6762	1.7887	1.7887	1.7887	2.0564	2.0781	2.0291	1.8056	1.4863
25000	1.5082	1.6954	1.4673	1.3576	1.7398	1.4137	1.8306	1.8306	1.7887
30000	1.2282	0.8147	1.7131	2.0026	1.7652	2.0781	1.3644	1.7131	1.4137
35000	0.9424	1.0341	1.1816	1.7887	0.8019	0.9138	1.4881	2.0026	2.0781
40000	0.9424	0.7412	1.0566	1.5977	1.5082	0.7888	1.4863	2.0026	2.0026
45000	0.732	0.8235	0.9316	1.4863	1.7887	0.7412	1.7887	1.7398	1.0246
50000	0.6758	0.8235	0.9316	1.4137	1.4137	0.6392	1.7652	1.1505	1.0246
55000	0.6758	0.8235	0.8235	0.8248	1.0248	0.7888	0.7746	1.0515	1.0246
60000	0.8235	0.8235	1.0248	0.8235	0.8233	0.7888	0.8298	0.8235	0.8852
65000	0.8235	0.8849	0.955	0.8849	0.8235	0.7888	0.8235	0.7908	0.854
70000	0.8235	0.8848	0.83	0.83	0.8298	0.8233	1.0248	0.8235	0.8233
75000	0.8235	0.8233	0.8233	0.8298	0.8247	0.8233	1.0248	0.8298	0.6603
80000	0.8235	0.8233	0.8295	0.8297	0.8233	0.8714	0.8233	0.8298	0.7109
85000	0.8235	0.8233	0.8295	0.8247	0.8233	0.8233	0.8233	0.83	0.8233
90000	0.8235	0.83	0.8295	0.8233	0.8233	0.8233	0.83	0.83	0.8233
95000	0.8235	0.8233	0.8233	0.8233	0.8233	0.8233	0.83	0.8589	0.8233

Table D.6: Simulation values of fixed cache capacity model

D.4 Simulation Output Values for Fixed Cache Capacity Model

Budget	Time to transfer 1Mb
0	2.910316
5000	2.561475
10000	2.385829
15000	2.257801
20000	2.078146

25000	1.713112
30000	1.467267
35000	1.27811
40000	1.265139
45000	1.088095
50000	1.023571
55000	1.011431
60000	0.830027
65000	0.830027
70000	0.830027
75000	0.830027
80000	0.830027
85000	0.830027
90000	0.830027
95000	0.830027

Table D.7: Simulation values for fixed cache capacity model

References

[ADEB02] A. Adebowale Oke. Design and implementation of Scalable Web Caching Infrastructure, Sasktel, Research proposal for TRLabs, 2002.

[ALBO99] D.H. Albonesi. Selective cache ways: On-demand cache resource allocation. Proceedings of the 32nd International Symposium on Microarchitecture, November 1999.

[ALME98] C., Almeida V., Meira W., Analyzing Performance of Partitioned caches for the WWW, Proceedings of the Third International Caching Workshop, Manchester, England, June 1998

[AVEL03] P. Avella, M. Boccia, R. Canonico, D. Emma, A. Sforza, and G. Ventre. Web cache location and network design in VPNS. http://www.grid.unina.it/projects/firb/PubblicazioniURCININA/23URCININA.pdf.

[BEST96] A. Bestavros. Speculative data dissemination and service to reduce server load, network traffic and service time for distributed information systems. In Proceedings of ICDE'96: The 1996 International Conference on Data Engineering, New Orleans, Louisiana, March 1996.

[BRES99] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In Proceedings of the IEEE Infocom '99, Boston, USA, March 21–25 1999.

[CARP] Cache Array Routing Protocol (CARP) and Microsoft Proxy Server 2.0.

http://www.microsoft.com/isaserver/evaluation/previousversions/wp_CARP.asp. Microsoft MSDN

[CCC] Cache Coherency Checks. http://www.cse.iitb.ac.in/~gracias/webcaching/html/node38.html

[CACHED] Cache Digest. http://www.squid-cache.org/CacheDigest/

[CAI97] J. Cai, K.-L. Tan, and B. C. Ooi. On incremental cache coherency schemes in mobile computing environments. In Proceedings of the 13th International Conference on Data Engineering (ICDE'97), pages 114--123, Birmingham, UK, October 1997.

[CAO98] P. Cao, J. Zhang, and K. Beach. Active Cache: Caching Dynamic Contents (Objects) on the Web. In Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98), The Lake District, England, September 1998.

[CHAN96] A. Chankhunthod, P. B. Danzig, C. Neerdaels, Schwartz M. F., and K. J. Worrell. A hierarchical internet objects cache. In In the Proceedings of the 1996 USENIX Technical Conference, San Diego, CA, January 1996.

[CUNH97] C. Cunha. Trace analysis and its applications to performance enhancements of distributed information systems. Boston Univ., Boston, MA, 1997.

[CZYZ97] J. Czyzyk, J. H. Owen and S. J. Wright. Optimization on Internet. OR/MS 1997.

[DANZ93] P. B. Danzig, R. S. Hall, and M. F. Schwartz. A case for caching file objects inside internetworks. In SIGCOMM, pages 239–248, 1993.

[DAVI01] B.D. Davison, Assertion: Prefetching with GET is Not Good, Proc. Sixth Int'l Workshop Web Caching and Content Distribution, Elsevier, Boston, June 2001. http://citeseer.nj.nec.com/davison01assertion.html.

[DAVISO] B D. Davison. Web Caching and Content Delivery Resources. http://www.web-caching.com/

[DIAMO1] J. Diamond. Web caching infrastructure planning at Sasktel. TRLabs, Winnipeg, Canada.

[DIAMO2] J. Diamond and M. Maheswaran. Optimization models for documentation placement and load balancing on content distributed networks. TRLabs, Winnipeg, Canada.

[DYNAMI] Dynamic Caching. http://www.foedero.com/dynamicCaching.html

[ERCE01] O. Ercetin and L. Tassiulas. Optimal Cache Allocation Policies in Competitive Content Distribution Networks Global Communication Systems. Number: TR 2001-4, 2001.

[FAN00] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area Web cache sharing protocol. IEEE/ACM Transactions on Networking, 8(3):281–293, 2000.

[FERR98] M. C. Ferris, M. Mesnier, and J. J. Mor' e, NEOS and Condor: Solving optimization problems over the Internet, tech. rep., MCS Division, Argonne National Laboratory, Argonne, Ill., 1998.

[FOUR02] R. Fourere, D. M. Gay and B.W. Kernighan. AMPL – A Modeling Language for Mathematical Programming. Second Edition, Duxbury Press / Brooks / Cole Publishing Company, 2002.

[GHOS04] D. Ghosh. Recent Advances in Web Caching Technologies – A Tutorial. International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'04). San Jose, 2004.

[GRACIA] A. A. Gracias. Web Caching Tutorial. IIT Bombay. http://www.cse.iitb.ac.in/~gracias/webcaching/html/node2.html.

[HEDD97] A. Heddaya and S. Mirdad. Webwave: Globally load balanced fully distributed caching of hot published documents. In International Conference on Distributed Computing Systems, pages 0–, 1997.

[INRC] Internet Caching Resource Center. http://www.caching.com

[KARG97] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin and R. Panigrahy, Consistent hashing and random trees: Distributed cachine protocols for relieving hot spots on the World Wide Web, In Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, pages 654-663 , 1997.

[KOSK03] T. Koskela, J. Heikkonen, and K. Kaski. Computer Networks: The International Journal of Computer and Telecommunications Networking Volume 43, Pages: 805 - 817, Year of Publication: 2003 ISSN: 1389-1286.

[KOSKEL] T. Koskela, J. Heikkonen, and K. Kaski. Web Cache Optimization with Nonlinear Model Using Object Features. Helsinki University of

Technology, Laboratory of Computational Engineering, FIN-02015 HUT, Finland.

[KOZUCH] M. Kozuch, A. Wolf, and A. Wolfe, "Network Caching Resource Allocation for Multimedia Objects ", Princeton University.

[KRIS99] B. Krishnamurthy and C. Wills, Proxy Cache Coherency and Replacement - Towards a More Complete Picture In Proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS), June 1999.

[KRIS98] P. Krishnan, D. Raz, and Y. Shavitt. Transparent en-route cache location in regular network topologies. In DIMACS Workshop on Robust Communication Networks: Interconnection and Survivability, 1998.

[KRIS00] P. Krishnan, D. Raz, and Y. Shavitt. The cache location problem. IEEE/ACM Trans. Netw., 8(5):568–582, 2000.

[KRI98] P. Krishnan and B. Sugla. Utility of Cooperating Web Proxy Caches in the Elsevier J. on Computer Networks and ISDN Systems 30, 1998, pp. 195-203. (This appears in the Proceedings of the 1998 World Wide Web Conference.)

[KROE97] T M. Kroeger, Darrell D. E. Long and Jeffrey C. Mogul. Exploring the Bounds of Web Latency Reduction from Caching and Prefetching. Proceedings of the USENIX Symposium on Internet Technologies and Systems Monterey, California, December 1997.

[LEYF99] S. Leyffer. User manual for MINLP BB. University of Dundee. March 1999.

[LI04] Li, K. and Shen H. Optimal placement of Web proxies for tree networks e-Technology, e-Commerce and e-Service, 2004. EEE '04. 2004 IEEE International Conference on, 28-31 March 2004 Pages:479 - 486

[LIED96] J. Liedtk. Caches versus Object Allocation. Proceedings of 5th International Workshop on Object-Orientation in Operating Systems ({IWOOOS}), IEEE Computer Society, Washington, DC, pp- 95—101,1996.

[LIU03] Y. E. Liu. Computer Network - Course notes. University of Manitoba, fall 2003. www.cs.umanitoba.ca/cs781.

[MALP95] R. Malpani, J. Lorch and D. Berger - Making World Wide Web Caching Servers Cooperate - Proceedings of the 4th International World-Wide Web Conference, pp.107-117-- 1995.

[NEISSE] G. Neisser & J. Heaton. Co-operative Caching, Manchester Computing, University of Manchester. http://www.g-ming.net.uk/Documents/JENC8/

[NEOS] Network Enabled Optimization System. http://www-neos.mcs.anl.gov/

[NOTTIN] M. Nottingham. Caching Tutorial for Web Authors and Webmasters. http://www.mnot.net/cache_docs/

[NS2] Network Simulator. http://www.isi.edu/nsnam/ns/

[PROV97] D. Provey, John Harrison, A Distributed Internet Cache, Proceedings of the 20th , Australian Computer Science Conference, Sydney, Australia, Feb. 5-7 1997.

[QIU02] L. Qiu, V. Padmanaban, G. M Voelker, "On the Placement of Web Server
Replicas", Proc. IEEE INFOCOMM 2001.

[RODR99] P. Rodriguez, C. Spanner and E. W. Biersack. Web Caching Architectures: Hierarchical and Distributed Caching. The International Web Caching Workshop, 1999.

[SAIL03] F. Sialer and J. Diamond. Caching Network Simulator. Trlabs, Winnipeg.

[SASKT1] SaskTel, Sasktel Network Diagrams: IP Core Nat/DHCP/LANSPAN_IP/STATIC/IP SCHEME, 2002.

[SASKT2] SaskTel, Weekly and Monthly Link Utilization Graphs, 2002.

[SIMU02] Simulation Performance Models, FrontRunner Computer Performance C Consulting 2002. www.frontrunnercpc.com/info/simulation.htm.

[TAMI96] A. Tamir. An O(pn2) algorithm for the p-median and related problems on tree graphs. Oper. Res. Lett. 19 (1996), 59-94.

[TAWSM] The Accelerated Web Server Model. http://developer.novell.com/research/appnotes/1997/october/01/03.htm

[TIMO03] V. Timonen. Simulation Studies on Performance of Balanced Fairness, Master Thesis, Helsinki University of Technology, Oct 03.

[WANG02] B.Wang, S.Sen, M.Adler and D.Towsley. Optimal Proxy Cache Allocation for Efficient Streaming Media Distribution, to appear in Proceedings of IEEE Infocorn, 2002.

[WANG99] J. Wang. A survey of Web caching schemes for the Internet. ACM Computer Communication Review, 25(9):36–46, 1999.

[WINS93] W. L. Winston. Operation Research: Application and Algorithm. International Thomson Publication, 3rd edition 1993.

[WCCP] WCCP .Cisco Systems. http://www.cisco.com/warp/public/732/Tech/switching/wccp/

[WCP] Web Cache Protocols. http://www.utdallas.edu/~ilyen/course/web/cache.PDF

[WCRPS] Web Caching-Related Protocols and Standards. http://www.web-cache.com/Writings/protocols-standards.html

[WIER02] A. Wierzbicki. Models for internet cache location. 7th International Workshop on Web Content Caching and Distribution, http://2002.iwcw.org/papers/18500049.pdf.

[WILL96] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox. Removal policies in network caches for World-Wide Web documents. In Proceedings of the ACM SIGCOMM '96 Conference, Stanford University, CA, 1.

[WILL99] C. E. Wills and M. Mikhailov. Examining the cacheability of user-requested web resources. In Proceedings of the 4th International Web Caching Workshop, San Diego, CA, March/April 1999.

[WOOS97] R.P. Wooster, M.Abrams. Proxy caching that estimates page load delays In Proc. Computer Networks and ISDN Systems, number 29, pp. 977 - 986, 1997. 20

[XUEY03] Xueyan Tang and S. T. Chanson. "Coordinated management of cascaded caches for efficient content distribution". Data Engineering, 2003. Proceedings. 19th International Conference on , 5-8 March 2003.

[YEUN01] K. Yeung Wong, K. Hau Yeung, "Site Based Approach to Web Caching Design", IEEE Internet computing online, September/October 2001 (Vol. 5, No. 5).

[ZHAN97] L. Zhang, S. Floyd, and V. Jacobson. Adaptive web caching, NLANR Web Cache Workshop, June 1997, Website: http://www-nrg.ee.lbl.gov/floyd.

[ZHAN98] L. Zhang, S. Michel, S. Floyd, V. Jacobson, K. Nguyen, and A. Rosenstein. Adaptive Web Caching: Towards a New Global Caching Architecture. In Proceedings of the Third International Caching Workshop, June 1998.