Analysing Redundancies in the World-Wide-Web

Master of Science Thesis

Jose Luis Puig Guerra
Jose.puig.guerra@aalto.com

Thesis supervisor: Jörg Ott
Thesis instructor: Pasi Sarolahti

Helsinki / 2014

Aalto University
School of Electrical Engineering

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Jose Luis Puig Guerra

Title: Analysing Redundancies in the World-Wide-Web

Date: 22.05.2014 Language: English Number of pages: 9+60

Department of Communications and Networking
Professorship: Code:

Supervisor: Prof. Jörg Ott
Instructor: Dr. Pasi Sarolahti

The World Wide Web is one of the most relevant Internet applications, and it is an important tool for our daily lives. Although it is widely extended, the current web access is still limited by two factors; the poor infrastructure in developing countries and the increasing bandwidth demand for services such as cloud computing or video streaming. Web caches have become a feasible solution to improve web access since improve on network infrastructures is very expensive.

Multiple studies in past years aimed to characterize web traffic in order to improve web caching. However, the WWW evolves very fast and previous studies about it are no longer reliable. Moreover, many of the studies are based on passive measurements by collecting traces at the edge of an organization. As a result, we miss little knowledge on current web traffic. This thesis attempts to study present web traffic and how caching systems can benefit from it.

We have developed an active measurement system that downloads popular web pages during a short period of time. We analyse this data set from two different points of view: compare old published web traffic and examine dynamic changes of web content. Finally, we investigate the unchanged content of this data set using both caching approaches traditional web caching and packet caching. Among our findings, we observe similar bandwidth saving for both approaches as well as an increasing number of objects per page.

Keywords: Caching, web site, traffic, bandwidth, web page, WWW, Unchanged bytes

This Page Intentionally Left Blank

Abstract

The World Wide Web is one of the most relevant Internet applications, and it is an important tool for our daily lives. Although it is widely improved over time, the current web access is still limited by two factors; the poor infrastructure in developing countries and the increasing bandwidth demand for services such as cloud computing or video streaming. Web caches have become a feasible solution to increase web access since improving network infrastructure is very expensive.

Multiple studies in past years aimed to characterize web traffic in order to improve web caching. However, the WWW evolves very fast and previous studies about it may not be reliable anymore. Moreover, many of the studies are based on passive measurements by collecting traces at the edge of an organization. Past studies have not analysed the change rate of web content over time, which is relevant information for assessing benefits of caching. This thesis studies present web traffic and how caching systems can benefit from it.

We have developed an active measurement system that downloads popular web pages at short time intervals. We analyse this data set from two different points of view: compare old published web traffic and examine dynamic changes of web content. Finally, we investigate the unchanged content of this data set using both caching approaches traditional web caching and packet caching. Among our findings, we observe similar bandwidth saving for both approaches as well as an increasing number of objects per page.

Contents

List of Acronyms and Abbreviations

AJAX	Asynchronous JavaScript and XML
ASCII	American Standard Code for Information Interchange
CDF	Cumulative Distribution Function
CDN	Content Delivery Network
CGMP	Cache Group Management Protocol
CRP	Content Routing Protocol
CSS	Cascading Style Sheets
DNS	Domain Name System
DOS	Denial of Service Attack
FTP	File Transfer Protocol
GPL	General Public License
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IP	Internet Protocol
ISP	Internet Service Provider
LAN	Local Area Network
LFU	Least Frequently Used
LRU	Least Recently Used
MIME	Multipurpose Internet Mail Extensions
RSS	Real Simple Syndication
RTT	Round Trip Time
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WAN	Wide Area Network
WPAD	Web Proxy Auto-Discovery Protocol
WWW	World Wide Web
XML	eXtensible Markup Language

List of Figures

List of Tables

Chapter 1

Introduction

As of today Internet is one of the most important tools that we have for business, a good way to keep us well informed about what is happening over the world and recently an important aspect about our social lives and leisure. Its relevance has become so important that United Nations has proposed Internet as a human right [1].

1.1 Problem and motivation

Internet is a platform where many services coexist. We see it in daily actions such as sending emails, watching streaming videos or read online newspapers. Also it is the core for new applications, for instance, cloud computing [2] or games [3]. At the same time, as new and old services evolve, the traffic generated on Internet grows. Although new applications generate larges amounts of data, actually peer-to-peer services stand as the main contributor of Internet's traffic followed by web traffic [4]. Despite its relevance in our life, web access is still limited by two factors: the growing demand of bandwidth by services such as Skype or Cloud computing, and limited network infrastructures in developing countries [4].

Web caching is a common technique to save bandwidth, reduce load from the servers and improve resilience. Web caches works by storing web documents visited by users in order to serve them in future requests. The cached content must be updated or the user may get an outdated document. There are many caching solutions depending on where web caches are placed, which method is used to determine which documents are cached or how these documents are maintained updated. Despite the multitude of solutions to deploy caches mechanisms, none of them differentiate between web services, even when they are clearly different.

The characterization and comprehension of redundant web traffic is necessary to generate synthetic workload for benchmarking on web servers [5]. Studies of web traffic also provide techniques to identify shortcomings. For example, by studying the loading time of a web page we could determine whether it exceeds the tolerable waiting time for users [6]. If this time is longer than the user is willing to wait, the user might not visit the web site again.

Analysing web traffic characteristics help us understand how useful web caching in the Internet is today. Different studies in past years [7] [8] [9] [10] aimed on characterizing and understanding web traffic, but according to [4], the number of studies in recent years has decreased because of the continuous evolution of the World Wide Web (WWW). On the other hand, many studies quantify the redundancy and cacheability of web sites in order to be implemented on caching mechanisms [11] [12] [13].

Many of the past studies are based on passive measurements, meaning they examine data generated by users. These studies do not control which web sites are accessed. Taking into account that popular web sites become more popular [4] and 90% of the users go deeper into the web site [14], much of the web sites are not completely analysed.

1.2 Goals

Since web access is difficult to be improved by updating the infrastructure for economical and practical reasons, web caches stands as a feasible solution to save bandwidth and increase web access. Before implementing a web cache, we need to determine the potential impact of this cache and its benefits. Studying the WWW is a tool to quantify the potential impact of web caches and its benefits. Since the World Wide Web change very fast, the properties of web content needs to be analysed regularly.

We have designed and implemented a measurement system for analysing web content characteristics and the content dynamics over time. We analyse changes in web content changes at well-defined constant intervals, which is not possible with passive measurements. We have designed and implemented a software solution based on active measurements, in other words, it is able to select the web sites. This application download all content in the web pages and analyse it.

We compare earlier published web traffic models with data collected by the measurement system, to determine possible long-term changes in the web traffic characteristics. Old published web traffic models need to be updated because the WWW evolves continuously. It does not mean necessarily that old traffic models are not valid, but we want to determine whether models have changed over time. We compare our results to earlier published works to determine possible changes in the traffic characteristics.

We study changes on the content of popular web pages over short time periods to analyse the potential impact of different caching strategies. Web traffic is a wide area to research. We have focused on examining unchanged content by storing web pages during a short period of time. The data is analysed by our measurement system identifying content changes and how these changes could benefit caching mechanisms.

1.3 Thesis outline

The thesis is structured as follows:

Chapter 2 describes the basics of WWW. It begins describing the protocols and content types used on the WWW, followed by previous researches on Web traffic.

Chapter 3 details different caching architectures as well as some enhancements on web caching. This section also covers other method of caching, packet caching. Finally, it ends describing common cache management algorithms.

Chapter 4 describes the purposed solution in detail. Firstly, explaining the methodology and metrics used and then describing the overall design. The next sections detail the specific parts of the collection and analysis of our dataset. Finally, the section ends evaluating the used tools and determining the present limitations of our proposal.

Chapter 5 evaluates the solution implemented in the abovementioned chapter with previous works. It also studies the current state of the WWW by determining the redundancy and how it could benefit web caches.

Chapter 6 gives the conclusions obtained through all this work. The section ends giving some ideas that could be helpful in future works.

Chapter 2

World Wide Web

Because this thesis compare old studies on web traffic, we will look at the prior work on most relevant work done on analysing web traffic. To understand the results from these works, we will summarize the factors involved on web traffic. In this section we will give the basics of HTTP/1.0 and HTTP/1.1 and how users access the web sites. In addition, we explain different types of resources present in web pages, because not all of them have the same cacheability and relevance on the WWW.

2.1 HTTP protocol

Today it is difficult to think on Internet without the World Wide Web. This is because its use has spread among all the daily services such as social networks, video streaming or electronic commerce [15]. In all these cases the protocol used to transport the information is HyperText Transport Protocol (HTTP), which has become the most used protocol on Internet [16]. It is a text-based protocol using the American Standard Code for Information Interchange (ASCII) as character encoding. It is very flexible and can be extended by modifying the methods, main point from where HTTP has become so popular. The communication is based on the request-response paradigm: a user queries a server and the server responds to the user. The WWW uses a Unique Resource Identifier (URI) [17], adapted to the web necessities called Uniform Resource Locator (URL) which identifies the server and where the item is located.

In mid 90's the increasing use of Internet represented a problem for web servers and service providers because the demand of content was not adapted to the available bandwidth. It was partly solved by the improved HTTP/1.1. That version reduced the number of packets sent and the elapsed time as well as better support for web proxies [18] [19]. It also improved its compatibility with different caching systems and it has been object of study ever since. Liang Shuai et al. [20] studied this protocol in the modern web [21] where discusses the rise of the web traffic as consequence of the use of bigger files and the increasing new services such as instant messages or updating applications.

Figure 2.1 shows a typical scenario for a HTTP request between client and server. The user types in the browser a URL belonging to a web site, for instance www.google.fi. The client needs to locate the resource to the remote server through a

Domain Name System (DNS) server. The DNS sends I.P address of the web server to the client. Then the HTTP client sends a HTTP request to the server with the resource (www.google.com/index.html). The server checks the HTTP header and responds to the client with the queried information, in this case index.html.

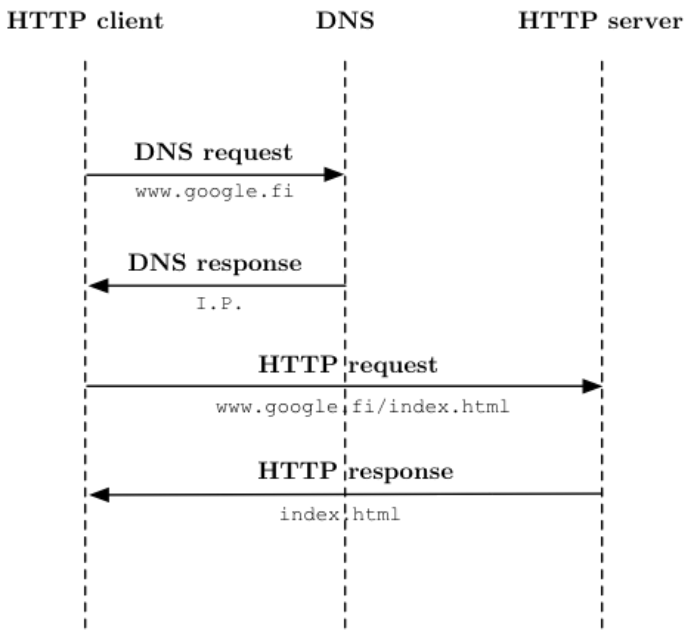


Figure 2.1 HTTP request scenario

Another remarkable point is that the behavior of HTTP is simple; the client contacts the server with a HTTP request, it could contain multiples options such as different languages, and the server responds with a preferred option in order to start the conversation based on a client-server paradigm, as depicted in Figure 2.1. Each message sent by either client or server has three parts: start line, header, and the body of the answer.

Table 2.1 Request methods.

Method	**Description**
GET	Request information for a specific URI
HEAD	Similar to GET but the response only contains the headers
POST	Allows data to be send to a server
PUT	Add a resource
DELETE	Delete the resource
OPTIONS	Request information about the available options at the server

Each HTTP request starts with a method that indicates the purpose of the request. Table 2.1 shows the basics methods used in webs and generated by the client. It has the same basic capabilities than a file: add, request and delete data, this is essentially because meshes of documents form the WWW.

On the server side, the different type of responses is quite large and out of scope in this thesis, therefore we only explain the GET response. Responses do not explicitly tell what the request was, but typically only GET results in cacheable body in response. This request returns the name of the resource. For instance, the file that contains the logo of Google is visible from the URL: http://www.google.com/images/srpr/logo4w.png. The first part of the URL is actually a DNS name (http://www.google.com), and the rest, (/images/spr/logo4w.png), is parsed by the HTTP server after a HTTP request. The information parsed by HTTP typically refers to the logical structure of the web site.

The web server responds with a three-digit number, which is interpreted by the client, and it acts consequently. Table 2.2 summarizes the messages sent by the servers. The first number identifies the type of response and the last two numbers identify the exact message. For instance if we receive the code 404 we know that the identified resource was not found, which is attributed as client error.

Table 2.2 Server response.

Response	Description
2xx	Success
3xx	Redirection
4xx	Client error
5xx	Server error

Nevertheless, web caches do not handle all responses, only successful requests are taken into account. Hence, caches operate only on responses like 200. Some caching systems might break the end-to-end paradigm and some applications may not tolerate (i.e., web commerce or security applications).

Table 2.3 HTTP/1.0 and HTTP/1.1 caching headers

HTTP/1.0	HTTP/1.1
If-Modified-Since	If-Modified-Since
Last Modified	Last Modified
Expires	Expires
Pragma	Pragma
	E-Tag
	If-None-Match
	If-Match
	Cache-control

Table 2.3 shows the HTTP headers defined to be used in caching systems. HTTP/1.0 [22] details only three headers whereas HTTP/1.1 [23] keep the previous

headers and increase the number of fields. The common headers in both versions of HTTP have not been modified to preserve retrospective compatibility between versions. The specific HTTP headers for caching are *If-Modified-Since*, *Last-Modified*, *Expires*, *Pragma*, *ETag*, *If-None-Match*, *If-Match* and *Cache-control*.

The *If-Modified-Since* header field is used along with the GET method to determine if the content has not been modified since the time specified in this field. If the content has not been modified since a specific date, the response is the same than a normal GET with a code 200. On the other hand, if the content has changed since that date, the server returns a modified response, which correspond the code 304.

The server responds with the header *Last-Modified* to indicate the time and date at which the resource was last modified. If the cache has a copy with a lower value in that field, the cached document must be considered stale.

The server could also response with the field *Expires*, which indicates the date and time after which the document is considered outdated. However, that time does not imply that the content has changed or it is going to be modified, it is just an approximation of how long a document could be not modified. If the resource has an *Expires* time earlier than the current date, the document must not be cached because it is still valid.

All three headers have a date as a parameter in standard representation format, defined in RFC 822 [24] and adjusted in RFC 1123 [25]. An example of one header is:

Last-Modified: Sat, 4 May 2013 12:43:22 GMT

The field header *Pragma* is not a specific cache field but it is used for additional instructions for HTTP processing, and it might forbid caches to store documents. Currently it is used in HTTP/1.0 because that version of HTTP has fewer fields dedicated to caching. The directive no-cache ensures that the request will be forwarded to the origin server without storing any copy at the cache. This allows users to receive an authoritative response from the origin server. Caching systems might use it to replace stale or corrupted resources from the cache.

Previous header fields rely on absolute timestamps with one second of resolution, which can lead to caching errors because of clock synchronization errors or lack of resolution. To solve these errors, HTTP/1.1 introduces several header fields such as *Age* or *max-age*. These fields work by counting the seconds until some criteria are fulfilled. The parameter, *Age*, indicates for how long the object has been in cache. It is determined by the web cache instead of the origin server to avoid clock skews. Another significant improvement is the introduction of an opaque identifier, called Entity Tag or *ETag* for brevity. This tag specify the version of the resource in a URL, thus when the resource is modified the origin server produces a new *ETag*. If the identifier does not match with the cached *ETag* means that the resource has changed.

The header *If-None-Match* is used with the method GET to make it conditional. A web client can check whether the resource is still the same by sending the *ETag* in the *If-None-Match* header to the origin server. In the case the *ETag* is not the same, the server will send to the cache an updated version of the resource. Moreover, a cache could request the header *If-Match* to verify if the resource has been modified. Both headers are similar but in the first case the cache checks the freshness of the resource meanwhile the second attempts to verify if the content is still the same.

HTTP/1.1 determines a new request field that must be obeyed by all caching mechanisms, both requests and responses, between the origin server and user called *Cache-control*. That field indicates caches that should first validate with the origin server the resource before send the content to the user. A common technique to prevent stale documents from being sent to the user without previous validation is by setting the directive `max-age=0`, which makes the document stale. *Cache-control* also provides other directives, such as `no-store` that indicates web browsers to not store content. This directive also specifies whether the content is public, cacheable, private or uncacheable, which allows browsers but not proxies to store information.

HTTP/1.1 defines multiple fields but none of them is mandatory for web developers. Around 56% of the resources do not fill the `max-age` field within the HTTP header and 3% have zero value [26]. As we commented above, if the field is left blank or has the value zero, the cache must request the content to the origin server. It means that only 41% of the resources are cacheable. Craig et al. [27] determine that 33% of HTML resources do not contain any cache directive and few contain expiration time.

2.2 Web content types

The WWW is not only composed by text, it has different types of resources such as videos, images or audio. These types of resources are registered by the IANA [28], and are known as Multipurpose Internet Mail Extensions (MIME) types. Originally MIME types were defined for sending other kind of information in email such as pictures or audio [29], but it has been adapted for web pages. The content types have two or more parts: the type, subtype and optional parameters. The top-level type of content indicates which kind of resource is, whereas the subtype indicates the encoding. For example, if the content type of a resource is `image/gif` we can determine that it is a bitmap image. Table 2.4 summarizes the basic content types.

Table 2.4 Media content-types.

Type	**Description**
Application	Application-specific data
Audio	Contains audio formats
Example	Used for examples
Image	Support multiple image formats
Message	For encapsulating mail messages
Model	Reserved for 3D models
Multipart	For objects contained in multiples parts
Text	Used for text and programming code
Video	Contains several videos formats

We do not have detailed each subtype in Table 2.4 because it is in constant expansion and open for new content subtypes though the IANA. Not all content types change in the same degree, some types change more frequently than others. For instance, `Application` messages transmit application data, therefore this data is less static than other MIME types such as `Images` or `Text` because it transmits application-specific data.

The web pages are composed by several MIME types, which together form a complex web page. HyperText Markup Language (HTML) is a mark-up language [22] to control content formatting, and widely language for web pages based on tags. The Cascading Style Sheet (CSS) is a language to describe formatting templates of documents and it is very common in HTML and eXtensible Markup Language (XML). JavaScript is a programming language used mostly in the client side and its use is extended in dynamic web sites. Typically, JavaScript is used to personalize content or to show content dynamically. There is another web technique where CSS, HTML and JavaScript works together called Asynchronous JavaScript and XML, also referred as AJAX [30]. It works at the client side and it is capable of retrieve information from a server without interfering with the behavior of the web page.

Images or other resources are common in web pages, and might be included in a web site as linked or embedded resources. Linked images are not part of the document itself, and have the following format:

```
<IMG SRC="http://www.google.es/images/icons/chrome-48.png">
```

The main drawback is that images cannot be accessed off-line, for that reason these resources use to be implemented as embedded. Unlike linked resources, embedded resources are part of the text and the content is identified using a content ID. The same image than before but embedded could be formatted as:

```
<IMG SRC="id:chrome-48">
```

As we see, there is no information about the format of the image or where it is located. Generally the ID is a mechanism to identify components within the web page, and it might be different than the name of the image (we have used the same file name than before for clarity).

The types `Text` and `Image` are suitable for caching as numerous studies shows [8] [31] [32]. Study the composition of web sites is important for understand the variety of content types, their traffic, as we will comment in the below section, and possible benefits for caching. These studies demonstrate that some content-types are more suitable for caching than others. For instance `text/html` reaches 24% of cacheable content while `image/gif` is about 48% but represents 33% of all bytes compared to the 18% of `text/html` accordingly. Z. Luwei [33] shows the cacheability of different content types and subtypes. Luwei determines a greater cacheability of dynamic content such as JavaScript [34] rather than cascade style sheet [35] (also known as CSS).

Douglis et al. [8] conducted a study through 950,000 web traces where the last-modified timestamp was analysed. They reveal that HTML resources changes more often than images, which almost never change. In addition, 5.9% of the traces had an explicit directive to not be cached via the header `Pragma: no-cache`.

2.3 Analysis of web traffic

Our work is not the first analysing web content, but our approach differs from many studies because we use active measurements rather than passive measurements and also analyse changes over time. Web traffic has been studied for many years in order to improve web performance and bandwidth efficiency. Results of these studies are the improvement on protocols [36] [37] [38] and browsers [39]. However, not all studies use the same methods, for instance, [40] verify the checksum and [41] compare words to determine if the document has changed.

Cho and Garcia-Molina [40] study how web pages evolve over time. They collected 720,000, and conducted several analyses and correlations to understand their evolution over time. They found that a page under domain `.com` does changes faster than domains like `.org` or `.edu`. They reveal that 20% of the pages, most of them under `.com`, changes within a day meanwhile the rest of domains change less than 10%.

The work of Cho and Garcia-Molina was extended by Fetterly et al. [41] by quantifying the degree of the change across different domains. They found a correlation between the frequency of change of a document and the top-level domain (i.e., `.com`, `.edu`). That effect is also seen in [40], where `.com` and `.net` domains change more often than `.edu` or `.gov`. The study also points that the size of a document is a strong predictor of the frequency and degree of the change. Larger documents change more deeply and often than smaller documents.

Cunha et al. [42] accumulated more than 500,000 requests for web documents between 1994 and 1995. They demonstrate that many characteristics of the WWW can be modeled using power-law distributions. They determine that the number of references to documents as function of the document's rank in popularity follows a Zipf's law. It states that the relative frequency of a request for the `i`'th most popular web page is proportional to `1/i` [43]. This power-law distribution helps web caches to determine the distribution of objects that may be cached by looking at the pattern of use of objects. This study is helpful when designing caching policies for web documents as multiples studies determined [44] [45] [46].

Butkiewicz et al. [47] have done a recent and accurate study. They used multiple metrics to determine how web traffic is composed. Moreover, they used Alexa categories to divide the sites. By doing this categorization, they have obtained interesting correlations between categories and metrics. They found that web sites contain around 40 objects per page, whereas web sites related to news exceeds this number largely. They confirmed the previous work done by Sunghwan and Vivek [48] where affirms the increasing use of JavaScript, a minimum of 6 objects per page, and CSS with two. They concluded that 30% of the objects and bytes belong to other domains (e.g., Facebook, Twitter) is even higher in *News* sites reaching 40% of the total number of objects. Their study also reflects the impact of the increasing number of resources at the client side, where more than half of the sites have loading times greater than 2 seconds, closely near to the threshold of the user's frustration [6].

Sunghwan and Vivek [48] have done a worldwide study about Web traffic along five years of data collection, between 2006 and 2010, in 187 countries. They confirm the increasing use of JavaScript, CSS and XML as a result of AJAX. Sunghwan and Vivek also points an increasing use of Flash Video, especially in United States and Brazil, where it is nearly the 25% of the total traffic. They determined that the increasing size of the pages as consequence of the advertisements and analytics, without neglecting the contribution of AJAX. Sunghwan and Vivek gives attention to caching systems where two trends were found; popular URLs gets more popular and the percentage of URLs that are accessed only once is increasing. Other interesting points are the 28% of uncacheable URLs, which represents 14% of the total bytes. Going further, they remark that a significant part of XML traffic is uncacheable, over 70% in China, because they implement Really Simple Syndication (RSS) feeds.

As summary, not all web pages use the available headers present in HTTP. As consequence the number of cacheable document is reduced. The cacheability is also reduced depending on the web content type because images or text are more likely to be cached rather than application data. Earlier works shows an increasing number of objects per page as well as a rising use of new technologies. The following section explains the most common caching architectures and several caching enhancements. The section also details the algorithms to cache management.

Chapter 3

Web caching

This thesis analyses dynamics on web sites to determine the possible impact on web caches. In order to know which factors affect them, we must understand first how caching systems works. In this section we will explain different caching architectures and their implementations as well as recent and old caching improvements. We briefly describe different algorithms used on maintaining caches up-to-date. At the end, we detail a caching approach based on detecting duplicated packets in the network.

3.1 Caching architectures

The goal of caching systems is to reduce the bandwidth and increases the availability of the content if the origin server is offline because the caching system has previously stored the document. The aim of web caches depends on where are located within the network. The most common architectures are proxy caching, reverse proxy caching and transparent caching.

Proxy Cache: Also known as simple caches [49], they are usually located at the edges of a network (i.e., firewall or gateway) to deal with the maximum number of users. Figure 3.1 depicts the typical scenario where a proxy cache receives for first time a user request. First, web clients must to configure their web browsers to identify proxies. The Web Proxy Auto-Discovery Protocol (WPAD) [50] has solved this issue by locating proxies within the Local Area Network (LAN). When clients send a HTTP request to the proxy, it checks whether the requested content is already in its cache. Since this is the first time the cache receives a HTTP request, it sends the request to the origin server. Once the response from the origin server arrives to the proxy it might store a copy before gives it back to the client, depending on the cache management algorithm.

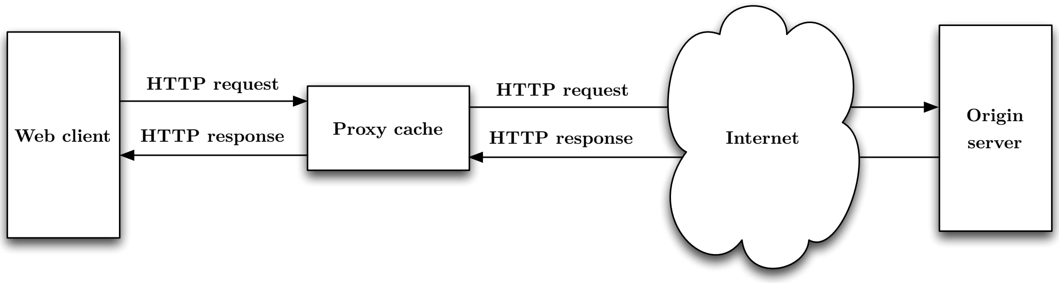


Figure 3.1 Proxy cache scenario

In a medium size network (typically 1,000 users or less) the proxy cache can save a reasonable bandwidth, reduce the response time and can serve data when the original server is unreachable. Despite all the benefits, it has some shortcomings. The main issue is that all queries are sent to the same machine increasing its processing load. All the queries produces a slow response from the server because it has to process all the queries. It also could produce a single point of failure because all requests go through the proxy cache, and the cache is the link between the client and the origin server.

Reverse Proxy Cache: This architecture is just the opposite of proxy cache because is deployed near the content instead of the user [49]. Many benefits such as load balancing, compression, encryption and security are provided by this architecture. It is also useful for hosting farms because all queries are sent to the proxy instead the content server. This feature is appropriate for being implemented at Internet Service Provider (ISP) or Content Delivery Network (CDN) since it is beneficial for content providers rather than users.

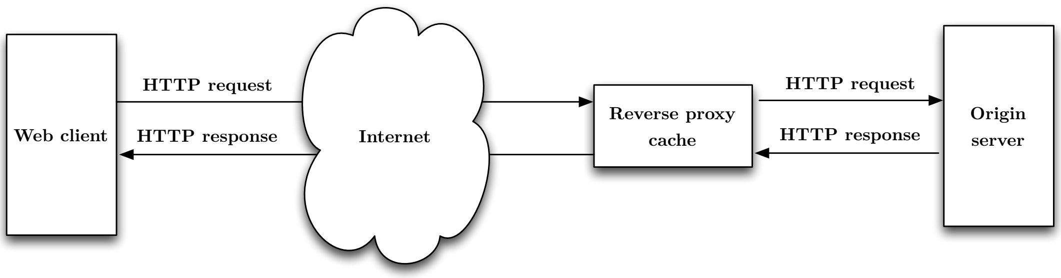


Figure 3.2 Reverse proxy cache scenario

Figure 3.2 depicts the typical scenario for using reverse proxy cache. The web client sends a HTTP request directly to the reverse proxy cache. The request could be forwarded to the origin server or could be served by the reserve proxy itself. This configuration is totally transparent to the client and it could work together with a proxy cache.

Transparent cache: Unlike the previous architectures, transparent caches are implemented on switches or routes. The main difference is that the client does not see the cache. The devices process the HTTP requests sent by the web client and forward them to web caches, clusters or end users [49]. The transparent cache itself does not store any content because are implemented on switches or routers, but redirect the request to an appropriate cache or end user like a load balancer would do. By forwarding the packet it violates the end-to-end principle, because it has to change the destination address. This might be a problem if the application requires keeping constant this principle.

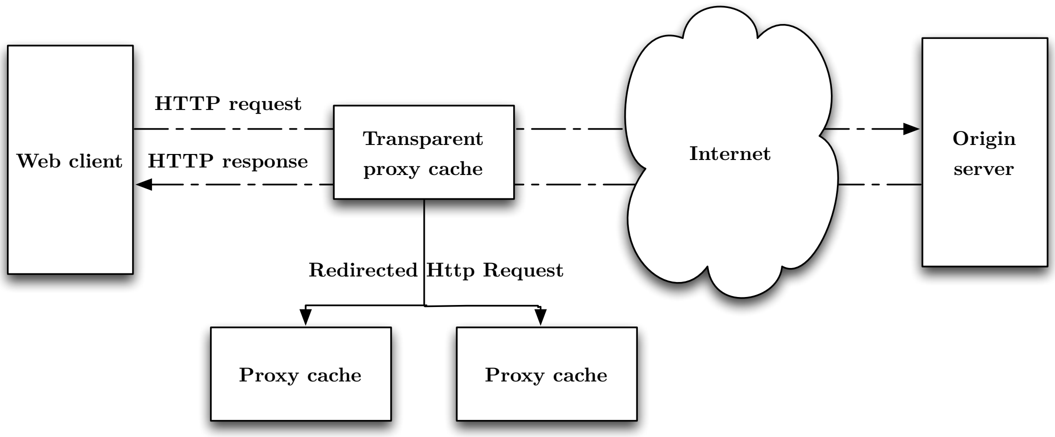


Figure 3.3 Transparent proxy cache scenario

The common implementation for transparent proxy cache is depicted in Figure 3.3. The web client requests a resource to the origin server. This HTTP request is intercepted by the transparent proxy and forwarded to a proxy cache or to the origin server on the user behalf. Abba et al. [51] studied the effectiveness of transparent web caching on a local area network. Their work conclude that the latency and download time is improved as well as a minor bandwidth saving.

Common software to implement any of the abovementioned architectures is Squid [52]. This software is free licensed under the General Public License (GPL or GNU), and works in different operating systems such as Microsoft Windows or Linux. Squid is very robust and can work with different protocols such as File Transfer Protocol (FTP), HTTP or HyperText Transfer Protocol Secure (HTTPS).

3.2 Caching enhancements

The principles behind caching is storing objects queried by clients, through proxies or browsers, and then serve them locally in further requests without requesting the web server. This principle has been deeply studied and as result some improvements have been made.

Adaptive web caching: Jacobson et al. proposed adaptive web caching [53], and studied in [54]. Later was improved by L. Zhang et al. [55]. It focuses on the "hot spot" phenomenon, which consists on some content becoming popular in a short period of time. After a brief period of time the numbers of requests for that content are reduced drastically to the normal number of requests. The hot spot might occur anywhere at anytime giving no time for re-provision caches. Adaptive web caching uses a mesh of distributed web caches capable of join and leave groups when the hot spot phenomenon appears. The ability to join and leave groups differs from *Squid's* hierarchical tree architecture where all the nodes are statics and well defined, hence unable to handle exponential growth on a dynamic environment. The Cache Group Management Protocol (CGMP) allows caches to create and leave meshes. CGMP uses multicast as technique to deliver large amount of messages efficiently within multicast groups. The Content Routing Protocol (CRP) detects and distributes cached content within the group of caches. The main issue of the hot spot phenomena is that a content might become popular anywhere. Hence, meshes of caches should be able to work and cooperate among different countries. This might arise some administrative problems because each country has its own legislation. Michel et al. [55] proposes a method to reduce the traffic between meshes whenever CRP distributes the fetched content. The study proposes a compressed hash tables with the URLs to reduce information between caches.

Active caching: Several studies along the last years have concluded that personalized content, for instance cookies or scripts, are increasing [56] [57] [11] [58] therefore unable to be cached in the traditional sense. Active caching [59] uses plug-ins as method to personalize objects on server's behalf. The queries sent to a proxy server for first time are forwarded to the web server. The origin server gives to the proxy the document and all cache applets required for process the queries. When a second request hits for the same document the cache invokes the corresponding applet for this hit. The cache applet decides whether allow the cache to give the document to the user, give a new document to the cache for sending it back to the client or redirect the query to the origin server. Furthermore, different information could be stored in a log object and send back to the origin server periodically, increasing the applet possibilities to log in users in the system, rotating advertisements or verify users permissions. That strategy relieves web servers from computational process but increases proxy cache load, important issue that as has been studied in [60].

Push caching: The aim is to proactively push data close to users before they have even requested it. The idea of *Geographical push caching* was introduced by Gwertzman [61] based on the premise that cached data should be keep it close, in geographically terms, to those users that are requesting this information. Unlike the previous approaches, this is server-initiated cache where the origin server has control over the content. Centralized registration services tracks push-cache servers to help them locating push-caching servers on demand. By knowing the IP address and users' access record, the server can decide where would be more efficient to set a copy of the requested document. Push caching uses IP prefixes to define the structure of the network. The topology consists on primary servers, which have the original document, and secondary servers where a copy is stored. When a primary server replicates the document, its load decreases because the user access to the other server. If the server's load increases too much, it can replicate the document again to another server and so on. That behavior maximize bandwidth and reduce load but only the origin server can override the document. This method needs some time to spread the new content to the rest of servers. In several articles [62] [63], Gwertzman and Seltzer discussed topology issues, when and how much data should be "pushed". They also determine that client-initiated combined with server-initiated gives a greater bandwidth savings rather than separately.

3.3 Packet caching

Packet level caching relies on the premise that duplicated packets are sent continuously. By detecting these redundant bytes, there is no need to transmit them again, only a code or "fingerprint" which represents these bytes. This architecture is based on Manber [64] for detecting duplicated files on a system and applied by Broder [65] to web documents.

Generally, packet-caching works by analysing a stream of packets and detecting repeated bytes from earlier packets. A token dictionary is computed using fingerprints between two caches, which represent the packets of the data. Typically, these two caches have different roles depending whether they are. A parent cache never queries to their children but they might querying the parent cache. The parent cache assumes the role of manager whereas child cache serves the queries from the users. Figure 3.4 depicts the basic scenario of a system where packet-level caching is used. When client *A* sends a query to a server, both parent and child caches store the packets (one packet represents one white square) and compute their fingerprints (one fingerprint depicts one grey square). Later, client *B* demand the same content, therefore the parent cache replace one packet for one token computed earlier from the access of client *A*. The *A* token is an identification of the packet, which consumes less bandwidth because is smaller than a packet. The token is transmitted between caches and translated to the original packet using the child's dictionary.

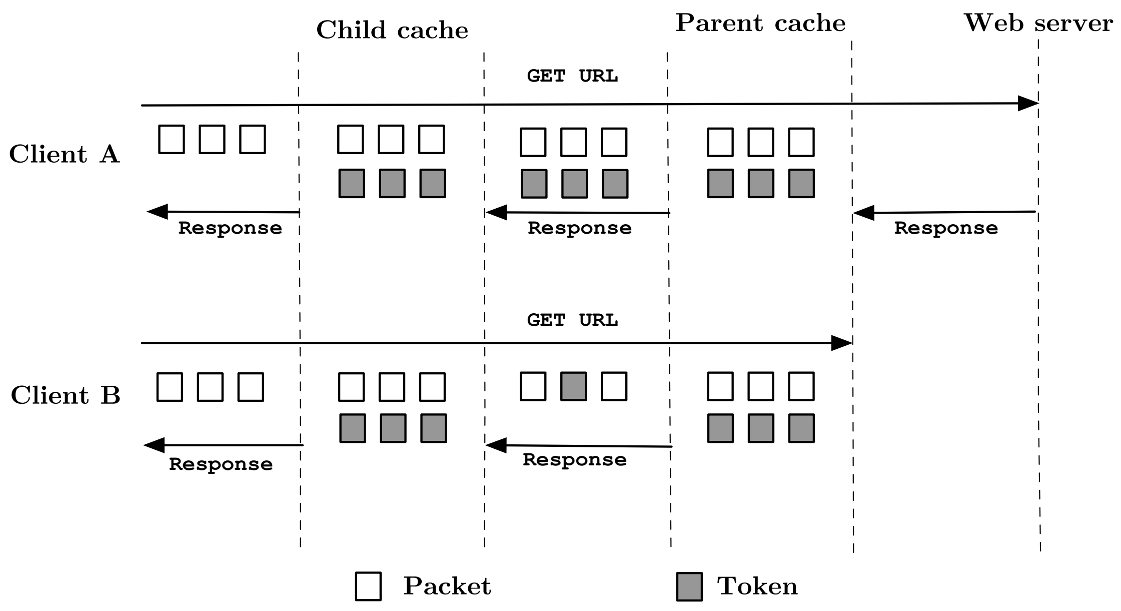


Figure 3.4 Packet-level caching basic scenario.

The difference between packet-caching algorithms relies on how the fingerprints are computed. Most of the solutions implement a variation of Rabin fingerprints [66]. Rabin's fingerprint is a method to identify uniquely resources by using polynomials. For instance, if two files differ only from one byte, the fingerprints for those documents might be completely different.

Spring et al. [67] was the first to develop a protocol-independent technique to detect and eliminate redundant traffic on networks. That protocol is based on Manber [64] and Mogul [32] [31] work. Several vendors such as Juniper [68] or Riverbed [69] have been encouraged by their results, along with other studies, and developed middleboxes called WAN optimization. The middleboxes are centered in links where bandwidth is limited (because of high demand or poor infrastructure), for instance an enterprise, ISP or datacenter.

Anand et al. [70] expanded Broder work to all routers, obtaining 10-50% bandwidth reduction on network links. Anand et al. [71] compare two solutions for eliminating redundant packets; redundancy suppression and data compression. Redundancy suppression identifies common strings of bytes in the current packet and packets previously stored in cache through a cache or dictionary. On the other hand, data compression applies compressing algorithms (e.g., deflate) to each packet. They determine a maximum bandwidth saving of 35% and 26% accordingly.

Rhea et al. [72] propose a different approach from the standard web caching by indexing data by its value instead of its name. They identify two factors for bandwidth wasting: aliasing and resource modification. The first one appears when the same content is identified by multiples URIs. It represents 54% of all data, which means 36% of all bytes transferred. The second factor is caused when the data identified by a unique URI has changed because then the new data is transmitted as well as old data.

3.4 Cache management algorithms

Extended and hard work has been done on cache management algorithms area, not only to improve basic algorithms but also because each has its own constraints. Depending on which metric is desired to improve (i.e., hit rate, latency, network traffic [73]) the algorithm differs. At that point it has to decide whether include the next document or not and which document is going to be replaced. During that decision is when algorithms differs. The following algorithms have been proposed for cache managing.

Least Frequently Used (LFU): LFU takes into account how many times a document is used and evicts the least frequently used.

Least Recently Used (LRU): LRU discards the least recently consulted document keeping records of what and when was it used [74].

LRU-Threshold: This algorithm is nearly the same than LRU but it takes into account the size of the item. Files with size is larger than a certain threshold are evicted [74].

LRU-Min: The algorithm expels the least recently used if its size is at least S. If there is no object that fulfills such criteria, it starts evicting documents with half that size (S/2) [74].

Size algorithm: The algorithm evicts the document with higher size first, favoring small items rather than larger [73]. The documents are evicted when the requested document exceeds the free space of the cache.

Pitkow/Recker algorithm: The algorithm uses two rules depending whether the document is accessed the same day or not. If the items are referenced the same day, then evicts the biggest object (same than size algorithm). When the document is accessed the same day the LRU algorithm is applied [75].

Abrams et al. [74] compare different versions of LRU, which are LRU-Min and LRU-Threshold. They determine that LRU-Min outperforms LRU because does not take into account the size of the document. When the documents' size is large, LRU performs better than LRU-Min. On the other hand, LRU-Threshold is comparable to LRU-Min when the available disk size is smaller than the theoretical cache's size, although the though configuration[1] makes it less attractive. They remark that a more

[1] Depends on the current disk workload and available disk size.

effective caching is done close to the origin server rather than the client side due to save disk space and can forecast which documents are worthy to catch for a long-term period.

This chapter details the most common caching architecture and some of their enhancements. Depending on the purpose of the cache it should be placed near the client or the origin server. This section also covers different cache management algorithms. For small documents LRU is better whereas for big files is LRU-Min. We introduce packet-caching systems, which consists on dividing a file into smaller pieces. The next section details our implementation and its limitations as well as the tools we have used for developing it.

Chapter 4

Experimentation setup

This thesis investigate the characteristics of different web sites and their changes over time to analyse how much data might be cached. The data set is obtained by active measurements of popular web sites from a host located at Aalto University. The metrics used for the analysis are detailed in the next section, followed by the implementation and limitations of this application.

4.1 Methodology

One goal of this study is the analysis of changed content in the WWW to evaluate how much content is unchanged over a delimited period of time. To understand and quantify the current web traffic, we need to look web sites from different points of view. For instance, we could examine the IP addresses from which the content is downloaded to describe the balancing performance of these web servers.

The data set for this study is composed by the 50 most visited web sites according to Alexa's ranking [76] [2]. This number of selected web pages is limited because we only have one measurement client to fetch and process all the data. Typically, web analysis tools and search engines use data centers to distribute load because of the vast quantity of data. Popular search engines, such as *Yahoo!* Or *Google* use web crawlers to index web pages in order to be found in their services. However, these services uses distributed programming models to process that data. A relevant model is MapReduce [77], which is implemented at *Google* for processing large datasets. The model distributes the task among the servers to make an efficient use of these machines. This technique is also used for other purposes such as machine learning, data mining or statistical machine translation.

Alexa's web site classifies domains, but some services use multiples domain names that appear as separate items in Alexa's classification. Analysing different top-level domain names of the same service would not give us a better understanding about the web site because of the small differences between domains. Therefore, we have keep one domain and excluded the rest (e.g., national Google sites).

[2] As of June 2011.

Our implementation fetches all objects in the index page as well as all documents linked to it. By index page we mean the top-level index page under that domain. In this thesis we also refer it as first load page or main page. We retrieve the web pages at a constant interval of 15 minute, enough to capture small changes and large enough to not overload the servers. Sending multiples request for a web site in a short period of time could be interpreted as Denial of Service Attack (DOS attack) [78]. The mechanism fetches content 24 hours per day, 7 days per week, starting at February 30th of 2012 and ending on October 30th of 2012. The implementation does not interpret JavaScript [79] embedded in HTML, thus it misses those types of objects. Even so, all the other objects contained within the web site are downloaded.

We have adopted a hierarchical approach aimed to characterize different aspects of the web pages in our data set. The levels of analysis are the followings:

Top-level page set: Web sites include all documents at the initial web page and the direct links from the index page. The links are obtained by recursive requests from the index page that goes one level further down the web site hierarchy. This level of analysis gives an overview about the composition of the web site to understand how changes are distributed. The study at this level provides a better understanding on the use of static and dynamic content through the study of unchanged bytes.

Index web page: That level represents the study of the index page at root, for instance www.site.com/index.html plus all embedded content. We compare our results to previous studies of the WWW [11] [12] [13] as well as determine its potential cacheability.

To extend our analysis we used two different comparison methods at the top-level page set and index web page. Each method aims to emulate different web caching approaches.

File level: Entire documents are compared between them. That comparison intends to determine possible benefits for file-level caching. This approach is common between web caches because it is simpler to implement and cheap.

Block level: Every document is divided into smaller pieces to identify and quantify the degree of change. This approach aims to analyse possible benefits for packet level caching. We compare a previous work [71] to determine which model provides a better detection of unchanged web content. This approach is used in both levels the index web page and the top-level page set level.

The comparison mechanism works along with the data collection. When the fetching process ends the list of web sites, the comparing process starts to analyse the data set. Identifying redundancies at these two levels with two different comparison methods provides a better understanding of web pages to assess the benefits of web caching mechanisms.

4.2 Metrics

This work uses several parameters at different levels to characterize redundant content. Different approaches require different metrics, although all levels attempts to identify content similarities. The metrics for the top-level page set are listed in Table 4.1.

Table 4.1 Top-level page set metrics.

Metric	Description
Size	Total size of the web site in kilobytes.
Unchanged [%]	Fractions of unchanged files for one implementation's iteration.
Unchanged [Kb]	Fraction of unchanged bytes of the Unchanged percentage.
Full match files	Number of files that are exactly the same for different web page versions.
Full match [Kb]	Size in kilobytes of full match files.
Existing files	Number of modified files that remain from previous iterations.
Total files	Total number of files per web site.

Previous works [4] [47] examines the impact of web site's size and their consequences to web caching. We compare the current size of web sites of these works to determine whether their models are still valid or have become outdated. To identify similar content we have used three parameters: *Unchanged [%], Full match files* and *Existing files.*

Unchanged [%] indicates the relation of similar content between two consecutive iterations of the same web site. This metric is calculated by averaging the unchanged percentage for all files belonging to a web site. Detecting relative percentage of redundant content is not enough, it is necessary to quantify the amount of data by the metric *Unchanged [Kb].* Moreover, a web site with high similarity and few unchanged bytes is less important to be cached rather than other with more duplicated bytes and less *Unchanged [%].* These metrics not only show that the document has changed, they also determine the amount of similar content that has changed, hence, how much content could be cacheable.

The continuous updating process does not affect all the documents; some of them keep unchanged over time. That part of the web site is also called permanent content [41] because rarely (or never) is modified (e.g.; favicon). Favicon is an icon that associates a web site to a small image. *Full match files* attempts to count the number of files that keeps exactly the same for two consecutive iterations of our program. We will compare it with an earlier work [41].

Some of the metrics for the top-level page set of analysis are no longer valid for the index web page level since only one web page is analysed. Table 4.2 shows the metrics used for this level.

Table 4.2 Index web page level metrics

Metric	Description
No. Objects	Number of objects in the index page.
Size	Size of the index web page.
Unchanged [%]	Fraction of unchanged data for one program iteration of the main web page.
Unchanged [KB]	Fraction of unchanged bytes of the Unchanged percentage.
Content-types (number)	Number of resources that may be something else in the index web page.
Content-types (bytes)	Size of the different web resources in the main page.

The *No. Objects* is a relevant metrics because it determines the number of files that caches should store to represent the full index page. We will compare them to an earlier study [80].

We use two similar metrics used in the top-level page set, *Unchanged [%]* and *Unchanged [KB]*. In this case, we use *Unchanged [%]* to determine the percentage of unchanged data within the index web page for one comparison iteration. It gives us and approximate value of how cacheable this web page could be. As we did in the top-level page set, we determine the number of bytes that this percentage of similar data represents and the impact on web caches.

We have abbreviated the last two metrics, *Content-types (number)* and *Content-types (bytes)*, because refers to five different web content types: HTML, CSS, JavaScript, images and others. The *Content-types (numbers)* counts the number of different web content-types within the index page level. *The Content-types (bytes)* takes into account the bytes used for each content-type. We aggregate the most common image formats (jpg, png and gif) into one metric: *Image*. The objects that do not fit into one of the previous content types are counted as *Others*. Typical objects in this category are icons and AJAX files. We counted the number of objects and their size for different content types because not all resources have the same cacheability. We evaluate the potential cacheability of these objects and we will compare to earlier studies [8] [48] [47].

Counting the number of *Total files* determines the rate of introducing files and identifies where the change has been made. In other words, it discerns whether the web sites update existing documents or create new files. That raises some questions:

when a web site change, does it change completely? Do caches need to store again the whole document? To answer these questions we have divided files into smaller pieces. With these pieces we identify those parts of the document that have actually changed.

The size of the chunks is 1460 bytes, which matches the typical Transmission Control Protocol (TCP) maximum segment size (not assuming TCP options) [81]. This approach simulates packet level caching, hence results can help to improve caches. Redundant blocks are measured with *Unchanged [%]*, which is determined by averaging the number of repeated chunks in a web site. We also determine the unchanged bytes with the metrics *Unchanged [KB]*.

We aggregate similar web sites into categories depending on their type of service. Thereafter, different categories (e.g.; search engine or social network) are compared in order to identify models of redundant content across categories. Alexa web service, from which we obtained the ranking of most visited web pages, has its own category list. Nevertheless, some categories are too general and in some cases not accurate. As example, www.youtube.com and www.google.com are clustered in the same category: "Internet", whereas each domain has different goals and content. Another example is www.facebook.com situated as "Activism" when www.twitter.com is categorized as "Internet". This study presents an alternative categories list where web sites are differentiated depending on their content rather than the use that we do with them.

Table 4.3 describes the different categories used in this work, and the complete list of web sites and their categories are detailed in Appendix A.

Table 4.3 List of categories.

Category	**Description**	**Nº of web sites**
Search service	Web sites that require keywords to produce results.	4
Content sharing	Web sites that prevalently exist for distribution of static content.	11
Video streaming	Pages dedicated to distribute video and audio.	8
E-Commerce	Entities that focuses on electronic commerce.	6
News	Pages dedicated to inform users about news.	6
Social network	Web pages where users are members of an online community.	9
Service portal	Pages that offer multiples services such as e-mail or news.	6

The study at this level implies a better understanding of the web sites according to the service that are offering and how could be applied to caching systems. Web sites whose categories have more changed content is more likely to be stored in a cache rather than web sites whose categories have less unchanged data.

4.3 Implementation

In this section we detail how we obtain the data set and how implemented our solution. The section begins with an overview of the implementation followed by detailing data collection and comparison processes. Tools and limitations of the solution are described next.

4.3.1 Overview

We developed a system where web sites are downloaded and examined at different levels. Our program is composed of three scripts, each one working independently from each other, to avoid functional errors. Each scripts aims to different analysis levels, as Table 4.4 shows.

Table 4.4 Relation between scripts and analysis levels.

Analysis level	**Script**
Top-level page set	Compare
Index web page	MainpageCompare

The first script is responsible of collecting the dataset, *Gather* script, is not included in Table 4.4 because it collects data instead of analyses.

The overall implementation is depicted in Figure 4.1 where circles represent scripts, rectangles folders and rectangles with bended tabs text files. The gather and comparing scripts are handled by a time-based job scheduler, in this case the daemon *cron* [82]. This daemon is configured to run the scripts every 15 minutes. The last step does not need a scheduler because is a statistical software that processes the data.

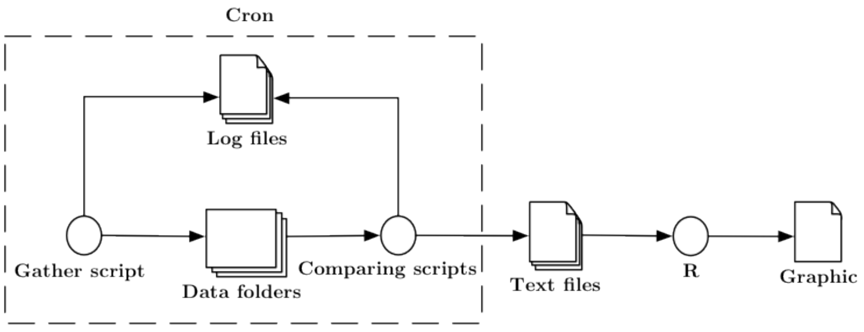


Figure 4.1. Overall implementation

For brevity, Figure 4.1 only depicts the general structure. Comparing scripts refer to our three analysing scripts: *Compare*, *MainpageCompare* and *BlockCompare*. All of them use the same structure but with different comparison methods.

Gather script is dedicated to query each web site to download all the documents within it. Information about gather process is stored on a *log* file in order to register any anomaly during the process. Along the execution of the program a set of folders are created, which is where the information is stored.

The second stage, *Comparing* scripts, begin to work when the data is obtained. Its function is to analyse the data of two consecutive queries from the same site looking for common parts. *Compare* script analyses the entire file, meanwhile *BlockCompare* split each file into smaller pieces and compare them. Finally, *MainpageCompare* analyses the index page of each domain separately. *BlockCompare* also analyse the content at the index web page level. Each script creates different *Text* files, based on different metrics, following a table structure to register the comparisons between documents.

The last part represent the data obtained on the previous stages with the statistical program *R* [83]. The result is a series of *Graphics* that are used to compare previous works and define new models for unchanged content.

4.3.2 Data collection

The measurements were done on a Linux-based dual-core AMD Opteron 2218 with 16 GB of RAM. The host has a gigabit network interface connected to the FUNET network [84].

The web sites extracted from Alexa are listed in a text file where the script reads them every time that it is executed by the daemon cron. The measurement client queries all webs within the file, web site after web site, as Figure 4.2 depicts.

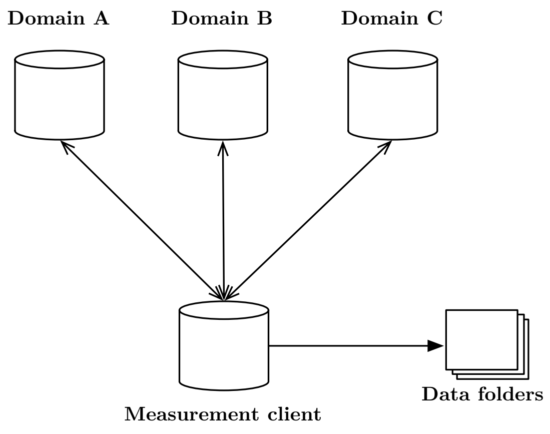


Figure 4.2. Query diagram

The server takes one domain at a time from our list, and downloads all data into the folders. Once the data is obtained from one web site, it jumps to another domain repeating that process until it finishes the web sites in the list. When 15 minutes have passed from the first query, the program starts the process again.

The queries are done using the tool *wget* [85] with some particularities at the request header. The script represents itself as Chrome client, because it is the most used web browser with more than 30% of market share around the world [86]. The used version string of Chrome is: `Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_6; en-US) AppleWebKit/534.16 (KHTML, like Gecko) Chrome/10.0.648.133 Safari/534.16`.

The full command for the fetching process is the following: `wget -r -l 1 –H -p --random-wait -wait=2 robots=off`. To download all content derived from the index web page, *wget* follows hyperlinks with the option `-r`. If no other option is settled, it downloads the web page indefinitely. We added the option `-l 1`, for limiting wget to the first's links from the index page in the web site hierarchy, as it is explained in section 4.1. Nevertheless, not all links refer to the same domain, some of them links to other domains. For instance, the main page of The New York Times has advertisements and links to social networks. Expanding the request to foreign host with the option –H downloads data from others domains Moreover, to get all objects needed (i.e., images or stylesheets) we included the `–p` option to display properly the web pages.

Along the years, some attacks have been performed against well-known web sites [87] [88]. To avoid undesired retrieval programs some web sites examine logs to forbid malicious software or denial of service attacks [89]. To mask *wget* as a common web browser we have used the options `–random-wait` and `-wait`. *Wget* wait a random time that varies from 0.5 to 1.5 times the parameter `wait`, which is settled to 2.

The last parameter issues the use of *Robot Exclusion Standard* [90], which limit search engines access to parts of a web site. It is useful for web developer to exclude from crawlers to access some part of the web page, which requires big resources of the server or parts where information is private. This file does not guarantee that the resources are not queried, at the end depends on user's purpose. The option `robots=off`, give full access to those parts while no login is required.

The data set is stored following a tree-like structure, as Figure 4.3 depicts. The framework has four levels determined by date, hour, browser and domains' name. That organization allows us to search easily for specific events that might occur at a specific time.

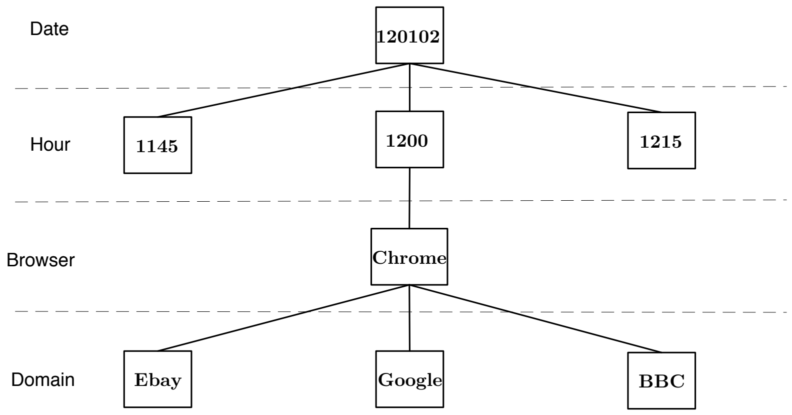


Figure 4.3. Folder's structure

Figure 4.3 is an example of our tree-like structure. In this case, Google's web page has been downloaded using the web browser Chrome at 12:00 the 2nd of January in 2012. We have added a level for the browser because different browsers have different behaviors. This feature is not implemented in this work, but it is discussed in section 6 as possible future work.

4.3.3 Data analysis

In these analyses we focus more on quantifying the degree of the change rather than the changed content.

The *Compare* script examine the files obtained after fetch all files within the index page and files linked to it, see section 4.1. The *Compare* script uses a recursive search of all files within the *domain's* folder, taking the files at different fetching times. If the file does not exist in one version of the web site, it is labeled as a new file with 100% of change. On the other hand, if both files have been located, represented as *File A* and *File B* in Figure 4.4, the script increases the existing file counter. This indicates the number of files that have the same name for two program intervals. At this point, we do not check whether both files, A and B, have the same content, this checking is collected by the metric *Full match files*.

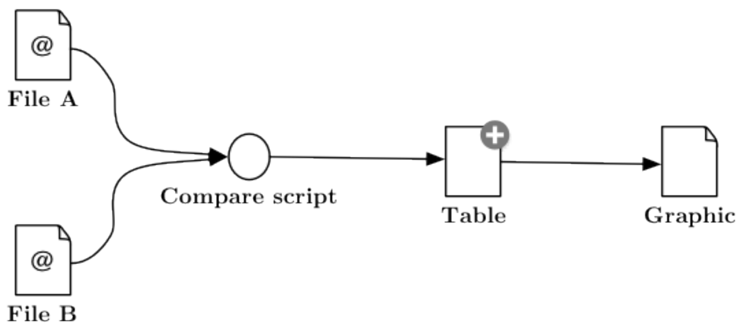


Figure 4.4. Comparison process

Each file is compared using the `Cmp` tool. If the files are exactly the same, it means that we have obtained 100% of unchanged data, therefore *Full match files* counter is increased. When the script has examined all the files within one domain, it averages the unchanged percentage and unchanged kilobytes for all files and stores them on a table along with other metrics.

MainPageCompare follows the same process but only for the index web page. It locates each index file for all web site in the list, depicted as *Index.html A* and *Index.html B* in Figure 4.5.

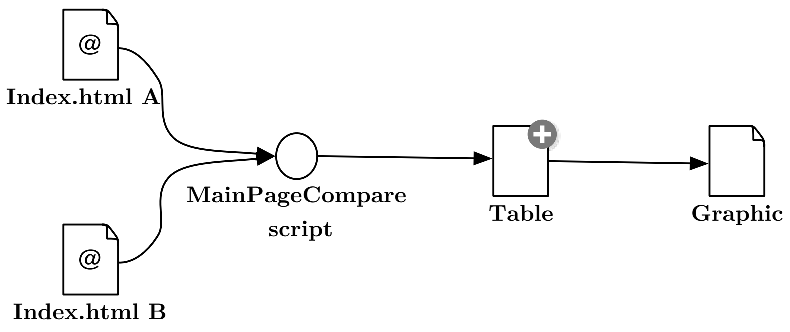


Figure 4.5 MainPageCompare analysis process

MainPageCompare also compares all objects contained in the index page plus embedded data. Typically these resources are images and JavaScript, but also CSS files.

The script *BlockCompare* uses a different approach for the analysis. It starts by locating each file within a domain at different fetching times, as *Compare* script and *MainPageCompare* does. This scripts works at both analysis levels. Once both files are located, *File A* and *File B*, the program divides both files into chunks of 1460 bytes each one, as Figure 4.6 depicts. This division is implemented in *C* because it provides better tools for reading a determined number of bytes.

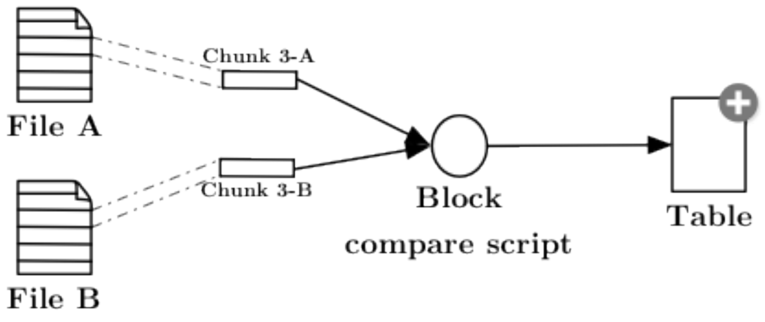


Figure 4.6 Chunks comparison process

When each chunk is selected, in this case *Chunk 3-A* and *Chunk 3-B*, the script compares them using the function *strcmp*. The function discerns whether *Chunk 3-A* is bigger, equal or smaller than *Chunk 3-B*. Once all chunks are compared with that script, the script averages the fraction of unchanged chunks of all files within one domain. Finally the scripts count the number of bytes that represent the unchanged blocks.

4.4 Tools

Alexa [76] is a web service that collects statistics on a large number of web sites worldwide. This service collects number of hits, traffic and time on site among others metrics, but we have used the ranking of the most visited sites that are expected to have most impact on Internet. There are other web sites with similar crawls such as Google trends [91] but Alexa has more than 30 millions of web sites in more than 125 countries.

Bash was chosen as scripting language (there could be have been many equally good script language alternatives in UNIX). We have used basic bash functions but also more specific tools that require special mention due to their specifics options.

The most important tool is *wget* because it is the responsible to download the web pages. *Wget* is very flexible and robust on slow or unstable connections because if a download fails, it keeps retrying until the document is download. This characteristic is especially useful for domains on countries where the distance to our machine could be a problem. There are similar tools such as *cURL* [92] or HTTrack [93].

R [83] is statistical software capable to represent data sets on multiples types of plots such as boxplots, histograms or pie charts. Moreover, it is capable to handle arrays or vectors but also calculate means and deviations. We have chosen this environment because is free, powerful and easy to use. There are several alternatives to R such as *S-Plus* [94] or *PSPP* [95].

4.5 Limitations

During our analysis we have found some shortcomings. Our intention to perform an analysis as heterogenic as possible found a shortcoming when we query to social networks. Any social network needs a login by the user, which is feasible to do it technically (if we have got passwords or some kind of credentials from users), but that invades the user privacy. We did not want to intrude the user's privacy, therefore, we have not compared any data beyond the index page for that category.

We suggested in section 4.1 the use of a distributed system where the load of the machines is shared. The amount of downloaded objects and their further comparison yields significant process consumption. By dividing the task into different servers (i.e., one server to collect traces, one to compare and another to represent) would increase the effectiveness and the possibility to spread the number of studied web sites. The benefit of this task division is detailed in [77] and it has been used in huge server's clusters at Google. Unfortunately we only have access to one server, which limit us to a relatively small number of sites. Although, our analysis is significant due to we analyse the most popular web sites.

As summary, in this section we have detailed the methodology followed by our implementation. We have introduced a categorization for different web sites according to their content. With that categorization we analyse two levels: Complete web site, Index web page level. Each level has its own metrics because we analyse web sites from different points of view. The comparison process is done by two different methods: file and block comparison. Later on we detailed the set of scripts to fetch the content, analyse our data set and plot the results. Finally this section ends explaining the tool we have used and the limitation of our implementation. The next section shows the results that we have obtained through all that process.

Chapter 5

Analysis of Results

This section compare earlier known 90's results with our data set. We examine number of objects, composition of the web sites and their sizes. Moreover, we study the dynamics of popular web pages in order to determine the potential impact of different caching techniques. We use categories, depending on their type of service to determine any correlation between web sites and categories.

5.1 Overall characteristics of web content

Along the years, the WWW has evolved, adapting to the time and capabilities of the surrounding technology. Strong indicators of this evolution are the number of objects per page, the number of resources within web pages and their size. All this metrics are studied in the corresponding analysing levels.

5.1.1 Overall analysis

To determine the number of objects per page, we have recorded and examined responses from different web pages for the index page. Figure 5.1 shows the Cumulative Distribution Function (CDF) for the number of objects required at the index load page when a user queried a web site. An earlier work by Hernandez-Campos et al. [80] studied the number of objects on web pages from 1998 to 2003, finding three or fewer objects per page for the 75% of their data set. These values are far lower than our results since 81% of web pages have 100 objects or more and none of them has three or less objects. It is because current web pages use new technologies, such CSS or JavaScript, which were less popular in the early 2000 than they are now. Advertisements are the other contributors for the growth of the number of objects per page.

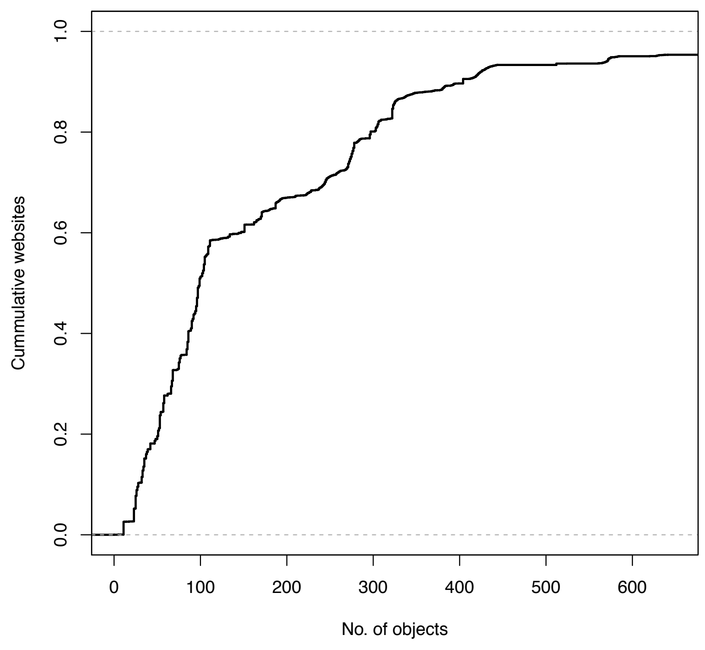


Figure 5.1 Average number of objects at the index page

The number of objects is a factor to take into account for web caches, but also the size of the objects. The reason is similar to the number of objects, since bigger objects needs more space in caches rather than smaller objects. Bigger objects load slower than smaller because it takes more time to be transmitted. Therefore some caching algorithms prefer bigger objects because the user downloads small files faster. Two HTTP/1.1 mechanisms: persistent connection and pipelining [18] favors the use of more objects per page. The reason is because these techniques allow transferring multiple HTTP requests without waiting for the corresponding responses on the server's behalf. Therefore, a higher number of objects could be sent regardless the acknowledgments. These two techniques allow web designers and web masters including more objects per page without excessively increase the load time of the web page.

Figure 5.2 shows the cumulative distribution of size of objects presents at the index page level. We observe that *HTML* is the main contributor followed by *JavaScript* resources. Contrary to Butkiewicz work [47], the median size of the resources is lower in most of the cases. One possible reason for reduction in size is the minification of the objects [96]. This technique removes all unnecessary characters from the web code without changing its functionality reducing the size. It is not the same process than compressing since minification does not require any uncompressing process. HTML, JavaScript and CSS are resources that may be minified with current tools such as Clousure Complier [97] or YUI Compressor [98].

Nevertheless, we have found an increasing size of image resources, the median size of an image is 8 KB whereas in Butkiewicz' study is less than 2 KB. The use of multiple icons per image, also called tiled images, is a possible factor for increase the size of images.

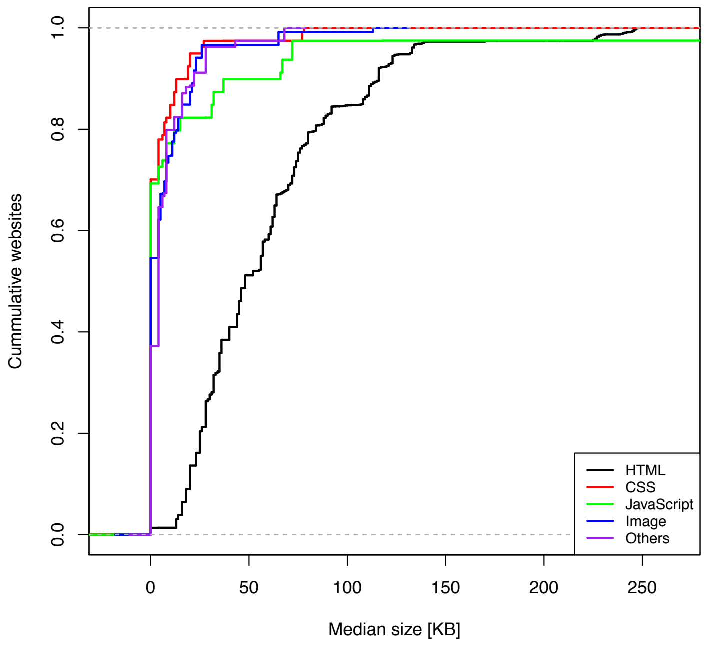


Figure 5.2 Object size for different web content types

Finally, we focus on the size of the web pages as Figure 5.3 shows. Sunghwan and Vivek [48] compare the evolution of the page size along several years. During the last two years the average size of the pages has remained similar, from 133 KB in 2010 to 124 KB in 2012 as median.

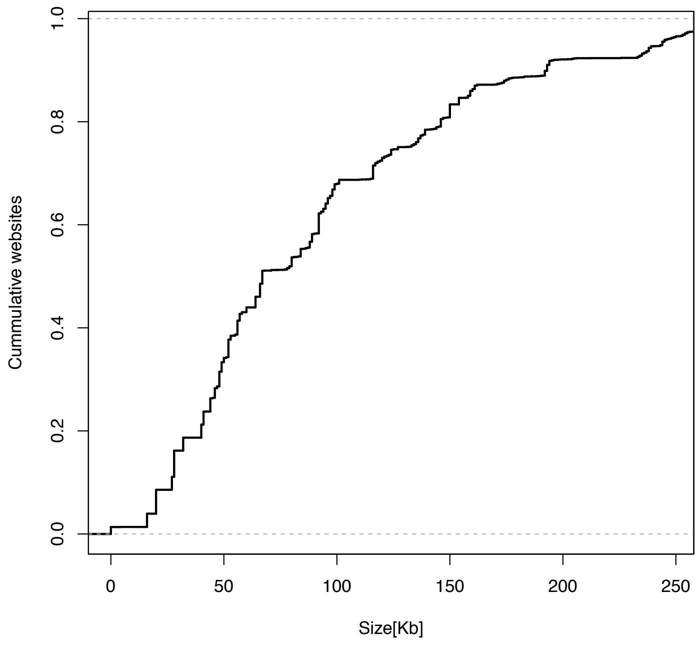


Figure 5.3 Average of index web page size

For the last few years the web pages have maintained their size, but the complexity of the web sites has increased because there are more objects per web page. Some types of objects have reduced their size through the minification process, but not images, which in fact have more bytes.

5.1.2 Distribution of different categories

There are many types of web services on the WWW. For instance, we can search for information with a search engine or buy an article from a web page. Each web service type is created to cover a specific goal, therefore the structure and approaches for achieve it are different from each web service. For example, social networks are based on dynamic content from the users meanwhile newspapers focuses on news from around the world. Some question arises from that diversity of web services: Have different web services the same structure and size? Do they use the same technologies? And, finally, in what degree different web services have changed their objects composition along the last years? To answer these questions we have used categories that enclose similar web services (e.g., www.ask.com and wwww.bing.com).

As described in the above section, the number of objects per page is an important factor when a web cache is deployed. Figure 5.4 plots the cumulative distribution function of objects per web page broken by categories. We have not performed any login for those web sites belonging to social networks because we did not want to fetch private data such as usernames or passwords. Therefore this category is at the lower range on the plot.

The category *News* exceeds the rest of categories by more than 200 objects. This is because the number of articles contained per page. Typically, an article consists of some text, several images and multiples references to other news. These factors increase the number of objects substantially.

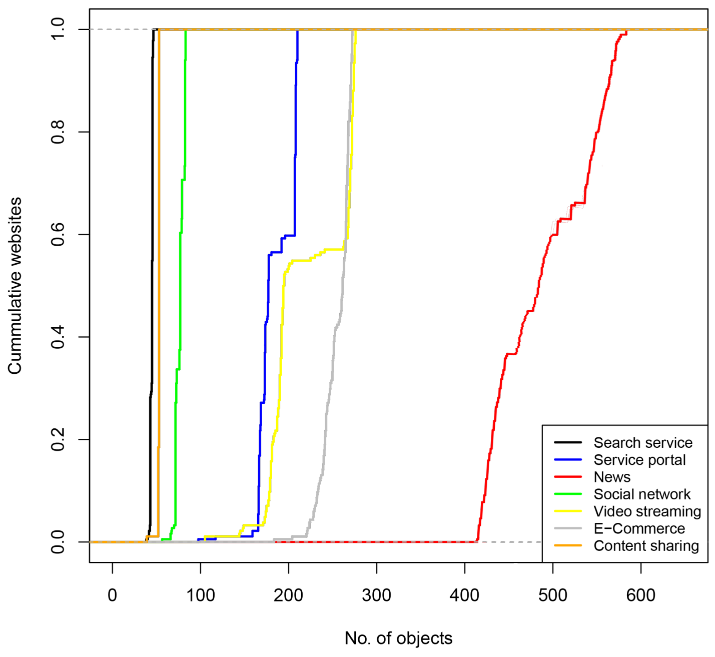
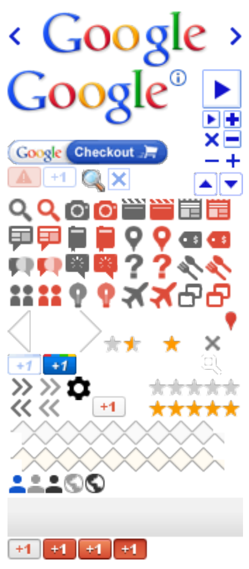


Figure 5.4 Number of objects at the index page level by category

Figure 5.5 shows an image from www.google.fi that contains different icons in one picture. These images are called tiled images. Not all the web sites use the same approach, for instance, www.nytimes.com or www.spiegel.de uses one image per icon. The use of small images contributes to increase the number of objects per page. Also increases the number of bytes and difficult the cacheability of the web site because multiple images needs to be stored in the cache.



a) www.google.fi

Figure 5.5 Google's tiled image

The search engines www.baidu.com and www.bing.com also uses this technique including several icons in one image. However, we have observed that some web sites do the same but only for sharing icons. For instance, for posting a comment on *Facebook* or send a message via *Twitter*.

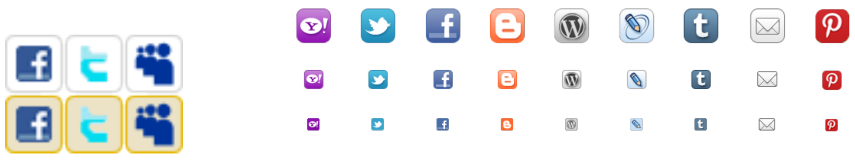

a) www.imdb.com b) www.flickr.com

Figure 5.6 Different sharing icons in one image

Figure 5.6(a) shows different sharing services grouped in one picture for the web site www.imbd.com. Figure 5.6(b) uses the same methodology but with more services such as www.wordpres.com or www.yahoo.com. Moreover, the name of the file is the same across all web sites, `sprite.png`. We suppose that this name is becoming a non-written agreement between web developers, similar to the favicon icon (see section 4.1).

Not only the number of objects has changed, the composition of the web pages also is different compared to previous years. This is a relevant factor because not all web content types have the same cacheability. Douglis et al. [8] did a study on the WWW in 1997. They found 24% of the resources were HTML files and 65% were different image formats. Their representation in bytes was 33% and 46% respectively. Figure 5.7 shows the composition of current web pages, index pages plus embedded objects from index page, and their percentage in bytes. *HTML* and *image* resources are the most common, although *CSS* and *JavaScript* have become relevant. One might expect that the *Video streaming* category is mostly composed by video files but typically videos are only downloaded by user's action (i.e., the stream starts by clicking on the play button). Typically videos are also in Flash files, which are not supported by our implementation. Our results are different compared to Douglis et al. work: we obtained 42.3% of HTML resources and 25.28% of image resources in average across all categories.

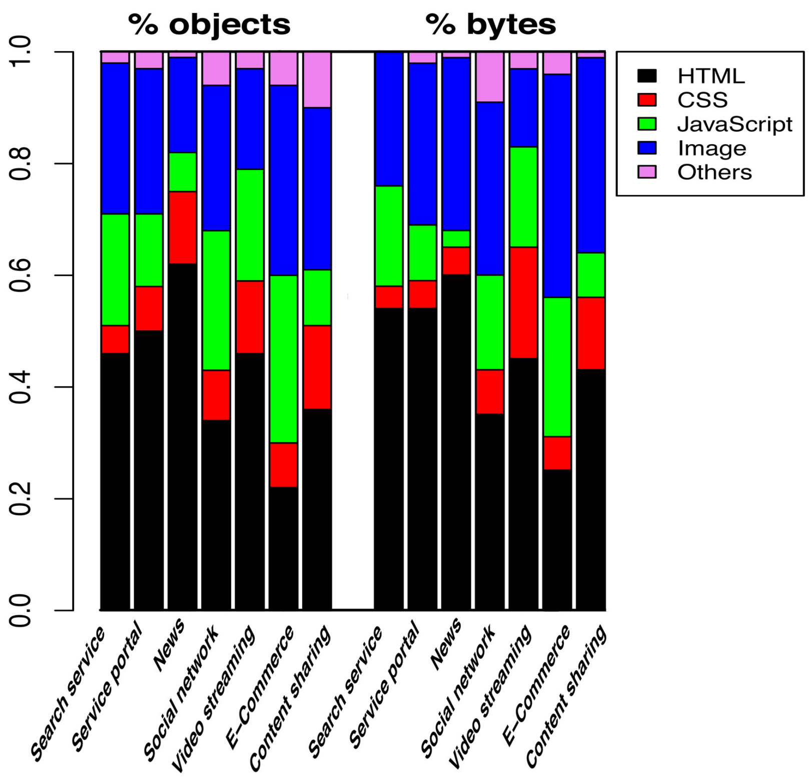


Figure 5.7 Composition of web content types per category

Sunghwan and Vivek [48] studied in 2011 that around 25% of the objects of their data set were HTML. We have obtained a higher percentage, 42% of HTML objects, but when we look at the percentage of bytes that they represents, this fraction raises to 45%. The fraction of HTML bytes obtained by Sunghwan and Vivek is just 32% of the total size of the web page. The increasing number of HTML resources confirms, in part, the trend of using more objects but with smaller size. In their work, they also analyse the number of image resources, obtaining more than 50% of image objects and 38% of images bytes. We have obtained lower percentage of image resources, just 25%, but it represents the 29% of the web sites' bytes. As we can see, the number of image objects has not increased, possible because some web sites use tiled images.

On the other hand, Sunghwan and Vivek found that 12% of the requests involves *JavaScript*, which represents the 36% of the web pages' bytes. Our results are similar in percentage, with 10%, but only 8% of bytes are *JavaScript*. We find more *JavaScript* objects but with smaller sizes. We noticed a high percentage of bytes for *Others* objects in *Social Network* category. We have found a representative number of AJAX resources in this category, less likely to be cached because is generated by client-side interactions.

The size per page across all categories is plotted in Figure 5.8. Similar to previous figures, the category *News* stands the first in terms of bytes, followed by *Video streaming* and *E-Commerce*, being 401 KB, 104 KB and 101 KB their medians respectively. As occurs in Figure 5.7 the category *News* has 4 times more bytes than the rest of categories in average.

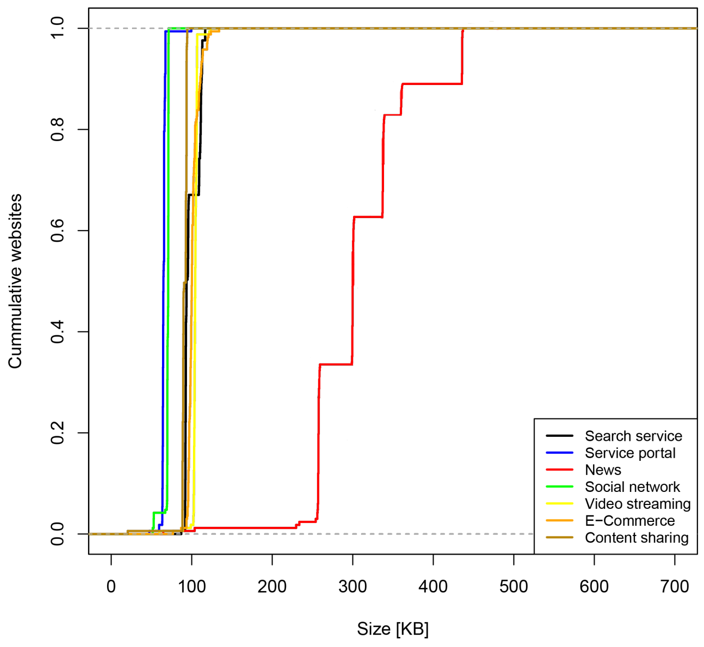


Figure 5.8 Index web page size grouped by category

Our results are roughly consistent with the analysis of Sunghwan and Vivek. They found a strong fraction of image resources across all categories as well an increasing use of *CSS* and *JavaScript*. The study also remarks the difference between the category *News* and the rest of categories in terms of size. This difference corresponds to the increasing use of objects per web page (Figure 5.1). Sunghwan and Vivek determined that 20% of web sites uses 100 objects or more, while we obtained 81% of web sites.

The structure of the web pages does not end at the index web page, in fact, most of the content is derived from that web page. Figure 5.9 depicts the size of the categories at the top-level page set. It clearly shows a remarkable increase on the size of the web sites. In average, we have found that data derived directly from the main web page represents 32 times the data at the index load page. *Content sharing* stands as the category with more relative deep content with 47 times the size of the index web page. *News* and *Video streaming* follows *Content sharing* with 39 and 32 times more deep data.

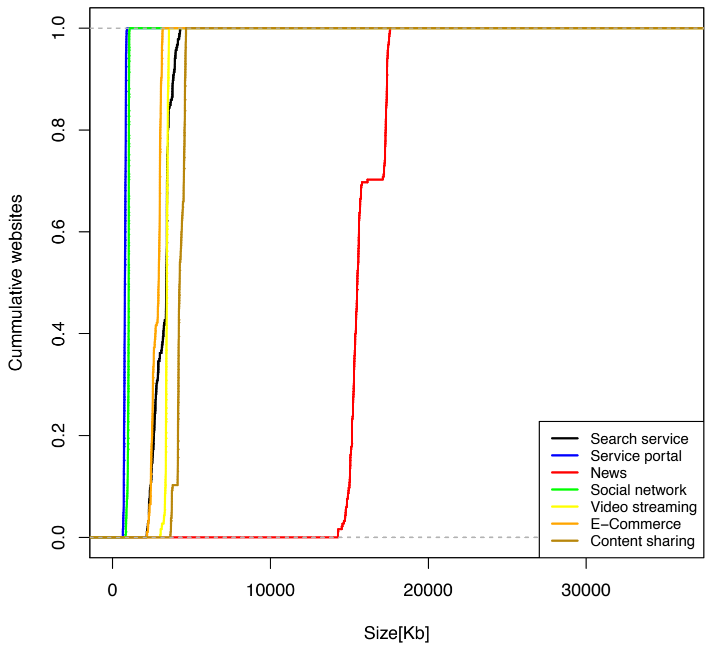


Figure 5.9 Top-level page set size grouped by category

We have confirmed a higher use of HTML objects but fewer number of image objects compared to the study of Douglis et al from 1997. Nevertheless, the number of objects has increased compared to 2011 but with lower size per file. We also found tiled images, which reduces the number of images per page and, a higher number of JavaScript objects but with less size. It may be explained by the improvements of HTTP/1.1 with the persistent connection and pipelining. We also have shown that different web services do not share the same structure and number of resources. The category News exceeds the rest of categories in number of objects and size. This is probably because these web pages have multiple articles, which contains multiple images and text.

5.2 Analysis of temporal differences

Analysing changes in content depends on many factors such as the comparison interval, which elements are compared or the comparison method itself. Detecting changes within an interval time too low will yield on no unchanged content whereas too long may obtain only changes. Therefore, we have selected a medium-range time of 15 minutes, enough to detect small changes within web sites. The same interval time was used for Mikhailov and Craig E. in their work [56]. The second factor for analysing changes in content depends on which elements are compared. We expand our research to those objects that are directly linked from the index web page. The objects from these links (top-level page set) provide a better understanding for web caches of the objects that may be accessed for the users and their potential cacheability.

5.2.1 File-level comparison.

We determine the cacheability of a web site by quantifying the fraction of content that is unchanged between two consecutive iterations of our implementation. We do this analysis at the index web page level and at top-level page set. Figure 5.10 shows the potential cacheability for different categories at the index web page level. The categories *Search service* and *Content sharing* rarely change when we compare two consecutive intervals of 15 minutes.

On the other side, *Video streaming*, *News* and *Social network* change more often, in median, 57%, 61% and 66% respectively. These percentages are lower than the 77% of unchanged data in average for all web sites across categories. For some categories, we can assume that the type of service of these sites is to update the content more frequently, such as *Video streaming* or *News*. For instance, web sites with news update more frequently than search engines, which rarely changes. The high percentage of changed content on *Social network* category is higher than expected without having performed any login. We have found that *Social network* uses different mechanisms to track the user's session and make more dynamic the index pages. For instance, *Badoo* uses different pictures for the index field. It is emphasized in this category by the low size of the web page because we only compare the login page. The use of AJAX is commonly particular in this category because the content is mostly personalized.

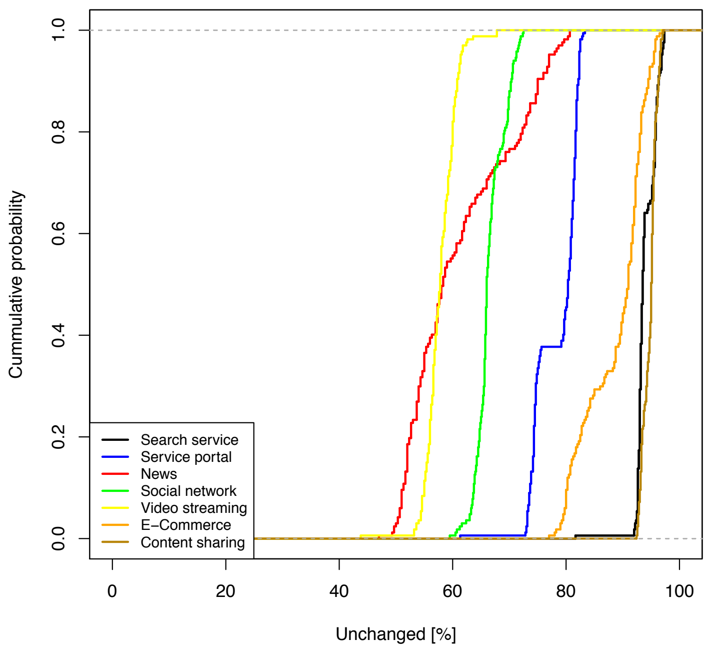


Figure 5.10 Fraction of unchanged data for 15-minute interval between two downloads of the same page at the index page level by category

In contrast to the index web page level, the number of objects compared at the top-level page set is much higher. Figure 5.11 plots the fraction of unchanged data for 15-minute interval for all files linked to the index web page. That picture shows the level of unchanged data of the index load page and the embedded objects directly linked from index page. *Search service* remains as the higher percentage of unchanged data along with *Content sharing* and *Social network*. All investigated web sites belonging to *Search service* category have a section called “News”, which is updated regularly. This section reduces the cacheability of these web sites because the content is modified regularly.

The fraction of unchanged data for social networks is similar in both levels index page and top-level page set. The number of objects that might be accessed without any login is lower than the number of objects accessed with a logged user. The rest of the available links in the web page without login are in most of the cases, links to “Terms”, “Privacy” or “About”, information that rarely changes.

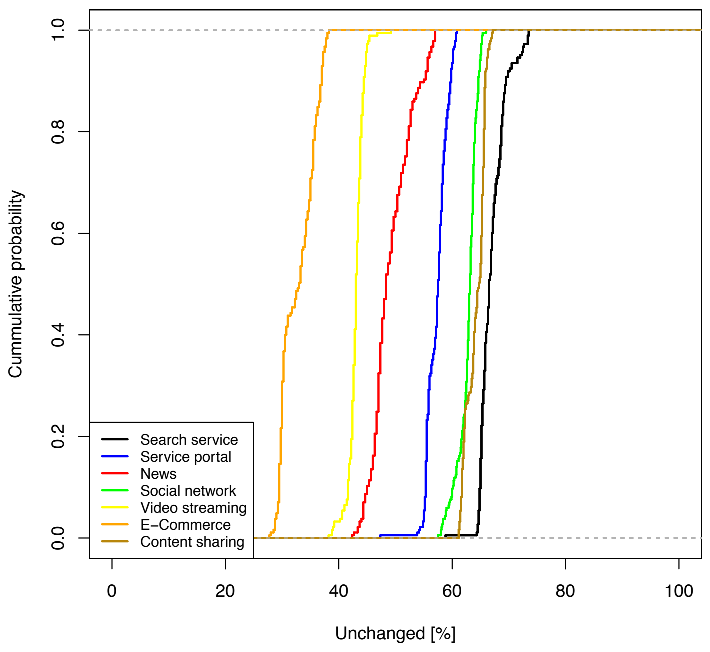


Figure 5.11 Unchanged percentage for 15-time interval between two downloads of the same site at the top-level page set by category

In average, the fraction of unchanged data is reduced by 21 % compared to the index page level. One might expect that unchanged percentage of data on the category *News* at the top level page set would be much lower than the index page, but the difference is just the 12%. The reason for such result is that index page is updated constantly, however the rest of the web site where old news are located, remains constant for several days. It produces a fast update but only at the index load page.

The fraction of unchanged content is not enough to characterize the potential cacheability of web sites; we have to quantify the number of unchanged bytes. Figure 5.12 shows the number of identical bytes present at the index load page level. The average amount of unchanged content over 15-minute interval is 83 Kilobytes across categories. All the categories have similar results but the category *News*. The results are closer between categories compared to the fraction of unchanged data depicted in Figure 5.10. It is because there is a difference in the size of the web page between categories.

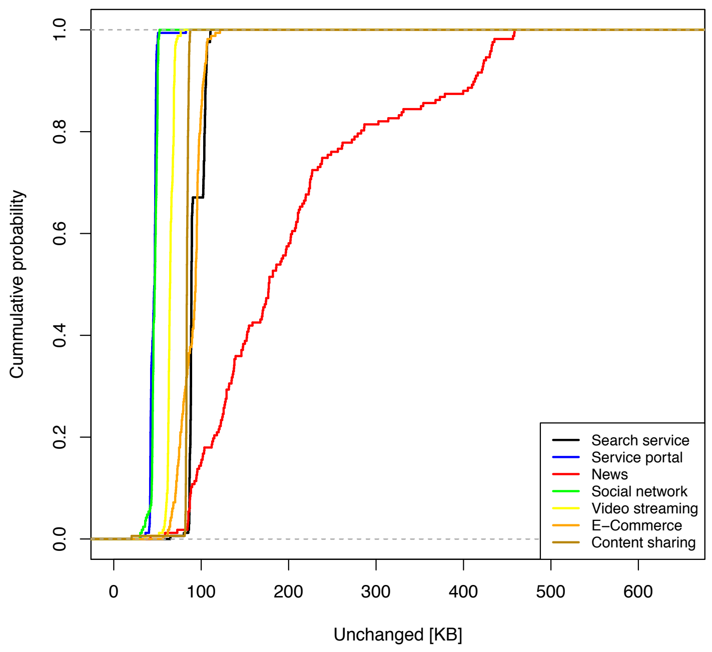


Figure 5.12 Unchanged bytes for 15-minute interval between two downloads of the same page at the index page level grouped by category

There is some similarity between the web site size and amount of unchanged data. Fetterly et al., [41] concluded that the document size is a good predictor of the degree of change. The changes in small objects produces lesser amount of unchanged bytes compared to the same fraction of big files. For instance, if an object has a large percentage of common data but its size is small (few kilobytes), the amount of unchanged data will be low. Otherwise if the fraction of unchanged data is small but the file is big (tens of kilobytes) the number of unchanged bytes will be high. That similarity is still valid when we compare the files at the top-level page set. Figure 5.13 depicts the redundant kilobytes for data beyond the first page. The median size of web pages is 3192 KB, whereas the unchanged content is just 1713 KB, which means that half of the size of the web page may benefit from caching.

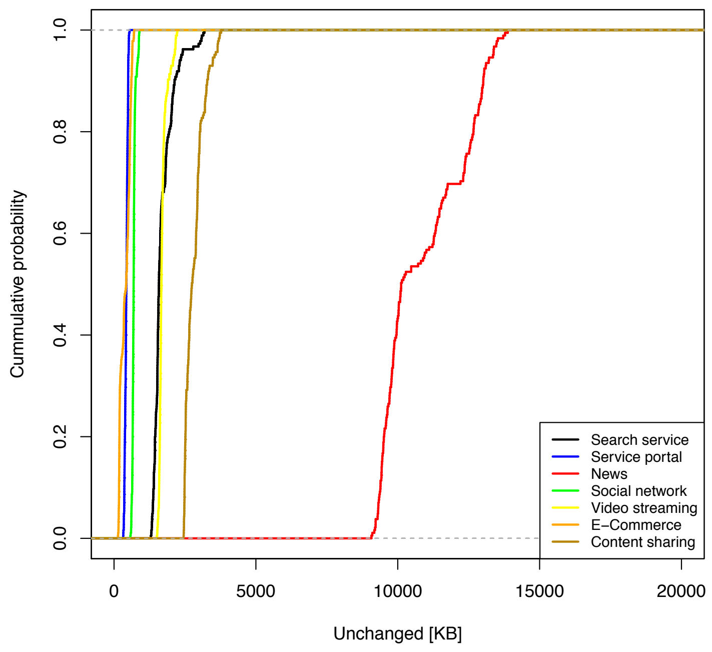


Figure 5.13 Unchanged bytes for 15-minute interval between two downloads of the same site at the top-level page set grouped by category

To determine where changes are produced we have compared all files at the top-level page set counting which files are new and which were present in previous 15-minute intervals. Figure 5.14 depicts the relation between previous versions of existing files and introduced files between intervals. Similar to previous plots, *News* stands as the category with the largest number of total files, 567 in median, followed by *E-Commerce* and *Content sharing*, with 162 and 153 files respectively. On the contrary, *Social network* and *Service portal* are the domains with fewer files, 30 and 22 accordingly. Earlier we have pointed the small number of objects for *Social network* and their effect on the results for that category.

It is interesting to examine the update rate of new files to determine whether a web site modifies existing files or introduces new data. The web pages related to *E-Commerce* stands as the category with the lowest ratio, just 11% of their files are constant between consecutive comparison intervals. The fraction of common files is 69% across all categories. Part of this low ratio is explained by the dynamism of that category because these web sites personalize the offers depending on the user's preferences.

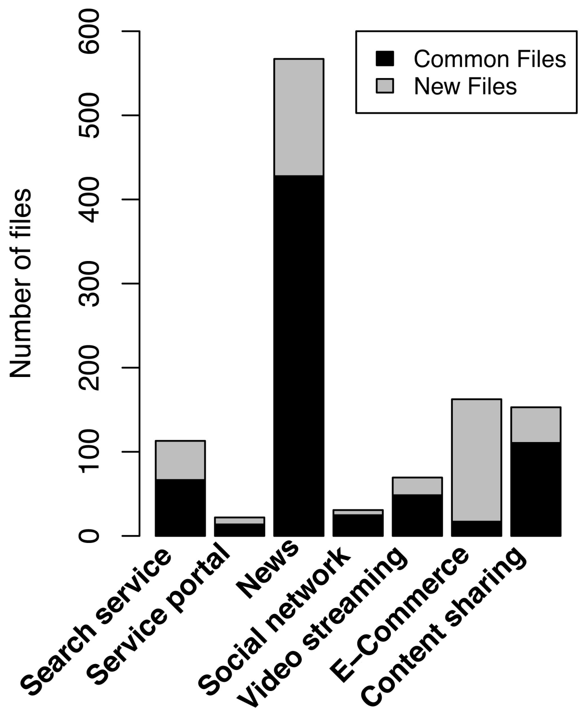


Figure 5.14 Total number of files at the top-level page set per category

The high percentage of common files represents that changes are produced mostly on existing documents. Thereby caches might use management algorithms of which metrics takes into account the time that the file has been in cache. For instance LRU or LFU, rather than algorithms that focuses only in their size.

Among the files that were present in previous comparison intervals, we detail in Figure 5.15 the corresponding number of files that are exactly the same for two consecutive intervals of 15 minutes. These are the files that potentially benefit from caching. *Content sharing*, *News* and *Social network* are the categories with higher percentage of identical files among the files that remained from the previous comparison interval with 98%, 95% and 81% respectively. In average across all categories, the 71% of files that are common between two comparisons iterations are identical, which means that changes are produced mostly in new files.

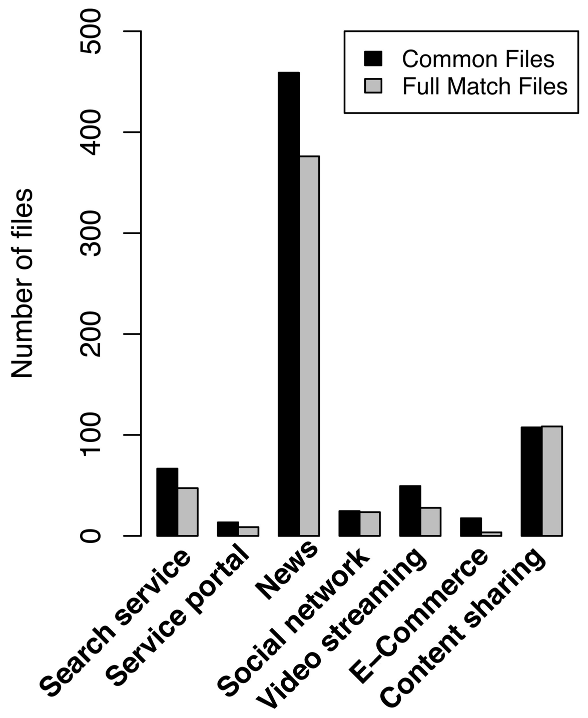


Figure 5.15 Identical files distribution at the top-level page set per category

We have confirmed that not all categories change likewise, probably because the type of service of some web services makes introduce content more often than others. Nevertheless the fraction of unchanged data is higher at top-level page set in all the cases and does not correspond to the number of unchanged bytes for the same analysis level. We have found a similarity between size and unchanged bytes because changes are in small files. We also found that the majority of files are constant between iteration and among these files most of the files are identical.

5.2.2 Block-level comparison

Finally, in this section we use a different approach to determine the potential cacheability of web sites based on packet caching. We have not examined the data at a low level as packet caching does, but we have compared blocks of data rather than the entire file. It is an approximation on packet-level caching because we divide files into smaller pieces. With this approach we determine the potential cacheability of the content for web caches that implements packet caching.

The following exhibit, Figure 5.16, depicts the percentage of unchanged data at the index page level using blocks of 1460 bytes, see section 4.1. This percentage is calculated by dividing the unchanged blocks at the index web page with the total number of blocks obtained from that web page. The average percentage of unchanged blocks is surprisingly low with 20% across all categories, which is lower compared to the 77% obtained by entire file comparison as it is shown in Figure 5.10.

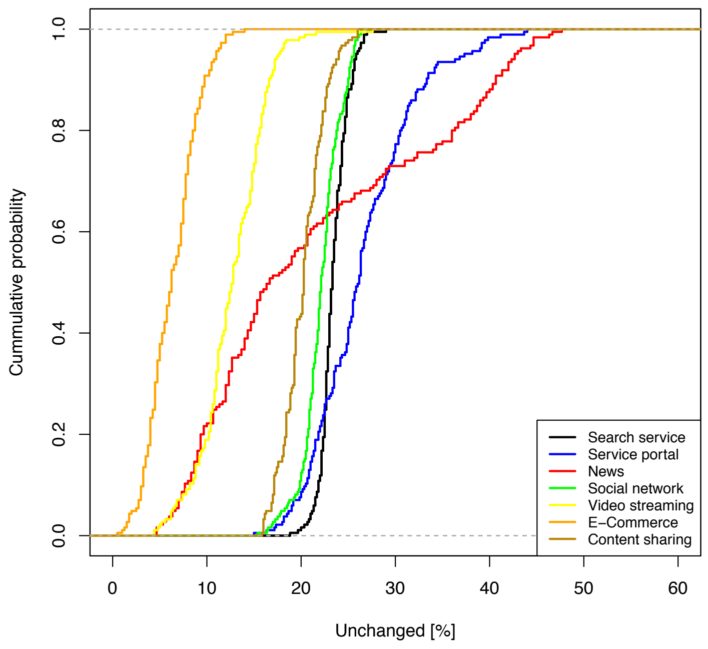


Figure 5.16 Unchanged percentage with block comparison for 15-minute interval between two downloads of the same page at the index web page by category

The main reason for that low percentage of unchanged data is that any change at the beginning of the web page affects deeply to the whole percentage. Since we are comparing constant blocks size, if any dynamic content or reference changes its length, the remaining bytes of that content will be shifted to the next block. Thereafter these block are different, hence the fraction of unchanged data is decreased even if the change is only in one byte. For instance, *YouTube* personalize content depending on what is popular at that time. In the code below we can see a sample of *YouTube's* personalization code.

```
yt.timing.info('e',
     "909520,914037,906040,907217,907335,921602,919306,919316,904455,
     912804,919324,912706,904452")
```

If we compare that sample with the same part of code at the next iteration of our implementation (below), the length of the sample differs from the above code.

```
yt.timing.info('e',
     "909707,907217,907335,921602,919306,919316,904455,912804,919324,
     912706,904452")
```

The difference of two values has a big effect on the rest of the blocks because the length of that part of code has changed and the rest of blocks will not match their corresponding blocks. This "shift effect" is caused as well for advertising due to ads do not have the same length. We have extracted two HTML advertising's tags for consecutive iterations in *Yahoo*'s website.

```
<div class="gdhp-product-pod gdhp-rounded-corners gdhp-product-pod-
     left"  title="Make your website run faster - From $4.24/mo"
     onclick="location.href='http://www.en.yahoo.com/hosting/web-
     hosting.aspx?ci=21391';" style="position:static">

<div class="gdhp-product-pod gdhp-rounded-corners gdhp-product-pod-
     left" title="Get more visitors to your website! - From $2.69/mo"
     onclick="location.href='http://www.en.yahoo.com/search-
     engine/seo-services.aspx?ci=57555';" style="position:static">
```

The length of the title and the URL where the ad is located differs from each sample, shifting part of the HTML element to the next block. That shifting effect is done in every index page that we have compared.

The results from Figure 5.16 differ for the index web page level when we determine the similarities at the one level web page, as Figure 5.17 depicts. To calculate the percentage of unchanged data we have performed several calculations. We have averaged the number of unchanged blocks within a web site with the total number of blocks that has this web site. Then, all web sites belonging to one category are averaged. By doing these calculations we observe that some categories have similar and, in some cases higher fraction of unchanged resources than we obtained in Figure 5.11. Such categories, *Search service*, *News* and *Content sharing* have more unchanged blocks. In fact, the average fraction of unchanged content using this method is 53%, slightly lower compared to the 57% that we have obtained with the file-level comparing method.

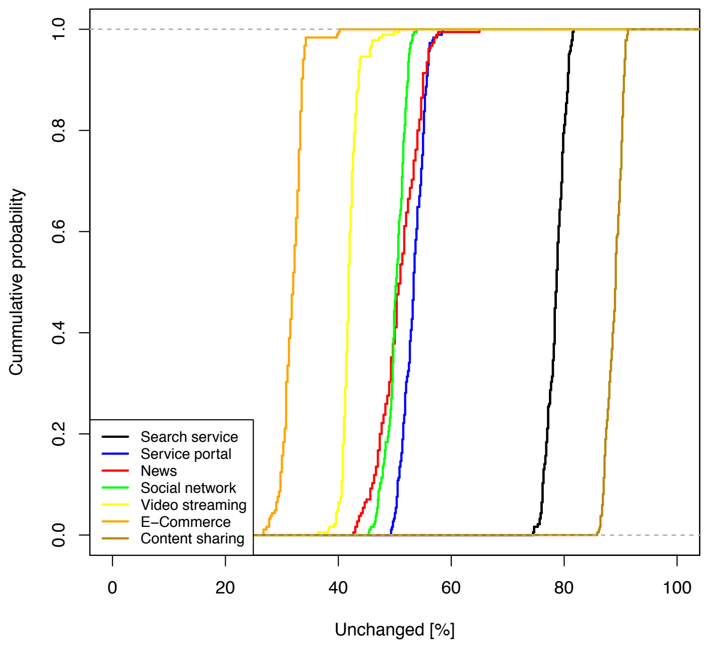


Figure 5.17 Unchanged percentage with block comparison for 15-minute interval between two downloads of the same site at the top-level page set by category

The narrow difference between both comparison methods, file and block levels, at top-level page set indicates that the data directly linked to the index page do not have the same level of personalization than the index load page.

The low percentage of unchanged data at the index web page level is traduced in less unchanged bytes as Figure 5.18 shows. The average amount of unchanged bytes across all categories is 20 KB, which is less than the 78 KB calculated with the traditional caching at the same level. *Service portal* is the category that produces the similar amount of unchanged bytes in both comparing methods with only 9 KB of difference.

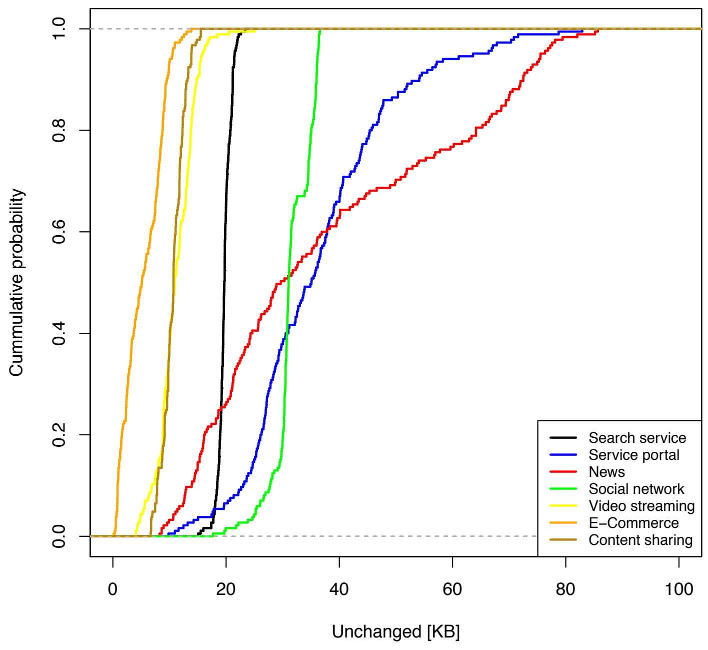


Figure 5.18 Unchanged bytes with block comparison for 15-minute interval between two downloads of the same page at the index web page level grouped by category

The same analysis at top-level page set with the file-level comparison yields different results as Figure 5.19 depicts. The average of unchanged bytes is 1772 KB slightly larger than the 1713 KB calculated with the file-level comparison. Only two categories have lower results, *News* and *Social network* with 8875 KB and 40 KB accordingly. In fact, if we exclude both categories from our analysis, the difference between comparison methods reduces to 111 KB of unchanged data. These differences confirm that at this level the tracking systems and personalization are implemented in a lesser degree.

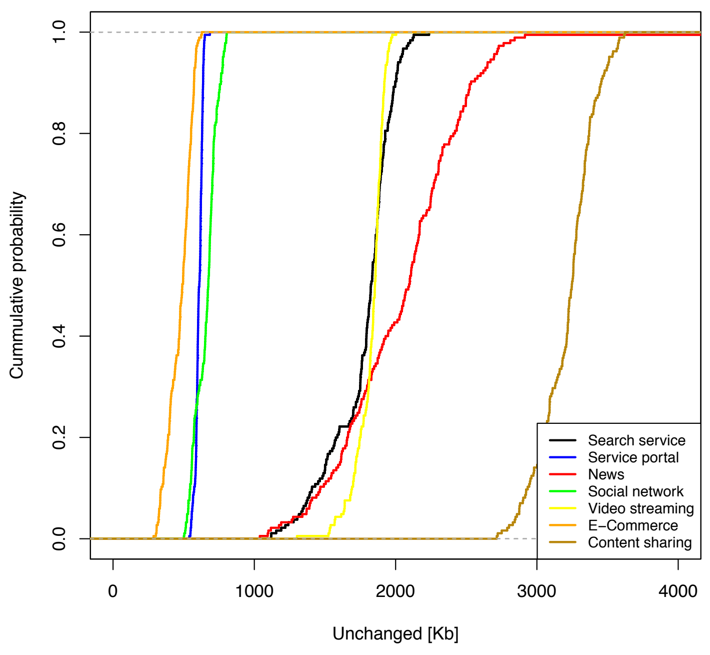


Figure 5.19 Unchanged kilobytes with block comparison for 15-minute interval between two downloads of the same site at the top-level page set grouped by category

We have noticed that the ideal size of the blocks strongly depends on the expected packet size in the network because all the packets do not have the same length. Our implementation is an approximation of the packet level caching, therefore our results are not exactly the same than an actual packet level cache. Nevertheless the level of personalization and tracking code at the index page level makes use of packet caching techniques more difficult.

As summary this section details the results that we have obtained with our implementation. At the index web page level we found that web pages mostly composed by HTML and CSS are the most static. By looking at the size of the index page, we can predict in some degree the cacheability of that web site. When we look at the top-level page set we observe a relevant increase of the size as well as a reduction of the unchanged data. If we look at the number of files, more than half of files are present in two consecutive comparison iterations. The block-level comparison yields lower results for the index load page level but obtains a greater percentage of unchanged data for the top-level page set. This difference indicates a lesser level of personalization for the web pages linked to the index page. The next section explains the conclusions and possible future work.

Chapter 6

Conclusions and future work

Along the years the WWW has faced multiples transformations, adopting emerging technologies. As consequence, we have observed an increasing number of objects per index page from few objects per page in 1998, to more than 100 objects for half of the compared web services. Technologies such as JavaScript, JavaScript or CSS, inexistent in earlier days of the WWW are now popular, being approximately 22% of the bytes, a relevant portion of the web page. Much of the web content is still HTML with more than 43% of the bytes per web site. We believe that percentage will increase as HTML5 gets adopted because of the improvements in multimedia media. Such improvements are focused on incorporating audio/video codecs and vectored graphics. The predominant use of HTML compared to JavaScript or AJAX favors caching systems because contains more unchanged bytes than other content types.

Although we have not noticed an increment of the total size of web pages for the last few years, the size of objects has become smaller. One contributor for that reduction in size is the minification process, which reduces the size of an object without compression. The reduction of the objects size involves less time to load the objects, decreasing the waiting time of the users. Hence, for small objects the Round Trip Time (RTT) becomes dominant performance factor. Therefore having the content close to the client (i.e. at cache) improves the performance because RTT can be expected to become smaller. Nevertheless, the number of images per page has decreased but their size has increased in the last few years. Probably this is because different images are merged into one picture, known as tiled images. Transmitting very small pictures separately it is not efficient because it adds one header for each transmitted image. Merging multiple images into one avoids overhead since only one image is transmitted.

Including or treating all web pages likewise has been the model for web caching systems for many years. We have defined some categories where similar web sites are joined according to the services that are offering. Our results are consistent with [47], finding that web sites with the same services behave similar.

At the Index web page level we have found that Search service and Content sharing stands as the most static web pages because are composed mostly by HTML and CSS objects. On the other side, Video streaming and News are the categories with less unchanged data. The type of service explains it, in part, because these categories update more regularly than others categories. However, the results differ when we look at the top-level page set. The fraction of unchanged data is reduced by 21% in average across categories. The categories with higher percentage of unchanged data at this level are Search service and Content sharing. That means that these categories update their index web pages rather than the content directly linked to the index load page.

The unchanged content, in terms of bytes, differs from the trend of percentage of redundant data. In fact, we can see different trends between the percentage of unchanged data and the number of unchanged kilobytes. We have found that the size of the index load page is a good predictor of the unchanged bytes for a web page. It could be used for caching systems to determine whether a web site is worth to be cached or not.

On the other hand, we have determined the number of similar files for each category. As median, 62% of documents were similar with previous iterations; therefore, the changes are mostly produced on these files. Web caches could use that information to not store documents related to some web sites. For instance, 89% of the files in E-Commerce are new between comparison iterations. Storing files from that category is not efficient because the files will change in a short period of time. In addition, we have detailed the number of files that are exactly the same from previous 15-time intervals. Content sharing, News and Social network stands as the categories with higher number of identical files between intervals. It means that these files remains identically over a medium time period hence are more suitable for being cached than files from E-Commerce.

We determine unchanged content with another method, splitting files into equal-sized blocks and quantifying their differences. Our results for the index load page level are not as good as our previous analysis with the file-level comparison method. In fact, redundant content has been reduced more than 70 KB in median. It is explained by the personalization, tracking systems and advertising. All these systems are generated dynamically and the length of that content varies between comparisons. Any change in the length of the content will shift the excess of bytes to the next fixed-sized block. That dynamism reduces the unchanged bytes obtained with the block-level comparison. However, the redundancy increases when we look to the top-level page set. Therefore dynamic data is strongly present at the index page but not for objects directly connected to the index page.

As for future work, we described in previous sections the creation of the folders we added and extra level for browsers. It is known that different browsers interpret the resources within a web page differently. For instance, web browsers use different engines to interpret JavaScript or uses different caching approaches on the client-side. We let an open door to investigate differences among browsers in terms on cacheability but it is a hard work because of the fast web browsers updating rate.

The traffic generated by mobile phones was eight times the traffic generated on Internet in 2000 according to [99]. Cacheability of mobile traffic has not been deeply studied and it could be an interesting research topic. Not only mobile phones are generating more traffic than computers, the data generated by tablets exceeds 3.4 times the traffic generated by smartphones [99].

When two users, implementing caching mechanism, access to one domain at different times, the cache stores the web site the first time and thereafter serves it to the others users the same web site locally, if the resource has not been modified. As we have discussed previously in section 4.3.3 that web sites do not change completely, usually they only change a portion of the content. An interesting application would be the use of advanced caching mechanisms capable to serve part of web sites rather than the entire site.

One comparison method used in this work is based on splitting files into smaller pieces in order to compare them. We have used blocks of 1460 bytes, but we encourage using different block sizes to determine which size fits better to each category.

Bibliography

[1] Frank La Rue, "Report of the Special Rapporteur on the promotion and protec- tion of the right to freedom of opinion and expression "United Nations General Assembly Human Rights Council, Resolution 7/36, 2011.

[2] Pekka Markkula, "Mobile Operators and Mobile Cloud (presentation)," in *Telecommunications Forum 2010*, Espoo, Helsinki, 2010.

[3] Osma Ahvenlampi, "Games in the Social Age," in *Telecommunications Forum 2010*, Espoo, Helsinki, 2010.

[4] S. Ihm, "Understanding and Improving Modern Web Traffic Caching" Computer Science, Princeston School of Engineering and Applied Science, Technical Report 908-11, 2011.

[5] Paul Bardord and Mark Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," in *In Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, vol. 26(1), 1998, pp. 151-160.

[6] Fiona Fui-Hoon Nah, "A study on tolerable waiting time: how long are Web users willing to wait?," *Behavior and Information Technology*, vol. 23, no. 3, pp. 153-163, 2004.

[7] Martin F. Arlitt and Carey L. Williamson, "Internet Web Servers: Workload Characterization and Performance Implications ," in *IEEE/ACM Transactions on Networking (TON)*, vol. 5(5), 1997, pp. 631-654.

[8] Anja Feldman, B. Krishnamurthy and J. Mogul Fred Douglis, "Rate of Change and other Metrics: a Live Study of the World Wide Web," in *USENIX Symposium on Internetworking Technologies and Systems (USITS)*, 1997.

[9] James E. Pitkow, "Summary of WWW Characterizations," *World Wide Web*, vol. 2, no. 1-2, pp. 1-13, 1999.

[10] Stephen Manley and Margo Seltzer, "Web Facts and Fantasy," in *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 1997.

[11] Craig E. Wills and Mikhail Mikhailov, "Towards a better understanding of Web resources and server responses for improved caching," in *In Proceeding of the 8th International World Wide Web Conference*, 1999, pp. 153-165.

[12] Xiangping Chen and Prasant Mohapatra, "Lifetime Behaviour and its Impact on Web Caching," in *Proceeding of the IEEE Workshop on Internet Applications*, 1999.

[13] Derek Eager and Carey Williamson Anirban Mahanti, "Temporal Locality and its Impact on Web Proxy Cache Performance," in *Performance Evaluation - Special issue on internet performance modelling*, vol. 43 (2-3), 2000, pp. 187-203.

[14] Ricardo Baeza-Yates and Carlos Castillo, "Crawling the Infinite Web," *Journal of Web engineering*, vol. 6, no. 1, pp. 49-72, Frebruary 2007.

[15] Ipoque. Ipoque. [Online]. http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009 Last visited on May, 2012.

[16] A. Feldman, V. Paxson and M. Allman G. Maier, "On Dominant Characteristics of

Residential Broadband internet Traffic," in *Proceedings of the 2009 Internet Measurement Conference (IMC 2009)*, May 2009. [Online]. http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009

[17] R. Fielding and L. Masinter T. Berners-Lee, "Uniform Resource Identifier (UR): Generic Syntax" W3C, RFC 3986, 2005.

[18] J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie and C. Lilley H. F. Nielsen, "Network Performance Effects of HTTP/1.1, CSS1, and PNG," in *Proceedings of the ACM SIGCOMM'97 conference on Applications, technologies, architectures, and protocols for computer communication*, vol. 27 (4), 1997, pp. 155-166.

[19] M. Allman and V. Paxson T. Callahan, "A Longitudinal View of HTTP Traffic," in *Proceedings of the 11th Passive and Active Measurement Conference (PAM 2010)*, 2010, pp. 222-231.

[20] Gaogang Xie and Jianhua Yang Liang Shuai, "Characterization of HTTP Behavior on Access Networks in Web 2.0," in *In proceeding on International Conference on Telecommunications*, 2008, pp. 1-6.

[21] Tim O'reilly. Oreilly. [Online]. http://oreilly.com/web2/ Last visited on Sep., 2005.

[22] R. Fielding and H. Frystyk T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.0" Network Working Group, The Internet Engineering Task Force (IETF), RFC 1945, 1996.

[23] J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee R. Fielding, "Hypertext Transfer Protocol (HTTP/1.1)" Network Working Group, RFC 2616, 1999.

[24] David H. Crocker, "Standard for the format of ARPA Internet text messages" RFC 822, 1982.

[25] R. Braden, "Requirements for Internet Hosts - Application and Support" Internet Engineering Task Force, RFC 1123,.

[26] HTTP Archive. HTTP archive. [Online]. http://httparchive.org/interesting.php#max-age Last visited on 2013.

[27] Craig E. Wills and Mikhail Mikhailov, "Examining the Cacheability of User-Requested Web Resources," in *In Proceedings of the 4th International Web Caching Workshop*, 1999.

[28] IANA. Internet Assigned Numbers Authority (IANA). [Online]. http://www.iana.org Last visited on October, 1970.

[29] N. Freed and N. Borenstein, "Multipurpose Internet Mail Extensions" Network Working Group, RFC 2046, 1996.

[30] W3C. W3C Working Draft (AJAX). [Online]. http://www.w3.org/TR/2012/WD-XMLHttpRequest-20121206/ Last visited on December, 2012.

[31] Jeffrey C. Mogul, "A trace-based analysis of duplicate suppression in HTTP" Western Research Laboratory, Compaq Computer Corporation, Palo Alto, California, Technical Report 97/4, 1999.

[32] Jeffrey C. Mogul, "Potential benefits of delta-encoding and data compression for HTTP," in *In Proceedings of the ACM SIGCOMM'97 conference on Applications, technologies, architectures and protocols for computer communication*, 1997, pp. 181-194.

[33] Zhang Luwei, "Cacheability study for web content delivery" Computer Science, National University of Singapore, Thesis 2003.

[34] ECMA, "ECMAScript Language Specification" ECMA International, Standard ECMA-

262, 2011.

[35] W3C. W3C Cascading Style Sheets, level 1. [Online]. http://www.w3.org/TR/CSS1/ Last visited on December, 1996.

[36] Jeffrey C. Mogul, "The case of persistent-connection HTTP," in *In proceeding of SIGCOMM'95*, 1995, pp. 299-313.

[37] Venakta N. Padmanabhan, Srinivasan Sesham, Mark Stemm and Randy H. Katz Hari Balakrishnan, "TCP behavior of a busy Internet server: Analysis and improvements ," in *In proceeding of 17th IEEE INFOCOM*, 1998.

[38] The Chrominum Projects. Chrominum. [Online]. http://www.chromium.org/spdy/spdy-whitepaper

[39] Leo A. Meyerovich and Rastislav Bodik, "Fast and parallel web page layout," in *In proceeding of the 19th International conference on World Wide Web*, 2010, pp. 711-720.

[40] Junghoo Cho and Hector Garcia-Molina, "The evolution of the web and implications for an incremental crawler," in *In proceeding of the 26th International conference on very large databases*, 2000.

[41] Mark Manasse, Marc Najork and Janet Wiener Dennis Fetterly, "A Large-Scale Study of the Evolution of Web Pages," in *In proceeding of the 12th International conference on World Wide Web*, 2003, pp. 669-678.

[42] Azer Bestavros and Mark E. Crovella Carlos R. Cunha, "Characteristics of WWW Client-based Traes" Computer Science Department, Boston University, Technical report BU-CS-95-010, 1995.

[43] G. K. Zipf, *Human behaviour and the principle of least-effort.*: Addison Wesley, 1949.

[44] Venkata N. Padmanabhan and Lili Qiu, "The content and access dynamics of a busy Web site: findings and implications," in *In Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, vol. 30(4), 2000, pp. 111-123.

[45] Pei Cao, Li Fan, Graham Philips and Scott Shenker Lee Breslau, "On the Implications of Zipf's Law for Web Caching," in *In Proceedings of the 3rd International WWW Caching Workshop*, Manchester, England, 1999.

[46] Pei Cue, Pei Cao, Li Fan, Graham Philips and Scott Shenker Lee Breslau, "Web Caching and Zipf-like Distributions: Evidence and Implications," in *In Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, vol. 1, 1999, pp. 126-134.

[47] Harsha V. Madhyastha and Vyas Sekas Michael Butkiewicz, "Understanding Website Complexity: Measurements, Metrics, and Implications," in *In proceeding of the ACM SIGCOMM conference on Internet measurement conference*, 2011, pp. 313-328.

[48] Sunghwan Ihm and Vivek S. Pai, "Towards Understanding Modern Web Traffic," in *In proceeding of the ACM SIGCOMM conference on Internet measurement conference*, 2011, pp. 295-312.

[49] I. Melve and G. Tomlinson T. Cooper, "Internet Web Replication & Caching Taxonomy" The Internet Engineering Task Force, RFC 3040, 2010.

[50] Josh Cohen, Martin Dunsmir and Charles Perkins Paul Gauthier, "Web Proxy Auto-Discovery Protocol" The Internet Engineering Task Force, DRAFT 1999.

[51] Marina Buzzi, Damir Pobric and Massimo Ianigro Laura Abba, "Introducing Trasparent Web Caching in local area network," in *Proceeding on the 26th International Computer*

Measurement Group Conference, 2000.

[52] Squid. Squid. [Online]. http://www.squid-cache.org Last visited on June, 2013.

[53] S. Floyd and V. Jacobson L. Zhang, "Adaptative Web Caching," in *Proceedings of the NLANR Web Cache Workshop*, 1997.

[54] K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd and V. Jacobson S. Michel, "Adaptive web caching: towards a new global caching architecture," in *3rd International WWW Caching Workshop*, 1998.

[55] Konstantinos Nikoloudakis, P. Reiher and L. Zhang B. Scott Michel, "URL forwarding and compression in adaptive Web caching," in *IEEE INFOCOM conference*, 2000.

[56] M. Mikhailov and C. E. Wills, "Evaluating a New Approach to Strong Web Cache Consistency with Snapshots of Collected Content," in *12th International World Wide Web Conference*, 2003.

[57] Brian E. Brewington and George Cybenko, "How dynamic is the web?," in *In Proceeding of the 9th International World Wide Web Conference*, 2000.

[58] Craig E. Wills and Mikhail Mikhallov, "Studying the Impact of More Complete Server Information on Web Caching ," in *Proceeding of the 5th International Web Caching and Content Delivery Workshoop*, 2000.

[59] J. Zhang and K. Beach P. Cao, "Active Cache: Caching Dynamic Contents on the Web," in *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, 1998, pp. 378-388.

[60] Manolis G. H. Katevenis, Dionisis Pnevmatikatos and Michail Flouris Evangelos P. Markatos, "Secondary Storage Management for Web Proxies," in *Proceeding fo the 1999 Usenix Symposium on Internet Technologies and Systems*, 1999.

[61] J. Gwertzman, "Autonoums replication in wide-are internetworks," in *Proceedings of the 15th ACM symposium on Operating systems principles*, 1995, pp. 1-31, 60-65.

[62] J. Gwertzman and M. Seltzer, "The case for Geographical Push-Caching," in *Proceedings of the 5th Workshop on Hot Topics in Operating Systems*, 1995.

[63] M. Seltzer and J. Gwertzman, "An Analysis of Geographical Push-Caching" Proceeding of 5th IEEE Workshop on Hot Topics in Operating Systems, 1997.

[64] Udi Manber, "Finding Similar Files in a Large File System," in *Winter USENIX Technical Conference*, 1994.

[65] Andrei Z. Broder, "On the resemblance and containment of documents," in *In Proceedings of the Compression and Complexity of Sequences 1997*, 1997, pp. 21-29.

[66] Michael O. Rabin, "Fingerprinting by random polynomials" Center for Research in Computing Technology, Harvard University, Technical Report CSE-03-01, 1981.

[67] Neil T. Spring and David Wetherall, "A Protocol-Independent Technique for Eliminating Redundant Network Traffic ," in *In Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2000, pp. 87-95.

[68] Juniper. Juniper. [Online]. http://www.juniper.net Last visited on 2012.

[69] Riverbed. Riverbed. [Online]. http://www.riverbed.com/us/solutions/wan_optimization/ Last visited on 2012.

[70] Archit Gupta, Aditya Akella, Srinivasan Sesahn and Scott Shenker Ashok Anand, "Packet Caches on Routers: The Implications of Universal Redundant Traffic Elimination ," in *In Proceedings of the ACM SIGCOMM Conference*, 2008, pp. 219-230.

[71] Chitra Muthukrishnan, Aditya Akella and Ramachandran Ramjee Ashok Anand, "Redundancy in network traffic: findings and implications," in *In Proceedings of the 11th international joint conference on Measurement and modeling of computer systems*, 2009, pp. 37-48.

[72] Kevin Liang Sean C. Rhea, "Value-Based Web Caching," in *In Proceedings of the 12th International World Wide Web Conference*, 2003, pp. 619-628.

[73] Marc Abrams, Charles R. Standridge, Chaleb Abdulla and Edward A. Fox Stephen Williams, "Removal Policies in Network Caches for World-Wide Web Documents (reviewed)," in *Proceedings on ACM SIGCOMM*, 1996.

[74] Charles R. Standridge, Chaleb Abdulla, Stephen Williams and Edward A. Fox Marc Abrams, "Caching Proxies: Limitations and Potentials," in *Proceeding on the 4th International World Wide Web Conference*, 1995.

[75] James E. Pitkow and Margaret M. Recker, "A Simple Yet Robust Caching Algorithm Based on Dynamic Access Patterns ," in *Proceedings on the 4th International World Wide Web Conference*, 1994.

[76] Alexa. Alexa: The Web Information Company. [Online]. http://www.alexa.com/topsites Last visited on June, 2011.

[77] Jeffrey Dean and Sanjay Ghemawat, "Mapreduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107-113, 2008.

[78] W. Eddy, "TCP SYN Flooding Attacks and Common Mitigations" Network Working Group, IETF, RFC 4987, 2007.

[79] B. Hoehrmann, "Scripting Media Types" Network Working Group, RFC 4329, 2006.

[80] Kevin Jeffay and F. Donelson Smith Ferlix hernandex-Campos, "Tracking the Evolution of Web Traffic: 1995-2003," in *Proceeding of the 11th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications System (MASCOTS)*, 2003, pp. 16-25.

[81] J. Mogul and S. Deering, "Path MTU Discovery" Network Working Group, RFC 1191, 1990.

[82] The Open Group Base Specifications, "Crontab- Schedule periodic background work" IEEE, IEEE Std 1003.1, 2004.

[83] R Fundation. R Project. [Online]. http://www.r-project.org Last visited on April, 2013.

[84] Center for Science and Culture. CSC - IT Center for Science. [Online]. http://www.csc.fi/english/institutions/funet/ Last visited on 1984.

[85] Micah Cowan. GNU Wget. [Online]. http://www.gnu.org/software/wget/ Last visited on August, 2012.

[86] Robin Wauters. Chrom rises: Google browser grabs 1/3 of the global market. [Online]. http://thenextweb.com/google/2012/08/06/chrome-rises-google-browser-grabs-13-of-the-global-market-statcounter/ Last visited on August, 2012.

[87] BBC News. BBC News. [Online]. http://news.bbc.co.uk/2/hi/science/nature/409980.stm Last visited on August, 1999.

[88] Jennifer Kahn, "The Homeless Hacker v. The New York Times," *The Wired*, vol. 12, no. 04, pp. 1-4, April 2004.

[89] Henry S. Baird and Mark Luk, "Protecting Websites with Reading-Based CAPTCHAs," in *Proceeding of the 2th Web Document Analysis*, 2003, pp. 50-56.

[90] M. Koster. A standard for robots exclusion. [Online].

http://www.robotstxt.org/orig.html Last visited on March, 2012.

[91] Google. Google TrendsGoogle Trends. [Online]. http://www.google.com/trends/

[92] cURL. cURL. [Online]. http://curl.haxx.se Last visited on September, 2012.

[93] Xavier Roche. HTTrack online. [Online]. http://www.httrack.com Last visited on December, 2000.

[94] TIBCO Software. Spotfire (S-Plus). [Online]. http://spotfire.tibco.com/discover-spotfire Last visited on November, 2010.

[95] GNU Project. PSPP. [Online]. http://www.gnu.org/software/pspp/pspp.html Last visited on October, 2009.

[96] Wesley Hales, *HTML5 and JavaScript Web Apps*, First edition ed., Simon St. Laurent and Meghan Blanchette, Ed.: O'Reilly Media, 2013, ISBN: 978-1-449-32051-5.

[97] Google. Google Developers. [Online]. https://developers.google.com/closure/compiler/ Last visited on 2012.

[98] Yahoo! YUI LIbrary. [Online]. http://yuilibrary.com Last visited on 2011.

[99] Cisco Systems, "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update" Cisco Systems, White paper 2012.

[100] Arun Iyengar, Junehwa Song and Daniel Dias Eric Levy-Abegnoli, "Desing and Performance of a Web Server Accelerator" Proceedings of the INFOCOM conference, 1999.

[101] T. Berners-Lee, "Hypertext Markup Language" The Internet Engineering Task Force, RFC 1866, 1995.

[102] P. Allen and S. Black V. Varadharajan, "An Analysis of the Proxy Problem in Distributed Systems ," in *In Proceedings of the 19991 IEEE Computer Society Symposium on Research in Security and Privacy*, 1991, pp. 255-275.

[103] Robert Auger, "Socket Capable Browser Plugins Result In Transparent Proxy Abuse "Paypal Information Risk Management Team, Vulnerability note 2009.

Appendix A

URL	Category	URL	Category
www.google.fi	Search engine	www.fc2.com	Service portal
www.facebook.com	Social network	www.go.com	Content sharing
www.youtube.com	Video streaming	www.bing.com	Search engine
www.wikipedia.org	Content sharing	www.bbc.com	Video streaming
www.yahoo.com	Service portal	www.cnn.com	Video streaming
www.photobucket.com	Social network	www.myspace.com	Service portal
www.badoo.com	Social network	www.vimeo.com	Video streaming
www.imgur.com	Social network	www.nytimes.com	News
www.twitpic.com	Social network	www.aol.com	Service portal
www.amazon.com	E-Commerce	www.mediafire.com	Content sharing
www.linkedin.com	Social network	www.baidu.com	Search engine
www.msn.com	Service portal	www.qq.com	Service portal
www.imageshack.us	Content sharing	www.ebay.com	E-Commerce
www.dailymotion.com	Video streaming	www.alibaba.com	E-Commerce
www.reddit.com	Video streaming	www.huffingtonpost.com	News
www.retuters.com	News	www.globo.com	News
www.spiegel.de	News	www.twitter.com	Social network
www.paypal.com	E-Commerce	www.netflix.com	Video streaming
www.flickr.com	Social network	www.4shared.com	Content sharing
www.craiglist.com	E-Commerce	www.blogspot.com	Content sharing
www.taobao.com	E-Commerce	www.espn.go.com	Video streaming
www.imdb.com	Content sharing	www.dropbox.com	Content sharing
www.weather.com	News	www.wordpress.com	Content sharing
www.rapidshare.com	Content sharing	www.tumblr.com	Social network
www.cnet.com	Content sharing	www.ask.com	Search engine