SELECTIVE CACHE DIGEST EXCHANGE MECHANISM FOR GROUP BASED CONTENT DELIVERY NETWORKS

A Thesis by

Sai Srivastava Tumuluri

Bachelor of Technology, TKRCET, JNTU, India 2010

Submitted to the Department of Electrical Engineering and Computer Science
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Master of Science

July 2013

SELECTIVE CACHE DIGEST EXCHANGE MECHANISM FOR GROUP BASED CONTENT DELIVERY NETWORKS

The following faculty members have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science with a major in Electrical Engineering.

_____
Ravi Pendse, Committee Chair

_____
Krishna Krishnan, Committee Member

_____
Abu Asaduzzaman, Committee Member

DEDICATION

To my family, my master, my teachers and all artists in the world

Time is the scarcest resource and unless it is managed nothing else can be managed.

- Peter Drucker

ACKNOWLEDGEMENTS

I would like to wholeheartedly thank my advisor and professor Dr. Ravi Pendse for his invaluable insight, guidance and constant encouragement throughout the thesis and Master's study at Wichita State University. I would also like to extend my appreciation to Mr. Amarnath Jasti for his guidance, advice and time during the thesis. I thank my committee members Dr. Krishna Krishnan and Dr. Abu Asaduzzaman for their precious time and suggestions. I am always grateful to my brother Ravi Teja who showed me the importance of hard work. Finally, I am very much obliged to family and friends for their constant encouragement and moral support during my education.

ABSTRACT

Content Delivery Networks (CDNs) are one of the most important techniques used for WAN optimization. CDN service providers install a cache server/distribution server close to the end users of the original data. The data from the original server of the CDN client is taken and pushed into the network of distribution servers, to make sure end-user is served from the nearest distribution server. Many algorithms were proposed for coordination between distribution servers, to exchange the information regarding content between distribution servers in the CDN environment with minimal error or minimal usage of resources. Different coordinating algorithms that currently exist include Internet Cache Protocol (ICP), cache digest algorithms and more. Most of these algorithms lacked an important factor of providing certainty of the information shared during process. Due to this issue, most algorithms use if-modified packets to validate information, sending them whenever a user requests CDN service. In this research work, a certainty factor was introduced and its effect on the environment is shown. Simulation was done to show the effect of using the certainty factor; simulation was also done with an increasing number of servers in the group, to show the scalability of the approach.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In today's world, the Internet has become a very important part of daily life and exists as one of the necessities in the modern era for communication, commerce and education. Upon its conception, the Internet found advantages in scalability that allows it to evolve for use with a large variety of devices in various areas. This gave rise to a lot of features and possibilities. However, this advancement in technology also posed the challenge of maintaining the Internet[1]. The increasing complexity of the Internet[1, 2] both in architecture and in amount of different types of rich content exchanged in the Internet, creates challenges like maintaining service quality for content providers [2, 3].

The ever-increasing availability of the Internet to most of the world's population even at remote places of the globe, has led the development of the Internet to concentrate on distributing the content, and making it available to all the users at all times. One of the solutions proposed to increase the accessibility, bandwidth availability and distribution of content to end-users is Content Delivery Networks (CDNs)[2].

Content Delivery Networks (CDNs) are distributed networks of interconnected servers that deliver content in reliable and fast way[2]. The main aim of CDN is to reduce the delay in serving end-user's requests. In order to achieve this, CDN service providers focus on redistributing content to the closest CDN cache server (known as proximity server) and routing the client's request to the proximity server [4].

Another issue addressed by the CDN technology is single point of failure in centralized content models of service providers. The centralized model has the

disadvantages of scalability and adaptability. The single source of content becomes a single-point when that source can't be reached in time. Distributing the content over the CDN creates multiple sources available for content serving local clients[4].

To clearly understand importance of CDN, consider the basic architecture difference involved. Figure 1, shows the content sharing without CDN, involving direct communication between the content provider's server and the end-clients.

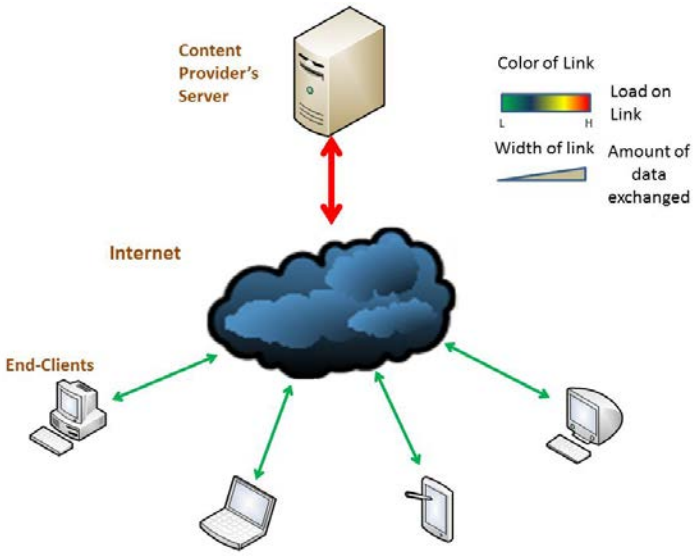

Figure 1: Simple Content Exchange without CDN

The uplink of the server is under constant load due to the large number of requests and becomes high risk failure. Failure of this link will lead to single point of failure. The server can easily become a victim of attacks (like Distributed Denial of Server (DDoS) attacks), causing interruption of content delivery to all end nodes. Figure 2, shows an example of content sharing in presence of a CDN.

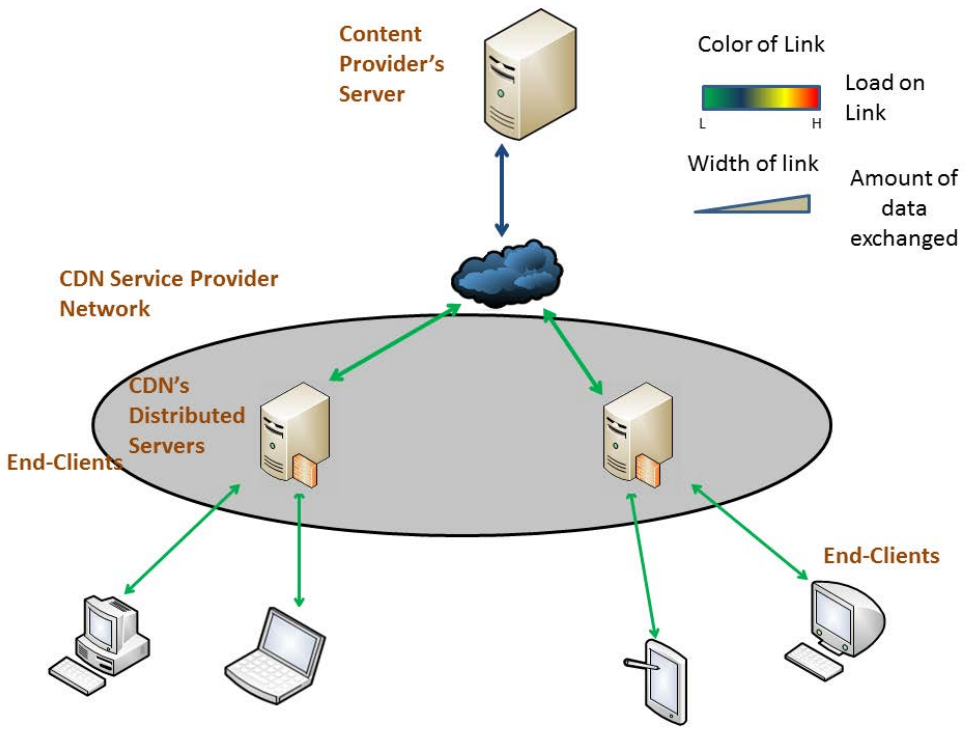

Figure 2: Content Sharing in presence of CDN

In the presence of CDN, the files from the content provider's server are moved to the CDN provider's distribution servers and the client will request from the nearest distribution server. From content provider prospective, the load is now distributed among the CDN's servers, and all of the end-client's requests will be served by the CDN. Now, the uplink of the content provider's server is no longer the single link of failure, as CDN acts in middle. From the end-client prospective, request packets need to travel less across the network enabling faster service. This example illustrates that the implementation of a CDN provides advantages both from a content provider prospective and end-user prospective.

1.1. Motivation

Development of CDNs began in late 90's, with continued development resulting in various improvements in various aspects of CDNs [2].Content distribution in CDNs consists of various mechanisms and concepts, such as positioning of distribution servers from client environment, content request routing mechanisms, content re-distribution mechanisms, content update mechanisms, and replica selection mechanisms. Each type of mechanism constitutes an individual field of research in CDNs.

This thesis work is motivated by works like Internet Cache Protocols (ICPs)[5], cache-digest mechanism[6] and hybrid architectures[7-9]. Research in these CDN techniques works to reduce either service delay or content delivery time. The research in this thesis uses cache sharing mechanisms, like cache digest, with hierarchical architectures[8] to continue to improve upon CDN technology.

Chapter 2

BACKGROUND

This chapter briefly covers various CDN key concepts and components necessary to introduce to CDN operation. The chapter also explains different basic CDN architectures necessary to understand the advantages of CDN architecture used in proposed technique. Final chapters sheds lights on research done in the world of CDN that inspired the thesis work.

2.1 Content Delivery Networks

CDNs work as shock absorbers for the service providers, absorbing any network shock caused by spontaneously heavy data loads generated by bursty traffic[2]. Various components and different techniques are used to create CDNs. This section describes different components, keywords, and key-concepts integral to understanding CDN architecture and its functions.

2.1.1 Content Provider

The Content provider is the customer for the CDN service provider, whose content is to be distributed and delivered to end-users/clients [2, 3]. The content provider makes Service Level Agreements(SLAs) with the CDN service provider, so that the end-users/clients can be assured high Quality of Experience (QoE)[3]. Typical content providers include online entertainment providers, large enterprises, e-commerce sites and educational organizations that provide interactive way of learning[2].

2.1.2 CDN Service Provider

CDN service provider is the company that maintains the CDN infrastructure allowing the delivery of content from content provider to clients with minimum delay[2]. CDNs provide an advantage over non-CDNs by allowing redistribution of content from different service providers using the same distributed network of servers.. This advantage reduces the cost of setting up a redistribution network for each company/content provider[3]. CDNs are located between the content provider and client

2.1.3 Client

End-users or clients access the information/content from the content provider [2]. Clients may have their own proxy server between the actual users and CDN servers. It is anticipated that in future, the CDN service provider will make QoS agreements with clients and charge the clients as per usage[2].

2.1.4 Servers

A typical CDN environment has two types of servers' origin server and cache server[2].

2.1.4.a Origin Server

An origin server is a content provider server where original content is located. Content from origin server is distributed across the CDN distribution network and is replicated close to end-user. During the process of serving the client request, content is provided from the CDN server. If the source selection mechanisms could not find an optimum replica server/cache server [2] among CDN servers, origin server will be last source to obtain file. Optimum condition varies among algorithms, but generally defined

as when retrieval time from the replica server falls under retrieval time from the original server [2].

2.1.4.b Cache Server

Cache servers are the servers deployed close to end-users by CDN providers to place the content of interest close to the end-user environment. Cache servers are also called edge servers or surrogate servers[2]. A web cluster is typically used for representing all edge servers of a CDN[2].

2.1.5 Content Distribution Algorithms

For CDNs to maintain SLA conditions, they must distribute desired content inside the network, such that the content exists on a server located close to the client [2]. To accomplish this, CDN service providers use algorithms called, content distribution algorithms. These algorithms based on different parameters like demand, file type, geo-location, and company policies help to distribute the content in fast and reliable manner[2].

2.1.6 Source Selection Algorithms

User-requested files not found in local cache require distribution servers to retrieve them. This retrieving process is done using source selection algorithms, which selects the source based on factors which include but not limited to source availability, congestion, and RTT. Source selection algorithms allow faster file retrieval and maintain load balancing. Few selection algorithms allow selection of more than one source for load balancing.

2.1.7 Request Routing Algorithms

Once the source selection algorithm chooses a best source server know as proximity server, request routing algorithms are used to route clients requests to proximity server in fashion to minimize the latency in retrieving the content[2].

2.1.8 Accounting in CDN Environment

CDN service providers charge content providers based on content access and usage of surrogate servers across the network. Accounting can be maintained by the CDN service provider or a third-party organization[2]. The accounting component of the CDN service maintains logs of user resource utilization during the process of content delivery. These logs include information regarding querying, distributing and content delivery with each depend upon factors like bandwidth usage, changes in traffic characteristics, size of content, number of edge servers serving the content provider, and more.

2.1.9 Existing CDN Architectures

CDNs can be broadly classified into commercial CDNs and academic CDNs. Commercial CDNs are commercially available CDN service providers in the market. They usually invest extensively, placing a large number of surrogate servers close to clients. For example Akamai a leading CDN service provider has more than twenty thousand servers distributed over globe[10]. Commercial CDNs basically use client-server models [2, 10] and tend to move towards peer-peer model used by academic CDNs [2].

Academic CDNs are maintained by volunteer server providers [1, 2, 10]. They form peer-to-peer distributed network of servers to share the content. Academic CDNs have

advantages in scalability and adaptability to sudden bursty traffic and failures. As the academic CDNs are maintained by volunteers, the number of distributed servers and QoS guaranteed depends on the number of volunteers and their commitment [1, 2, 10].

Commercial CDNs are maintained by individual companies, they face issues like scalability because of high costs associated with providing an increasingly-sized company with appropriate resources [1-4, 10] to maintain a commercial CDN. Commercial CDNs face other disadvantages, including one commonly called ‘”islands of CDN”. Content providers typically make contracts with a single CDN service provider due to reasons like high cost. This can result in the formation of isolated CDN islands belonging to individual CDN service providers. To overcome this disadvantage, CDN service providers tend to collaborate with each other to maintain the SLA agreements guaranteed to content providers by extending the reachability towards clients.

2.2 LITERATURE SURVEY

Commercial CDNs generally have a client-server architecture, whereas the academic CDNs have peer-to-peer architectures. Each architecture has its own set of pros and cons. Therefore, a new hybrid architecture was used [2, 3, 7-10] to obtain the advantages of both architectures. Considering the advantages of a hybrid architecture, researches [7, 8], proposed to enhance the working of CDNs. The group-based hybrid architecture includes the formation of groups on servers with hybrid architecture.

The research in [7], concentrates on developing an automated group forming mechanism. Automated group formation does not create a large amount of traffic and does not affect the scalability feature of group based CDNs. As scalability is not affected,

this research work used hybrid group-based architecture as the base architecture for research.

Content request routing mechanisms aim to route requests to a proximity server. Edge servers discover the proximity server using different methods, like sending a broadcast request to neighbors, using source selection strategies like in [9] or using queries like Internet Cache Protocol (ICP) [6]. ICP uses a query technique to discover the presence of content on neighboring servers. Although this technique works well to get a good server hit rate, ICP still faces the challenge of delay occurred due to querying[6]. To overcome this challenge Cache Digest was proposed in [6].

Cache digest is an exchange mechanism [6] for exchanging digest/summary of the cache of CDN surrogates with their neighbors. It is implemented in Squid proxy servers. Squid proxy servers are HTTP based proxy servers. Squid software is an open software developed by individuals [11]. The Cache Digest technique uses bloom filter, a database representation technique for representing contents stored in cache. The digest summary represented by the bloom filter is exchanged between servers. This will eliminate the need for query messages and reduces delay. The bloom filter has disadvantage of false miss, which is paid in return to the space advantage provided [12-15].

In [8] the author proposed a content routing algorithm, to route the content between groups and proposed a semi-hashing algorithm to share summary of cache inside the group. The semi-hashing algorithm only works inside a group, so the author proposed a work around of content request routing mechanism to route the content request between groups. In general, for inter-group communication, cache sharing algorithms, like pure-hashing, semi-hashing and cache-digest fail due to large summary size[8]. Work to

reduce the cache-summary size will allow the cache digest exchange mechanism to be used for inter-group sharing or sharing cache between high cache capacity servers.

Considering the semi-hashing algorithm proposed in [8], further work was done by Björkqvist and others in [9]. Authors segregated the content cache on local cache into three groups, gold, silver and bronze based on the content popularity, cache capacity, and, other factors. However, they did not use any Internet Cache Protocols (ICPs). They used LRU (Least Recently Used), cache algorithm to cache the files on local cache and proposed an adaptive caching using LRU properties.

User pattern is one of the important aspects in CDN environment used in selecting appropriate caching technique. User request rate is considered to follow Poisson like distribution [9, 16]. Prediction of successive values in Poisson distribution is done in [17]. Research in this thesis also considers the user request rate follows a Poisson like distribution and a technique is proposed to solve following general limitations of cache digest exchange mechanisms given in [6].

The following are some of the limitations addressed by the technique proposed in this thesis work to improve cache digest. First, in [6] the idea of expiration time or certainty of cache digest is not addressed clearly, the information is updated after fixed amount of time irrespective of changes at source server. In the proposed technique, a precise procedure to give an approximate value for validity of information known as Time To Live (TTL) is explained. TTL represents the time for which digest is certain to be valid. TTL is send along with cache digest update information to neighboring servers. Second, the TTL value, is also used to solve the question of, how frequently cache summary need to be

exchanged. The TTL received from the source server is used as timer and an update request is sent when timer falls below a threshold value.

Third, preparing a digest of all files in the cache is processor intensive[6]. In the current thesis work, using TTL value, a digest of only a few files of interest can be prepared. This helps in reducing burden on the CPU and the size of digest updates. Chapter 3 explains the technique in detail.

CHAPTER 3

PROPOSED TECHNIQUE

Mechanisms that utilize sharing of cache information aim to reduce the delay caused by ‘replica selection mechanisms’. One such mechanism, called cache-digest, shares the complete summary of all cache files with all neighbors. Digest sharing mechanisms use different data representation techniques to share a complete hash based digest. This thesis work focuses on proposing a new technique for sharing a digest of selected files, which is an improvement on legacy mechanisms that shares digest of all files in cache. By choosing only the content whose information needs to be updated, this proposed technique primarily aims to reduce false hits caused by update mechanisms.

Developments proposed in this thesis are motivated by previous work done on digest sharing and developments in increasing scalability of digest sharing techniques in CDNs. This current thesis work tries to exploit the advantages of existing CDN hybrid architecture and existing analysis on user access patterns. The proposed enhanced cache exchange mechanism aims to address the issue of false hits with the cache digest mechanism by providing certainty to the information in the digest.

3.1. Group-based CDN Architecture

The architecture of a CDN is composed of the Content Provider's origin Server (CPS), Control Surrogate Server (CS), Distributed Surrogate Server (DS), and Clients (C). Group based architecture [7, 8], have advantages of grouping, hierarchical, and peer-to-peer structures together. The architecture consists of two logical layers of mesh topologies, which are logically interlinked at CS. The upper layer consists of control servers (Control Surrogate Servers) and the lower layer consist of distributed servers (Distributed Surrogate Server) which are close to client's location. Figure 3 shows the default architecture of hybrid CDN.

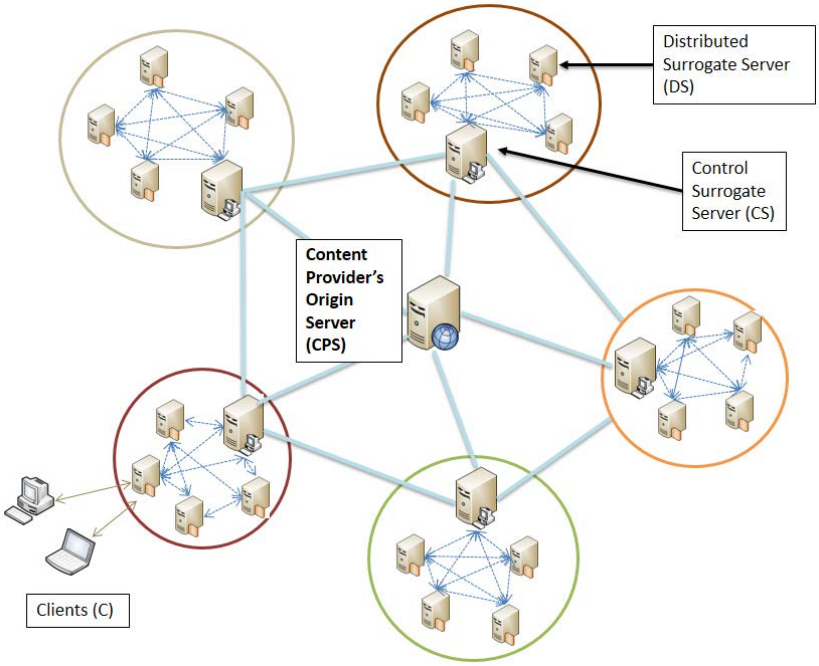

Figure 3: Architecture of group based CDN Network

Communication in the architecture happens at the following places.

- Source Provider links (CPS – CS)
- Control links (CS –CS)
- Group Links (CS –DS and DS-DS)
- Edge Links (DS - C)

Distributed Servers (DSs) are edge servers in the CDN environment, and are close to client's environment. Considering the geographical distance and communication latency between DSs (excluding the highly variable constraints like temporary congestion), DSs are grouped or clustered together either by manual configuration or self-organizing scheme [7, 8]. One server from the group is made the Control Server (CS) either by manual assignment or election based scheme [7, 8], which will depend on many factors like server's memory capacity, processing scheme, and its RTT time with the CS of other group.

CSs of different groups form a peer to peer network and are also connected to the Content Provider's origin Server (CPS). This peer-to-peer connectivity helps with fast content and content request distribution. Communication latency between two devices across a WAN equals the sum of propagation delay, processing latency at edge nodes, processing latency at each hop and congestion across the path. Out of the stated latencies, processing latency is ignored due to ignorable contribution to overall latency when compared to others. RTT (Round Trip Time) represents the total latency, the packet has experienced during network travel. RTT can be viewed as sum of propagation latency and congestion latency.

The following list outlines the suggested conditions for a group based CDN architecture, under which groups of peer-to-peer connected servers' are interlinked for fast request routing:

1. The maximum of RTT on a group link is comparatively less than half the value of RTT between the DS and the CPS.
2. The RTT control link(s) between two groups is minimum, when compared to, RTT on group links in both the groups.
3. The RTT between the CS and the CPS is less than any other RTT values between the DS and the CPS.

This research work assumes that each DS will cache the popular objects in its own client environment as per local popularity level and request rate pattern. In later sections, detail explanation is provide on how local caching is used to reduce the amount of update information exchanged in environment. Each DS will exchange a summary of cached objects with other DSs in the group and also the CS of the group. The overall content retrieval process can be summarized as follows:

1. Try to satisfy the client request using local cache of the DS.
2. If Step 1 fails, try to satisfy using DSs of the group.
3. If Step 2 fails, try to satisfy using adjacent groups.
4. If Step 3 fails, try to retrieve the file from the origin server.

In the original architecture (as per [8]), cache digest is exchanged on group links. If the original DS confirms that the file of interest is not present on neighbor DS cache, inter-group content request routing is done on control links. If inter-cluster request routing fails the file is retrieved from origin server. It is very obvious that typical cache summary

cannot be used for sharing inter-group content information, due to high possibility of large file size. The proposed algorithm tries to reduce the update size, so that the cache digest mechanism can be used for inter-group content summary sharing. Figure 4 shows view of communication in the architecture. In Figure 4, the dotted lines indicate the links on which digest is exchanged between servers. The solid lines indicate the links on which no digest being shared.

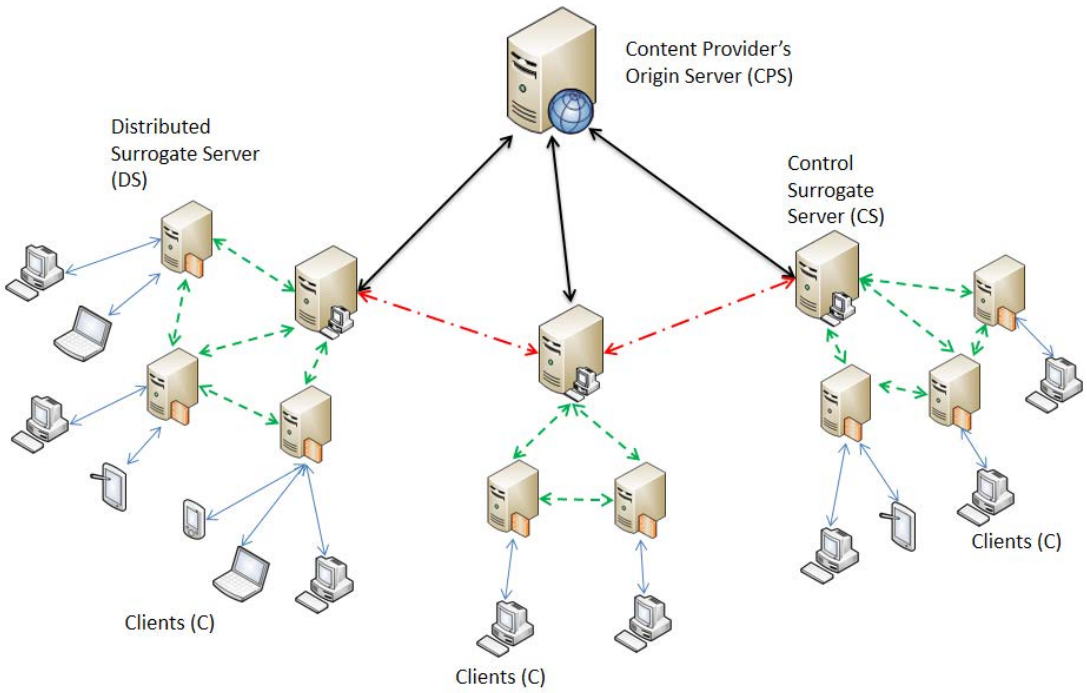

Figure 4: View of Communication Architecture with proposed technique

3.2. Local Content Caching on DS

Each DS maintains a local cache that contains files cached based on local popularity, which is determined by the number of requests obtained for each specific

file[9]. This approach helps each DS maintain a cache more specific to the clients being served in the local environment, thereby increasing the local hit rate.

The number of files to be cached is given by a threshold value depending on the local popularity. Most of the object caching algorithms based on popularity uses Zipf's law, where the popularity of the content follows a Zipf like distribution [2, 9, 18]. The threshold can be calculated from the value of 'k' in [18], so that each DS can make sure to have a pre-defined hit ratio of 'h.

3.3. Intra-Group Communication

Each DS in a group will share their cache summary with other DSs and CS of the group, as in legacy cache digest [2, 8]. As per the proposed exchange mechanism, when a new DS joins the group for the first time, the local cache of the DS will be empty. All the other DSs in the group will share the complete summary of their cache to the new DS for the first time. Summary from each DS will give information regarding the files that the DS currently has cached along with TTL (Time to Live) for each file. This TTL value represents a measure of time for which the file is expected to exist on the source server. If an existing DS (from another group) is added to the group, it will also share the complete summary of its cache with all the other DSs in the group for first time.

In this thesis work, a DS which sends its cache summary is called a source DS and a DS which requests or receives a cache summary is called a target DS. After every source DS shares its complete summary for the first time, the new target DS in the group will compare the summaries received with its local cache summary, and prepare a summary table (similar to a routing table on a router). The summary table will contain the

file ID and TTLs for all DSs in the group that currently have that file in their cache. Table 1 shows an example summary table, where C_1, C_2 .. Cn represents content/files and m represents the TTL_1, TTL_2 … TTLm represents TTL from neighboring DS1, DS2 and so on in the group

TABLE 1: Example View of Cache Information at Target Server

Content ID	TTL_1	TTL_2	TTL_3	TTL_4	...	TTL_m
C_1	t_1	---	t_3	t_4	…	tm
C_2	---	-----	t3	---	…	---
…	…	…	…	…	…	…
C_n	----	t2	----	t4	…	tm

DSs use the pull based approach to obtain delta updates. When the target DS identifies the information in summary table as slate, it sends an 'if-modified' request to the source, the source DS. The source DS will respond with the updated TTL value and any unexpected deletions from the cache (represented by negative TTL). Optionally, if a content was added to the cache very recently (just before an update request is received), the new content can be neglected in the update summary to suppress the effect of sudden spikes in the popularity of the content at the cost of losing source information for target DS.

3.4. Variable TTL

TTL value is the maximum time for which the receiving server can assume the cache summary is valid. TTL can be manually set by the administrator, based on content

type, trustworthiness of the source DS and other factors as in [19]. In the proposed technique, a variable TTL value is used for the cached files and this value is set by the source DS. Instead of having the constant TTL value for the complete digest shared, a unique TTL value is assigned per content. This value is calculated by the source DS, based on the user access pattern in its environment. This helps to determine a better approximation for the TTL of the content.

This thesis research considers three main assumptions. First, users retrieval requests in an environment of unique files, follows a Poisson like[9] distribution. Second, in [20], author shows the access pattern increase and decrease linearly along day, and same access pattern is considered in this research. Third, user request rate pattern is not bursty in nature. Figure 5, shows an example of access pattern for a file used for explaining the concept of variable TTL.

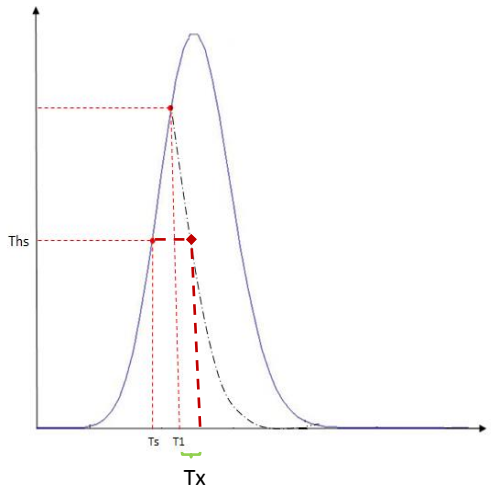

Figure 5: An example of access pattern for a file

In Figure 5, let 'T_S' be the time at which the content user request rate has reached the threshold 'Th_S', and at this point the local DS starts caching the content assuming DS has sufficient space and doesn't have to evict other file which has more popularity than

current file. Consider a cache digest update is send at time 'T_1'. Proposed technique considers, the content has reached its maximum popularity and assumes, request rate will decrease from current point of time. Considering this assumption, the content is estimated to be in the local cache for time (T_X), as per the current popularity of files in environment. So, the source DS will use (T_X) as the TTL for the content. This will inform the destination DS, which received the summary update, regarding the time for which a file is expected to remain cache with current environment. The value of TTL send by the source DS will change based on the current popularity situation in the source environment.

Let C_1 and C_2 be two contents/files cached at source DS and are not cached at target DS. Target DS is statically configured with constant TTTL say TTL_{const} to request updated cache summary from neighbors. Based on the user access pattern at source C1 and C2 are estimated to be in cache for time TTL_1 and TTL_2 respectively by source server. TTL_1 and TTL_2 are related to TTL_{const} in following way.

$$TTL_1 > TTL_{constant} \quad \text{eq(1)}$$

$$TTL_2 < TTL_{const} \quad \text{eq(2)}$$

For files like C_1, any update request from target server and any digest creation for the file at source DS is an extra usage of resources like processing and bandwidth. For files like C_2 the constant TTL_{const} at target DS will make target DS to hold slate information for time (TTL_{const} - TTL_2).

In the presence of proposed technique, for each file cached at source DS an estimated TTL value is calculated and sent along with update. As this TTL value is based on user access pattern for the file, it is close to TTL_1 and TTL_2. So, the target DS can only

request for files whose information becomes slate and source DS can prepare digest of only files of interest, which consumes less processing when compared to creation of digest for all files. Hence, the proposed technique uses variable TTL configured by source DS for reducing processing load, and bandwidth.

3.5. Cache Digest Sharing and Update Process

Each group in the architecture has a CS and DSs connected together. If a new group is formed all DSs share the cache summary with other DSs and CS of the group for the first time. If new DS is added to an existing group, existing DSs will share the complete cache summary to new DS added to the group as the first update.

All the receiving DSs will make their own cache summaries by analyzing the received cache summaries send by all other DSs. The updates in the group from here after will be only delta updates. Delta updates provide only a summary of changes in the cache and updated TTL. If a traditional push based is used, the source DS will send an update every fixed amount of time, but this method has a disadvantage. It is likely that the destination DS may or may not need a full summary[8]. If a traditional pull-based method is used, the DS will be using pre-configured TTL. A modified pull based approach is used instead, to take advantage of variable TTL in this research work.

A pull based approach is used when the DS requests for updated information from the source DS of the content. Consider the complete summary table stored on a target DS. Table 2 represents notations used in this thesis work.

TABLE 2: Notations Used in Thesis Work

Symbol	Representation

S	Total number of servers in a group
Ø	Total number of objects in a CDN environment
N_x ; $x \in [1, S]$; $0 \leq n_x \leq Ø$	Total number object that are cached on a 'xth' DS
C_y ; $y \in [1, O]$;	'yth' Content out of N_x
K_{Cy} ; $1 \leq K_{Cy} \leq S$	Number of source DS(s) in a group having content C_y
W_{Cy}; $w \in [1, S]$	Represent 'wth' source DS from of K_{Cy} number interested source DSs
RTT_z ; $z \in [1, S]$	RTT between target DS and a 'zth' source DS
TTL_{wcy}	Current Variable TTL for content 'C_y' from 'wth' source DS

A target DS will consider the summary information for a content C_y as slate, based on following condition:

$$max\left(TTL_{\mathrm{WC}_y}\right) \leq \max(RTT_z) + \Delta_{RTT} \quad \text{Eq (3)}$$

This condition exists to make sure that the summary is synchronized more accurately and has control on the number of times information is requested about a file. Meeting this logical condition means that at least one neighbor server contains the file. In general, the update information is requested when TTL reaches zero, but this has a disadvantage. If the update request is sent when TTL reaches zero, the cache summary on the target DS will remain in slate for the RTT time period, and there may be a change in the cache digest on source DS. Good chance of having false hits for RTT time period on target DS if the file is gaining popularity at target environment and cache digest may

not be synchronized close to perfect. A factor "Δ_{RTT}" is introduced, as a variation factor in Eq (3). It can be any positive real number. It can be considered statically or calculated dynamically by observing the variation in different RTTs on group links, to take congestion into control and to select files which will become slate in near future. It is very likely that updated cache summary of an object from source DSs, which is cached on local target DS, is of less interest than an object which not cached on local target DS.

The target DS will send an if-modified request to the source DSs with the time of last update received from source DS and will receive delta updates from all DSs. Delta updates will have the objects with updated TTL and changes in the cache on the source DSs, compared to previous update.

Once the slate objects are selected, an if-modified request packet (a collective if-modified request for all content related to source DS) is sent using one of the two methods:

- Method 1: Send an update request to each DS recorded as source from previous update.
- Method 2: Send an update request to each DS in the group.

The update reply from the source DS will be a multicast packet to both the target DS and CS of the group, containing updated TTL value information with any cache additions on source DS.

Sending an update request to each source DS has an advantage of a reduced number of updates, and sending updates to all DSs has an advantage of more updated information. Choosing one of the above mentioned methods to follow depends on the minimum stability of cache of DSs seen in the group. One example scenario is, when an

update request is send to only previous recorded source servers, the same object might me cached in other servers after previous update, in such cases the method 2 is useful. The choice between two approached methods depends on stability of environment at source DS, which depends on the factors like:

1. Variation of demand for files in local environment of DS
2. File Distribution in the group among servers

Method 1 is considered when the local environments have less variation in demand and less number of files in common. The second approach to send requests is considered when multiple DSs may have the file and a considerable variation in cached files at source can be seen. Method 2 will work well in combination with content replication algorithms, which try to replicate the file based Zipf's law and criteria in a P2P network.

Chapter 4

SIMULATION AND RESULTS

In this chapter a detail analysis of the proposed technique is done along with simulation results comparing the proposed technique with legacy approach of statically updating cache digest.

4.1. Calculation of TTL at Source Server

Consider content C_y, cached on a local cache of a source Distributed Server(DS). The content C_y will be cached on local cache when the request rate of file is relatively high when compared to other files i.e. satisfying the minimum threshold R_{th} and does not have to evict other files which have more popularity than current file. Consider the content is cached at a request rate of $R_S$at time T_S where $R_S > R_{th}$.

Whenever the information regarding a source in summary table at the target DS becomes slate, the target DS will request an update of information from source DS. Let an update was sent by the source server regarding the content at time $T_n$and an previous update was sent at T_{n-1} and corresponding request rate is R_n and R_{n-1}. To estimate TTL, parameters are saved for each cached file; these parameters are listed in Table 3.

TABLE 3: List of Parameters Used For Calculating TTL

Parameters	Description
$\boldsymbol{R_S}$	The request rate at which content was cached
$\boldsymbol{T_{ST}}$	Most recent time at which request rate reached R_S
$\boldsymbol{T_S}$	Latest time at which content C_y is added to cache
$\boldsymbol{R_n}$	Request rate at which current update is send
$\boldsymbol{R_{n-1}}$	Request rate at last update
$\boldsymbol{T_n}$	Time at current update
$\boldsymbol{T_{n-1}}$	Time at previous update
$\boldsymbol{R_{th}}$	Minimum request rate needed to cache a file

Considering all the above parameters, TTL (at current update time T_n) sent by the source DS is calculated using Eq (4)

$$TTL_{C_Y} = \begin{cases} |TTL_{n-1} - T_{ST}| & for\ \ R_n = R_{n-1}\ and\ R_{n-1} > R_s \\ \left|(R_{n-1} - R_{th}) * Tan\left(90 - ATan\left(\frac{R_s - R_{n-1}}{T_{n-1} - T_{ST}}\right)\right)\right| & for\ \ R_n = R_{n-1}\ and\ R_{n-1} < R_s \\ |T_{val}| & for\ \ R_n = R_{n-1}\ and\ R_{n-1} = R_s \\ |T_n - T_{val}| & for\ \ R_n > R_{n-1}\ and\ R_{n-1} \geq R_s \\ \left|(T_n - T_{n-1}) + \left|(R_{n-1} - R_{th}) * Tan\left(90 - ATan\left(\frac{R_s - R_{n-1}}{T_{n-1} - T_{ST}}\right)\right)\right|\right| & for\ \ R_n > R_{n-1}\ and\ R_{n-1} < R_s \\ \left|(R_n - R_{th}) * Tan\left(180 - 2 * ATan\left(\frac{R_s - R_n}{T_n - T_{ST}}\right)\right)\right| & for\ \ R_n < R_{n-1}\ and\ R_{n-1} \geq R_s > R_n \\ \left|(R_n - R_{th}) * Tan\left(180 - 2 * ATan\left(\frac{R_{n-1} - R_n}{T_n - T_{n-1}}\right)\right)\right| & for\ \ R_n < R_{n-1}\ and\ R_{n-1} < R_s \\ \left|T_{n-1} - T_S - \left|\frac{R_{n-1} - R_n}{\tan\left[\cos^{-1}\left(\frac{T_n - T_{n-1}}{\sqrt{(T_n - T_{n-1})^2 + (R_{n-1} - R_n)^2}}\right)\right]}\right|\right| & for\ \ R_n < R_{n-1}\ and\ R_n > R_s \end{cases} \quad \text{Eq (4)}$$

4.2. Cache Table Update at Target Server

The target DS, receives the update for a file $C_y$with TTL, 'TTL_{WC_y}' from source 'W' and subtracts the propagation delay (P.D) from the TTL_{WC_y}, and added to complete summary table on target DS, so the TTL value on target DS TTL_{WC_y}, where 'S' represent the neighboring DS in the group,is given by Eq (5).

$$TTL_{SC_y} = TTL_{WC_y} - P.D \quad \text{Eq (5)}$$

Table 4 shows the local cache summary table on the target DS. In this table content is mapped with TTL value from Eq (5).

TABLE 4: Example View of Information Table at Target Server after Receiving Updates

Content	**TTL_1**	**TTL_2**	**…**	**TTL_S**
C_1	TTL_{1C_1}	TTL_{1C_1}	…	TTL_{sC_1}
C_2	TTL_{1C_2}	---	TTL_{s-1C_2}	---
….	…	…	…	…
C_y	----	----	…	TTL_{sC_n}

A target DS will consider the summary information for a content C_y as slate, based on following condition, where RTT_Z represent Round Trip Time (RTT) to zth DS out of available sources

maximum (TTL_{WCy}) ≤ maximum (RTT_z) + selection processing delay

Let n_{sl} be total number of slate objects in the table, and n_{slS} be number of objects per source server. When the target DS, sends an update request to the source servers, the number of objects (δ_n) in the update packet returned by the source is:

$$\delta_n = n_{slS} + \Delta_n \quad \text{Eq (6)}$$

Where Δ_n represents, the number of new objects added and deleted to local cache on the source DS. With traditional cache digest, a complete digest of all objects on the local cache of the source DS is sent as update, thus exchanging changes in the cache. The proposed technique will not sent a complete cache digest, only the changes are exchanged reducing update size. This reduces the impact of bursty updates on the data traffic.

4.3. Simulation

Simulation was done in MATLAB®, simulating source DS and Target DS. The algorithm was tested in a control environment with request rate increasing and decreasing at same rate. A runtime environment of 240 seconds is taken to dub the time period of 1 day. Likewise, static updating performed every 10 seconds equated to sending an update every hour in real-time. Bloom filter is lossy type of representation, and false prediction of files can happen due to overlapping of bits during representation[6]. A bloom filter, filter with negligible error rate occupies same as MD5 with 128-bit hash from value of ‘m’ from [6]. An MD5 128-bit hash algorithm was used to represent the digest instead of a bloom filter, to remove the effect of false hits that happen due to overlapping in representation. The simulation was done in two scenarios, and second scenario is tested with two methods discussed in Chapter 3. The two scenarios and methods are as follows:

1. Scenario 1：The scenario has a source server and a target server.
2. Scenario 2: The scenario has two source servers, a target server. In scenario 2, the target server sends an update request using any of the following methods.
    a. Method 1: Send an update request to each DS recorded as source from previous update.
    b. Method 2: Send an update request to each DS in the group.

Figure 6 compares amount of digest data exchanged for complete simulation time when 50 files are cached in scenario 1. In Figure 6, the solid line in the graph indicates the amount of digest data sent by source server when using static updating to update the complete digest and dotted line indicates the amount of data exchanged when using variable TTL. Comparing the two methods for updating the complete digest, selective updating using variable TTL exchanges less data between servers. This helps to visualize the usage of cache digest for inter-group digest exchange.

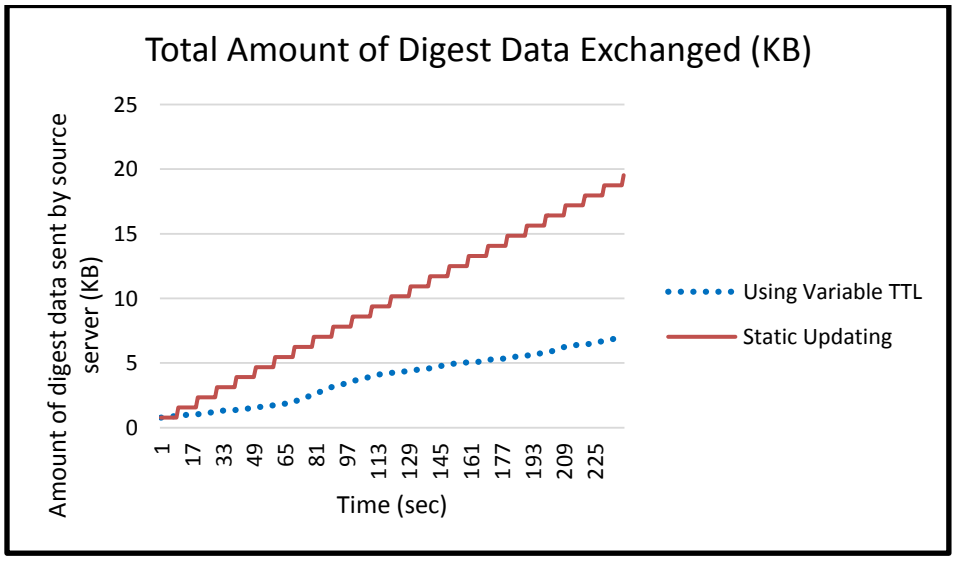

Figure 6: Total Amount of Digest Data Exchanged (KB)

Figure 7, shows comparison cumulative sum of false positive for two algorithms over time in scenario 1. The solid line represents the cumulative sum of the number of requests in a static update environment marked as false positive, (i.e., the target DS assumes that the file is still present at the source as per digest information, but the file has been removed from cache at the source server). The dotted line indicates the cumulative sum of number of requests marked as false positive when variable TTL is used. In a more stable environment, not many deletions in cache are seen, the number of false positives reduces in both algorithms. False positives are seen when popularity of a file decreases more suddenly than expected.

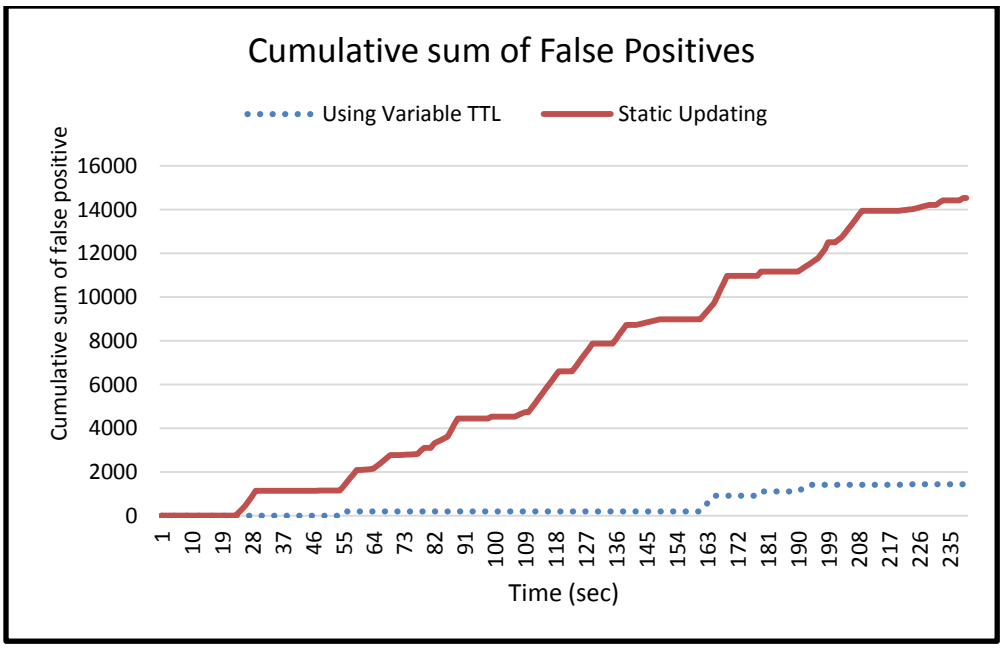

Figure 7: Cumulative sum of False Positives

Figure 8 shows the comparison of the number of false positives for both algorithms at every instance. The solid line indicates the number of requests at the target server marked as false positives when static updating of digest is used. Dotted lines indicate the number of requests marked as false positives over time when proposed method is used. As discussed above the false positives depend on the variation of the source DS cache and the number of requests received at the target server for the file deleted at the source DS.

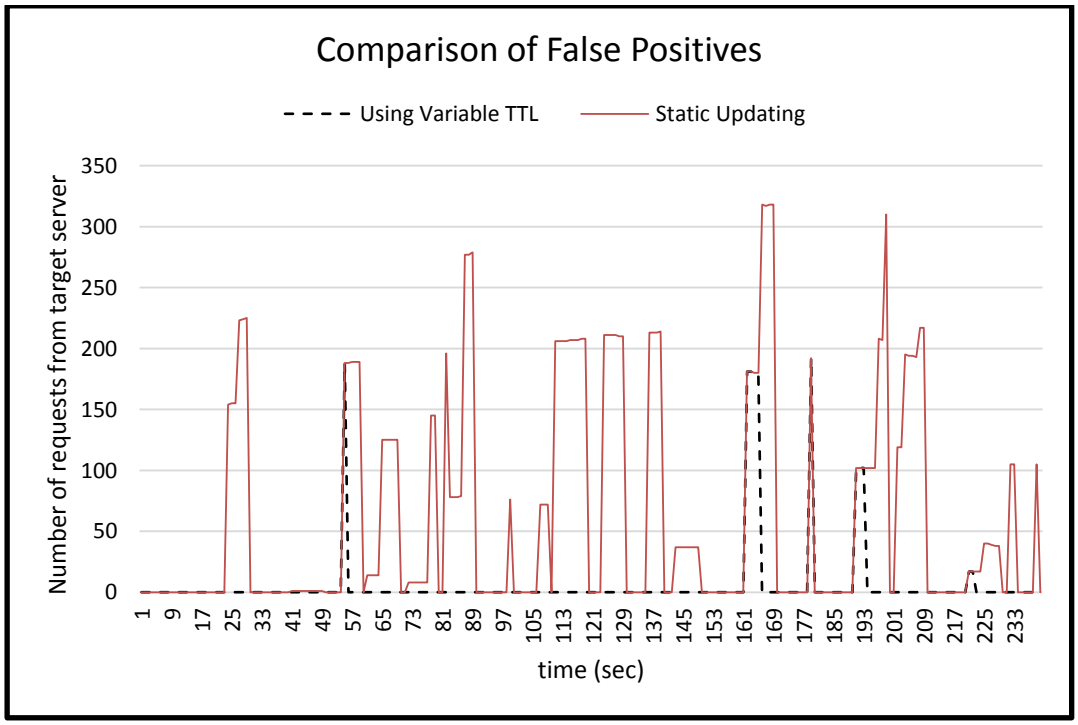

Figure 8: Comparison of False Positives

Figure 9, compares the cumulative sum of false negatives over time for both techniques. The solid line indicates the cumulative sum of requests that are marked as false negatives; the target DS predicts that the source DSs do not have the file as per previous digest summary. The dotted line indicates the cumulative sum of false negatives when dynamic TTL is used. The number of false negatives are low in this case, due to the frequent updates of summary at the target DS.

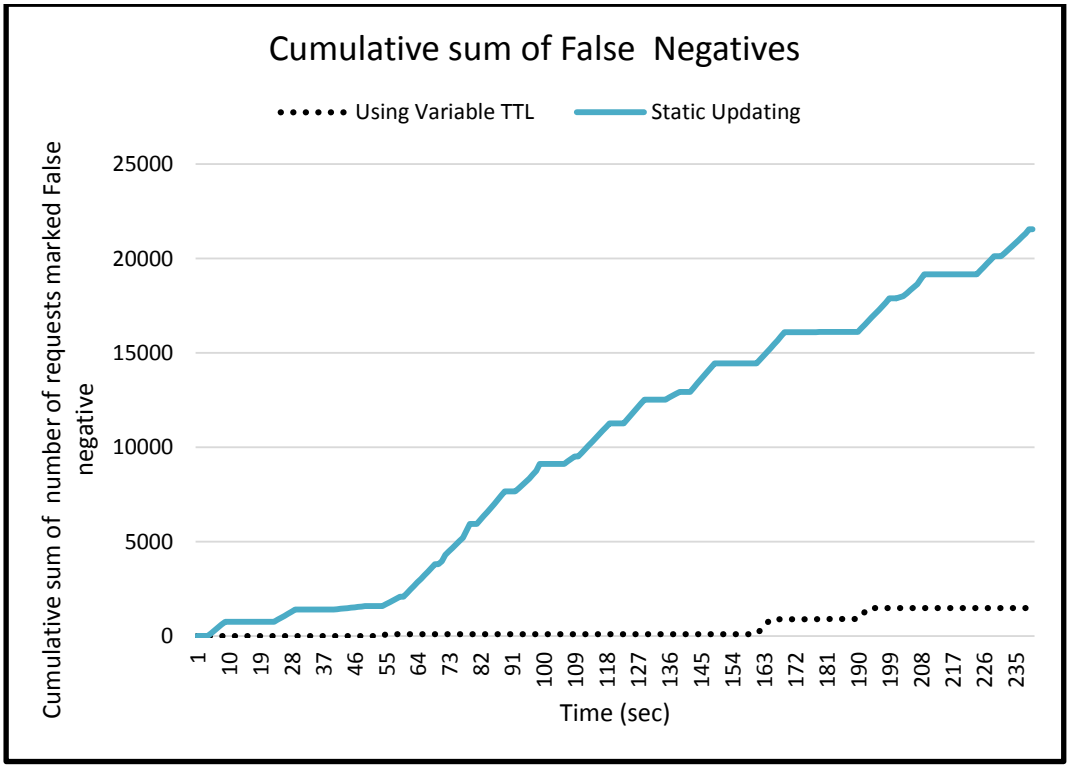

Figure 9: Cumulative sum of False Negatives

Figure 10 shows the number of requests marked as false negatives at every instance. As the environment at source and target server has more dynamic changes in popularity for various files, there are many changes in the list of objects cached at both source and destination server.

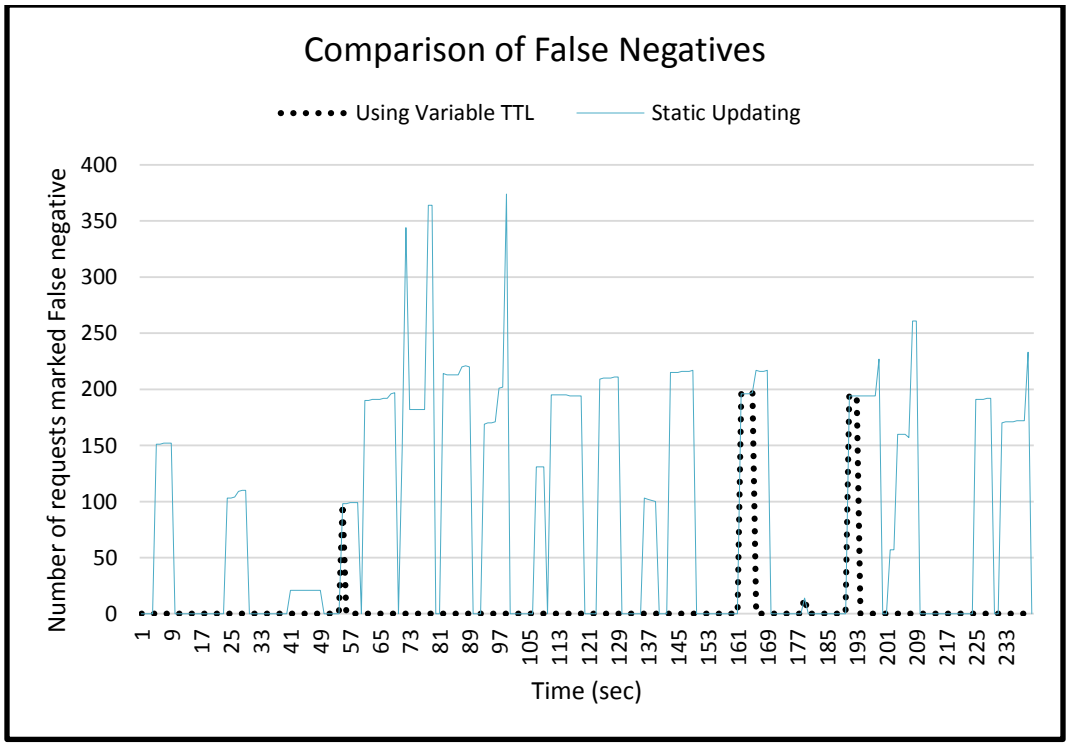

Figure 10: Comparison of False Negatives

To further analyze the scalability of the technique, the proposed algorithm was tested in scenario 2. This scenario has two source servers with one source server taken from scenario 1 and an entirely new source server. A target server was added into group and simulation was performed. Figures from 11 to 16 shows the behavior in the environment when each of the two approaches to request an update are used as discussed above. The first approach is when a file becomes slate on the requesting

server; the update request is sent only to the sources from which it was previously recorded.

Figure 11 shows the amount of digest sent only by source server 1 which is common in both scenarios. In Figure 11, the solid line indicates the amount of digest data send only by source server 1, when it was the only source of file. The dotted line indicates the amount of digest sent only by source 1, when another server exists as a second source for the file. The amount of digest sent reduces with the increasing number of sources for a file, as the target server will not request the update information if there exists one valid TTL belonging to any of the source DSs. When the DSs in a group has more files in common, the amount of information exchanged decreases more as the probability of the file becoming slate reduces.

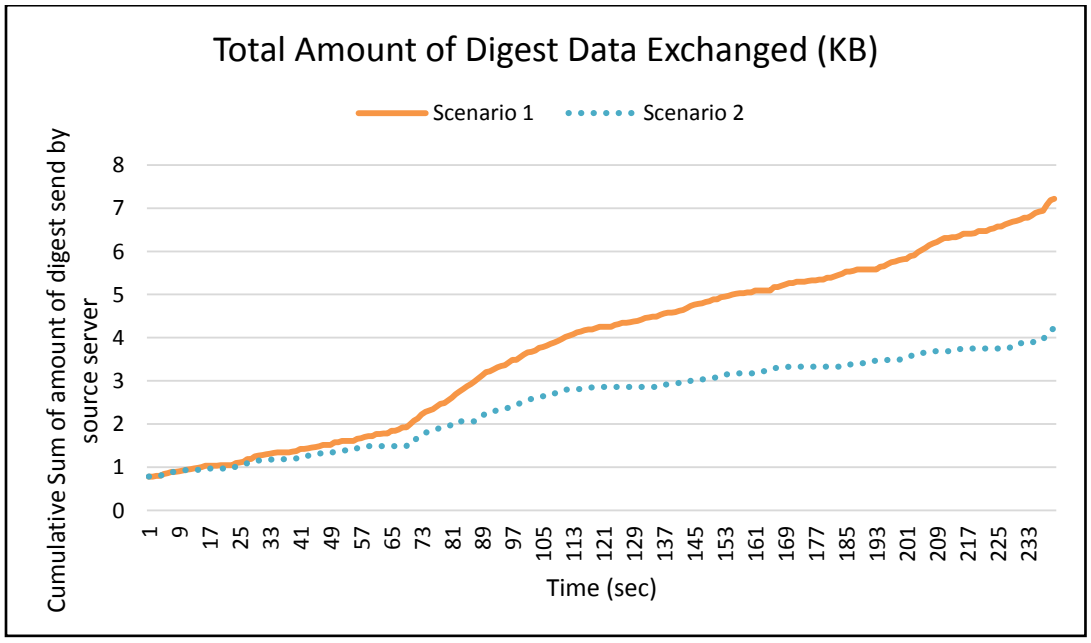

Figure 11: Method 1 - Total amount of digest data exchanged (KB) for different scenarios

Figure 12, shows number of requests marked as false positive for both scenarios at every instance. In Figure 12, the solid line indicates the times when user requests at the target server are marked as false positive for scenario 1 and dotted line indicates the times when requests are marked as false positive for scenario 2. In general, number of times false positives appear continuously reduces as more number of valid sources for the file are available for file and a less varying cache is seen at sources. The number of requests that are marked as false positive depends upon number of requests obtained at target server for the file, which was deleted at source server and not notified due time difference between successive updates.

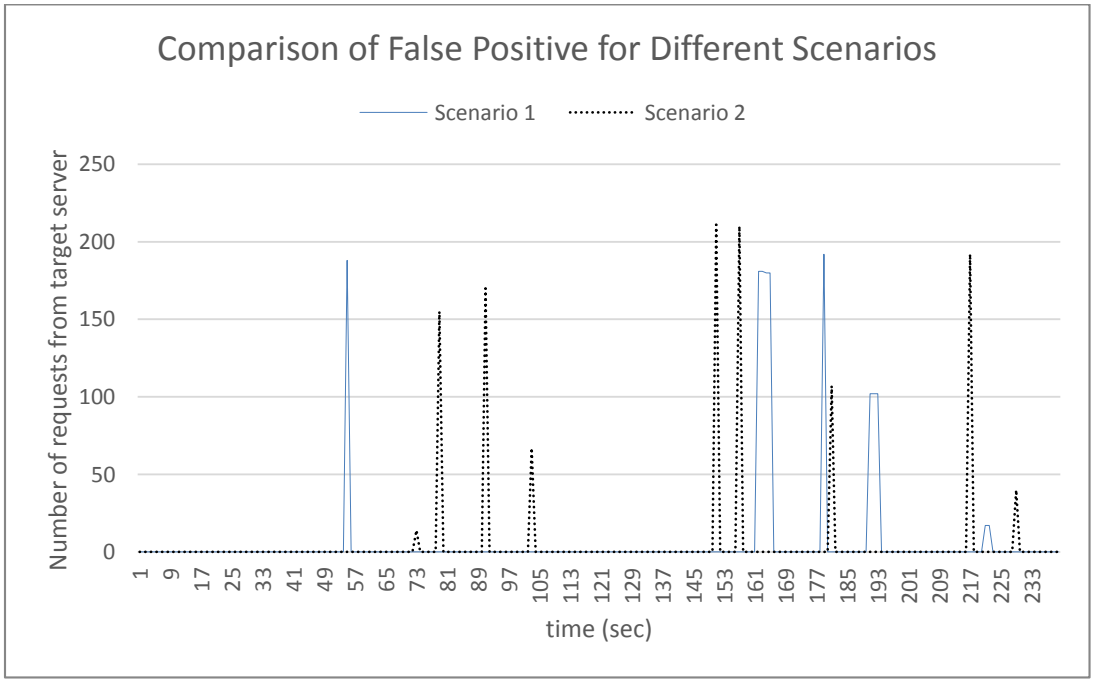

Figure 12: Method 1 - Comparison of False Positives for Different Scenarios

Figure 13 compares the number of times requests are marked false negatives for both scenarios. In Figure 13, the solid line indicates the false negatives in scenario 1 and the dotted lines indicates the false negatives in scenario 2. The initial spikes in scenario 2, are due frequent changes in the cache at second source server and user requests appear at the target server at the same time. This can be seen as disadvantage of method one as the information about file is only obtained from the previously recorded sources.

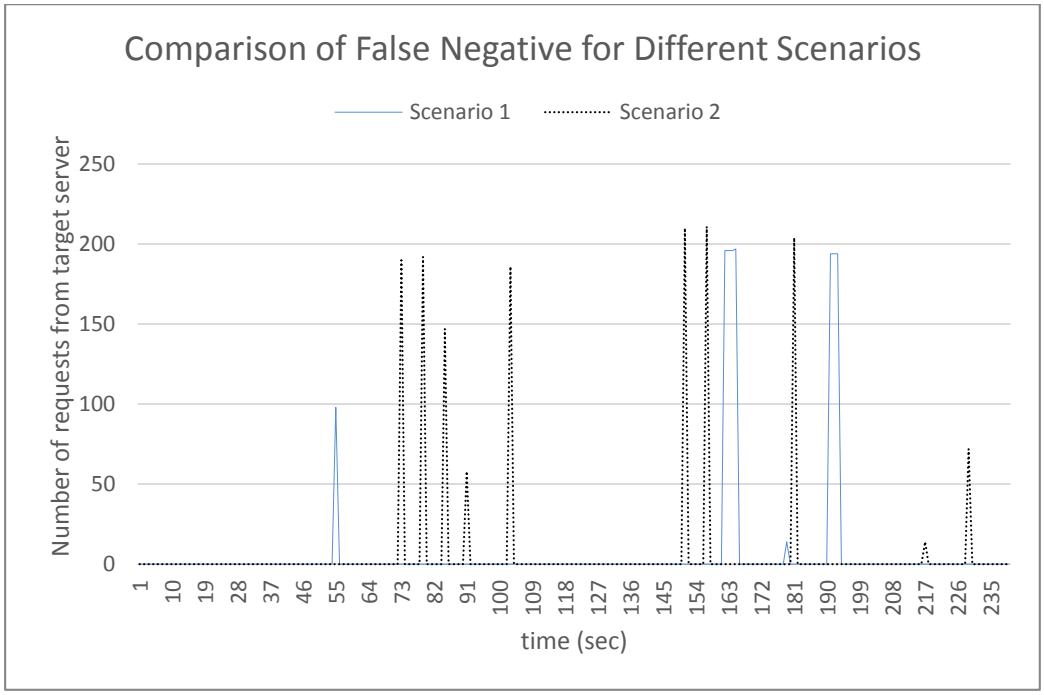

Figure 13: Method 1 - Comparison of False Negative for Different Scenarios

Considering the method two of scenario 2 of sending updates. When a file becomes slate on the requesting server, an update request for the file is sent to all neighbors. The second method was tested in a similar test bed used for method 1. The source server 2 is less stable than source server 1, but has common files cached.

Figure 14, shows the amount of digest information sent by only the source server 1, between scenario 1 and method 2 of scenario 2. Figure 14, compares the amount of digest information sent by the source server 1 in both the scenarios. The amount of data is almost same because the request is sent to source 1, even if the information for the file recorded from source 2 becomes slate.

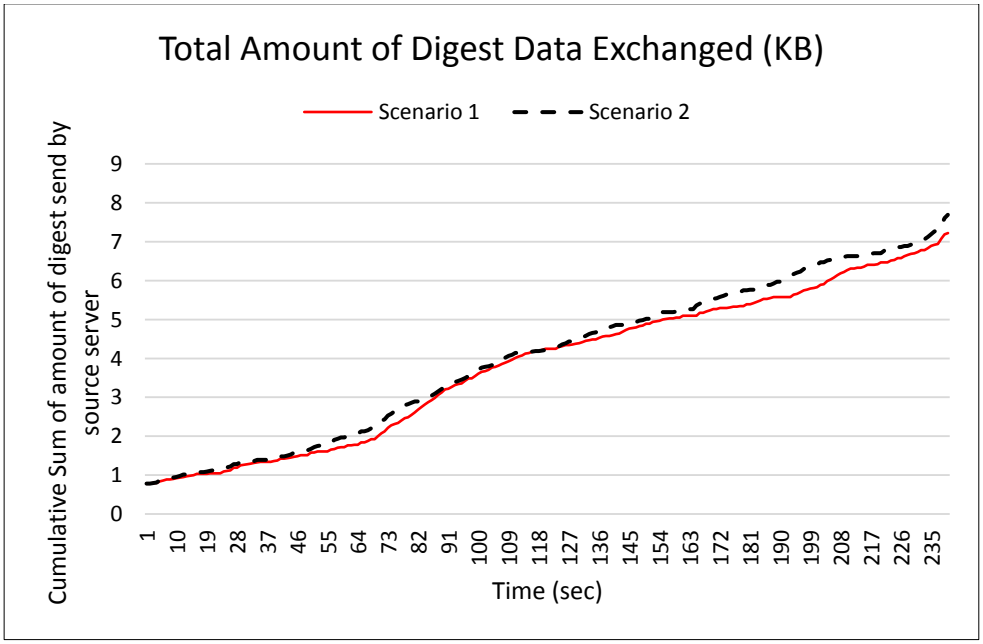

Figure 14: Method 2 - Total Amount of Digest Data Exchanged (KB)

Figure 15, shows user request marked as false positives in both scenarios. In Figure 15, dotted line indicates the number of false positives recorded in scenario 1 and the solid line represents the number of false positives recorded in scenario 2 when method 2 is used. The false positives is almost negligible as the summary table is frequently updated. If the environment has less number of new popular files, which is considered as an ideal case for digest sharing techniques [6], with the proposed technique less number of false positives can be seen as in Figure 15, with less amount of digest data exchanged between source and target DSs.

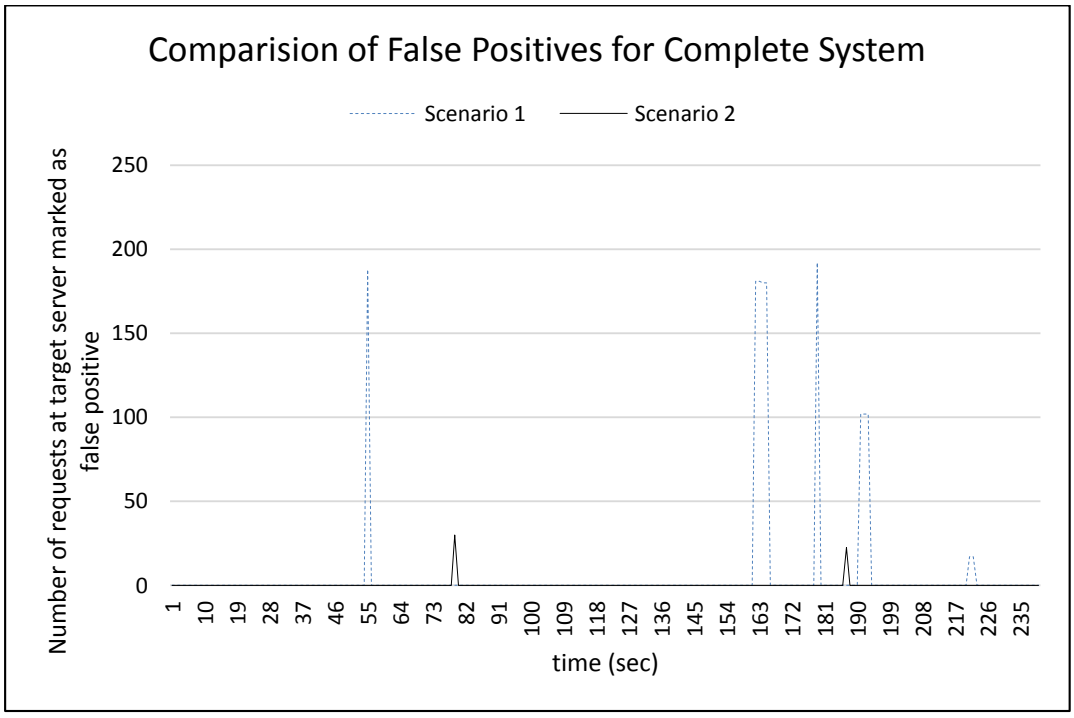

Figure 15: Method 2 - Comparison of False Positives for Different Scenarios

Figure 16, dotted line indicates the number of false negatives in scenario 1 and the solid line indicates the number of false negatives in scenario 2 with method 2. It can be observed that false negatives are almost negligible in scenario 2, as the target DS's summary table is updated more frequently than in method 1.

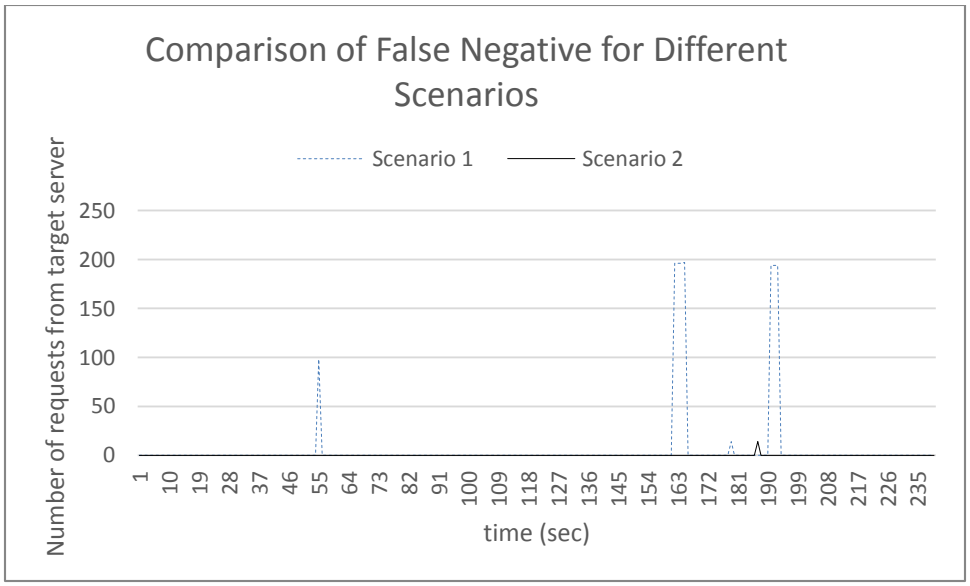

Figure 16: Method 2 - Comparison of False Negative for Different Scenarios

Thus, using proposed technique for the user access pattern seen along the time period can be used to tune cache digest exchange mechanism. The proposed techniques performs better than legacy static updating of complete digest, by reducing the amount of digest information exchange between source DS and target DS in scenario 1. The number false positives and false negatives are low compared to static updating in scenario 1. The false positives and false negatives are almost negligible in method, with almost same amount of digest information exchange compared to scenario 1, but less than legacy static updating.

CONCLUSION

This research presents a new line of research that can be used to improve the certainty of information that is exchanged between servers by providing an estimated Time To Live (TTL) for files that are shared as part of exchange. This mechanism is suitable for access patterns which are not bursty in nature and are popular among regional end clients. As the content popularity varies across different clusters, a well distribution algorithm along with the proposed algorithm can provide considerable advantages in reducing the number of false positives and reducing bandwidth wastage. The proposed mechanism increases the number of updates, but each update decreases in size because only the digest of selected files is exchanged. This reduces the impact of sharing bursty types of updates on low-speed links, which is limitation of traditional static updating of complete digest [6]. Hence, dynamic updates which have stream like traffic pattern can be useful for sharing digests on low-speed links. Future work could be done to analyze the feasibility of using such access patterns to improve other techniques like hash based and semi-hash based mechanisms.

REFERENCES

REFERENCES

[1] A. M. K. Pathan, "Utility-oriented internetworking of content delivery networks," PhD, Engineering - Computer Science and Software Engineering, The University of Melbourne, 2009.

[2] R. Buyya, M. Pathan, and A. Vakali, *Content Delivery Networks* vol. 9: springer, 2008.

[3] J. P. Mulerikkal and I. Khalil, "An Architecture for Distributed Content Delivery Network," in *Networks, 2007. ICON 2007. 15th IEEE International Conference on*, 2007, pp. 359-364.

[4] N. Bartolini, E. Casalicchio, and S. Tucci, "A WalkThrough Content Delivery Networks," *In Proceedings of MASCOTS 2003, LNCS,* vol. 2965/2004, pp. 1-25, April 2004.

[5] D. Wessels and K. Claffy. (September 1997). *Internet Cache Protocol (ICP), version 2*. Available: http://www.ietf.org/rfc/rfc2186.txt, [August, 2012]

[6] A. Rousskov and D. Wessels, "Cache Digests," *Computer Networks and ISDN Systems,* 1998.

[7] J. Lloret, M. Garcia, D. Bri, and J. R. Diaz, "A group-based content delivery network," presented at the Proceedings of the third international workshop on Use of P2P, grid and agents for the development of content networks, Boston, MA, USA, 2008.

[8] N. Jian, D. H. K. Tsang, I. S. H. Yeung, and H. Xiaojun, "Hierarchical content routing in large-scale multimedia content delivery network," in *Communications, 2003. ICC '03. IEEE International Conference on*, 2003, pp. 854-859 vol.2.

[9] M. Bjorkqvist, L. Y. Chen, M. Vukolic, and Z. Xi, "Minimizing retrieval latency for content cloud," in *INFOCOM, 2011 Proceedings IEEE*, 2011, pp. 1080-1088.

[10] J. P. Mulerikkal and I. Khalil, "An Architecture for Distributed Content Delivery Network," Minor, School of Computer Science and Information Technology Science, Engineering, and Technology Portfolio, Royal Melbourne Institute of Technology, 2007.

[11] *Squid Introduction*. Available: http://www.squid-cache.org/Intro/, [September, 2012]

[12] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM,* vol. 13, pp. 422-426, 1970.

[13] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal, "The Bloomier filter: an efficient data structure for static support lookup tables," presented at the Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, New Orleans, Louisiana, 2004.

[14] M. Ripeanu and A. Iamnitchi. *Bloom Filters – Short Tutorial*. Available: www.cs.uchicago.edu/~matei/PAPERS/bf.doc, [January, 2013]

[15] D. Suciu. (Wednesday, June 2, 2010). *Lecture on Bloom Filters*. Available: http://www.cs.washington.edu/education/courses/cse444/10sp/lectures/lecture24.pdf, [January, 2013]

[16] H. Yu, D. Zheng, B. Y. Zhao, and W. Zheng, "Understanding user behavior in large-scale video-on-demand systems," *SIGOPS Oper. Syst. Rev.,* vol. 40, pp. 333-344, 2006.

[17] G. Barnett and A. Tong, "DEPARTMENT OF ACTUARIAL STUDIES RESEARCH PAPER SERIES."

[18] D. N. Serpanos, G. Karakostas, and W. H. Wolf, "Effective caching of Web objects using Zipf's law," in *Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on*, 2000, pp. 727-730 vol.2.

[19] Jeff. (May 7, 2012). *Amazon CloudFront - Support for Dynamic Content*. Available: http://aws.typepad.com/aws/2012/05/amazon-cloudfront-support-for-dynamic-content.html, [December, 2012]

[20] K. Harlin. (April 24, 2012). *How time of day affects content performance*. Available: http://www.imediaconnection.com/content/31577.asp, [June, 2013]