ABSTRACT

PERFORMANCE STUDY OF WEB CACHING ALGORITHMS IN A HOME NETWORK WITH VIDEO TRAFFIC

By

Peter Suh

August 2015

Internet traffic is growing at a significant pace, especially with video content that already accounts for more than 50% of total of internet traffic. The reasons for this trend are increasing numbers in mobile devices, shifting focus from traditional television media to the online content, and advancement in internet technology. Internet infrastructure is facing challenges as it tries to meet the demand for video traffic. One solution is web caching, an area of research with many different proposed algorithms and implementations. This thesis examines the performance of a set of algorithms using a popular web caching software, Squid. The tests are done using a web automation tool, Selenium, to browse the web from a home network in three different scenarios: checking web content, videos, and a mix of web content and videos.

PERFORMANCE STUDY OF WEB CACHING ALGORITHMS IN A HOME NETWORK WITH VIDEO TRAFFIC

A THESIS

Presented to the Department of Computer Engineering and Computer Science

California State University, Long Beach

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

Committee Members:

Tracy Bradley Maples, Ph.D.
Burkhard Englert, Ph.D.
Mehrdad Aliasgari, Ph.D.

College Designee:

Antonella Sciortino, Ph.D.

By Peter Suh

B.S., 2011, University of California, Irvine

August 2015

ACKNOWLEDGEMENTS

I want to thank Dr. Lam for guiding me in this program early on. I would like to thank Nick Tran for helping me with this by sharing his experiences and advices. I would not have been able to graduate without the help from Mr. Valdivia, Dr. Sathianathan, Dr. James, and Dr. Englert. Finally, this thesis could not have been done without the advice and direction from my advisor, Dr. Maples.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AMD	Advanced Micro Devices
CPU	Central Processing Unit
CBS	Columbia Broadcasting System
CDN	Content Delivery Network
DSL	Digital Subscriber Line
DVD	Digital Video Disc
GDS	Greedy Dual Size
GDSF	Greedy Dual-Size Frequency
HBO	Home Box Office
HD	High-Definition
ISP	Internet Service Provider
LFU	Least Frequently Used
LFUDA	Least Frequently Used with Dynamic Aging
LRU	Least Recently Used
MB	Megabyte
Mbps	Megabits Per Second
RAM	Random Access Memory
TV	Television

CHAPTER 1

INTRODUCTION

In today's internet, with so much content available, from streaming video to online social networks, high access speed is of critical importance. We have become dependent on the internet for almost every aspect of our lives. It is certain that internet traffic is increasing substantially. According to Cisco, a well-known company specializing in network and internet technology, 51 zettabytes of data have been transmitted over the internet in 2013. A zettabyte is equivalent to one trillion gigabytes. Last year, it increased to 62 zettabytes. It is projected to increase steadily over the next few years [1].

There are several factors affecting this increase in internet traffic. First, the number of mobile devices have exceeded the global population and this number is estimated to double by 2018 [1,2]. Second, people are moving away from cable and television and into online streaming services for flexibility of viewing and to avoid the high cost of cable subscriptions [3,4]. Third, televisions are now equipped with internet access that allows viewing of online content, with some devices getting content strictly from the internet, such as Google's Android TV [3]. Fourth, higher video quality is available, from HD to 4K [5]. Fifth, many broadcast companies such as CBS and HBO are starting to distribute their media through online streaming services [3]. Finally, current internet capacity is higher than 16 Mbps and it is expected to reach 42 Mbps by 2018 [1]. As a result, the majority of the current internet traffic is video.

The impact of video’s percentage in total internet traffic has become significant. As of last year, 63% of total internet traffic, which is 39 zettabytes, was video [1]. In other words, it would take a person 60 million years to watch everything currently being streamed on the internet [1]. The percentage of video is projected to increase to 80% by 2018. That will be equivalent to 105 zettabytes of video streaming across the internet [1]. By that time, a person must watch 72 billion DVDs to what is being transmitted on the internet [1]. Currently, the majority of videos are coming from Netflix and YouTube [6]. There are a wide variety of video presentations in terms of both resolution and duration. Just on Youtube, over 100 hours of videos are uploaded every minute [7]. With such a large growth in video and internet traffic, the internet faces obstacles to its underlying network infrastructure.

Users and content providers notice problems when the network infrastructure has difficulty handling the ever-increasing internet traffic. Viewers have experienced videos stopping momentarily waiting for more data to load which is known as buffering. Accessing web pages starts to take longer, especially if there is a lot of content. There are a number of possibilities for these problems. First, it may be that the home internet connection is not designed to handle the traffic being requested. In other words, it does not have sufficient bandwidth. Another problem may be that the source of the content is at a great distance from the destination which will cause delay. The solution for the first problem would be to upgrade the connection to the ISP since it is expected to reach 42 Mbps by 2018, an increase of 200% since last year. Telecommunication companies are upgrading their networks and constantly investing in faster technologies [8]. Although this is viable, some critics argue that the capacity of local connections may still not be able to keep up with the growth of internet traffic [3]. The solution to the second problem, the distance of the requested data, would be to bring the content closer to the users. One approach is Content Delivery Networks (CDN) [9]. CDNs are data centers located in densely populated areas that re-distribute original content [9]. They essentially bring content closer to the user. This is effective, but it requires a large investment by the service providers. The second approach to fixing this problem is to use web caching, which is to store web data temporarily. Web caching is a technology used in proxy servers and web browsers that accelerates content delivery and reduces network congestion. Proxy server is a server that transfers messages between clients and other servers. Many internet service providers, campus networks, and businesses use proxy servers [10]. Since web caching technology is incorporated as part of the network software, there are many ways for improvement. The disadvantages of web caching are that it cannot store everything as CDN does, it is typically bound by more limited memory and connection speed, and proxy servers must retrieve new data from content servers. It is important to note that none of the solutions mentioned can resolve the latency of live-streaming which is also growing rapidly [6].

Web caching offers a flexible way to relieve internet traffic because it is software-based. Although hardware can be improved, this can be costly and time-consuming so a solution is to utilize the current available memory and make an algorithm that is efficient. Web caching temporarily stores requested web content. When the same content is requested again, it is delivered to the client faster than delivering it from the source. If there is no more room to store content, the cache will remove old or unused items. Web caching uses algorithms to try to make the best decisions on what to keep and remove.

There are three types of web caching: client-side, content server-side, and proxy server. Client-side web caching is done by browsers where it stores contents of visited web pages. The disadvantage to client-side caching is that this is restricted to one user or device. Content server-side caching is used to cache redundant computations and database retrieval [10]. This reduces the load on the server. Proxy servers are located between multiple users and content servers. They use web caching software to handle all incoming web traffic. Their use reduces network traffic and response time by relieving the load on content servers and reducing retrievals of data across the internet. There has been significant research into developing various effective algorithms for web caching in all three approaches. These algorithms will be discussed in further detail shortly.

Efficient caching algorithms keep popular items and dispose of rarely used items, thus translating to better proxy performance and increased access speed. This thesis examines the performance of a popular web caching software used in a proxy, Squid, through empirical testing with a subset of today's web traffic. The testing is done using Selenium, a tool for web automation. Based on the results of the tests, some recommendations are made to maximize the performance of Squid.

This thesis is divided as follows: in Chapter 2, background information on web caching and various web caching algorithms is covered. Chapter 3 describes the software tools used in this thesis. Chapter 4 discusses the web caching experiment design and collection of data. Chapter 5 discusses the results of the experiment. Finally, Chapter 6 provides conclusions and future work about this web caching experiment.

CHAPTER 2

BACKGROUND

Web caching is a technology that stores or caches web content or objects for future reuse. The web objects can be almost anything, such as images, videos, texts, audio, documents, web page components, and scripts. Web caching was developed to reduce latency, bandwidth use, and load on internet content servers [10]. One of the major problems with web caching is keeping unused objects in the cache for a long time which is known as cache pollution. This occurs because algorithms do not perform efficiently either because they are too simple or because certain network conditions do not work well with a particular algorithm. Another problem occurs when an algorithm is not caching data effectively. In this case, the stored objects get removed from the cache too early. Although there have been many algorithms developed for web caching, finding the best algorithm is still a major challenge [10]. Today's web caching software, such as Squid, still uses basic algorithms despite the availability of many other advanced algorithms. The next few paragraphs will talk about how web caching works in further detail and then discuss metrics used to measure performance. In addition to web caching, a brief background on web pre-fetching will be covered as both are related.

When a web page is requested from a client for the first time, a proxy server will retrieve web pages from the source, which may be a long distance away. The retrieved pages are cached into individual objects in a proxy server and sent to the client. Each time a cached object is requested, the proxy server checks if the object is still fresh, meaning the current date and time have not passed the expiration date and time. If it is fresh, then the proxy server will send the cached object. If it is not fresh, then the proxy server must retrieve it from the source and replace it with the expired object. When the same content is accessed again from the same or a different client, the content is retrieved from the proxy server, not from the source. Therefore, the delivery time is reduced significantly. A proxy server will cache objects until the memory is full. When the memory is exhausted and new objects arrive, the proxy server must decide what objects to replace with the new ones or whether to keep the new objects at all. Algorithms make these important caching decisions.

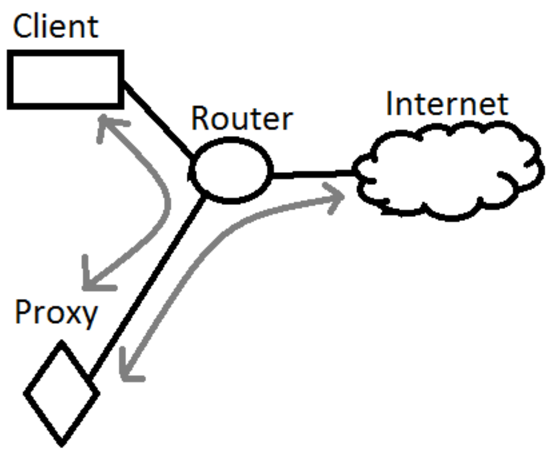


FIGURE 1. Proxy server in a network. The client retrieves new or cached objects from the proxy and the proxy retrieves new objects from the internet.

Web Caching Algorithms

One of the most popular web caching software available is Squid. We will discuss Squid in detail in Chapter 3. Squid uses three algorithms: Least Recently Used (LRU), Least Frequently Used with Dynamic Aging (LFUDA), and Greedy Dual-Size

Frequency (GDSF) [11]. Least Recently Used works by adding a timestamp to each object which represents the last time it was either cached as a new object or reused [12,13]. When there is not enough space, it simply removes the object with the oldest timestamp [12,13]. LRU was first used in memory and CPU caching, and it has proven effective there but in web caching it is not as effective because the size of objects are not uniform [14]. The advantage of using LRU in web caching is that it is fast due to its simplicity. The disadvantage is that it may discard an object that is used frequently. It is one of the simplest algorithms developed and there are enhanced variations [15,16]. Although there are no distinct advantages, it still provides satisfactory performance that is why it is the default choice in Squid [11,12,17,18].

LRU = last accessed timestamp (1)

Least Frequently Used with Dynamic Aging keeps track of how many times an object gets reused and its time in the cache or age [12,17]. It removes objects with lowest frequency of use. The predecessor of this algorithm is Least Frequently Used. Least Frequently Used, just as Least Recently Used, is a simple algorithm that ranks objects based on frequency. Frequency is stored as a counter that increments only to the cached object that is accessed. LFUDA was also used in memory and CPU caching. Keeping objects based on frequency allows some objects to be stored longer than necessary, which is known as cache pollution. This is due to an object having a high frequency resulting in objects with lower frequency being replaced even though the objects with lower frequency may have higher potential to be accessed again. To prevent this, an age variable is implemented to balance the frequency [19]. The age counter increments every time a cached object is replaced and the value is added only to the newly cached object.

This forces objects that have not been accessed for a long time to be pushed back in the queue. A score is kept by summing age and frequency. When the cache is full, LFUDA removes objects with the lowest score, based on equation 2. A new object that gets cached is given a score by summing the age value and a frequency of one. Then all of the objects are re-ordered based on their scores. An advantage is that it keeps popular objects, regardless of size. A disadvantage is that caching large, popular objects takes a significant amount of memory space.

LFUDA = age + frequency (2)

Greedy Dual-Size Frequency is similar to Least Frequently Used with Dynamic Aging except that it takes into account an object's size in order to maximize the usage of memory space. The predecessor of this algorithm is Greedy Dual. Greedy Dual was developed for uniform-size cache replacement. It sorts objects by cost where cost is a measure of transferring objects into the cache. The improved algorithm, known as Greedy Dual Size (GDS), was developed to consider variable size. An age was also implemented to prevent cache pollution. Finally, frequency was added for determining access patterns which improved performance further, known as GDSF. In Squid, the cost variable is removed [19]. It uses equation 3 to remove objects with the lowest value. The advantage is that it maximizes the number of objects that can be cached. The disadvantage is that it penalizes large and popular objects [14].

GDSF = age + $\frac{frequency}{size}$ (3)

In addition to Squid's algorithms, there are many more web caching algorithms developed. There are six types of primary caching algorithms: recency-based, frequency-based, size-based, function-based, random, and intelligent. LRU is categorized as recency-based, LFUDA is categorized as frequency-based, and GDSF is categorized as function-based. Function-based algorithms consider multiple parameters to meet specific requirements. The advantages of function-based algorithms vary by their design, and the disadvantage is the difficulty of putting appropriate weight on each parameter. GDSF is function-based because it considers more than one factor: frequency, cost, and size. Greedy Dual and GDS are considered function-based due to cost and size factors. LFUDA can also be considered function-based because of the age factor but the algorithm is primarily concerned with frequency and age is implemented solely for preventing cache pollution. Same argument can be made for GDSF for its use of the age factor. Size-based type prefers to store smaller objects. The advantage of this is that it can get high hit ratio because it can store many objects. The disadvantage is that it discards large frequently visited objects so it cannot get high byte hit ratio. Random types remove cached objects randomly. There is no definitive advantage or disadvantage to using it. Intelligent type algorithms do complex analysis to cache objects. It analyzes logs, web environments, and web traffic by applying machine learning techniques. The advantage is that it is highly effective. The disadvantage is that it takes a significant amount of time and computation overhead. Within each type there are numerous algorithms. For example, there are twelve different recency-based algorithms and seven intelligent algorithms. Each algorithm has its own advantages and disadvantages. A summary of each type is shown in Table 1. The next paragraph will discuss how to measure performance of these algorithms [10].

TABLE 1. Algorithm Types

Algorithm	Description	Advantage	Disadvantage
Recency-based	Adds last access timestamp to each object.	Simple implementation. Fast computation.	Disregards frequency.
Frequency-based	Adds frequency counter to each object.	High hit rate.	Cache pollution. Low byte hit rate.
Size-based	Keeps smaller objects.	High hit rate.	Low byte hit rate. Removes frequently visited large objects.
Function-based	Considers multiple parameters.	Depends on an algorithm's design.	Difficulty assigning appropriate weight on each parameter.
Random	Randomly removes objects.	Not definitively known.	Not definitively known.
Intelligent	Analyzes caching pattern by applying machine learning technique.	High hit rate and byte hit rate.	Large computation overhead.

Web Caching Metrics

An algorithm's performance is measured by metrics. Two metrics are widely used in web caching: hit rate (HR) and byte hit rate (BHR) [10,12,13,14,17,18,19,20]. Hit rate is a measure of how many times a cached object is requested over total requests. It is a way of measuring how well an algorithm caches objects. High hit rate translates to good performance. There are three ways to get high hit rate. First, the same object is requested many times. Second, many cached objects are requested. The first case is unlikely because there are many different objects in the internet. The second scenario is more likely to occur. To get high hit rate in the second scenario, the number of cached objects should get close to the number of requests. That means the number of cached objects must get high. Since there is a limit on cache space, the number of cached objects can only be maximized by decreasing their size. In order to get high hit rate, an algorithm should store as many small objects as possible. The third scenario is to keep and store objects that will be requested frequently. This requires heuristics to predict what objects may be requested again in the future. Greedy Dual-Size Frequency was designed to achieve high hit rate by increasing the chances of getting hits by storing smaller objects [12,14,17].

Byte hit rate is a measure of how much data is coming from the cache versus how much comes from the content server. It is a way of measuring how efficiently the bandwidth is used. High byte hit rate is important for good network performance. There are two scenarios to achieve high byte hit rate. The first case is getting hits on a large object. The second case is getting hits on many small objects. These two cases are unlikely to occur commonly due to the nature of the internet. This also requires heuristics. Least Frequently Used with Dynamic Aging was designed to achieve high byte hit rate by allowing larger objects to get hits [12,14,17].

It is difficult to achieve high percentage in both of these metrics because they are contradictory [14]. If high hit rate is desired, then smaller objects should be stored and this will not provide a high byte hit rate. On the other hand if high byte hit rate is desired, then larger objects should be stored and this will not provide a high hit rate due to fewer number of cached objects. Getting high hit rate with large objects is also difficult due to object size. Getting high byte hit rate with small objects is difficult because the total size of all small objects will not be large compared to the total size of all large objects. For example, the sizes of small and large objects are one kilobyte and five kilobytes, respectively. Then if five cached objects are requested, sending small objects only have a total size of five kilobytes while sending large objects only have a total size of 25 kilobytes. Sending large objects only have higher byte hit rate. On the other hand, if the total size of requested objects is five kilobytes, then only five small objects or one large object can be sent at a time. Sending small objects only have higher hit rate. To get both a high hit rate and a byte hit rate is nearly impossible because the cache must contain most of the objects requested, but users always visit new web pages. For example, GDSF cannot achieve high byte hit rate and LFUDA cannot achieve high hit rate.

Web Pre-Fetching

One way to improve performance further is by using web pre-fetching. Web pre-fetching is similar to web caching except it caches objects that have never been requested. This technique can be applied at client-side, content server, and proxy servers as well. It works by predicting what objects will be requested based on users' navigation patterns. In order for web pre-fetching to work, it must incorporate web caching. The performance of web caching and web pre-fetching can increase by 100% if the two work well together. The disadvantage is that web pre-fetching may cache objects that may never get requested, resulting in increased load and traffic. This can lead to worse performance than just utilizing web caching. Web pre-fetching has two types: history-based and content-based. History-based types analyze visited web pages and determine what similar pages will be requested in the near future. For example, if users check online news from Yahoo or CNN, it will cache those pages as soon as new content are available. Content-based types use keywords found in web pages to predict what pages with the same keywords will be visited by users. For example, users visiting Amazon will most likely visit other online shopping sites such as eBay and Walmart using keywords such as “shopping,” “buy,” and “sale.” There are different metrics used to measure the performance of web pre-fetching algorithms [10].

The effectiveness of pre-fetching algorithms is measured by 6 metrics: precision, byte precision, recall, byte recall, traffic increase, and latency per page ratio. Precision is the rate of how many pre-fetched objects were hit over total pre-fetched objects. Byte precision is similar to precision except in terms of bytes. Recall is similar to the hit rate from web caching except it counts the number of pre-fetched hits over the number of requests. Byte recall is also similar to byte hit rate except it calculates the total bytes of pre-fetched objects over total bytes transferred. These four metrics desire high percentage for performance. Traffic increase is the difference of bytes passing through the network with pre-fetching and no pre-fetching. Latency per page ratio is a latency ratio between using pre-fetching and not using pre-fetching. These two metrics are desired to be minimal [10].

CHAPTER 3

SOFTWARE TOOLS

This section introduces the software tools used in the proxy server and in testing the performance of web caching algorithms. These tools were found through online search and references in published articles. Although there are many other software options out there, these were selected based on documentation, popularity, availability, and compatibility. The software documentation had to be thorough to be able to install and run the software in a short span of time. Popular tools tend to have more resources available online through websites, forums, and streaming video sites such as Youtube. In terms of availability, only free tools were identified as candidates, mainly for financial reasons. The tools had to be compatible with the existing hardware, and only the Windows operating systems was available. The tools should not require more than two computers and a router. These requirements were all satisfied with some challenges.

Squid

Squid is an open-source proxy software developed for Linux and Windows operating systems which started in the early 1990's [21]. Besides web caching, it has many other features such as web access control and Domain Name System [11]. It offers instructions on how to setup in a Windows environment as the servers used in this experiment were Windows-based. Currently, it is being used by many internet providers [11]. It comes with a log analysis tool that provides information such as hit rate and byte hit rate. Its settings such as selecting different algorithms and cache storage size are edited through a configuration file. When Squid is running, its settings cannot be modified until it is stopped. Although there are many other software tools out there, Squid was selected based on documentation, popularity, availability, and compatibility.

Other candidates were Varnish, Nginx, Apache Traffic Server and Polipo. Nginx, Apache Traffic Server, and Polipo were not considered because these do not support Windows operating system that is the only operating system available for testing [22,23,24]. Varnish supports Windows but the monitoring tool did not have byte hit rates [25,26,27,28].

Selenium

Selenium is an open-source web automation tool primarily used to validate contents and functionalities of web pages. It is popular in many companies for front-end validation. It supports many browsers, such as Firefox, and it is offered in many programming languages, such as Java, for writing automation scripts. It also has a record-and-play plug-in available in Firefox so it can be used by beginners. It supports Linux, Windows and Macintosh operating systems. For this thesis, it is used to automate web browsing. This was also selected based on the same criteria as Squid: documentation, popularity, availability, and compatibility.

Other candidates were Wisconsin Proxy Benchmark, Polygraph, and Chrome Browser Automation. To make Squid work is to imitate how a user would browse the internet, which is to visit websites and do it for many hours. Since this would require a lot of manual labor, automation or benchmarking software was to be used. Benchmarking software is specifically designed to test the performance of proxy servers by generating web traffic and providing performance data. Wisconsin Proxy Benchmark was a good candidate because of many factors. It provided caching statistics including hit rate and byte hit rates [29]. The setup required a client, a proxy, and a content server [29]. Figure 2 shows the setup. It was referenced in published journals so it proved its purpose [12,19]. However, the documentation was very limited in building and running the software. It did not mention what operating systems it supports so it would require trial and error.

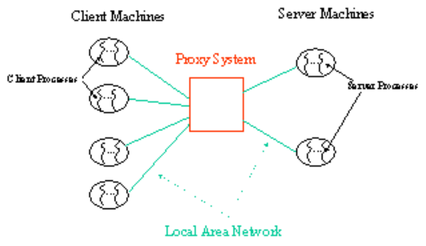


FIGURE 2. Architecture of Wisconsin Proxy Benchmark [29].

Polygraph was another benchmark software that was better than Wisconsin Proxy Benchmark in many aspects. First, it was well documented [30]. It had details on how to setup, build and run it on Linux and Windows environments [31]. It also had scripts designed for specific scenarios such as testing proxy servers [32]. It was referenced in published journals so it proved its purpose as well [19,33,34]. However, one major problem was its steep requirements on hardware that was not attainable at the time. It requires, in addition to proxy and client servers, a Domain Name System server, a content server, and a switch [32]. Figure 3 shows the setup. Because this software tries to be as realistic as possible by creating and using domain names, it was strongly recommended for a closed network for security purposes [35]. The setup requirements were unable to be fulfilled due to lack of resources so it was no longer considered.

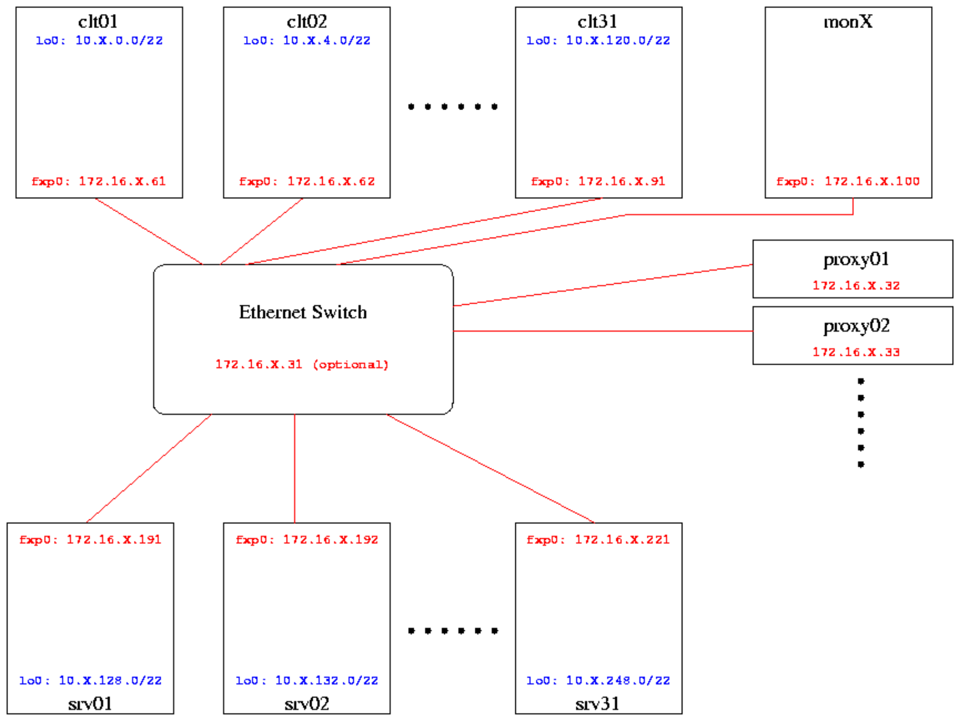
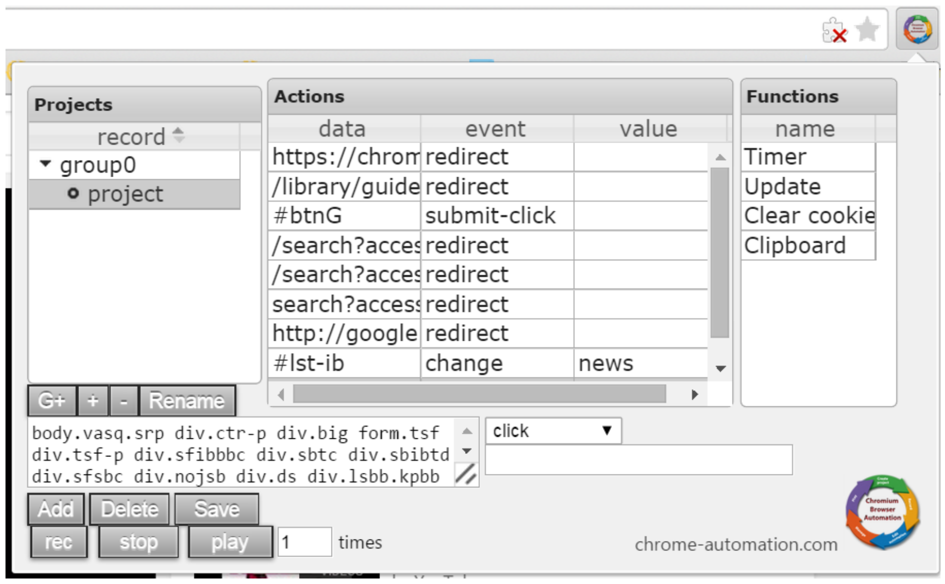


FIGURE 3. Architecture of Polygraph [32].

Chrome Browser Automation was a Chrome plug-in and it was promising because it was based on record and play function. Figure 4 shows the interface. The plug-in can record every action performed in the browser and replay the session. However, there were some major limitations. The interface was basic and limited for editing. It would crash after some usage. This required constant monitoring and restarting experiments that were time-consuming. The plug-in would not wait for the web pages to fully load so adding delays were attempted so that all of the objects were cached. This failed often as they were frequently skipped. Overall, it was not fit for this experiment.



data	event	value
https://chrom	redirect	
/library/guide	redirect	
#btnG	submit-click	
/search?acces	redirect	
/search?acces	redirect	
search?access	redirect	
http://google	redirect	
#lst-ib	change	news

name
Timer
Update
Clear cookie
Clipboard

```
body.vasq.srp div.ctr-p div.big form.tsf
div.tsf-p div.sfibbbc div.sbtc div.sbibtd
div.sfsbc div.nojsb div.ds div.lsbb.kpbb
```

FIGURE 4. Screenshot of a Chrome Browser Automation interface.

CHAPTER 4

EXPERIMENT

The hardware setup is done as follows. For the client computer, Compaq Presario desktop is used. Its operating system is 64-bit Windows 7 Home Premium. Its processor is AMD Athlon II X2 215 with a clock speed of 2.70 gigahertz. Its RAM is 3.00 gigabytes. For the proxy server, Lenovo Thinkpad laptop is used. Its operating system is 64-bit Windows 7 Enterprise. Its processor is Intel Core i5-4300M with a clock speed of 2.60 gigahertz. Its RAM is 8.00 gigabytes. Both computers are connected to an Actiontec DSL router via Ethernet connections. This completes the hardware setup.

The experiments are designed as follows. Selenium runs on a client server and Squid runs on a proxy server. Web browsing is performed by a client server and all of its network traffic is re-directed to a proxy server via a router. The web browser is configured not to cache anything. Figure 5 shows the hardware setup.

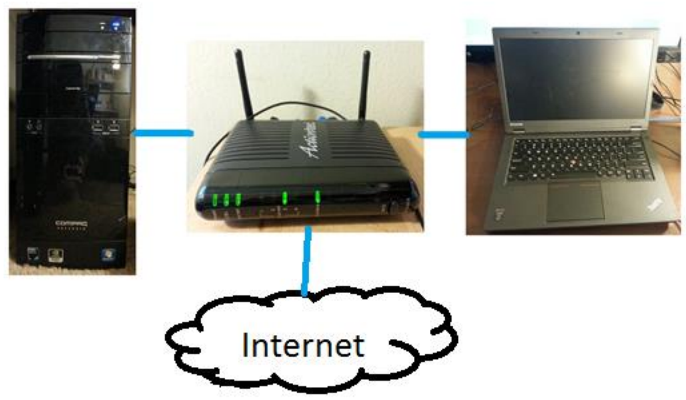


FIGURE 5. Hardware setup for this experiment.

The first test opens home pages of 30 websites that have a large amount of web content, such as Yahoo and eBay. The list of 30 websites is selected from the top 500 most visited websites worldwide according to a web analytics site, and having a large amount of content [36]. Popular websites such as Google are not considered due to lack of content on its homepage. The sizes of the objects in the list of 30 websites vary from a few hundred bytes to hundreds of kilobytes. There are a total of 2,891 objects in all of the web pages that is 40 megabytes in total. Thus the average size of an object would be about 14 kilobytes.

The second test views streaming videos. Because Squid requires extensive modification in the configuration to cache streaming videos, for this experiment it is substituted to accessing large images. This is determined by comparing the size of few-minute videos from Youtube and large images, and it is found that they are identical in size. The large images are found from Google. From this point onwards, the images are referred to as streaming videos. The sizes of the videos or objects vary from a megabyte to tens of megabytes. There are a total of fourteen objects, one per web page with a total of fourteen web pages, which is 56 megabytes in total. The average object size would be approximately four megabytes. The first and second tests are to represent two ends of a spectrum where the first test finds many small objects and the second test finds few, large objects. In other words, it is to represent two types of web users: users who check web content and other users who watch streaming videos.

The third and last test is the mix of web content and streaming videos to see the overall effect. The total size of objects is therefore 96 megabytes. Each test uses three algorithms: Least Recently Used, Least Frequently Used with Dynamic Aging, and

Greedy Dual-Size Frequency. Each algorithm is tested with 10%, 18%, 25%, 50% and 75% cache storage sizes with respect to the total size of all web content, streaming videos and a mix of web content and streaming videos. Five different cache storage sizes are selected to show a broad spectrum. For example, with the cache storage size at 50% for visiting web content pages, Squid would store about 1,450 objects or 20 megabytes in total and the test will still visit all of the web pages. Thus the total number of experiments is 45; it is a product of three types of tests, three algorithms, and five cache storage sizes. Each experiment is conducted for one hour based on initial findings where the data were becoming consistent after running for one hour. The web pages are visited randomly at each test. One instance of a random run is analyzed in Figure 6. At the end of each test, Squid's configuration file, a screenshot of the log analysis tool at one hour, and logs are saved as experimental data. The logs and cache folders are cleared before the next test. This covers the approach of the experiment.

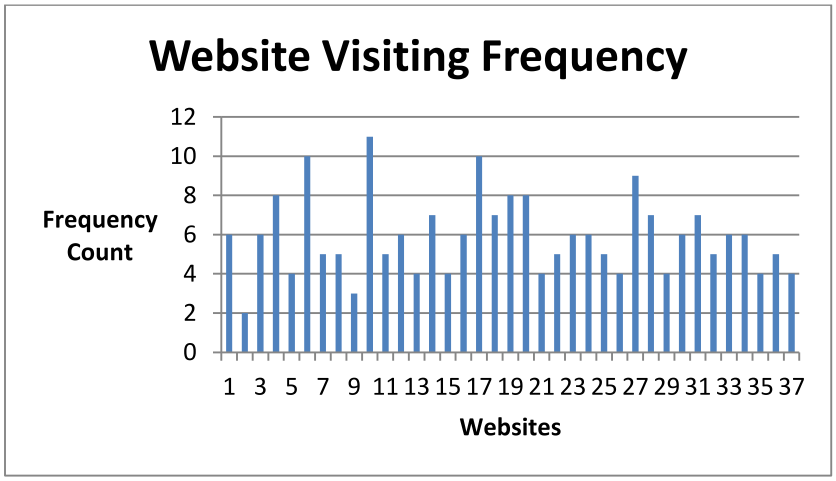


FIGURE 6. Statistical analysis of a test run. This shows how frequently each web page was visited in this run. For example, website one was Yahoo which was visited six times and website two was CNN which was visited twice. The statistics vary for each run.

CHAPTER 5

RESULTS

This section analyzes hit rates and byte hit rates from three tests: web content, streaming videos, and a mix of web content and streaming videos. All graphs have either hit rate or byte hit rate in the y-axis and cache storage size on the x-axis. The value for y-axis is in percentage and for x-axis it is in megabytes. The cache storage size represents how much of the total objects visited are cached. The total size of visited web objects is different for each test: 40 megabytes of web content, 56 megabytes of streaming videos, and 96 megabytes in a mix of web content and streaming videos.

The first graph, shown in Figure 7, is the hit rate for visiting web content. The hit rate for visiting web content shows that Greedy Dual-Size Frequency outperformed other algorithms. This is expected because the design of GDSF maximizes the cache storage by choosing smaller objects. GDSF's hit rate increases four-fold from 7.6% to 30.2%. Least Frequently Used with Dynamic Aging performs the worst because it caches objects of any size, thus reducing chances of getting hits. LFUDA's hit rate increases eight-fold from 2.4% to 25.4%. Least Recently Used does worse than GDSF because LRU also stores any sized objects. It does slightly better than LFUDA as the cache size increases. This is probably because LFUDA caches large and frequently visited objects so it holds fewer objects than LRU. The hit rate of LRU increases at the highest: nine-fold from 2.9% to 28%. All seem to have a uniform increase over cache storage sizes. It is clear that increasing storage size has the expected positive effect on web caching.

TABLE 2. Hit Rate for Web Content

Size	LFUDA	GDSF	LRU
4 MB	2.4%	7.6%	2.9%
7 MB	7.2%	11.9%	8.6%
10 MB	10.2%	15.6%	11.9%
20 MB	18.7%	24.4%	22.5%
30 MB	25.4%	30.2%	28.0%

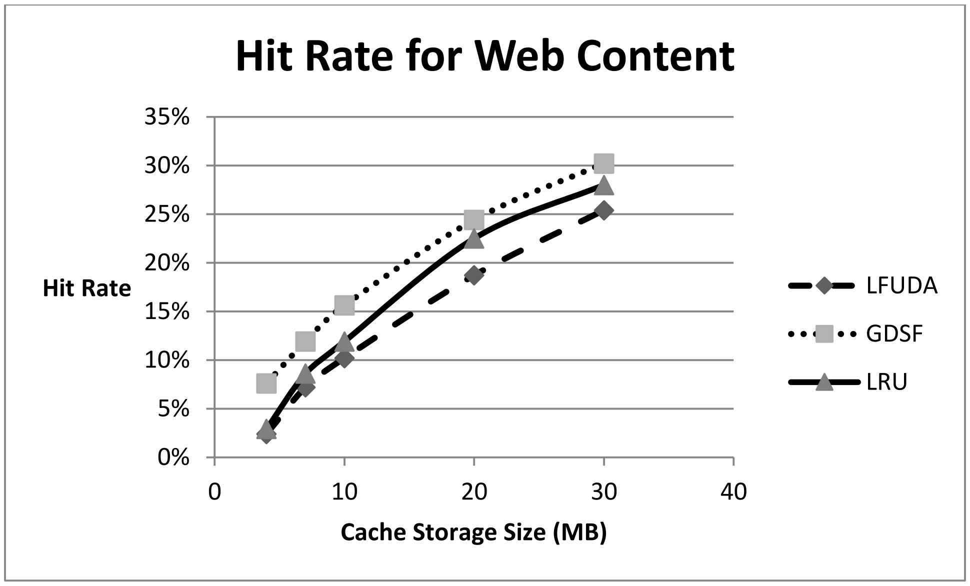


FIGURE 7. Hit rate for visiting web pages with content.

The next graph, shown in Figure 8, is the byte hit rate for visiting web content. LRU maintains highest byte hit rate at all cache storage sizes. This is expected because its algorithm does not consider frequency as others do. Since the tests are created to visit web pages randomly, the frequency factor actually has a negative effect on byte hit rate. Visiting web pages randomly means that algorithms need to ignore frequency if they need to get adjusted to new patterns quickly. Since LFUDA and GDSF place some weight on frequency, frequency makes GDSF and LFUDA harder to reorganize their cache for new objects. GDSF performs the worst because it prefers to store smaller objects. In order to get high byte hit rate, it requires many small objects to get hit which is more difficult statistically. LFUDA does better than GDSF but worse than LRU because of the same reasons explained earlier, regarding frequency and size. LFUDA keeps any sized objects, therefore it has higher chances of getting hits on larger objects than GDSF. LRU and LFUDA seem to have the most dramatic increase in byte hit rate by a factor of nine, from 3.7% to 34.2% and 2.7% to 32.3%, respectively. GDSF increases by a factor of almost eight, from 2.7% to 28.2%. It is clear that increasing storage size has the expected positive effect on web caching.

TABLE 3. Byte Hit Rate for Web Content

Size	LFUDA	GDSF	LRU
4 MB	2.7%	2.7%	3.7%
7 MB	10.1%	6.0%	10.9%
10 MB	13.0%	9.9%	16.3%
20 MB	22.0%	21.6%	30.4%
30 MB	32.3%	28.2%	34.2%

The next two paragraphs discuss hit rate and byte hit rates for viewing streaming videos. Again, GDSF outperforms others in the hit rate, shown in Figure 9, because it can store more objects based on smaller size. LRU performs worst overall. However, it has the highest increase in hit rate with a factor of about four, from 6.1% to 21.7%. GDSF and LFUDA do not show significant increase except between the cache storage sizes of ten and fourteen megabytes. The reason could be that objects with size greater

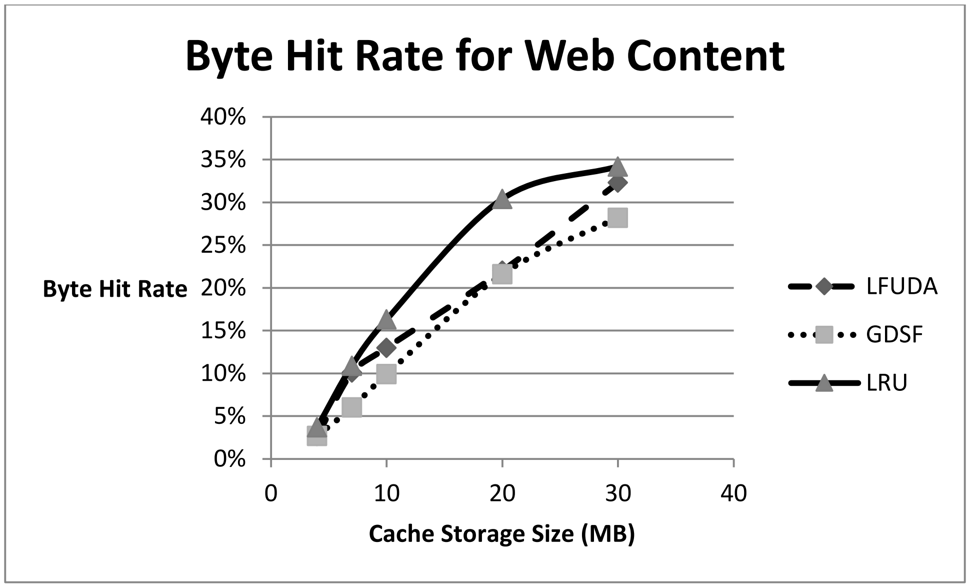


FIGURE 8. Byte hit rate for content.

than ten megabytes are now able to be cached. This could also explain why the hit rates of LFUDA and GDSF do not increase proportionately to the cache storage size. However, LRU's hit rate starts increasing substantially and this may be because of its disregard to frequency. The hit rates for GDSF and LFUDA increase by a factor of 1.5. The overall hit rate is less than the overall hit rate from web content. This is expected because there are fewer objects. It is clear that increasing cache storage size has positive effect on caching on all algorithms.

TABLE 4. Hit Rate for Streaming Videos

Size	LFUDA	GDSF	LRU
6 MB	11.8%	14.4%	6.1%
10 MB	12.4%	15.2%	9.0%
14 MB	16.2%	19.7%	8.2%
28 MB	18.3%	22.2%	15.6%
42 MB	18.1%	23.5%	21.7%

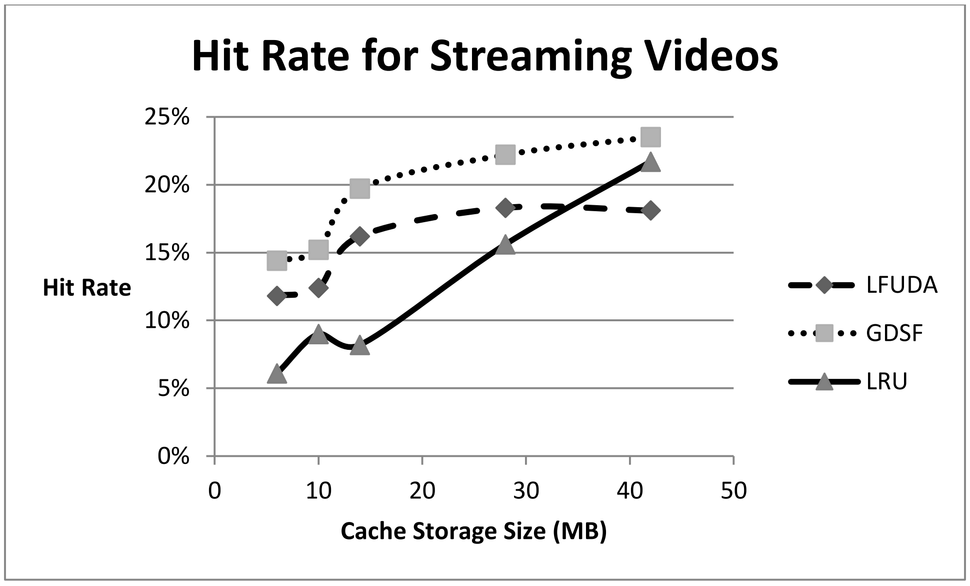


FIGURE 9. Hit rate for streaming videos.

According to Figure 10, LRU has the highest byte hit rate for streaming videos, with the same reasoning from byte hit rate for web content. GDSF and LFUDA are almost the same at each cache storage size. A probable reason for showing similar performance would be because there are very few objects to experiment with. LRU grows by a factor of eleven, from 5.4% to 61.8%. LFUDA and GDSF grow by a factor of nine, from 5.8% and 4.8% to 45.5% and 48%, respectively. It is clear that increasing cache storage size has positive effect on caching on all algorithms. The overall byte hit rate is much larger here than for visiting content web pages that is expected because every hit translates to large byte hits.

The last two paragraphs discuss the results of mixing web content and streaming videos. GDSF does best overall in hit rate, as shown in Figure 11, which is expected

TABLE 5. Byte Hit Rate for Streaming Videos

Size	LFUDA	GDSF	LRU
6 MB	5.8%	4.8%	5.4%
10 MB	7.0%	6.5%	14.1%
14 MB	18.9%	19.9%	20.3%
28 MB	30.7%	35.4%	39.5%
42 MB	45.5%	48.0%	61.8%

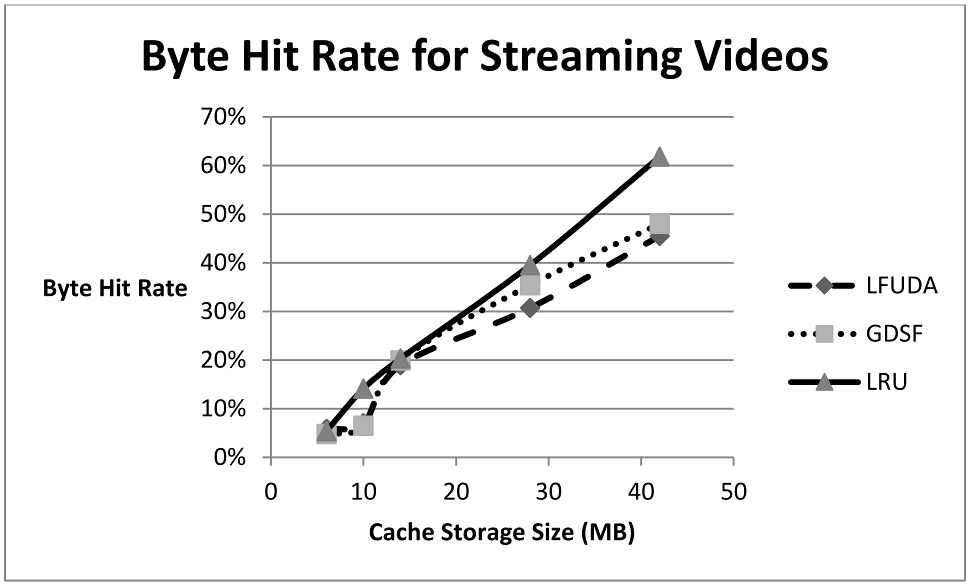


FIGURE 10. Byte hit rate for streaming videos.

because it outperformed in previous two tests. Its hit rate doubled from 16.6% to 38.7%. LFUDA and LRU are evenly matched, with their hit rates growing by a factor of four. A sudden jump in hit rate at seventeen megabytes could be due to the cache storage size being big enough to cache objects with size greater than ten megabytes, as discussed earlier in regards to Figure 9. The overall hit rate is also larger than any previous scenario, which is expected since this is a mix of web content and streaming videos. It is clear that increasing cache storage size has positive effect on caching on all algorithms.

TABLE 6. Hit Rate for Web Content and Streaming Videos

Size	LFUDA	GDSF	LRU
10 MB	10.7%	16.6%	7.9%
17 MB	9.8%	24.4%	9.0%
24 MB	12.3%	24.1%	14.7%
48 MB	20.9%	35.7%	24.2%
72 MB	36.4%	38.7%	34.1%

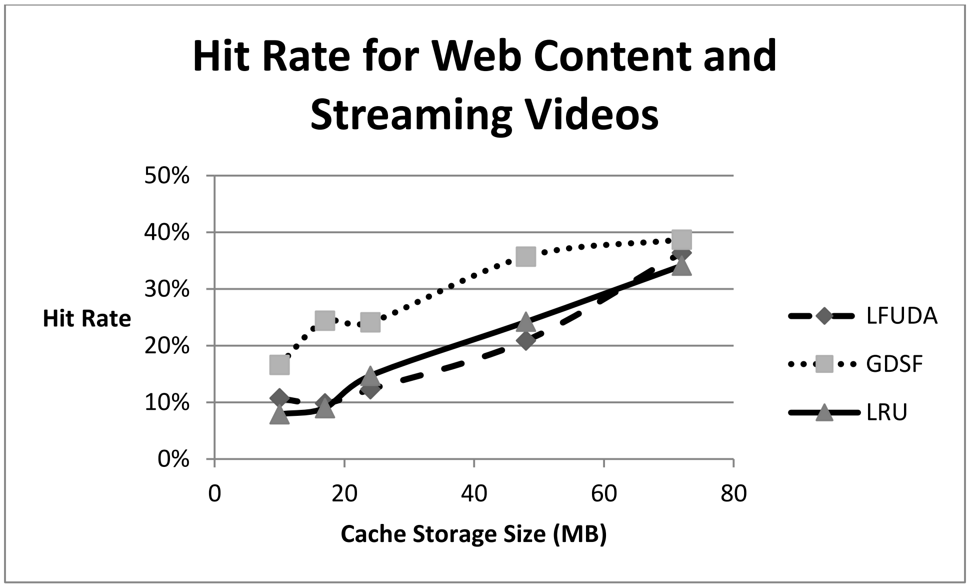


FIGURE 11. Hit rate for web content and streaming videos.

Finally, LRU does best in byte hit rate, as shown in Figure 12. This is expected since it did well in previous scenarios. GDSF and LFUDA are almost identical. The byte hit rate of LRU increases by a factor of five, from 10% to 48.7%. GDSF and LFUDA increase by a factor of seven. The overall byte hit rate is better than the byte hit rate of web content and worse than the byte hit rate of streaming videos. This is because smaller objects have a negative impact while larger objects have a positive impact on byte hit rate. It is clear that increasing cache storage size has positive effect on caching on all algorithms.

TABLE 7. Byte Hit Rate for Web Content and Streaming Videos

Size	LFUDA	GDSF	LRU
10 MB	8.5%	5.6%	10.0%
17 MB	7.3%	10.4%	11.1%
24 MB	12.6%	12.4%	18.3%
48 MB	32.3%	33.5%	35.4%
72 MB	41.1%	41.7%	48.7%

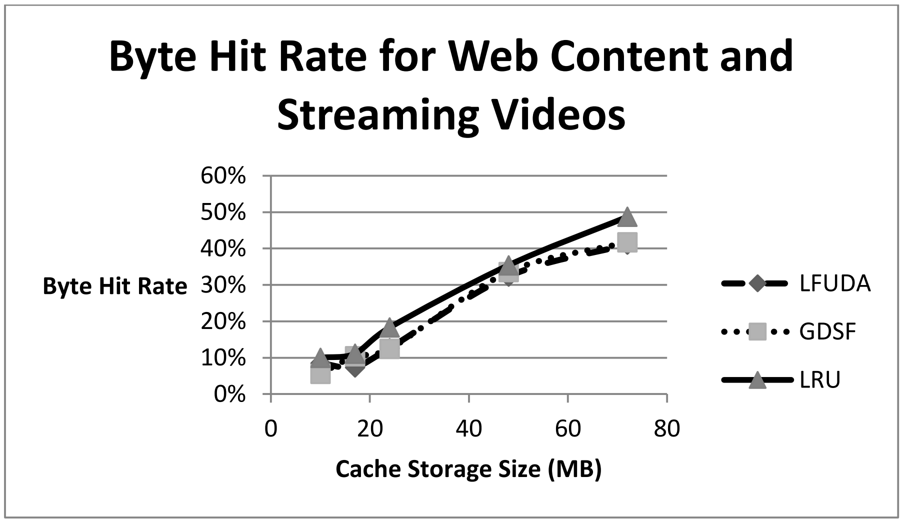


FIGURE 12. Byte hit rate for content and streaming videos.

CHAPTER 6

CONCLUSION

In this study, the performance of a web caching software on a home network was examined. The motivation for this experiment was due to significant increase in web traffic, with streaming video taking an ever-larger percentage of the total web traffic. A brief overview of web caching algorithms was discussed to provide a basic overview. For this experiment, Squid was used to compare performances of its set of web caching algorithms. The tests were composed of viewing web content, streaming videos, and a mix of web content and streaming videos.

The results showed that Greedy Dual-Size Frequency performed best in hit rates while Least Recently Used was the highest in byte hit rates. GDSF had the best hit rate because it maximized caching by keeping smaller objects which was expected. LRU had the best byte hit rate because it did not consider frequency. In comparison to research papers, the results nearly matched on hit rates even though the tests were not similar. Every result showed that the hit rate and byte hit rate increased proportionately to cache storage size, as expected.

There is no clear choice on which algorithm should be used to satisfy both metrics. However, since the testing was performed randomly and had a limited scope, additional testing must be done to be able to draw generalized real-world recommendations. In real internet traffic, there may be patterns to the searches users perform. Further tests are needed to determine which algorithms work best taking into account the possible non-random access done by users. With less random testing, patterns may emerge in web traffic that result in changes in algorithm performance. It is quite possible that the frequency aspect of the web caching algorithms will play a larger role. In hit rate, GDSF may still have the highest performance. In byte hit rate, Least Frequently Used with Dynamic Aging may have the highest performance.

Future Work

In order to perform experiments more realistically, more exhaustive testing on the web caching algorithms must be performed. First, if patterns exist in web browsing behavior, the patterns must be identified and a method for replicating them must be developed. The tests will need to be designed so that visiting the web pages is done following the users' patterns. This may be able to be achieved by modifying the automation script so that some web pages are referenced more frequently than others. Another approach would be to incorporate the web browsing history of real users where the automation scripts visit web pages based on users' web browsing history. Furthermore, using benchmarking software such as Polygraph may show how results can vary based on the methods of testing. Another aspect that can be improved is the scale of testing. Significantly increasing the number of websites and streaming videos will need to be done. Increasing the duration of streaming videos from 30 to 120 minutes is suggested to represent television shows and movies. Since there are other web caching software tools available, other software can also be tested for comparison. There are many variations that can improve this experiment.

REFERENCES

REFERENCES

[1] Cisco, “Cisco Visual Networking Index: Forecast and Methodology; 2013–2018,” 2014; http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html

[2] GMSA Intelligence, "Global Data," 2015; https://gsmaintelligence.com

[3] C. Palermino, "Online Video Will Account for 80 Percent of the World’s Internet Traffic by 2019," 2015; http://www.digitaltrends.com/home-theater/online-video-will-dominate-internet-traffic-by-2019/

[4] I. Greenwald, "The Cable Bill's Too High. Here's Why," 2013; http://www.forbes.com/sites/igorgreenwald/2013/01/15/the-cable-bills-too-high-heres-why/

[5] Digital Media World, "Akamai Sees a Growing Future for Streaming Video On Demand," 2015; http://www.digitalmedia-world.com/Disrupt/akamai-streaming-video-on-demand

[6] D. Rayburn, "Carriers Facing New Challenge of Caching Huge Demand for Live Video Traffic," 2014; http://blog.streamingmedia.com/2014/04/carriers-facing-new-challenge-caching-huge-demand-live-video-traffic.html

[7] The Economist, "As Online Video Continues to Boom, Publishers are Exploring New Ways to Deliver their Content Reliably," 2014; http://www.economist.com/news/technology-quarterly/21635325-online-video- continues-boom-publishers-are-exploring-new-ways-deliver

[8] A. Thompson and C. Rahn, "Netflix May Strain European Networks on Video Demand," 2014; http://www.bloomberg.com/news/articles/2014-09-03/netflix-may-strain-european-networks-as-streaming-demand-swells

[9] M. Z. Shafiq, A. X. Liu, and A. R. Khakpour, “Revisiting Caching in Content Delivery Networks,” *SIGMETRICS*, June 2014, pp. 567-568.

[10] W. Ali, S. Shamsuddin, and A. Ismail, “A Survey of Web Caching and Prefetching,” *Int. J. Advance. Soft Comput. Appl.,* March 2011, Vol. 3, No. 1.

[11] Squid, "Squid: Optimising Web Delivery," 2013; http://www.squid-cache.org

[12] S. Jarukasemratana and T. Murata, "Web Caching Replacement Algorithm Based on Web Usage Data," *New Generation Computing*, July 2013, pp. 311-329.

[13] W. Ali et. al., “Intelligent Web Proxy Caching Approaches Based on Machine Learning Techniques,” *Decision Support Systems*, May 2012, pp. 565-579.

[14] L. Cherkasova, “Improving WWW Proxies Performance with Greedy-Dual-Size-Frequency,” Hewlett Packard, November 1998, HPL-98-69.

[15] E. Markatos, “On Caching Search Engine Query Results,” *Computer Communications,* February 2001, pp. 137-143.

[16] T. Johnson and D. Shasha, “2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm,” *Very Large Data Bases*, September 1994, pp. 439-450.

[17] A. Songwattana, T. Theeramunkong, and P. Vinh, “A Learning-Based Approach for Web Cache Management,” *Mobile Networks and Applications*, March 2014, pp. 258-271.

[18] W. Ali, S. Shamsuddin, and A. Ismail, “Intelligent Naïve Bayes-based Approaches for Web Proxy Caching,” *Knowledge-Based Systems*, March 2012, pp. 162-175.

[19] J. Dilley, M. Arlitt, and S. Perret, "Enhancement and Validation of Squid’s Cache Replacement Policy," Hewlett Packard, May 1999, HPL-1999-69.

[20] S. Jin and A. Bestavros, “Popularity-Aware Greedy Dual-Size Web Proxy Caching Algorithms,” *Distributed Computing Systems*, April 2000, pp. 254-261.

[21] Squid, “What is Squid?,” 2013; http://www.squid-cache.org/Intro

[22] Nginx, "Nginx for Windows," 2015; http://nginx.org/en/docs/windows.html

[23] Traffic Server, "Supported Operating Systems," 2015; https://cwiki.apache.org/confluence/display/TS/Supported+Operating+Systems

[24] Polipo, "Polipo - a Caching Web Proxy," 2015; http://www.pps.univ-paris-diderot.fr/~jch/software/polipo/

[25] Varnish Cache, "Installing Varnish from Source Code," 2015; https://www.varnish-cache.org/trac/wiki/Installation

[26] Varnish Cache, "Stats Explained," 2015; https://www.varnish-cache.org/trac/wiki/StatsExplained

[27] Kristian Lyngstøl's Blog, "Varnishstat for Dummies," 2009; http://kly.no/posts/2009_12_08__Varnishstat_for_dummies__.html

[28] Varnish Cache, "Varnish Settings," 2011; https://www.varnish-cache.org/ trac/wiki/Performance

[29] J. Almeida and P. Cao, “Wisconsin Proxy Benchmark 1.0,” 2015; http://pages.cs.wisc.edu/~cao/wpb1.0.html

[30] Web Polygraph, "Documentation," 2015; http://www.web-polygraph.org/docs

[31] Web Polygraph, "Getting Started," 2015; http://www.web-polygraph.org/docs/userman/start.html

[32] Web Polygraph, "PolyMix-4," 2015; http://www.web-polygraph.org/docs/workloads/polymix-4/

[33] D. Lee and K. Kim, “A Study on Improving Web Cache Server Performance Using Delayed Caching,” *IEEE*, April 2010, pp. 1-5.

[34] A. Rousskov and D. Wessels, "High Performance Benchmarking with Web Polygraph,” *Software: Practice and Experience,* January 2004, pp. 187-211.

[35] Web Polygraph, "Using DNS," 2015; http://www.web-polygraph.org/docs/userman/dns.html

[36] Alexa, "The Top 500 Sites on the Web," 2015; http://www.alexa.com/topsites