Web Optimizer: Reducing the Intermediate Web Traffic

Submitted by:
Al-Imtiaz
ID: 2014-2-96-004
Department of computer Science and Engineering,
East West University

Supervised by:
Dr. Mohammad Rezwanul Huq
Assistant Professor
Department of computer Science and Engineering,
East West University

East West University
Aftabnagar, Dhaka, Bangladesh, September 2016.

Web Optimizer: Reducing the Intermediate Web Traffic

Submitted by:
Al-Imtiaz
ID: 2014-2-96-004
Department of computer Science and Engineering,
East West University
2016

Supervised by:
Dr. Mohammad Rezwanul Huq
Assistant Professor
Department of computer Science and Engineering,
East West University

This thesis is submitted to the department of Computer Science and Engineering as the requirement for the degree of Master of Computer Science and Engineering at

East West University
Aftabnagar, Dhaka, Bangladesh, September 2016.

Abstract

Usage of Internet is growing rapidly at this time in all over the world. Among all other services, the World Wide Web is the most popular Internet applications. Internet contains various useful information and therefore, the dependency on Internet is increasing day by day. Providing Internet facilities as a tool to complete mundane activities becomes quite common in schools, colleges, government and non-government organizations etc.

However, the process of digitalization has not been achieved as desired due to the lack of infrastructure, awareness and proper use of technologies. Developing countries like Bangladesh are still struggling to provide Internet facilities to all other citizens. The availability of Internet at the user end and high expense of Internet data are the main obstacles right now.

There are so many technologies working together on server side to handle the thousands of client's requests at minimum possible time. Those thousands requests come back with thousands responses. All of those messages and contents use Internet as their communication medium and those contents are also known as web traffic.

Existing Technologies like web cache and Content Data Network (CDN) work to reduce the web traffic by caching it locally to serve it later. There are few scopes to improve the current technologies. The goal of this thesis is to reduce the web traffic that will lead to faster web browsing experience. Here "Cache Tree" and "LiteWeb" have been introduced and few existing technologies have been modified to work as "Cache Broadcasting" and "Cache Sharing".

Declaration

I hereby declare that, this thesis was done under CSE599 and has not been submitted elsewhere for requirement of any degree or diploma or for any purpose except for publication.

Al Imtiaz
ID: 2014-2-96-004
Department of Computer Science and Engineering,
East West University

Letter of Acceptance

The this entitled “Web Optimizer: Reducing the Intermediate Web Traffic” submitted by AL Imtiaz, ID: 2014-2-96-004 to the Department of Computer Science and Engineering, East West University, Dhaka 1212, Bangladesh is accepted as satisfactory for partial fulfillment of requirements for the degree of Masters of Science (MS) in Computer Science and Engineering on September, 2016.

Acknowledgement

Firstly, my most heartfelt gratitude goes to my beloved parents for their endless support, continuous inspiration, great contribution and perfect guidance from the beginning to the end.

I wish to pay my heartiest gratitude to Dr. Mohammad Rezwanul Huq, Asst. Professor for his encouragement and guidance to compete this thesis.

Table of Contents

List of Figures

Abbreviation and Acronyms

WO = Web Optimizer
CSWO = Client Side Web Optimizer
SSWO = Server Side Web Optimizer
LLpCC= Long Live Public Cacheable Content

Chapter 1
Introduction

In this age of digitalization and globalization the appeal of better internet facility is become one of the major issues. Web traffics have been increasing tremendously and it becomes a fundamental issue to provide a better experience to the end user by managing those traffics. These growths of web traffic are expected to continue, not only as the web becomes a de facto front-end for many emerging cloud-based services, but also as many applications getting migrated to the Web.

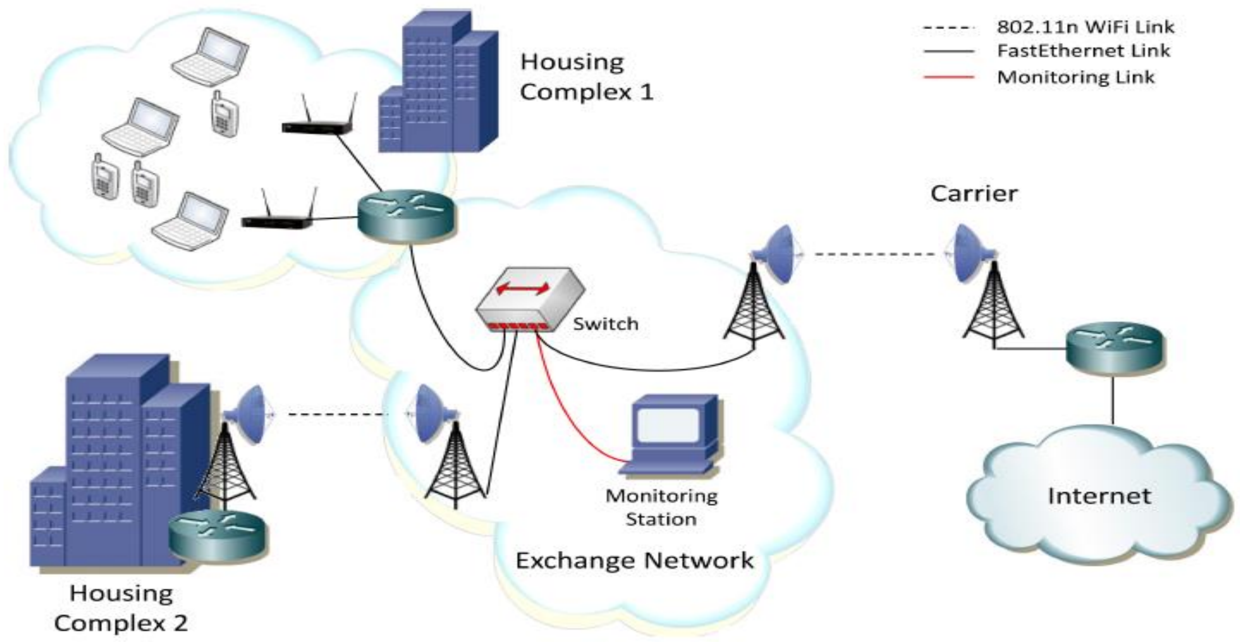


Figure 1: An approach of monitoring Web Traffic

The World Wide Web (www) is an information space where documents and other web resources are identified by URLs, interlinked by hypertext links, and can be accessed via the Internet. It has become known simply as the Web. The World Wide Web was central to the development of the Information Age and is the primary tool billions of people use to interact on the Internet.

Web pages are primarily text documents formatted and annotated with Hypertext Markup Language (HTML). In addition to formatted text, web pages may contain images, video, and software components that are rendered in the user's web browser as coherent pages of multimedia content. Embedded hyperlinks permit users to navigate between web pages. Multiple web pages with a common theme, a common domain name, or both, may be called a website. Website content can largely be provided by the publisher or interactive where users contribute content or the content depends upon the user or their actions. Websites may be mostly informative, primarily for entertainment, or largely for commercial purposes.

Web traffic is the amount of data traveled on the internet and those data are sent and received by the visitors of a web site and server. This internet has already become giant networks which have to maintain huge amounts of web traffic in every moment. This huge web traffic may cause the slowness of the entire system. The amount of Global digital information created and shared will be double within next two years according to the current growth rate of web traffic.

Web traffic is being acquired from the access log of web data. It only assumes simple and static Web pages, and once Web pages are identified, other contents can be derived, such as the number of embedded objects, total page size, total page time, and inter-arrival time.

Browser stores all cacheable web contents for a while. When a user browses the same page within a few minutes then the content will be served from the cache memory of the browser. Sometimes browser keeps those contents for a long time. And also each time there is a request for that cache content the browser also checks it is valid/fresh or not.

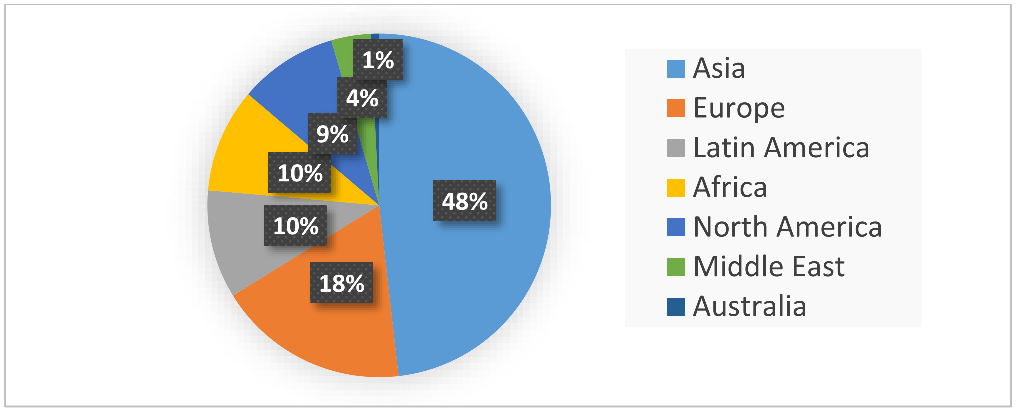


Figure 2: Internet user in the World by Regions

The web contents are increasing day by day. Hundreds of Giga Bytes are traveling this internet in each second. To serve better server are sending larger contents like images, videos to clients. Cache is a smart solution to reduce those static contents by saving those in local computer's memory. There are many different technologies such as Content Delivery network, proxy cache server to improve the overall performance of internet.

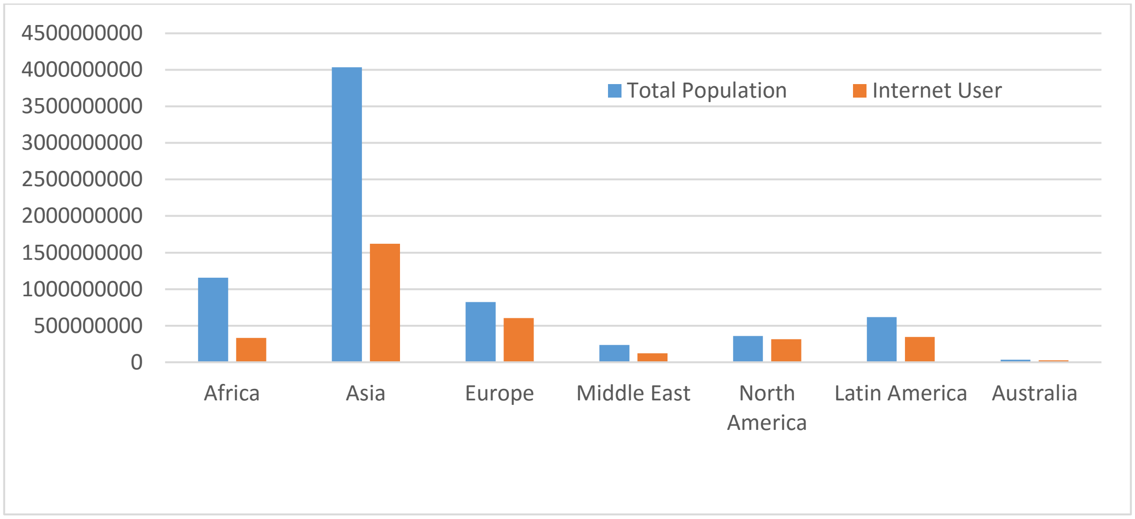


Figure 3: World internet usage and population Statistics

There are very few popular websites those are browsed more frequently.

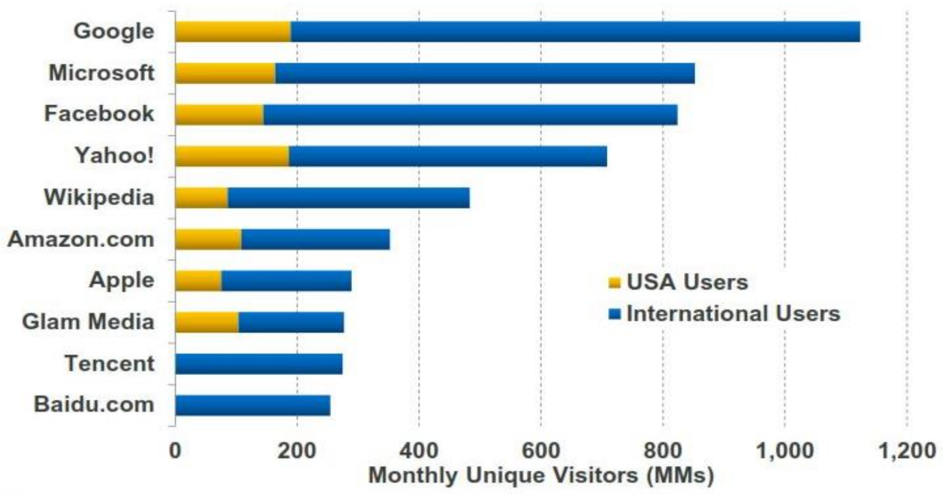


Figure 4shows the top 10 web sites. Among them 80% of top ten global internet properties are owned by USA [3].

In most of the time the popularity of a website depends on the geographical region. In most of the cases the users of a network browse same types of website (News portal, popular sites). Those contents have to be served from server. Among those content some are dynamic (very light weighted content) and others are static (images, weight of those contents is very high).

Each time a user browses a page, browser stores few contents to server it later. This mechanism is applied in all modern browsers.

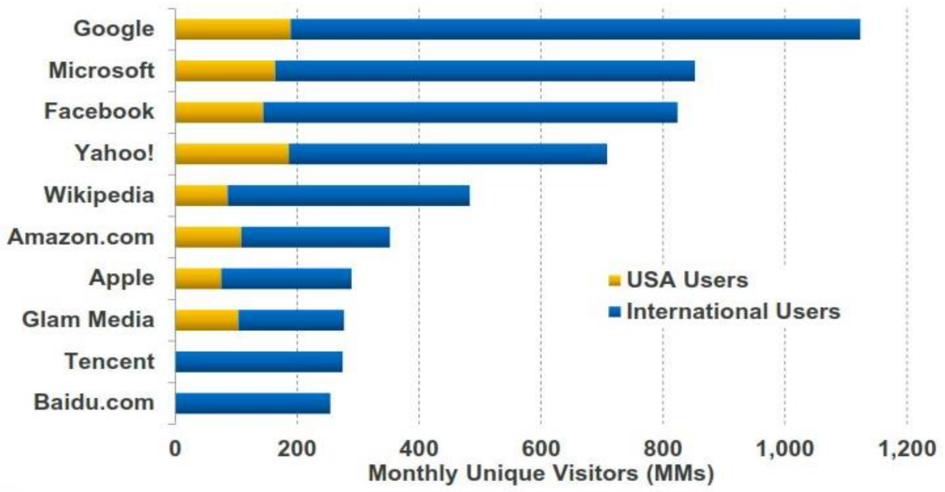


Figure 4: Top 10 Internet Properties by global monthly unique visitors, Feb, 2013 [3]

According to HTTP Archive, amongst the top 300,000 sites (by Alexa rank), nearly half of all, the downloaded responses can be cached by the browser, which is a huge savings for repeat page views and visits. Some application will have 50% of resources that can be cached: some sites can cache 90%+ of their resources, while others may have a lot of private or time-sensitive data that can't be cached at all. By 2019, mobile-connected tablets will generate nearly double the traffic generated by the entire global mobile network in 2014. [10]

Fetching something over the network is both slow and expensive: large responses require many round-trips between the client and server, which delays when they are available and can be processed by the browser, and also incurs data costs for the visitor. As a result, the ability to cache and reuse previously fetched resources is a critical aspect of optimizing for performance.

There are few existing technologies that help to improve the web browsing experience. Among them Web Cache and Content Delivery Network are most common technology used by developers. The working procedures of Web cache and CDN have been discussed on chapter 2.

Every browser has HTTP cache management architecture by default. It has to ensure that each server response provides correct HTTP header directives to instruct the browser on when and for how long the response can be cached by the browser.

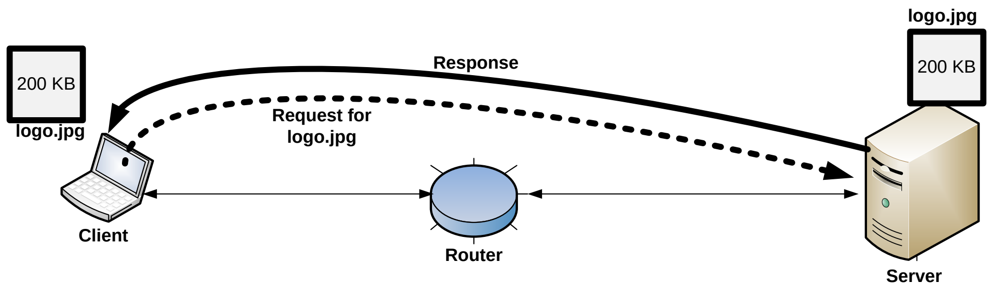


Figure 5: Content have been served from server

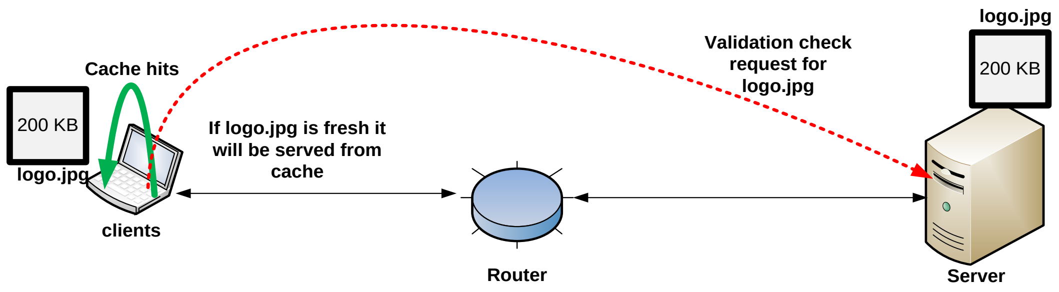

Cache hits
200 KB
logo.jpg
clients
If logo.jpg is fresh it will be served from cache
Validation check request for logo.jpg
logo.jpg
200 KB
Router
Server

Figure 6: Content have been served form browser's cache

Together, freshness and validation are the most important ways that a cache works with content. A fresh representation will be available instantly from the cache, while a validated representation will avoid sending the entire representation over again if it hasn't changed.

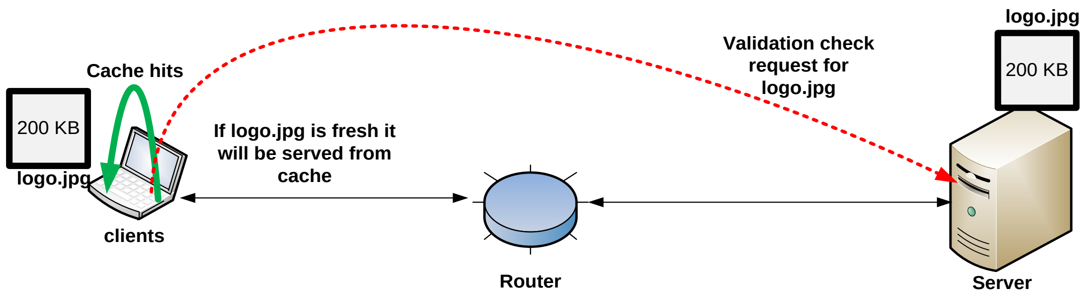

Cache hits
200 KB
logo.jpg
clients
If logo.jpg is fresh it will be served from cache
Validation check request for logo.jpg
logo.jpg
200 KB
Router
Server

Figure 6shows a generic view of cache control.

HTTP cache has to be enabled at server side to make contents cacheable at user ends.

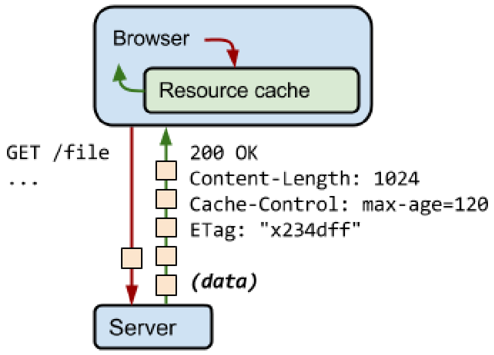


Figure 7: HTTP Response message form server

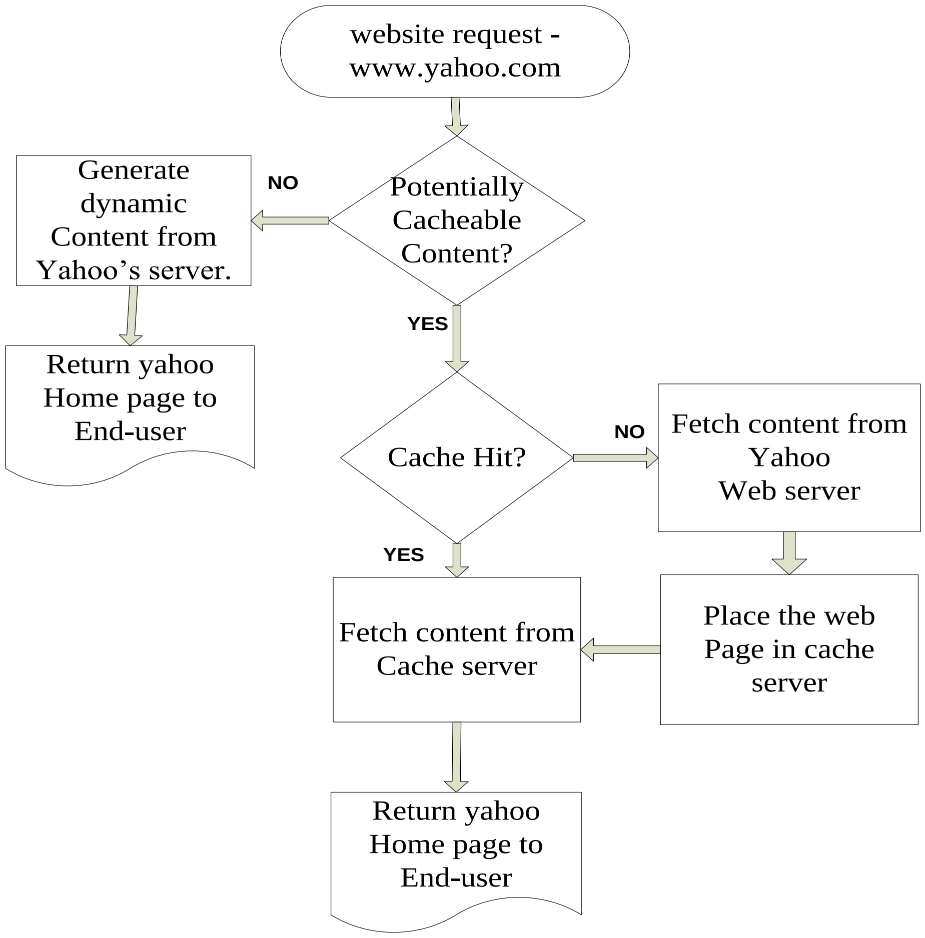

website request - www.yahoo.com
Potentially Cacheable Content?
NO
Generate dynamic Content from Yahoo's server.
Return yahoo Home page to End-user
YES
Cache Hit?
NO
Fetch content from Yahoo Web server
YES
Place the web Page in cache server
Fetch content from Cache server
Return yahoo Home page to End-user

Figure 8: Flow chart of web cache validation procedure

When the server returns a response it also emits a collection of HTTP headers, describing its content-type, length, caching directives, validation token, and more. For example, in the above exchange the server returns a 1024-byte response, instructs the client to cache it for up to 120 seconds, and provides a validation token ("x234dff") that can be used after the response has expired to check if the resource has been modified.

Web cache helps to reduce the web traffic by serving content locally shown in Figure 9, but the browser has to check the validation each cacheable contents separately to be ensured that he is serving the right contents.

Web cache can save a huge amount of web traffic, where a cache validation is required as shown fig 8.

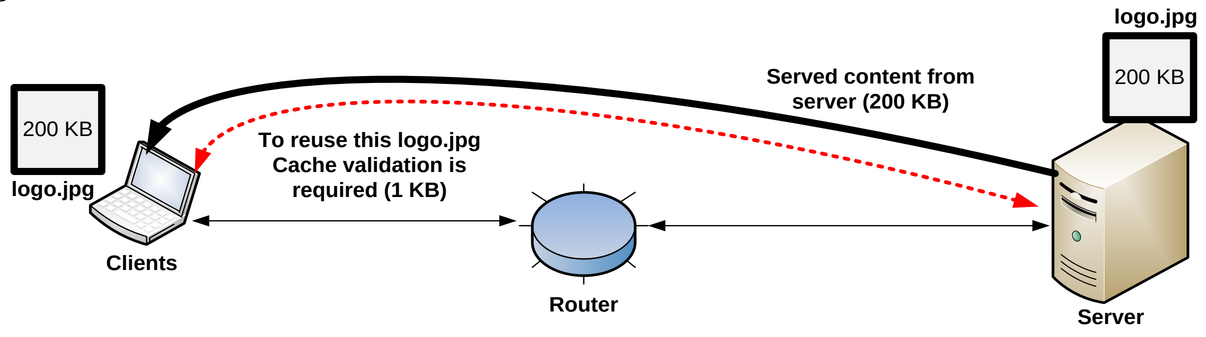

logo.jpg
200 KB
Served content from server (200 KB)
200 KB
logo.jpg
To reuse this logo.jpg Cache validation is required (1 KB)
Clients
Router
Server

Figure 9: Web content saved from cached contents after validation check

Web cache reduces the amount of downloadable content and it helps to browse website faster. On the other hand, CDN work with the distance between client and server.

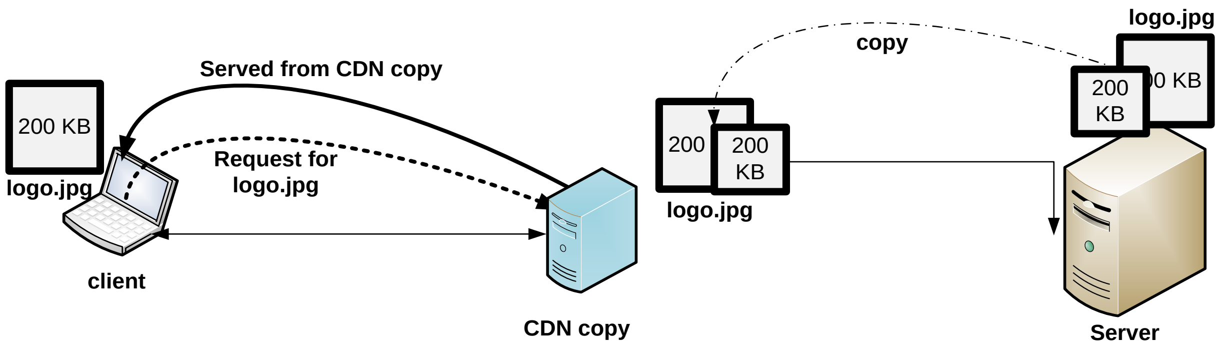


Figure 10: Content Delivery Network (CDN) helps to reduce the distance between client and content

The goal of "web optimizer" is to optimize the present technology by decreasing the cache validation request through "Cache Tree", "Cache Broadcasting" and let users browse key content based on their bandwidth limitation by using "LiteWeb" and "Cache Sharing" will reduce the web traffic by sharing and resolving cache request locally.

1.1 Scenario Analysis:

1.1.1. *Redundant* data transmission

Internet is networks of networks. All the computers of every small network are connected through internet. Every ends of a network are a client/server. Every request/response message has to travel from client to server and the distance might be thousands of miles.

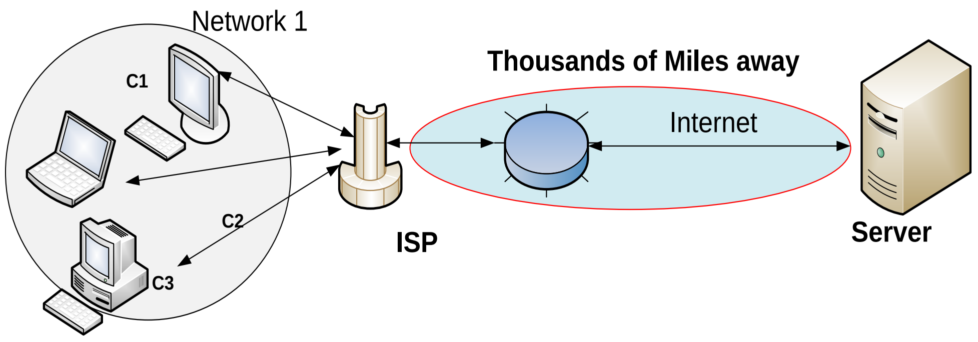


Figure 11: Local Network connected with internet through an Internet provider.

Suppose, there is network consist of 30 users on a network named as "Network 1". They all are contacted to same ISP/local server/internet provider. They are intra connected and can communicate without internet. Through an internet provider they are connected with internet.

Let, three users of Network 1 are browsing googl.com. They will send individual request to the server and server will send the response. So the intermediate path has to carry same content for three times. The aim of "Cache Sharing" is to avoid those redundant web traffics from internet.

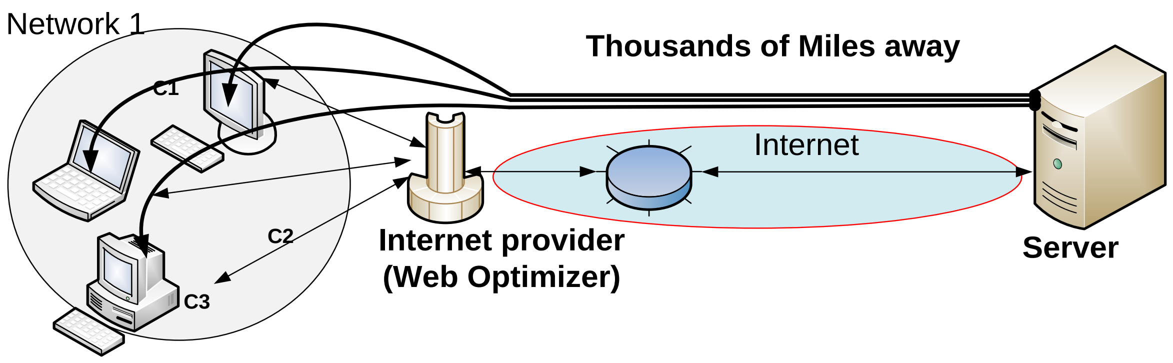


Figure 12: Redundant Web Traffic on internet

An example scenario has been discussed on chapter 3. To solve this problem "cache sharing" have been proposed. The goal of Cache broadcasting is to reduce the intermediate traffic of server and client with the help of SSWO and CSWO. Here long live public cacheable content (LLpCC) are suitable contents for broadcasting, because they do not change very frequently and could be cached for a long time. So the broadcasting requests have to be called very frequently.

1.1.2. Cache validation check

Suppose there is a website named abc.com and it contains few cacheable contents. Among them 10 contents are cacheable and others are not cacheable. Browser will store/cache all of those cacheable contents.

During the second request for the same page browser will send 10 "freshness" check request to be ensured that all contents are fresh and 2 regular requests to download rest of the contents. After the first request, every request will contain 10 "freshness" check requests and 2 regular requests. Most of the time those "freshness" check requests will return with negative result.

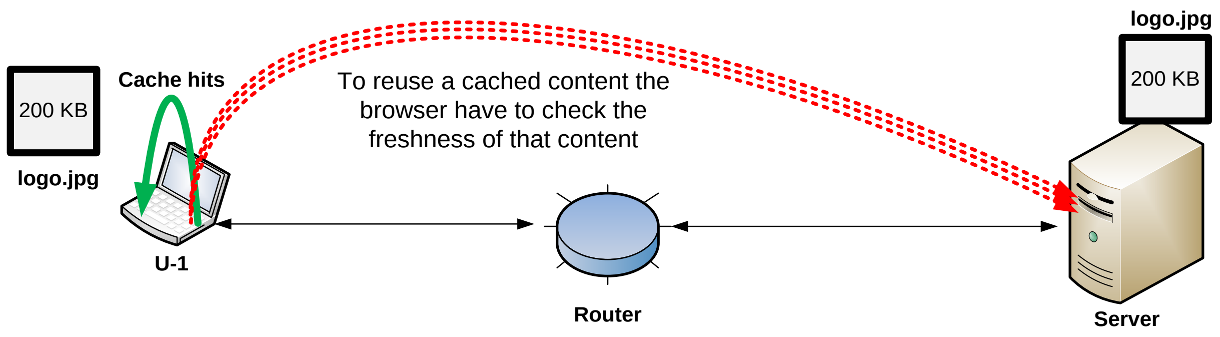

200 KB
logo.jpg
Cache hits
To reuse a cached content the browser have to check the freshness of that content
logo.jpg
200 KB
U-1
Router
Server

Figure 13: Multiple Cache validation check request to reuse content

To minimize this huge amount request 'Max-age' attribute of cached contents could be used. But there is no appropriate value of 'Max-age'. If the value is large then there are possibilities to loss some update and again if the value is small, then the number of unnecessary request will increase.

1.1.3. Content priority

Every web page contains important contents named as "key content". Those are the required information requested by client. But to represent and beatification some additional contents have been sent to the client's PC. If the bandwidth of client's PC is not sufficient to download all of those contents within a short time, then the users have to wait for those less important contents to be downloaded.

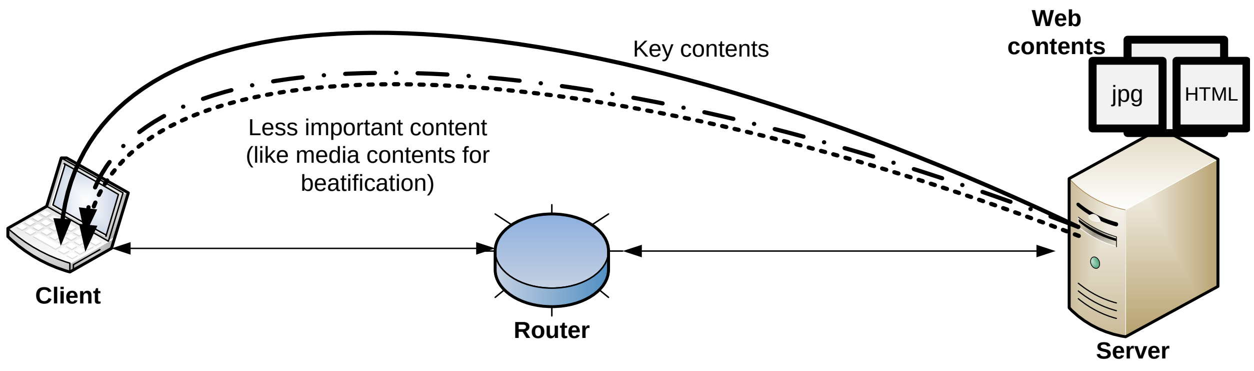

Web contents
Key contents
jpg
HTML
Less important content (like media contents for beatification)
Client
Router
Server

Figure 14: Clients have to download different types of contents from server to render a web page.

But user may want to browse only the key contents due to the bandwidth and data limitations.

1.2 Proposed solution:

World Wide Web is the heart and the internet is the backbone of virtual world. The Web is growing day by day. Simultaneously the web traffic is also increasing with respect to the internet users. There are four different modules of Web Optimizer and their goals are:

1. **Cache Sharing:** To make reusable cached content by peer to peer sharing them with other computers of same networks.
2. **Cache Tree:** To reduce the freshness check request by grouping cache content.
3. **Cache Broadcasting:** To Reduce network traffic by broadcasting public cacheable content from server to client.
4. **LiteWeb:** Responsive Web Content based on the bandwidth of client’s computer

1.2.1. Proposed architecture:

This Web Optimizer will work with traditional Client-server architecture. It has two parts:

1. Server Side Web Optimizer (SSWO)
2. Client Side Web Optimizer (CSWO)

Both will act as proxy server with few additional responsibilities.

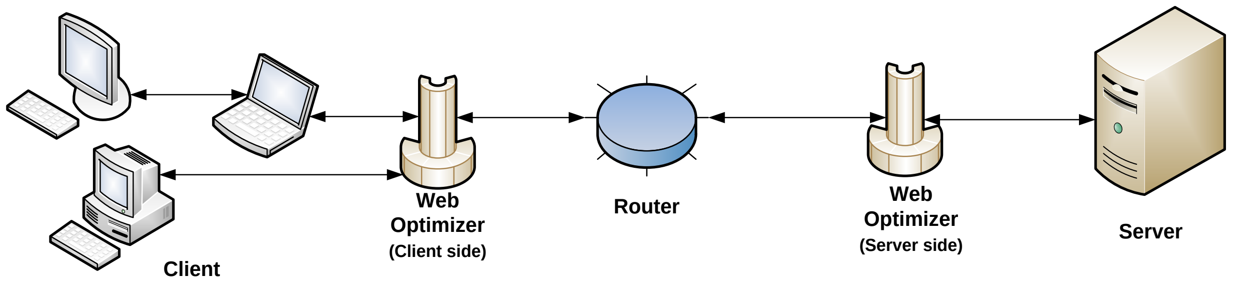


Figure 15: Client side and server side web optimizer

This Client side web optimizer will work as proxy server. CSWO will track the public cacheable contents at client side and SSWO will help to broadcast Long Live Public Cacheable Contents. The Details working procedure of "Web optimizer" have been discussed in chapter 3.

Web cache is one of the most used web optimization technique to reduce the web traffic. Storing be contents to use it during next request for that website is known as "Web cache". Content Delivery Network (CDN) is another web optimization technique that stores a copy of all files from server. The architecture of Web cache and CDN has been discussed in next chapter.

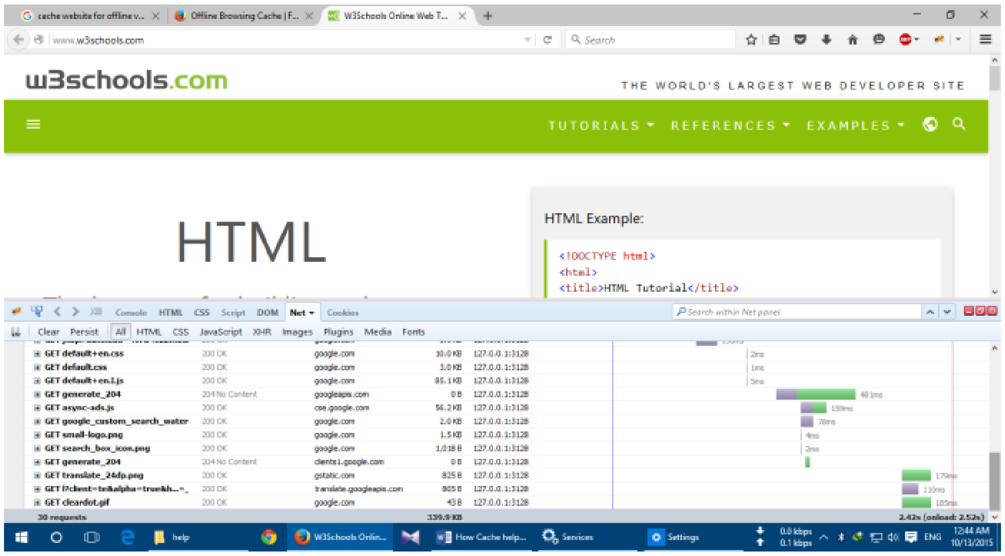
Chapter 2
Existing Technologies

2.1 Web cache

Web caches are being used to handle this huge amount of web traffic by storing frequently browsed web content on the browser of client's computer or local proxy server. Wide range of researchers has been done in this field and numbers of architecture have been developed for cache management. Goals of Cache are: [9]

- To minimize the response time
- To reduce the Web Traffic
- To reduce the server processing time

Here "Firebug" application is being used to demonstrate the advantages of web cache. At first browser have to download 340KB data and it takes 2.42Sec (Figure 19). During that time browser stores some data as cached content. From next time browser will use those cache contents and so that it has to download few contents from server. Less content requires less time and that's why the website will appear faster than before.

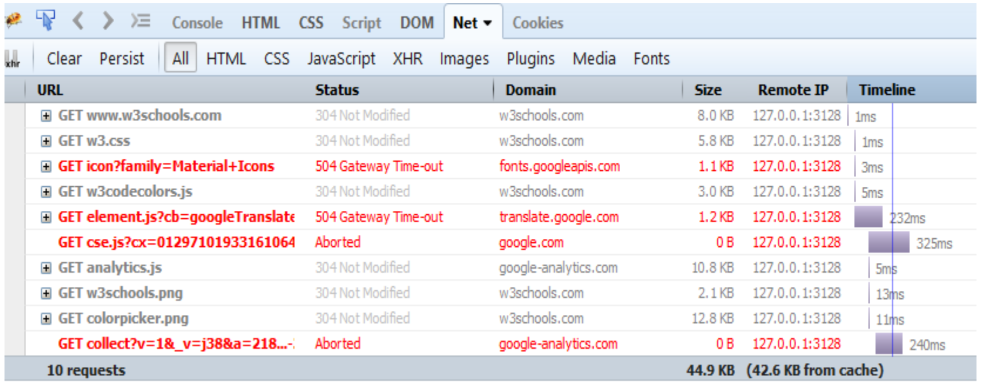


GET default+en.css	200 OK	google.com	10.0 KB	127.0.0.1:3128	2ms
GET default.css	200 OK	google.com	3.0 KB	127.0.0.1:3128	1ms
GET default+en.I.js	200 OK	google.com	85.1 KB	127.0.0.1:3128	5ms
GET generate_204	204 No Content	googleapis.com	0 B	127.0.0.1:3128	481ms
GET async-ads.js	200 OK	cse.google.com	56.2 KB	127.0.0.1:3128	159ms
GET google_custom_search_water	200 OK	google.com	2.0 KB	127.0.0.1:3128	78ms
GET small-logo.png	200 OK	google.com	1.5 KB	127.0.0.1:3128	4ms
GET search_box_icon.png	200 OK	google.com	1,018 B	127.0.0.1:3128	2ms
GET generate_204	204 No Content	clients1.google.com	0 B	127.0.0.1:3128	
GET translate_24dp.png	200 OK	gstatic.com	825 B	127.0.0.1:3128	179ms
GET l?client=te&alpha=true&h...=_	200 OK	translate.googleapis.com	805 B	127.0.0.1:3128	110ms
GET cleardot.gif	200 OK	google.com	43 B	127.0.0.1:3128	185ms
30 requests			339.9 KB		2.42s (onload: 2.52s)

w3schools.com
THE WORLD'S LARGEST WEB DEVELOPER SITE
TUTORIALS
REFERENCES
EXAMPLES
HTML
HTML Example:
Console HTML CSS Script DOM Net Cookies
Clear Persist All HTML CSS JavaScript XHR Images Plugins Media Fonts
Search within Net panel

```
<!DOCTYPE html>
<html>
<title>HTML Tutorial</title>
```

Figure 16: Browsing w3school without cached content



Console HTML CSS Script DOM Net Cookies
Clear Persist All HTML CSS JavaScript XHR Images Plugins Media Fonts

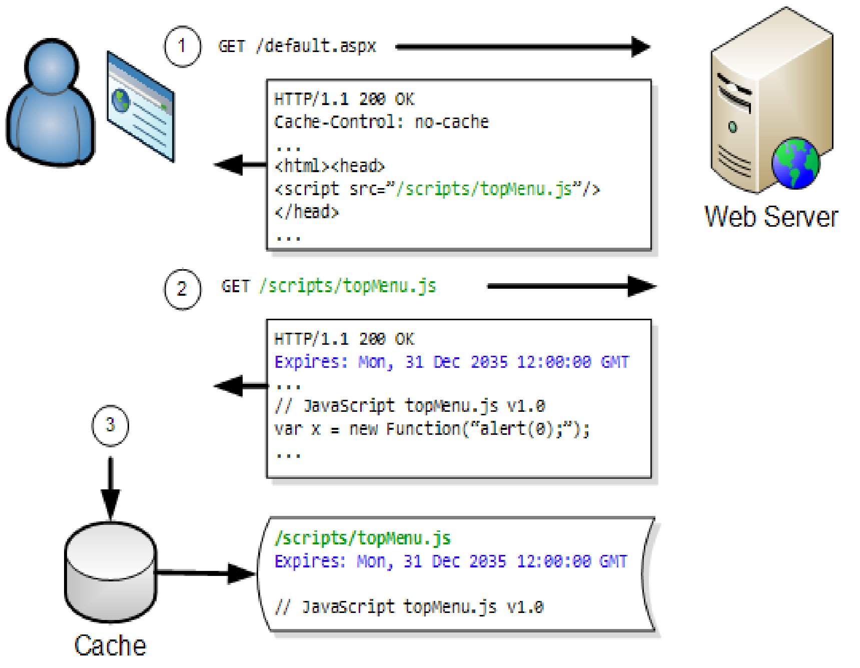
URL	Status	Domain	Size	Remote IP	Timeline
GET www.w3schools.com	304 Not Modified	w3schools.com	8.0 KB	127.0.0.1:3128	1ms
GET w3.css	304 Not Modified	w3schools.com	5.8 KB	127.0.0.1:3128	1ms
GET icon?family=Material+Icons	504 Gateway Time-out	fonts.googleapis.com	1.1 KB	127.0.0.1:3128	3ms
GET w3codecolors.js	304 Not Modified	w3schools.com	3.0 KB	127.0.0.1:3128	5ms
GET element.js?cb=googleTranslate	504 Gateway Time-out	translate.google.com	1.2 KB	127.0.0.1:3128	232ms
GET cse.js?cx=01297101933161064	Aborted	google.com	0 B	127.0.0.1:3128	325ms
GET analytics.js	304 Not Modified	google-analytics.com	10.8 KB	127.0.0.1:3128	5ms
GET w3schools.png	304 Not Modified	w3schools.com	2.1 KB	127.0.0.1:3128	13ms
GET colorpicker.png	304 Not Modified	w3schools.com	12.8 KB	127.0.0.1:3128	11ms
GET collect?v=1&_v=j38&a=218...-	Aborted	google-analytics.com	0 B	127.0.0.1:3128	240ms
10 requests			44.9 KB	(42.6 KB from cache)	

Figure 17: Browsing w3school with cached content

Web cache is a mechanism to store the web contents temporary (caching), such as HTML pages and images, supporting style sheet and other scripts this. Cache helps to reduce Network traffic, bandwidth usage, loading time, and lighten the load on a Web server by reducing the number of incoming requests; browsers retrieve portions of data from the cache rather than directly from the server. Web cache stores copies of documents passing through it; then, if there is another request for the same URL, it can use the response that it has stored previously, instead of asking the origin server for it again. There are three types of web cache.

a) Browser Caches

Browser's "Cache" is used for the temporary storage of Web pages on a hard disk. Every web page retrieves from the server for the first time, the browsers cheek the cache first time, if there are the same content then it server form the cache. Then the browser attempt to read the page from its "cache", instead of spending the extra time and network traffic to download it again.



```
GET /default.aspx
```

```
HTTP/1.1 200 OK
Cache-Control: no-cache
...
<html><head>
<script src="/scripts/topMenu.js"/>
</head>
...
```

```
GET /scripts/topMenu.js
```

```
HTTP/1.1 200 OK
Expires: Mon, 31 Dec 2035 12:00:00 GMT
...
// JavaScript topMenu.js v1.0
var x = new Function("alert(0);");
...
```

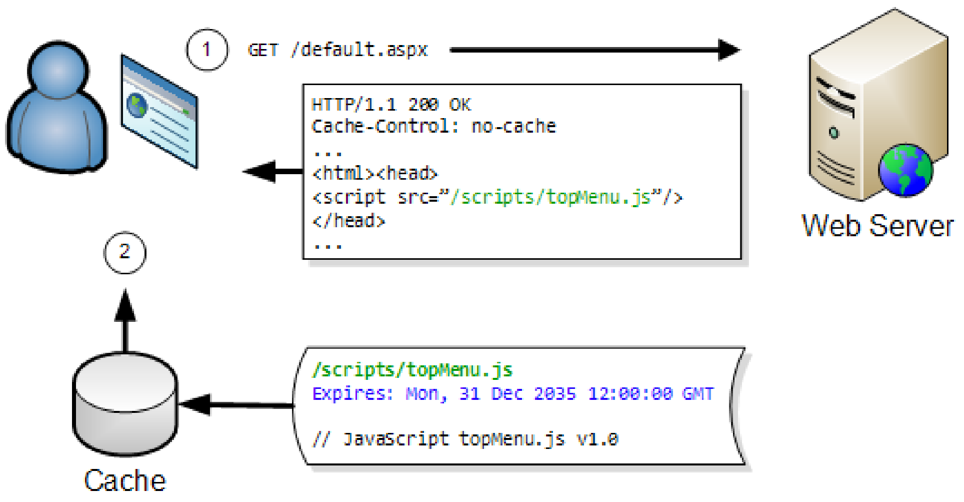
```
/scripts/topMenu.js
Expires: Mon, 31 Dec 2035 12:00:00 GMT

// JavaScript topMenu.js v1.0
```

Figure 18: Browser store cacheable contents

The Figure 18shows how a JavaScript file is saved into the browser cache on the first visit to a web page and

Figure 19shows how browser uses that cached contents:



```
HTTP/1.1 200 OK
Cache-Control: no-cache
...
<html><head>
<script src="/scripts/topMenu.js"/>
</head>
...
```

```
/scripts/topMenu.js
Expires: Mon, 31 Dec 2035 12:00:00 GMT

// JavaScript topMenu.js v1.0
```

Figure 19: Browser cache is being used.

This browser cache helps to avoid client-server-client data transmission by using local content which is typically 100 to 1000 times faster than downloading the file proxy server.

b) Proxy Caches server

A proxy cache server is a shared network device that can undertake Web transactions on behalf of a client, and, like the browser, the proxy cache server stores the web content. Subsequent requests for this content, by this or any other client of this network of the cache will trigger the cache to deliver the locally stored copy of the content, without downloading the original content from server.

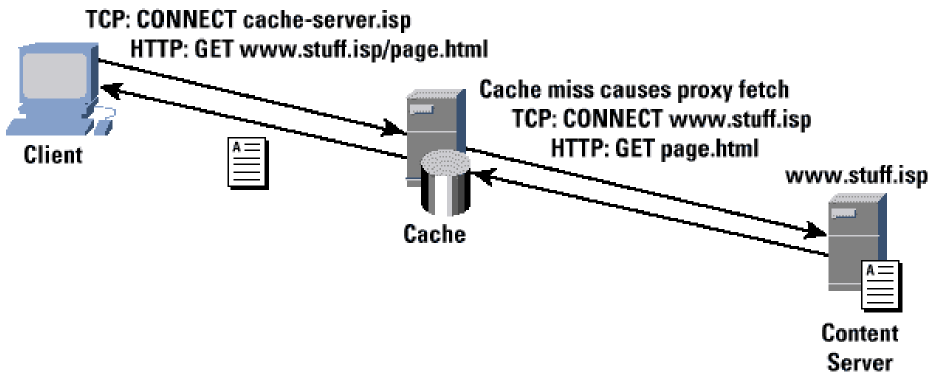

TCP: CONNECT cache-server.isp
HTTP: GET www.stuff.isp/page.html
Client
Cache miss causes proxy fetch
TCP: CONNECT www.stuff.isp
HTTP: GET page.html
Cache
www.stuff.isp
Content Server

Figure 20: Proxy Caches Server

Proxy caches aren't part of the client's content or the server content. Proxy caches are one kind of shared cache; rather than just having one person using them, they usually have been used by a large number of users by this they are reducing latency and network traffic. That's because popular representations are reused a number of times.

Proxy servers may also perform additional tasks, such as compressing data and optimizing documents. Systems that assist in such way are referred to as web accelerators. Sometimes caching proxy servers are most useful in distributed environments, where the proxy server can act as both a load balancer and optimizer.

c) Server Side Caches

Also known as “Reverse proxy caches” or “Gateway caches,” Server Side Caches are also intermediaries, but instead of being deployed by network administrators to save bandwidth, they’re typically deployed by Webmasters themselves, to make their sites more scalable, reliable and better performing.

2.1.1. Web Cache Architecture:

Multiple Web-caching servers can be used together to provide for more efficient caching. There are two basic caching architectures that use multiple caching servers working together:

- Distributed Caching
- Hierarchical Caching

As the name implies, distributed caching distributes, or spreads, the cached Web objects across two or more caching servers. These servers are all on the same level on the network. The figure below illustrates how distributed caching works.

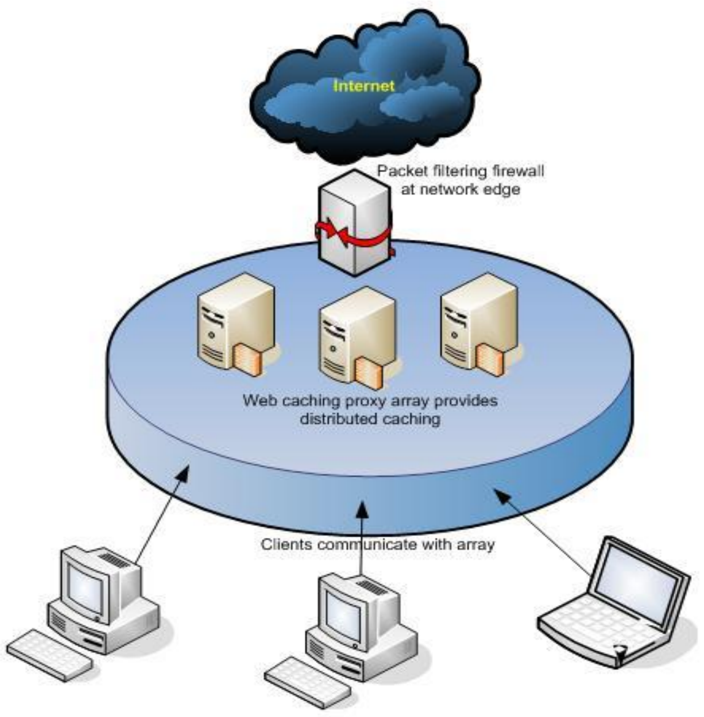

Internet
Packet filtering firewall at network edge
Web caching proxy array provides distributed caching
Clients communicate with array

Figure 21: Distributed architecture of web cache

Hierarchical caching works a little differently. In this setup, caching servers are placed at different levels on the network. Upstream caching servers communicate with downstream proxies. Hierarchical caching is more efficient in terms of bandwidth usage, but distributed caching is more efficient in terms of disk space usage. [7]

There are several methods of controlling a distributed proxy cache network in order to improve both the utilization of the cache information and the access concentration by using Internet Cache Protocol (ICP) information and round-trip time (RTT) estimation [11].

2.1.2. How “Web Cache” works

HTTP protocol helps to manage the web Cache. It includes some information in HTTP header that helps browser to detect and validate cacheable contents.

2.1.2.1. Validating cached responses by E-Tags

HTTP 1.1 introduced a new kind of validator called the ETag. E-Tags are unique identifiers that are generated by the server and changed every time the representation does. Almost all caches use Last-Modified times as validators. This is the time that the document last changed. When a cache has a representation stored that includes a Last-Modified header, it can use it to ask the server if the representation has changed since the last time it was browsed.

Most modern Web servers will generate both ETag and Last-Modified headers to use as validators for static content (i.e., files) automatically. ETag validation is also becoming prevalent. Validators are very important; if one isn’t present, and there isn’t any freshness information (Expires or Cache-Control) available, caches will not store a representation at all.

Validation token is communicated by the server via the ETag HTTP header Validation token enables efficient resource update checks: no data transfer if the resource has not changed.

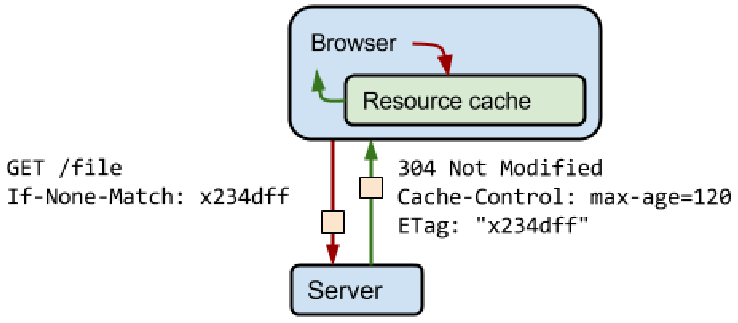


Figure 22: Cache control by E-Tag

In above example the client automatically provides the ETag token within the “If-None-Match” HTTP request header, the server checks the token against the current resource, and if it has not changed returns a “304 Not Modified” response which tells the browser that the response it has in cache has not changed and can be renewed for another 120 seconds.

2.1.2.2. Cache-Control by Max-age

Each resource can define its caching policy via Cache-Control HTTP header Cache-Control directives control who can cache the response, under which conditions, and for how long.

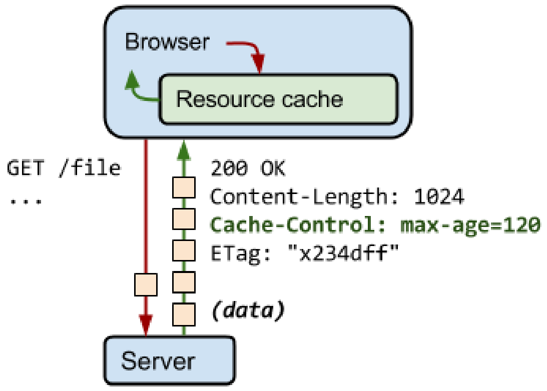


Figure 23: Cache control by Max-age

Cache-Control header was defined as part of the HTTP/1.1 specification and supersedes previous headers (e.g. Expires) used to define response caching policies. All modern browsers support Cache-Control.

a) "no-cache" and "no-store"

"no-cache" indicates that the returned response cannot be used to satisfy a subsequent request to the same URL without first checking with the server if the response has changed. As a result, if a proper validation token (ETag) is present; no-cache will incur a roundtrip to validate the cached response, but can eliminate the download if the resource has not changed.

b) "public" vs. "private"

If the response is marked as "public" then it can be cached, even if it has HTTP authentication associated with it, and even when the response status code isn't normally cacheable.

By contrast, "private" responses can be cached by the browser but are typically intended for a single user and hence are not allowed to be cached by any intermediate cache - e.g. an HTML page with private user information can be cached by that user's browser, but not by a CDN.

c) "max-age"

This directive specifies the maximum time in seconds that the fetched response is allowed to be reused for from the time of the request - e.g. "max-age=60" indicates that the response can be cached and reused for the next 60 seconds.

Table 1 : Details of Cache control attributes

Cache-Control directives	Explanation
max-age=86400	Response can be cached by browser and any intermediary caches (i.e. it is "public") for up to 1 day (60 seconds x 60 minutes x 24 hours)

private, max-age=600	Response can be cached by the client's browser only for up to 10 minutes (60 seconds x 10 minutes)
no-store	Response is not allowed to be cached and must be fetched in full on every request.

2.1.2.3. Defining optimal Cache-Control policy

Google has designed an optimal cache control policy.

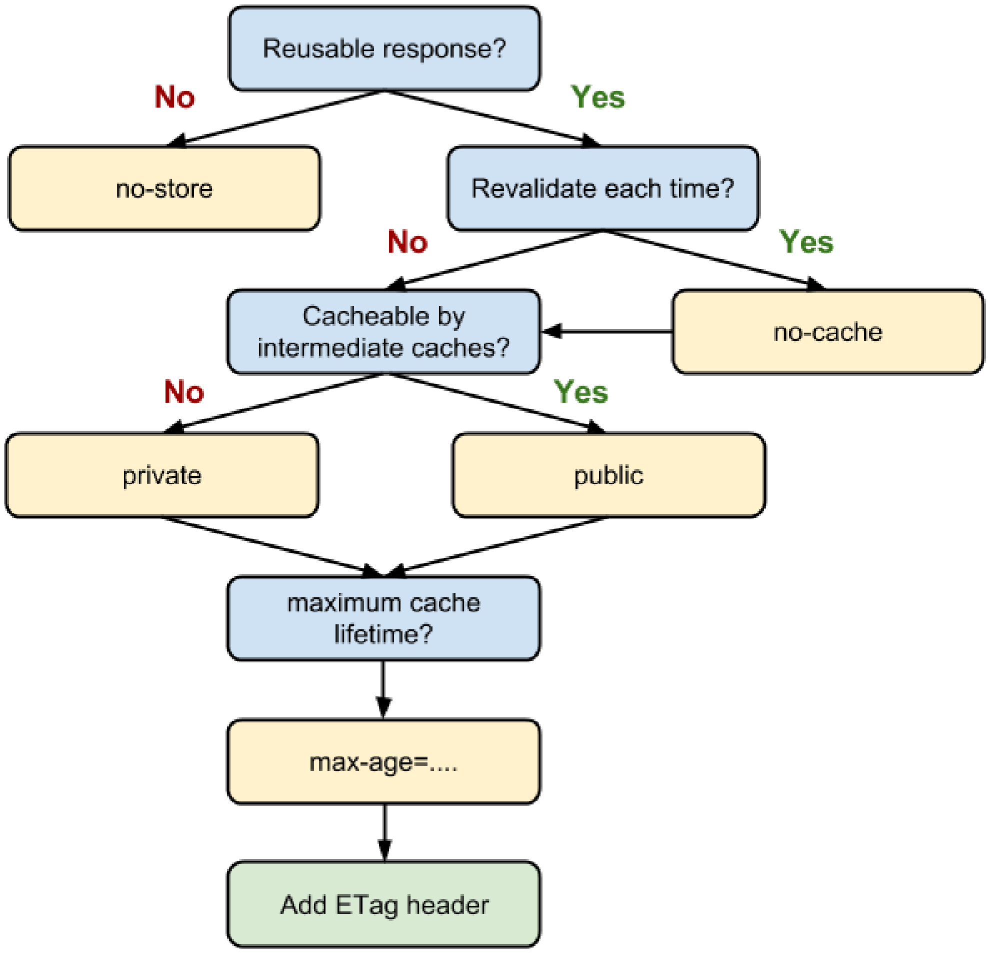


Figure 24: Optimal Cache control policy by Google

Follow the decision tree above to determine the optimal caching policy for a particular resource, or a set of resources used by an application. Ideally, the developer should aim to cache as many responses as possible on the client for the longest possible period, and provide validation tokens for each response to enable efficient revalidation.

2.1.2.4. Invalidating and updating cached responses

Client-side caching and quick updates can achieved be changing the URL of the resource and force the user to download the new response whenever its content changes. Typically, this is done by embedding a fingerprint of the file, or a version number, in its filename - e.g. style.x234dff.css.

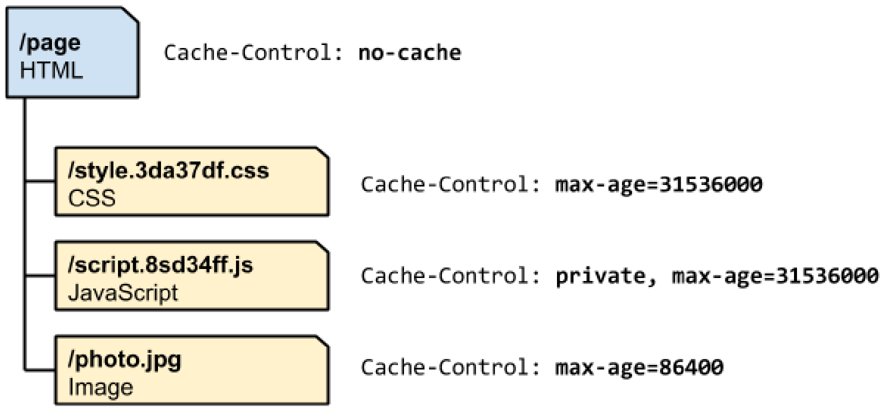

/page
HTML
Cache-Control: no-cache
/style.3da37df.css
CSS
Cache-Control: max-age=31536000
/script.8sd34ff.js
JavaScript
Cache-Control: private, max-age=31536000
/photo.jpg
Image
Cache-Control: max-age=86400

Figure 25: Cache control precedence

The ability to define per-resource caching policies allows us to define "cache hierarchies" that allow us to control not only how long each is cached for, but also how quickly new versions are seen by visitor. For example, let's analyze the above example:

- The HTML is marked with "no-cache", which means that the browser will always revalidate the document on each request and fetch the latest version if the contents change. Also, within the HTML markup the developer embeds fingerprints in the URLs for CSS and JavaScript assets: if the contents of those files change, then the HTML of the page will change as well and new copy of the HTML response will be downloaded.
- The CSS is allowed to be cached by browsers and intermediate caches (e.g. a CDN), and is set to expire in 1 year. Note that "far future expires" of 1 year could be used safely because embed fingerprint its filename: if the CSS is updated, the URL will change as well.
- The JavaScript is also set to expire in 1 year, but is marked as private, perhaps because it contains some private user data that the CDN shouldn't cache.
- The image is cached without a version or unique fingerprint and is set to expire in 1 day.

The combination of ETag, Cache-Control, and unique URLs allows us to deliver the best of all worlds: long-lived expiry times, control over where the response can be cached, and on-demand updates.

2.1.2.5. Expiration Model

HTTP caching works best when caches can entirely avoid making requests to the origin server. The primary mechanism for avoiding requests is for an origin server to provide an explicit expiration time in the future, indicating that a response MAY be used to satisfy subsequent requests. In other words, a cache can return a fresh response without first contacting the server.

The expiration mechanism applies only to responses taken from a cache and not to first-hand responses forwarded immediately to the requesting client. If an origin server wishes to force a semantically transparent cache to validate every request, it MAY assign an explicit expiration time in the past. This means that the response is always stale, and so the cache SHOULD validate it before using it for subsequent requests.

If an origin server wishes to force any HTTP/1.1 cache, no matter how it is configured, to validate every request, it SHOULD use the "must- revalidate" cache-control directive. Servers specify explicit expiration times using either the Expires header, or the max-age directive of the Cache-Control header.

An expiration time cannot be used to force a user agent to refresh its display or reload a resource; its semantics apply only to caching mechanisms, and such mechanisms need only check a resource's expiration status when a new request for that resource is initiated. See section 3.3 for an explanation of the difference between caches and history mechanisms.

a) Age Calculations

In order to know if a cached entry is fresh, a cache needs to know if its age exceeds its freshness lifetime. A response's age can be calculated in two entirely independent ways:

1. Now minus date_value, if the local clock is reasonably wellsynchronized to the origin server's clock. If the result is negative, the result is replaced by zero.
2. age_value, if all of the caches along the response path implement HTTP/1.1.

There are two independent ways to compute the age of a response when it is received, these can be combined as

corrected_received_age = max (now - date_value, age_value)

Because of network-imposed delays, some significant interval might pass between the time that a server generates a response and the time it is received at the next outbound cache or client. If uncorrected, this delay could result in improperly low ages.

Because the request that resulted in the returned Age value must have been initiated prior to that Age value's generation, the developer can correct for delays imposed by the network by recording the time at which the request was initiated. Then, when an Age value is received, it MUST be interpreted relative to the time the request was initiated, not the time that the response was received. So, corrected_initial_age = corrected_received_age+ (now - request_time)

Where "request_time" is the time (according to the local clock) when the request that elicited this response was sent.

b) Expiration Calculations

The age is calculated as described in section 3.2.3; this section describes how to calculate the freshness lifetime, and to determine if a response has expired. In the discussion below, the values can be represented in any form appropriate for arithmetic operations.

The term "expires_value" is to denote the value of the Expires header. The term "max_age_value" is to denote an appropriate value of the number of seconds carried by the "max-age" directive of the Cache-Control header in a response.

The max-age directive takes priority over Expires, so if max-age is present in a response, the calculation is simply:

freshness_lifetime = max_age_value

Otherwise, if Expires is present in the response, the calculation is:

freshness_lifetime = expires_value - date_value

Note that neither of these calculations is vulnerable to clock skew, since all of the information comes from the origin server.

If none of Expires, Cache-Control: max-age, or Cache-Control: s- maxage appears in the response, and the response does not include other restrictions on caching, the cache MAY compute a freshness lifetime using a heuristic. The cache MUST attach Warning 113 to any response whose age is more than 24 hours if such warning has not already been added.

Also, if the response does have a Last-Modified time, the heuristic expiration value SHOULD be no more than some fraction of the interval since that time. A typical setting of this fraction might be 10%.

The calculation to determine if a response has expired is quite simple:

response_is_fresh = (freshness_lifetime>current_age)

2.1.2.6. Caching checklist

There is no one best cache policy. There are few basic techniques on caching strategy:

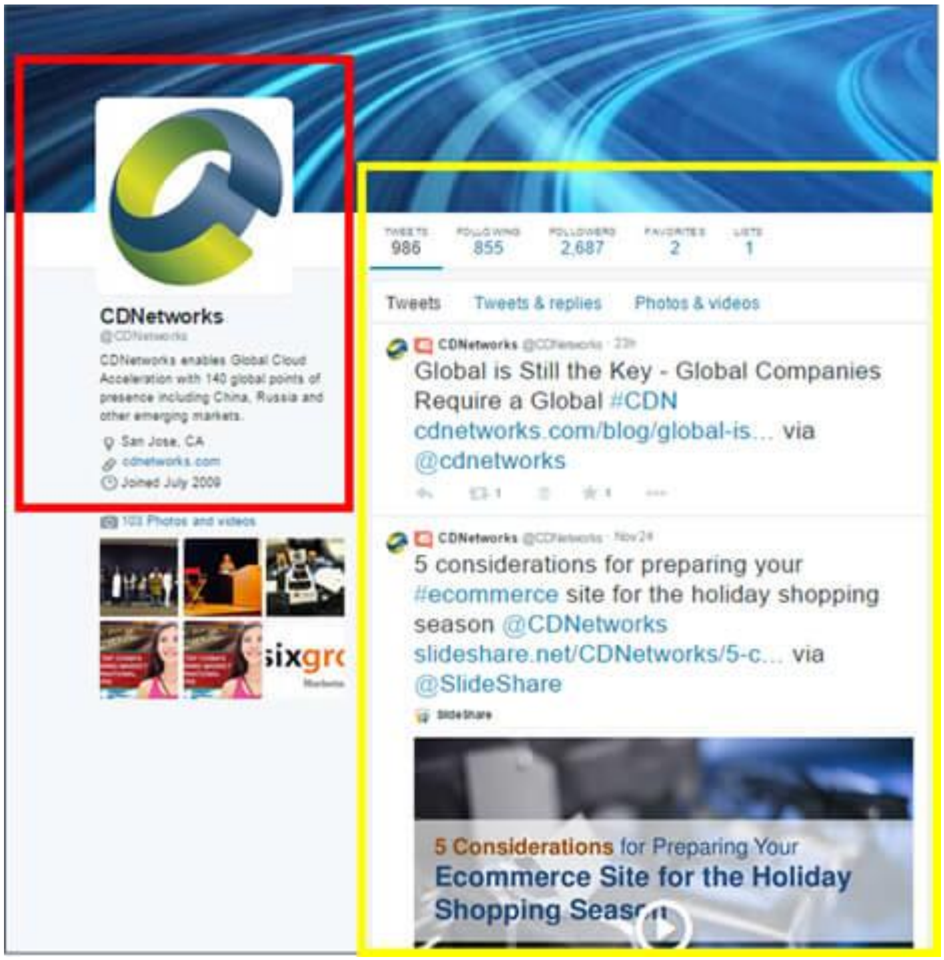
1. **Use consistent URLs:** if anyone serve the same content on different URLs, then that content will be fetched and stored multiple times.
2. **Ensure the server provides a validation token (ETag):** validation tokens eliminate the need to transfer the same bytes when a resource has not changed on the server.
3. **Identify which resources can be cached by intermediaries:** those with responses that are identical for all users are great candidates to be cached by a CDN and other intermediaries.
4. **Determine the optimal cache lifetime for each resource:** different resources may have different freshness requirements. Audit and determine the appropriate max-age for each one.
5. **Determine the best cache hierarchy for a site:** the combination of resource URLs with content fingerprints, and short or no-cache lifetimes for HTML documents allows anyone to control how quickly updates are picked up by the client.

Depending on a traffic patterns, type of data served, and application-specific requirements for data freshness, the developer will have to define and configure the appropriate per-resource settings, as well as the overall “caching hierarchy”.

2.2 Content Delivery Networks (CDN)

The interaction between a user and online content is far more complex today than it was a few years ago. Today's users are much more likely to be streaming a longer video from a mobile phone or accessing a SaaS portal when working from home. These are far more complex experiences that did not exist five or so years ago. Given the expected growth of the CDN market in the coming years, this post will define exactly what a CDN is and how it operates.

A content delivery network, also called a CDN, improves the performance, security and reliability of a website. Since the start of the internet, websites have evolved in how they deliver content to the end user. In the example below, a Twitter stream. Highlighted in red, is what is known as static content. The logo and profile descriptions are basic graphics and HTML text that rarely change. Highlighted in yellow, is what is known as dynamic content. The Twitter stream represents content that is always changing and moving.


CDNetworks
@CDNetworks
CDNetworks enables Global Cloud Acceleration with 140 global points of presence including China, Russia and other emerging markets.
San Jose, CA
cdnetworks.com
Joined July 2009
103 Photos and videos

TWEETS 986 FOLLOWING 855 FOLLOWERS 2,687 FAVORITES 2 LISTS 1

Tweets Tweets & replies Photos & videos

CDNetworks @CDNetworks · 23h
Global is Still the Key - Global Companies Require a Global #CDN cdnetworks.com/blog/global-is... via @cdnetworks

CDNetworks @CDNetworks · Nov 24
5 considerations for preparing your #ecommerce site for the holiday shopping season @CDNetworks slideshare.net/CDNetworks/5-c... via @SlideShare
SlideShare
5 Considerations for Preparing Your Ecommerce Site for the Holiday Shopping Season

Figure 26: Cacheable contents on web pages

It is the growing popularity of dynamic online experiences, like Twitter, that are driving the growth of CDNs.

2.2.1. How CDNs Work:

A CDN has Points of Presence (PoPs) or data centers that are situated around the world. Within each PoP are thousands of servers. Both the PoPs and servers help accelerate the speed at which content is delivered to the end user.

Below, is a representation of how a web page works without a CDN. An end user requests a page in which all the page's components (images, HTML, dynamic content) are retrieved from the website's server or origin, which could be anywhere in the world. In the example below, the user is in North America, while the origin that holds the content requested is all the way across the globe.

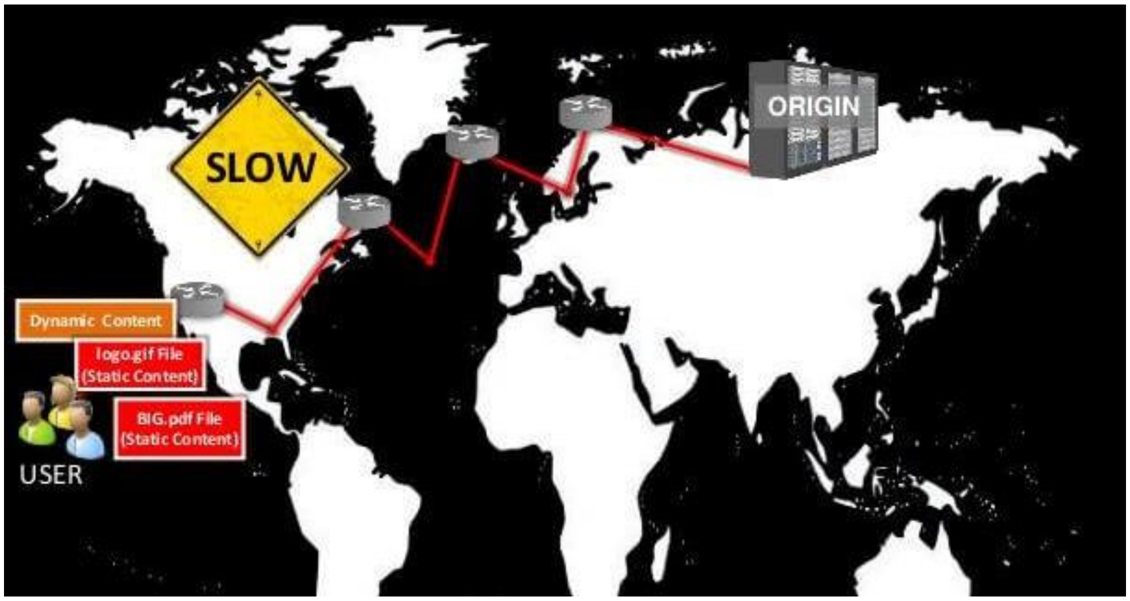


Figure 27: Serving content without CDN

Below are images that represent how a web page loads content using a CDN. A CDN caches all files locally (images, HTML, etc.) in the local PoP they have established. As a result, when an end user requests a web page, it loads faster. If the CDN does not have an image or file that is requested by the user, it will load from the origin as needed.

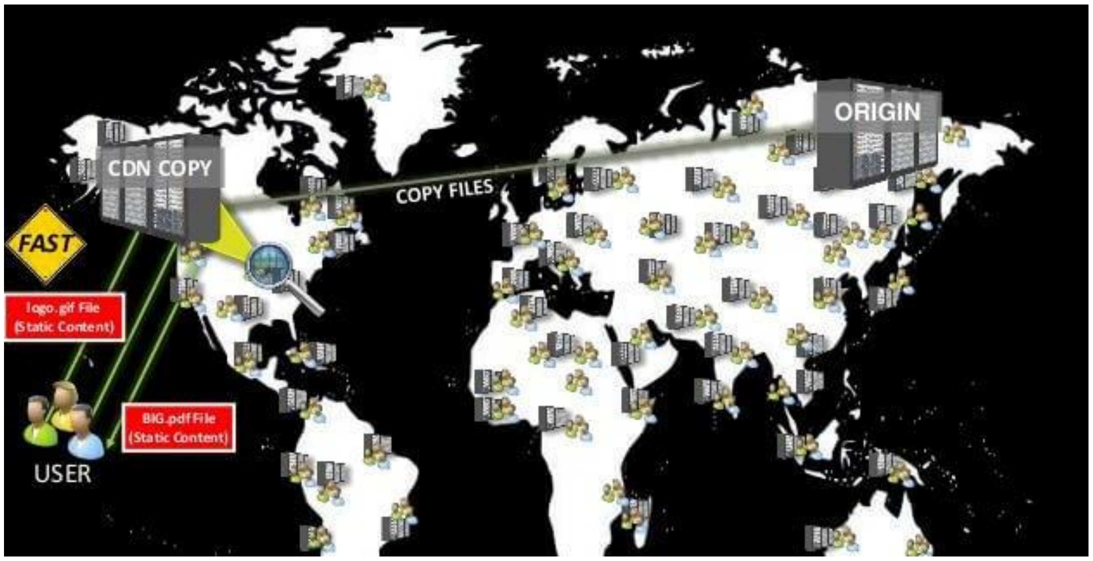


Figure 28: Serving static content through CDN

A CDNs approach changes for sites that have dynamic content. CDNs that support dynamic content create a “super highway” to accelerate the delivery of content across a long distance. An individual ISP cannot provide this. In the image below, the “super highway” helping to facilitate the delivery of dynamic content from the origin to the end user.

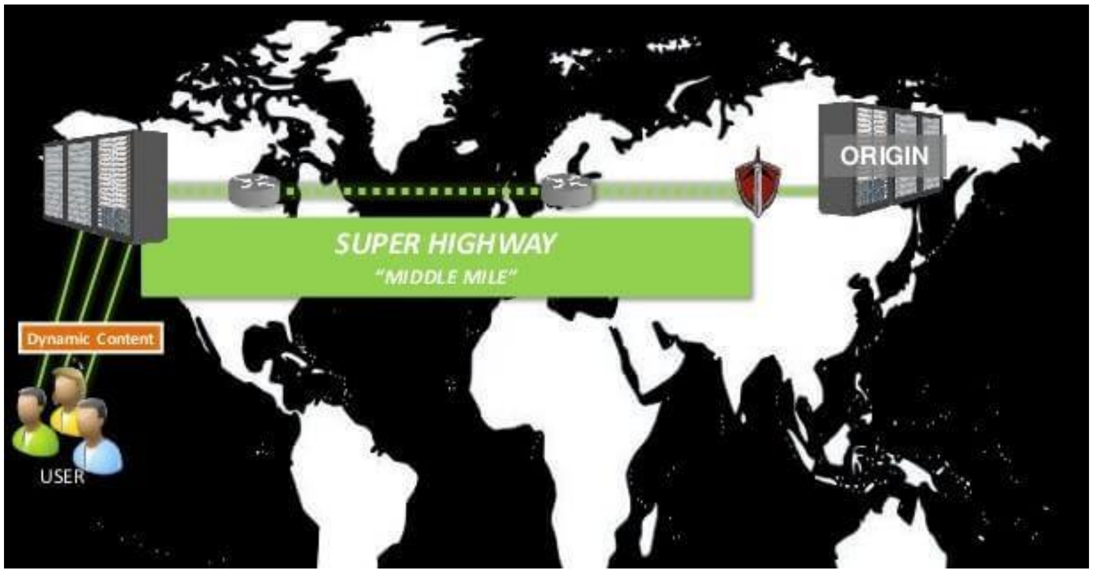


Figure 29: Serving dynamic content through CDN

2.2.2. Security:

Lastly, a CDN should protect itself and all the data it stores as well as a website’s origin from any data breaches or internet hackers. The most common data threats are various forms of distributed denial of service (DDoS) attacks. A CDN has proper techniques for preventing, detecting and correcting different forms of DDoS attacks. Prevention measures such as HTTP load balancing, have an always-on approach, making sites less vulnerable to attacks. Detection systems within a CDN look for suspicious behavior and highlight the need for additional investigation and action. An example of detection might be a traffic surge in which case the CDN provider has systems in place to automatically notify those responsible for the website. Corrective measures such as secured DNS will block out unwanted requests while continuing to allow trustworthy requests to continue being served. Advance CDN with p2p content sharing is another new technology discussed on [14].

2.3 Recommended Practices and Rules to optimize web pages

The performance of World Wide Web depends on web contents, web technologies and networks. The technology behind the WWW is improving day by day. Many researchers are working on it. Yahoo's Exceptional Performance team has identified 34 rules that affect web page performance. YSlow's web page analysis is based on the 23 of these 34 rules that are testable.

1. Minimize HTTP Requests
2. Use a Content Delivery Network
3. Avoid empty src or href (Avoid Empty Image src)
4. Add an Expires or a Cache-Control Header
5. Gzip Components
6. Put Style Sheets at the Top
7. Put Scripts at the Bottom
8. Avoid CSS Expressions
9. Make JavaScript and CSS External
10. Reduce DNS Lookups
11. Minify JavaScript and CSS
12. Avoid Redirects
13. Remove Duplicate Scripts
14. Configure ETags
15. Make AJAX Cacheable
16. Use GET for AJAX Requests
17. Reduce the Number of DOM Elements
18. No 404s
19. Reduce Cookie Size
20. Use Cookie-Free Domains for Components
21. Avoid Filters
22. Do Not Scale Images in HTML
23. Make favicon.ico Small and Cacheable

Few important rules are briefly discussed here.

2.3.1. Minimize HTTP Requests

80% of the end-user response time is spent on the front-end. Most of this time is tied up in downloading all the components in the page: images, stylesheets, scripts, flash, etc. Reducing the number of components in turn reduces the number of HTTP requests required to render the page. This is the key to faster pages.

One way to reduce the number of components in the page is to simplify the page's design. Here are some techniques for reducing the number of HTTP requests, while still supporting rich page designs.

Combined files are a way to reduce the number of HTTP requests by combining all scripts into a single script, and similarly combining all CSS into a single stylesheet. Combining files is more challenging when the scripts and stylesheets vary from page to page, but making this part of release process improves response times.

Reducing the number of HTTP requests in a page is the place to start. This is the most important guideline for improving performance for first time visitors 40-60% of daily visitors to a site come in with an empty cache. Making a page fast for these first time visitors is the key to a better user experience.

2.3.2. Use a Content Delivery Network

As a first step to implementing geographically dispersed content, don't attempt to redesign the web application to work in a distributed architecture. Depending on the application, changing the architecture could include daunting tasks such as synchronizing session state and replicating database transactions across server locations. Attempts to reduce the distance between users and the content could be delayed by, or never pass, this application architecture step.

Remember that 80-90% of the end-user response time is spent downloading all the components in the page: images, styleSheets, scripts, Flash, etc. This is the *Performance Golden Rule*. Rather than starting with the difficult task of redesigning the architecture of application, it's better to first disperse its static content. This not only achieves a bigger reduction in response times, but it's easier thanks to content delivery networks. The working procedure has been discussed on section 2.2.

2.3.3. Add an Expires or a Cache-Control Header

There are two aspects to this rule:

- For static components: implement "Never expire" policy by setting far future Expires header
- For dynamic components: use an appropriate Cache-Control header to help the browser with conditional requests

Web page designs are getting richer and richer, which means more scripts, stylesheets, images, and Flash in the page. A first-time visitor of a page may have to make several HTTP requests, but by using the Expires header those components could be make as cacheable. This avoids unnecessary HTTP requests on subsequent page views. Expires headers are most often used with images, but they should be used on *all* components including scripts, stylesheets, and Flash components.

```
Expires: Thu, 15 Apr 2010 20:00:00 GMT
```

This example of the ExpiresDefault directive sets the Expires date 10 years out from the time of the request.

2.3.4. Gzip Components

The time it takes to transfer an HTTP request and response across the network can be significantly reduced by decisions made by front-end engineers. It's true that the end-user's bandwidth speed, Internet service provider, proximity to peering exchange points, etc. are beyond the control of the development team. But there are other variables that affect response times. Compression reduces response times by reducing the size of the HTTP response.

Starting with HTTP/1.1, web clients indicate support for compression with the Accept-Encoding header in the HTTP request.

```
Accept-Encoding: gzip, deflate
```

2.3.5. Put Style sheets at the Top

Front-end engineers that care about performance want a page to load progressively; that is, the client wants the browser to display whatever content it has as soon as possible. This is especially important for pages with a lot of content and for users on slower Internet connections. The problem with putting stylesheets near the bottom of the document is that it prohibits progressive rendering in many browsers, including Internet Explorer. These browsers block rendering to avoid having to redraw elements of the page if their styles change. The user is stuck viewing a blank white page.

2.3.6. Put Scripts at the Bottom

The problem caused by scripts is that they block parallel downloads. The HTTP/1.1 specification suggests that browsers download no more than two components in parallel per hostname. If the images served from multiple hostnames, the web browser will get more than two downloads in parallel. While a script is downloading, however, the browser won't start any other downloads, even on different hostnames.

2.3.7. Make JavaScript and CSS External

Using external files in the real world generally produces faster pages because the JavaScript and CSS files are cached by the browser. JavaScript and CSS that are inlined in HTML documents get downloaded every time the HTML document is requested. This reduces the number of HTTP requests that are needed, but increases the size of the HTML document. On the other hand, if the JavaScript and CSS are in external files cached by the browser, the size of the HTML document is reduced without increasing the number of HTTP requests.

2.3.8. Reduce DNS Lookups

The Domain Name System (DNS) maps hostnames to IP addresses, just as phonebooks map people's names to their phone numbers. When anyone type www.yahoo.com into browser, a DNS resolver contacted by the browser returns that server's IP address. DNS has a cost. It typically takes 20-120 milliseconds for DNS to lookup the IP address for a given hostname. The browser can't download anything from this hostname until the DNS lookup is completed.

DNS lookups are cached for better performance. **Most browsers have their own caches**, separate from the operating system's cache. As long as the browser keeps a DNS record in its own cache, it doesn't bother the operating system with a request for the record.

2.3.9. Minify JavaScript and CSS

Minification is the practice of removing unnecessary characters from code to reduce its size thereby improving load times. When code is minified all comments are removed, as well as unneeded white space characters (space, newline, and tab). In the case of JavaScript, this improves response time performance because the size of the downloaded file is reduced. Two popular tools for minifying JavaScript code are JSMin and YUI Compressor. The YUI compressor can also minify CSS.

2.3.10. Avoid Redirects

Redirects are accomplished using the 301 and 302 status codes. Here's an example of the HTTP headers in a 301 response:

```
HTTP/1.1 301 Moved Permanently
Location: http://example.com/newuri
Content-Type: text/html
```

The browser automatically takes the user to the URL specified in the Location field.

If a domain name change is the cause of using redirects, an alternative is to create a CNAME (a DNS record that creates an alias pointing from one domain name to another) in combination with Alias or mod_rewrite.

2.3.11. Remove Duplicate Scripts

Unnecessary HTTP requests happen in Internet Explorer, but not in Firefox. In Internet Explorer, if an external script is included twice and is not cacheable, it generates two HTTP requests during page loading. Even if the script is cacheable, extra HTTP requests occur when the user reloads the page.

2.3.12. Configure ETags

Entity tags (ETags) are a mechanism that web servers and browsers use to determine whether the component in the browser's cache matches the one on the origin server. (An "entity" is another word a "component": images, scripts, stylesheets, etc.) ETags were added to provide a mechanism for validating entities that is more flexible than the last-modified date. An ETag is a string that uniquely identifies a specific version of a component. The only format constraints are that the string be quoted. The origin server specifies the component's ETag using the ETag response header. The working procedures of ETag have been discussed on 2.1.2 and 2.1.3.

2.3.13. Post-load Components

Contents absolutely required in order to render the page initially could be stored separately. The rest of the content and components can wait.

JavaScript is an ideal candidate for splitting before and after the onload event. For example, if anyone has JavaScript code and libraries that do drag and drop and animations, those can wait, because dragging elements on the page comes after the initial rendering. Other places to look for candidates for post-loading include hidden content (content that appears after a user action) and images below the fold.

2.3.14. Preload Components

Preload may look like the opposite of post-load, but it actually has a different goal. By preloading components, anyone can take advantage of the time the browser is idle and request components (like images, styles and scripts) will need in the future. This way when the user visits the next page, most of the components already in the cache and the page will load much faster for the user.

2.3.15. Reduce the Number of DOM Elements

A complex page means more bytes to download and it also means slower DOM access in JavaScript. It makes a difference if anyone loop through 500 or 5000 DOM elements on the page when developer wants to add an event handler for example.

A high number of DOM elements can be a symptom that there's something that should be improved with the markup of the page without necessarily removing content.

2.3.16. Split Components Across Domains

Splitting components allow to maximize the parallel downloads. Do not use more than 2-4 domains because of the DNS lookup penalty. For example, anyone can host HTML and dynamic content on www.example.org and split static components between static1.example.org andstatic2.example.org.

2.3.17. Minimize the Number of iframes

Iframes allow an HTML document to be inserted in the parent document. It's important to understand how iframes work so they can be used effectively.

2.3.18. No 404s

HTTP requests are expensive so making an HTTP request and getting a useless response (i.e. 404 Not Found) is totally unnecessary and will slow down the user experience without any benefit.

2.3.19. Reduce Cookie Size

HTTP cookies are used for a variety of reasons such as authentication and personalization. Information about cookies is exchanged in the HTTP headers between web servers and browsers. It's important to keep the size of cookies as low as possible to minimize the impact on the user's response time.

- Eliminate unnecessary cookies
- Keep cookie sizes as low as possible to minimize the impact on the user response time
- Be mindful of setting cookies at the appropriate domain level so other sub-domains are not affected
- Set an Expires date appropriately. An earlier Expires date or none removes the cookie sooner, improving the user response time

2.3.20. Avoid Filters

The IE-proprietary AlphaImageLoader filter aims to fix a problem with semi-transparent true color PNGs in IE versions < 7. The problem with this filter is that it blocks rendering and freezes the browser while the image is being downloaded. It also increases memory consumption and is applied per element, not per image, so the problem is multiplied.

The best approach is to avoid AlphaImageLoader completely and use gracefully degrading PNG8 instead, which are fine in IE. If anyone absolutely need AlphaImageLoader, use the underscore hack _filter as to not penalize IE7+ users.

2.3.21. Make favicon.ico Small and Cacheable

The favicon.ico is an image that stays in the root of a server. It's a necessary evil because even if anyone don't care about it the browser will still request it, so it's better not to respond with a 404 Not Found.

It's small, preferably under 1K.

2.3.22. Avoid Empty Image src

Image with empty string **src** attribute occurs more than one will expect. It appears in two forms:

1. straight HTML <imgsrc="">
2. JavaScript varimg = new Image();
   img.src = "";

Both forms cause the same effect: browser makes another request to the server.

- **Internet Explorer** makes a request to the directory in which the page is located.
- **Safari and Chrome** make a request to the actual page itself.
- **Firefox** 3 and earlier versions behave the same as Safari and Chrome, but version 3.5 addressed this issue [bug 444931] and no longer sends a request.
- **Opera** does not do anything when an empty image src is encountered.

Empty source is bad.

1. Cripple a server by sending a large amount of unexpected traffic, especially for pages that get millions of page views per day.
2. Waste server computing cycles generating a page that will never be viewed.
3. Possibly corrupt user data. If anyone is tracking state in the request, either by cookies or in another way, he has the possibility of destroying data. Even though the image request does not return an image, all of the headers are read and accepted by the browser, including all cookies. While the rest of the response is thrown away, the damage may already be done.

The root cause of this behavior is the way that URI resolution is performed in browsers. This behavior is defined in RFC 3986 - Uniform Resource Identifiers. When an empty string is encountered as a URI, it is considered a relative URI.

Firefox, Safari, and Chrome are all resolving an empty string correctly per the specification, while Internet Explorer is resolving it incorrectly, apparently in line with an earlier version of the specification, RFC 2396 - Uniform Resource Identifiers (this was obsoleted by RFC 3986). So technically, the browsers are doing what they are supposed to do to resolve relative URIs. The problem is that in this context, the empty string is clearly unintentional.

The src attribute must be present, and must contain a valid URL referencing a non-interactive, optionally animated, image resource that is neither paged nor scripted.

Hopefully, browsers will not have this problem in the future. Unfortunately, there is no such clause for <script src=""> and <link href="">. Maybe there is still time to make that adjustment to ensure browsers don't accidentally implement this behavior.

Though there are so many technologies to optimize the web traffic and to improve the internet browsing experience, but there are few scopes to improve the present technologies. In next chapter few example scenarios have been discussed and new architectures have been proposed to solve those problems.

Chapter 3
Proposed Solution

The goal of Web Optimizer is to reduce the intermediate web traffic. There are four different of web optimizers and their goals are: (discussed in section 1.2).

1. **Cache Sharing**
2. **Cache Tree**
3. **Cache Broadcasting**
4. **LiteWeb**

Proposed Architecture:

This Web Optimizer will work with traditional Client-server architecture. It has two parts:

3. Server Side Web Optimizer (SSWO)
4. Client Side Web Optimizer (CSWO)

Both will act as proxy server with few additional responsibilities.

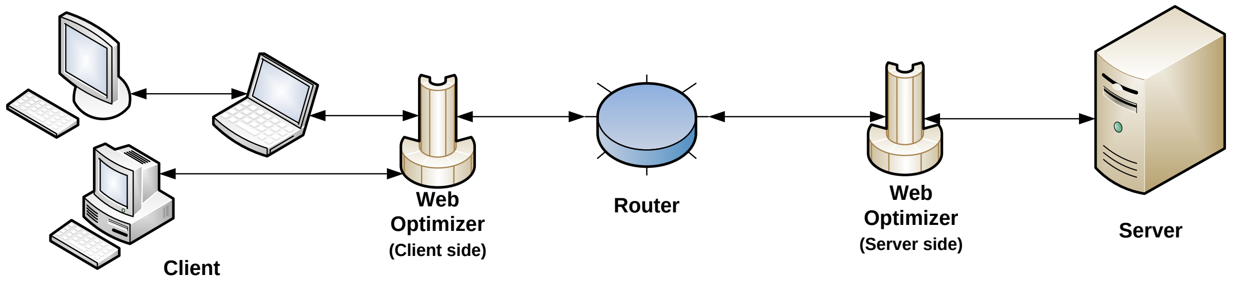


Figure 30: Client side and server side web optimizer

a) **Server Side Web Optimizer (SSWO)**

The goal of server side web optimizer is to reduce the load of server as acting as proxy server and optimize the web content by tracking, caching and broadcasting web content.

- SSWO will keep track of client/CSWO and it will broadcast all ***long live public cacheable content (LLpCC)***.
- SSWO will send ***key-content*** as a lighter version of website if client's bandwidth is low.
- Maintain a ***Cache Tree*** based on the data type and their popularity. It will help to find any update of cacheable content.

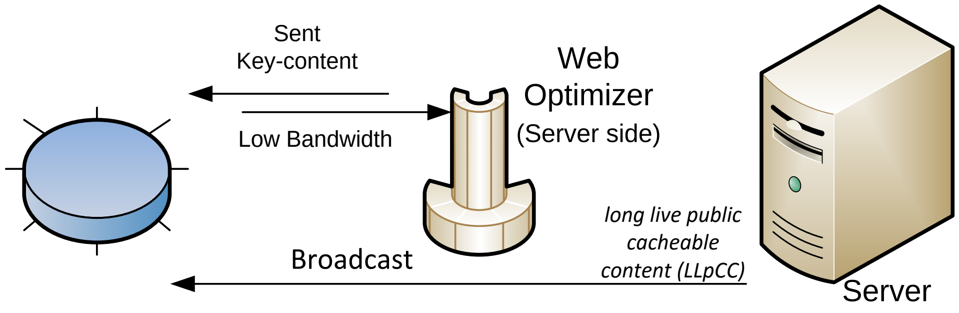


Figure 31: Server Side Web Optimizer

b) Client Side Web Optimizer (CSWO)

The goal of Client Side web optimizer is to keep log of cacheable content of client's pc and reduces web traffic by sharing web contents in between end users.

- Broadcast LLpCC who have requested for that content previously.
- Keep track of cache content and browser's request. Connect Clients locally to let them to share their cached content.
- Maintain a ***Cache Tree*** based on the data type and their popularity. This tree will help to find out any changes on server side quickly and help client to synchronous cacheable content with server.

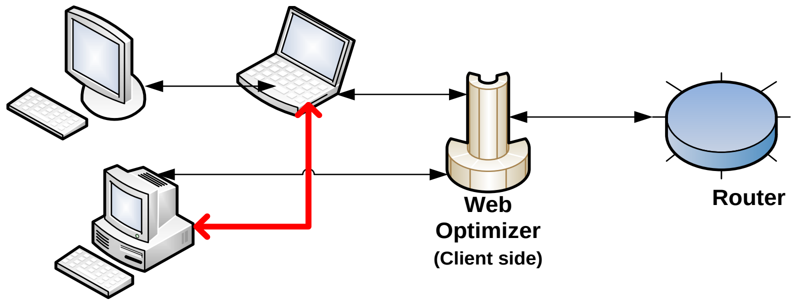


Figure 32: Client Side Web Optimizer

3.1. Cache sharing

Browser stores all cacheable web contents for a while. When a user browses the same page within a few minutes then the content will be served from the cache memory of browser. Sometimes browser keeps those contents for a long time. And also each time there is a request for that cache content the browser also checks is it valid/fresh or not.

In most of the time the popularity of web site depends on the geographical region. In most of the cases the users of a network browse same types of website (News portal, popular sites). Those content have to be served from server. Among those content some are dynamic (very light weighted content) and others are static (images, weight of those content is very high).

Each time a user browses a page, Browser stores few contents to server it later. This mechanism is applied in all modern browsers.

The aim of this "Cache Sharing" is to decrease the distance between server and client by serving the content from a neighbor's computer. It will improve the browsing experience by decreasing the downloading time and also it will decrease the web traffic.

3.1.1. Cacheable contents:

There are different types of contents on a web page. All contents will be grouped/ categorized based on their properties.

1. Long Live public cacheable Content (LLpCC).
2. Temporary Cacheable Content (CC)
3. Key Content (KC)
4. Non Cacheable Content (NCC)

An Extra tag could be added with contents to track the all cacheable contents:

Content-type: LLpCC

Most of websites contain LLpCC like

- Images (Logo and branding images)
- Static images (Those does not change frequently)
- Downloadable Content/Media Files
- Style sheets (CSS files)
- Basic JavaScript files and so on.

And also there are few important contents of websites are known as Key Contents (KC). Key contents are the main part of websites. Those key contents have to be cached very carefully and check the validation with high priority.

- HTML pages
- Content related images
- Frequently modified JavaScript and CSS

Some contents are sensitive and should avoid caching, those are:

- Sensitive data (banking information, password etc.)
- Dynamic Content (based on request, result)
- Frequently changed contents (News portal)

Here this internet provider may work as "Client Side Web optimizer" or another proxy server has to set for it. This web optimizer will maintain a log of all LLpCC.

3.1.2. Scenario Analysis

Internet connects millions of computers with another. This internet could be divided into so many small networks. There exist many computers in a network and they are connected to internet through an internet provider. The distance between a client pc and server could be thousands of miles. All the web contents have to travel this long path to serve the client.

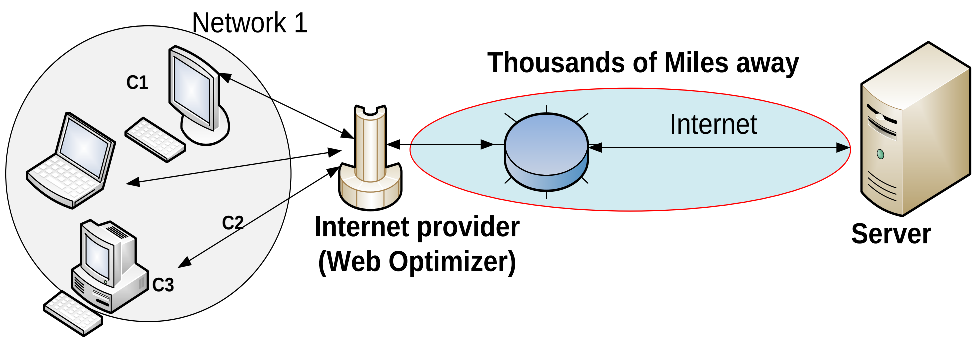


Figure 33: Local networks connected with internet through an Internet provider.

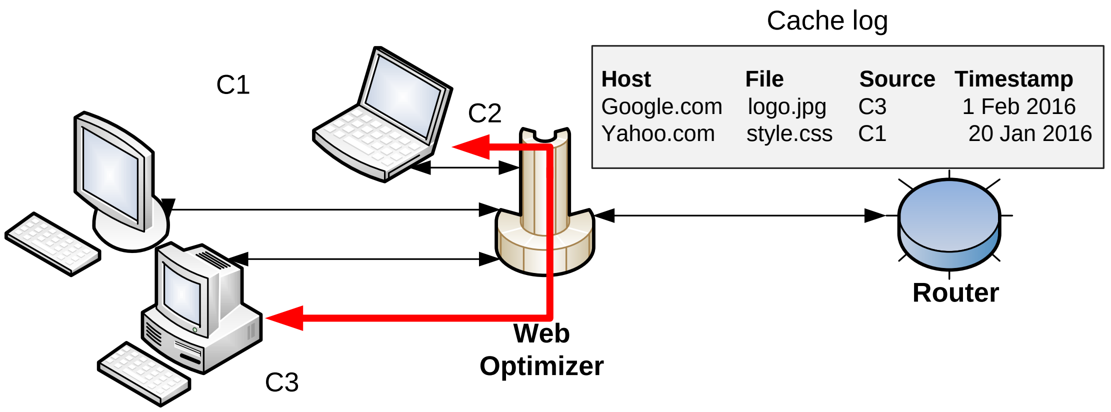
Suppose, there is network consist of 30 users on a network named as "Network 1". They all are contacted to same ISP/local server/internet provider. They are intra connected and can communicate without internet. Through an internet provider they are connected with internet.

Let, three users of Network 1 are browsing googl.com. They will send individual request to the server and server will send the response. So the intermediate path has to carry same content for three times. The aim of "Cache Sharing" is to avoid those redundant web traffics from internet.

3.1.3. How "Cache Sharing" works

Client side web optimizer (CSWO) will keep track of cacheable content of client's PC. If CSWO receive a new request that could be served from client side, then it will not send the request to the server. CSWO will connect host pc (who has that content) with the requester and establish a peer to peer connection to transmit the requested content. The goal "Web Optimizer" could be achieved by with the help of Client side web optimizer. Shareable browser cache will reduce the load of proxy server, network traffic and web pages will be faster. Normally Browser caches are stored in user's computer. The aim of "Cache Sharing" is to make shareable those caches with others.

The CSWO have to maintain a log of public cache contents of end user. A Browser plug-in "Cache Manager" could be introduced to handle intra-networked Figure 5 cache requests shows how the proposed architecture will work. The working procedure of p2p content sharing have been discussed on [8] [12].



Host	File	Source	Timestamp
Google.com	logo.jpg	C3	1 Feb 2016
Yahoo.com	style.css	C1	20 Jan 2016

Figure 34: Cache log of CSWO

3.1.3.1. Data Transaction:

If the content is available on the network and fresh, then that content will be served.

- At first there will be peer to peer connection. This connection will be established by the proxy server.
- Then the content will be served through this connection.

Here the CSWO is playing a vital role in this architecture. CSWO will check the availability of cache provider, which have the requested cache and alive on the local network. If there are any, then it will set a peer to peer connection with client and cache provider. The CSWO is responsible to check the freshness of cache by using the E-Tag or/and modification date of that contents. If the cache failed to pass the validation test, then the cache will be fetched from main server.

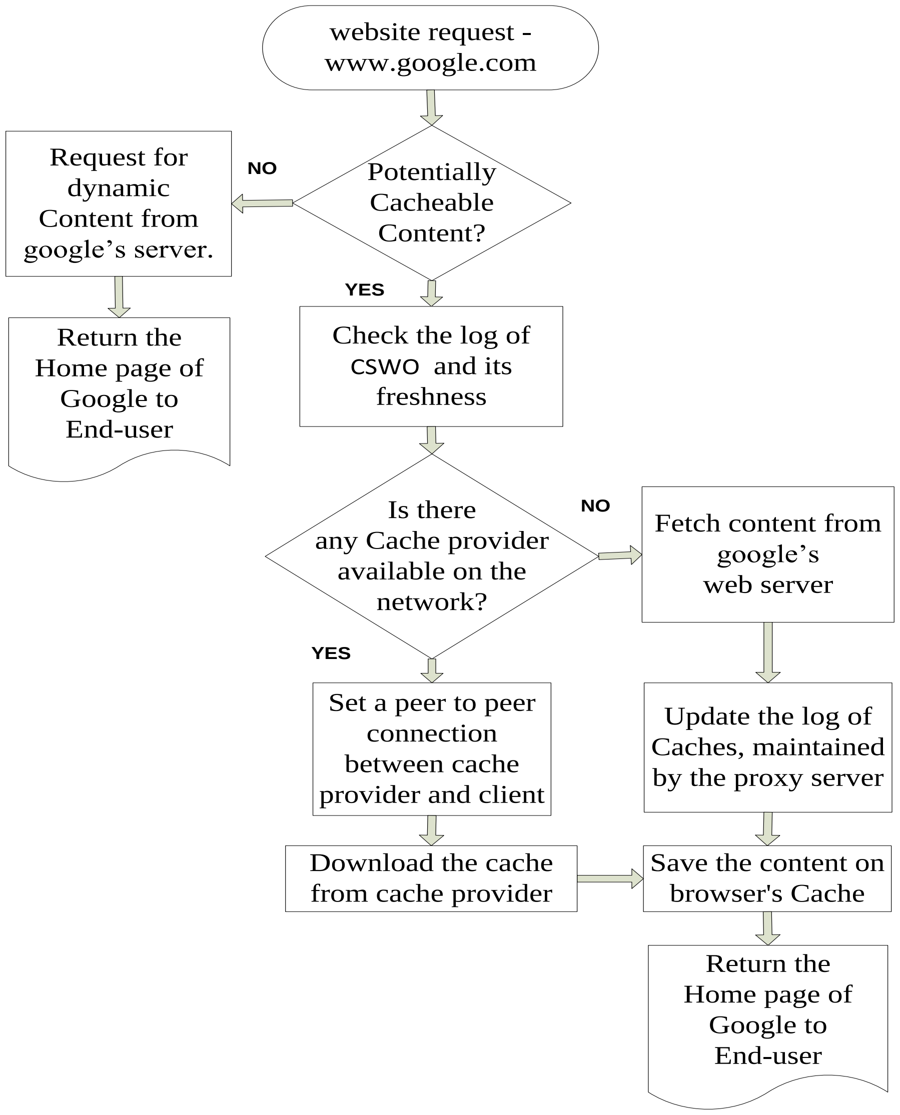

website request - www.google.com
Potentially Cacheable Content?
NO
Request for dynamic Content from google's server.
Return the Home page of Google to End-user
YES
Check the log of CSWO and its freshness
Is there any Cache provider available on the network?
NO
Fetch content from google's web server
YES
Set a peer to peer connection between cache provider and client
Update the log of Caches, maintained by the proxy server
Download the cache from cache provider
Save the content on browser's Cache
Return the Home page of Google to End-user

Figure 35: Flow chart of proposed cache management architecture

3.1.4. Challenges

There are few sensitive issues of this architecture. Sharing contents without any authentication is very risky but normally the cached contents are encrypted by browsers.

a) Security:

The security of caches is one of the major issues of this proposed solution. The caches will be locally stored and shared among each other. Any one may inject malware or virus with those caches. Encryption could be a handy solution for this issue.

The cache contents have to be encrypted in browser cache memory. Only public cacheable content could be send through this connection. Proxy server is responsible to check the malicious content.

b) Multiple Browsers:

Most of the user may use more than one browser on their computer. And the user may not install this plug-in on all browsers, if they do then synchronizing and maintaining those caches of all browsers is another issue.

c) Dynamic Contents:

Dynamic contents are not cacheable, because the contents have been served from the database of main server.

d) Content availability:

To find out requested contents in this network, all public cacheable contents have to be tracked. Then the availability of that content with address will be traceable. There have to be a computer/proxy server to keep track of all those caches.

e) Freshness of contents:

There should be a validation check before each transaction. CSWO may help regarding this issue.

3.1.5. Advantages of Cache Sharing

This cache sharing technique will reduce the web traffic. Less web traffic and request is needed for faster internet.

Assume,

n is number of cacheable content.
d is the distance from client to web optimizer
D is the distance from web optimizer to server (Where D >> d)
c is the contest time of server, required to process a single request.
A is the amount of data is being transmitted from web optimizer to server
a is the amount of data is being transmitted from client to web optimizer.

m is the validation check request by web optimizer to check the validation of cached content.

Table 1. Comparison between traditional system and cache sharing.

Criteria	**Traditional**	**Cache Sharing**
Data transmission	(d*a+D*A)*n	d*a*n+m
Server processing time required	n*c (individual request have to be processed)	m*c (request send by web optimizer)

3.2. Cache Tree

Popular web sites have been browsed thousands of times in a day. Most of the contents of those sites are dynamically generated [Result publication sites/ shear market] or few of them updates continuously [News portal / Blog]. On the other hand, there are private, password protected data of social networking websites, Banks, E-mails and those are not cacheable.

There are some less sensitive contents on every regular website, among them style sheets (CSS file), action scripts (JavaScript), images (logo/ photos) are very common and does not change very frequently. If the administrator finds any bug or wants to alter the look or wants to add some new features, then they change/update those contents. That means every sites have cacheable contents. Those cacheable contents are being cached in cache server and locally stored in browser’s cache memory.

All cacheable contents could be distributed into different cache server. But there are some general issues related with distributed web caching architecture. Such as Extra overhead, size of cache, cache coherence problem, scalability, robustness, hit ratio, and Load balancing, low latency and so on. On the other hand, web servers have to perform lots of functionality to cautiously server the content. Here the validation check request is an overhead for server.

The browser sends a validation check request to the server to check the freshness of that particular resources requested by a user to make sure that the cached resources stored in browser cache are still fresh. Those requests create some extra web traffics on the internet.

The aim of "Cache Tree" is to find out an efficient and effective solution to reduce web traffic by optimizing cache validation check request.

3.2.1. *Case* Study

Suppose there is a website named abc.com and it contains few cacheable contents. User may visit those pages separately or one from other. This website contains

Content type	Cacheable content	Key content
Image file	3	1
CSS file	5	0
HTML file	0	1
Js file	2	0

Now there is a request for abc.com. Home page requires 12 files. According to the traditional system browse will send 12 different requests and download them to render the home page of abc.com. Among them 10 contents are cacheable and others are not cacheable. Browser will store/cache all of those cacheable contents.

During the second request for the same page browser will send 10 "freshness" check request to be ensured that all contents are fresh and 2 regular requests to download rest of the contents. After the first request, every request will contain 10 "freshness" check requests and 2 regular requests. Most of the time those "freshness" check requests will return with negative result. The main aim of "Cache Sharing" is to reduce this validation check requests.

Let, 1 million Facebook users are browsing this website 5 times in a day and each time they browse five different pages in an average and there are 10 files to check the freshness, then the total number of validation check request will be.

Facebook user	10, 00,000
Daily visit	5 times

Number of pages	5
Number of Resources	10

Total: 1000000 x 5 x 5 x 10 = 25, 00, 00,000

25cr validation check request required for only one site. In most of the case those request returns with negative result that means there are no update for that resources. And also the servers have to process those request and send response message to the browser. So there is more 25Cr responses message from server.

To minimize this huge amount request 'Max-age' attribute for cached contents could be used. But there is no appropriate value of 'Max-age'. If the value is large then there are possibilities to loss some update and again if the value is small, then the number of unnecessary request will increase.

The aim of "Cache Tree" is to find out an efficient and effective solution for popular site to eliminate this Validations check request.

3.2.2. Cache Tree

A cache tree will be maintained as secondary index for all cacheable contents. All contents will be distributed into different cache trees. This tree will work as secondary index. A single file could be member of multiples trees. All cache trees could be categorized into two types.

- Grouped Cache Tree (Various Types of file required for a specific task).
- Regular Cache Tree (Based on MIME types)

Among those files LLpCC will not update frequently. Those are the static contents for a website. It may contain logo, style file and images to represent the theme of that website.

During each request all contents will be checked their freshness. This group will ensure that all required files are fresh or not. It will reduce the freshness check request and find out any changes with minimum time and process by using the cache tree.

To check any updates for a specific grouped contents the browser has to send a validation check request with cached time and group name. If the last modified content of that grouped is older than cached time, then no contents will be served from server.

"Regular Cache Tree" will contain all cacheable contents of a website. This tree will be generated based on the file types.

The data structure of cache tree is very important. According to the aim of "Cache Tree" the most updated contents have to be on the top of the tree. So that most recent file will get higher priority than others. If any old files get modified, then it should be on top.

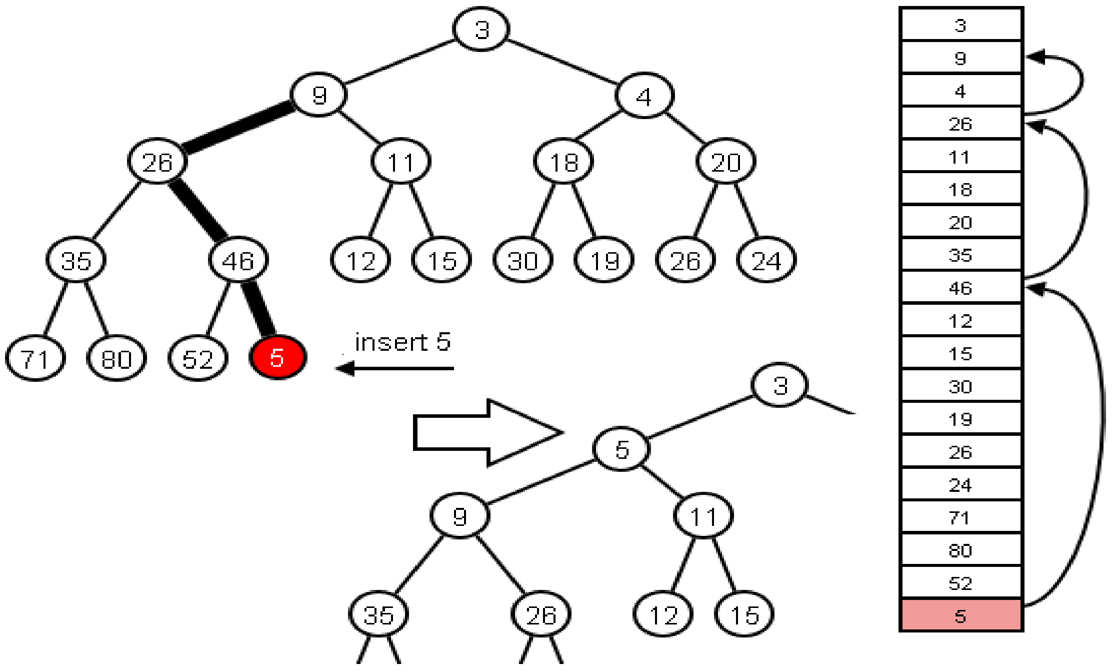


Figure 36: Insert new item on the heap

Cache tree will be maintained based on a Heap, where the most recent will get the highest priority.

i. Select a content:

The most recent element will get the highest priority, is always on top, so it will take Θ (1) to.

ii. Add a content:

Since the cache tree is a complete binary tree, it has minimum height, and the worst case number of moves up is logarithmic, Θ (log n) according to the heap size (n).

iii. Remove a content:

Removing content from the cache tree is less used than others. This step also required Θ (log n) to remove content from the tree.

Cache tree will be always rearranged based on the modification time of file. If a file gets modified, then it will be inserted into first position. So always the top node is the newest file or most updated file of this tree.

3.2.3. How "Cache Tree" works:

The Cache tree is heap where recent content will get higher priority.

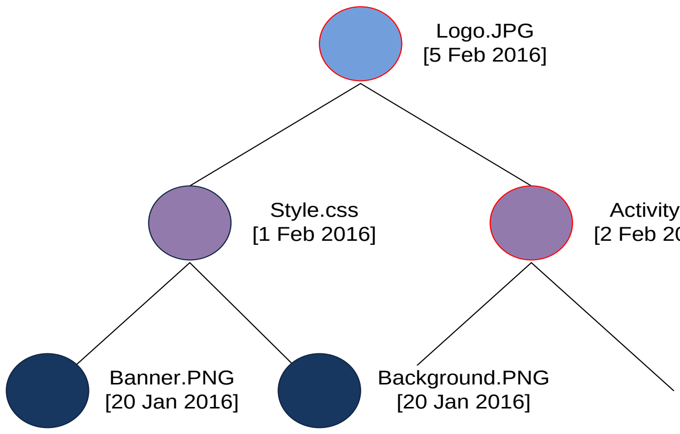


Figure 37: Regular Cache Tree (Server Side)

Same cache tree will be maintained in server side and client side. And the cache tree of client side will be synchronized by downloading the new content from server.

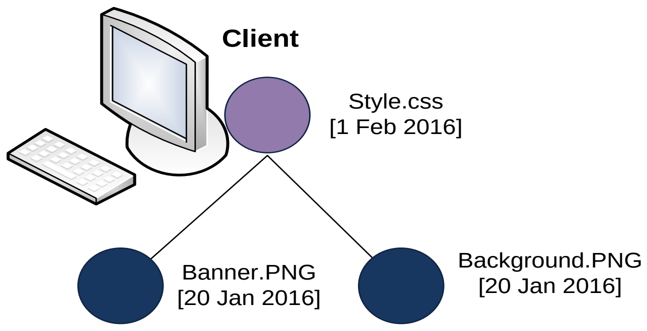


Figure 38: Cache Tree (Client side)

The timestamp of root node will be send with http request.

CacheTree Name: Blue

CacheTree Type: Regular cache tree
CacheTree Timestamp: 1 Feb 2016

Figure 39: Additional information with HTTP request for cache tree

All updated contents, newer than CT timestamp will be sent to the client and also the cache tree will be synchronized. It will also update the "CT timestamp" of client's cache tree. Client will use the new "CT timestamp" for next request.

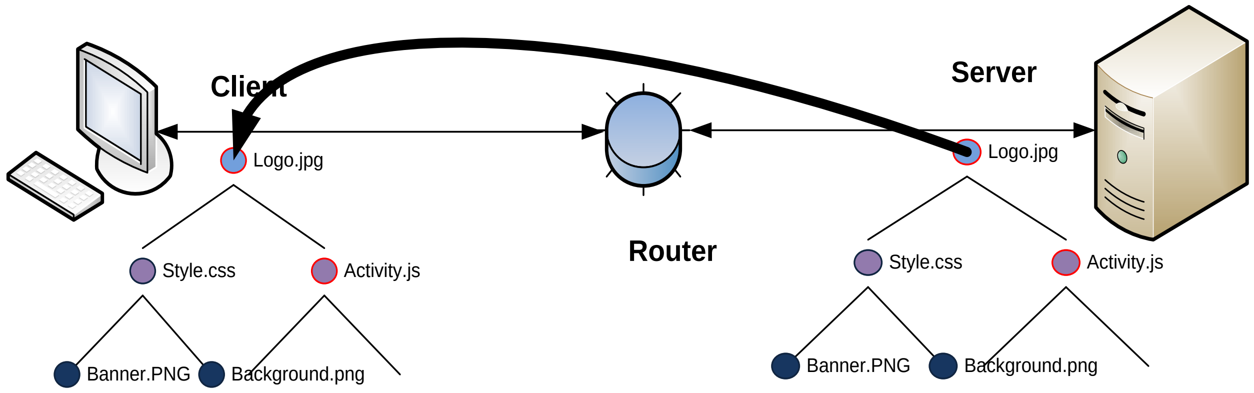


Figure 40: New contents have been served from server to client

3.2.3.1. Working procedure of Cache Tree:

Same cache tree will be maintained on client side and server side. Client's browser will add the last updated time for each group of cacheable contents with request header. An attribute "cached-time: 15:30:00 1 Jan 2016" will be added with the HTTP request header. If any content of that group has been updated in the server side, then the cached-time for that group will be changed after synchronizing and downloading all new contents.

1. **New user:** All contents will be sent to the clients Browser. Browser will store those file till there is no request to update those files from server.
2. **Server has update:** If server has any updates for a particular group of cacheable content. Then server will update its cache tree.
3. **Request from previous user:** previous user means, who have already browsed this site before. Client's browsers have the cached contents. At this point there could be two scenarios.
    a. Server has new "Cacheable Public Contents". It will compare the "cached-time" of client's contents with the modified time with server content. If the server content is updated, then new/ modified contents of that group will be sent to the client's browser. All Contents will be up-to-dated, and also the cached-time will be updated.
    b. If the server has not any updates, then content will be severed from the browser cache and "Cached-Time" will be updated.

3.2.3.2. Example Scenario:

Group a1 (required for home page) requested on 5 January. All cacheable contents have been saved in browser cache.

A CSS file has been changed on 7 January and the cache tree of server side has been updated.

Group a1 is requested on 10 January. Browser will send the last updated time of server with http request. Server will compare this date with modification date of top node of list which contains all cache files for Home page.

Server will send all content newer than "updated date".

<u>5 January</u>

Client Side

1.css		2.css		5.css			
Jan 5		Jan 5		Jan 5			

Server side

1.css		2.css		5.css			
Jan 1		Jan 1		Jan 1			

<u>7 January</u>

Server side

5.css		1.css		2.css			
Jan 7		Jan 1		Jan 1			

<u>10 January</u>

<u>Client side:</u>

- Send **request with** “Last Update date: Jan 5”.

<u>Server side:</u>

- Receive the request and send all newer content than Jan 5. Here “5.css” is newer.

- Server will send the file to the client and client will also update its cache tree.

Client side:

5.css		1.css		2.css			
Jan 7		Jan 1		Jan 1			

15 January

Client side:

- Send **request with** “Last Update date: Jan 10”.

Server side:

- Will receive the request and checked it with the modification date of top node of Group a1.

If the modification date is same or older, then no data will be send. A negative response will be send. (8 different request and response messages have been resolved by only one request and response message. Here it also saves the server processing time)

File compression technology could be used to reduce more web traffic. If there are more than one newer file, then only one compressed file will be sent to the client.

3.2.4. Challenges

Cache tree has to be maintained by client PC and the server. In client side all cache and cache tree is managed by the web browser application (such as, Opera, Google chrome). The cache handling mechanism is different from one browser to another. So the cache tree has to be implemented for such a way that can handle multiple caching mechanisms simultaneously.

3.2.5. Advantages of Cache Tree

This cache tree will reduce the cache validation request and it will also reduce the web traffic. Less web traffic and request is needed for faster internet.

Assume,

n is number of cached content (validation is required to reuse them).
m is number of positive response message per request (with content).
o is number of negative response from server (no update is required)

c is the contest time of server, required to process a single request.

TABLE I. COMPARISON BETWEEN TRADITIONAL SYSTEM AND CACHE TREE

Criteria	Traditional	Cache Tree
Validation check request for single content	1	1
Validation check request for multiple Contents	n	1
Response message from server to client	n	m where, m<< n
Web traffic on the internet	Request: n Response : o+m Total: n+(o+m)	Request: 1 Response: m Total: 1+m
Server processing time required to find new sendable content	n*c (individual request from client have to be processed)	1*c (only 1 request to find new cacheable content)

3.3. Cache Broadcasting

Internet technologies are changing day by day. Web developer may update their web services to provide the most updated features. According to the web architecture client have to send individual request to the server and server will serve the content. This request could be reduced by using cacheable content with 'cache tree', max-age, E-Tag.

To serve individual request of client the server sends the response and it travels a long path of internet. Here the aim of "Cache Broadcasting" is to reduce the intermediate web traffic by broadcasting Public cacheable contents.

3.3.1. Scenario Analysis

Assume that, there are 200 users and they are browsing "facebook.com". So the server of Facebook has to serve the logo of Facebook (for example) at least 200 times through path-1, path-2 path-3 (Figure 41). If the content is cacheable then the client can store that content either the server has to send those contents again and again. If this content is not cacheable or end user does not keep any cache of it then the server has to send this content again and again.

If there is an update for this content on server then according to the traditional system, the file has to be sent to the end user individually if they request for it.

Let,
The file size is 10Kb.
There are 200 end-users.

Total "server to client" data transmission is 2000Kb and also the server has to process 200 requests which have been requested by the end user.

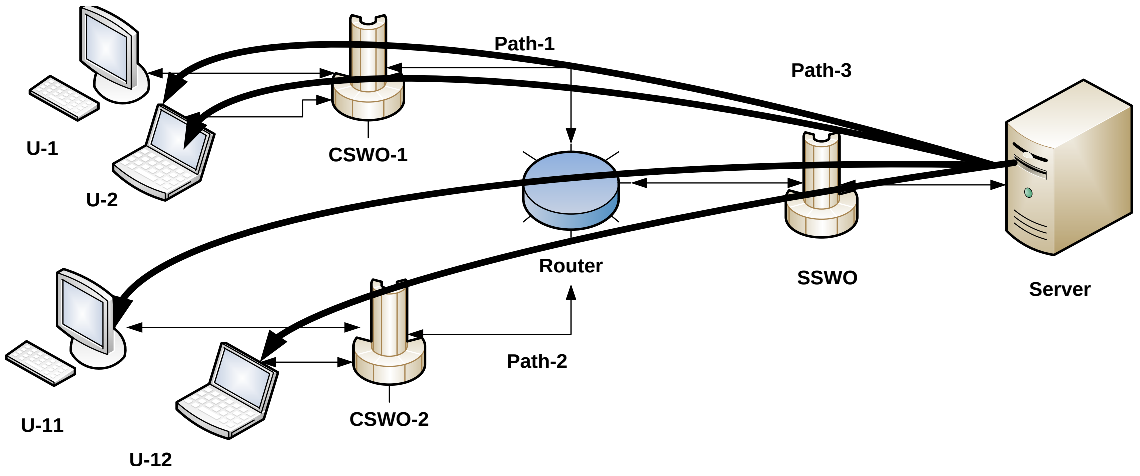


Figure 41: Response message are being served from server

If it is cacheable contents then, 200 individual requests from client to server after the first visit to check the freshness of those contents.

The goal of Cache broadcasting is to reduce the intermediate traffic of server and client with the help of SSWO and CSWO. Here LLpCC are suitable contents for broadcasting, because they do not change very frequently and could be cached for a long time. So the broadcasting requests have to be called very frequently.

3.3.2. Cache Broadcasting

Long Live Public Cacheable Content (LLpCC) does not change very frequently. If there are any changes in server side those will be broadcasted to the client side. This process will reduce the freshness check request and save server processing time and data transmission time. Those LLpCC are also locally shareable.

3.3.3. How "Cache Broadcasting" works

Server Side Web Optimizer will handle this cache broadcasting with the help of client side web optimizer.

1. Server sider web optimizer (SSWO) is responsible for broadcasting LLpCCs. Server will send the broadcasted content to the SSWO and it will send all contents to all known Client Side Web Optimizer (CSWO).

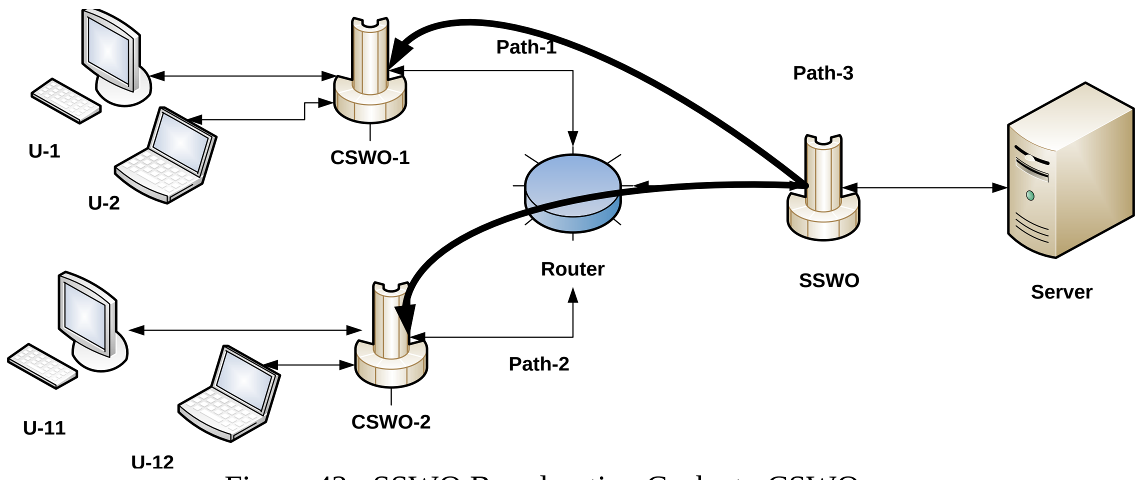


Figure 42: SSWO Broadcasting Cache to CSWO

2. CSWO have to keep track of all end users who have browsed that content earlier and contain those LLpCCs. CSWO receive the broadcasting request from SSWO and then send it to end user. (force update might be an issue for security)

Or

CSWO will keep those LLpCC and during the next request for this content then CSWO will serve it.

3. During regular browsing, the CSWO is responsible to check the freshness of those LLpCC. Cache Tree could be implemented to check the freshness by only one request to be ensured that all LLpCC are fresh or CSWO will check the freshness periodically (once a day) According to the old system they have to send individual request for each contents. In most of the cases those responses are negative.

This broadcasting will reduce the server to client data transmission.

3.3.4. Example Scenario

With the help of broadcasting LLpCC, if there are updates in the server the server will send it to the SSWO. Then the SSWO will broadcast it to the CSWO.
CSWO will serve it to the end user. If the file size is 10Kb then the data transmission between

- Server to SSWO is 10Kb
- SSWO to CSWO is 20Kb (there are 2 CSWO)
- CSWO to End user is 2000Kb (200 End user)

Here, it saves at least 1980Kb data transmission from server to CSWO for a single update.

3.3.5. Challenges:

The Broadcasting is simple way to send content but it causes problems like traffic jams, network congestion and so on.

1. Possibilities of Network Jam.
2. Security is one of the major issues of cache broadcasting. There are some possibilities of harmful contents being broadcasted and downloaded by the clients.
   This security risk could be handled by ensuring that the client is accepting the broadcasted contents when he needs it.

3.3.6. Advantages

This Cache Broadcasting will reduce the web traffic by reducing the redundant server to client data transmission.

3.4. LiteWeb

Internet is now medium of communication, services, education and business. Many organizations are providing so many services through internet. But this internet is not available to all of us. In many countries, Internet is a dream for them. Some Developing countries are trying to improve their internet facilities, but due to so many constraints the end users are not receiving the optimum services. Most of the mobile operators offer internet facilities with "data limit". In most of the country, the bandwidth of internet is very poor. To make a website more interactive developers use large images and lots of scripts.

LiteWeb will be helpful to browse a website with low bandwidth. Many web sites contain so many style sheet, animations, and videos to make web pages more attractive to the viewers. Except key-contents most of them are avoidable. A web page that contains only key-content will be treated as example of LiteWeb.

3.4.1. Scenario Analysis:

Let Internet Speed of User-12 is very slow. User-12 is trying to browse abc.com, but it seems very slow. It takes more times to download all scripts and images of that website and render all of it on the browser.

abc.com contains

50 Kb of text
750 Kb of script
2000Kb of images (15 images, including logo and other theme related files)

Total 2800Kb data have to be transferred from server to client. Among them 50 Kb of text and 3 200Kb of are key contents. That means 250 Kb data are the important information and rest of contents are sent to decorate the page.

If the bandwidth of user is 50Kbps then it will take almost 1 minute to download all the contents.

3.4.2. Introduction to LiteWeb

LiteWeb are consists of all key contents. Those key contents will be severed to the client with the help of web optimizer. Before that, the developers have to decide what contents will be treated as "key contents". Few more information could be added with contents to solve this problem.

Content-type: Key-Contents

MIME is used to describe the content type. A new tag "Key-Content" will be added to track all key contents of a website. On the other hand, a tree similar with "Cache Tree" could be used to track all key contents of a web site.

According to the client's request or based on the feedback of CSWO all those Key contents will be sent to the clients. This response will be lighter than regular response, that's why it is named as "LiteWeb".

3.4.3. How "LiteWeb" works

Client side Web optimizer will send request for LiteWeb content based on the bandwidth of clients. Clients may also change the settings as

- Automatically detect and request for LiteWeb.
- Only LiteWeb
- LiteWeb (key content) will get higher priority

CSWO will also keep track of client's bandwidth and send it to the SSWO. SSWO will check the bandwidth of client side computer. It will check the settings (embedded information with request) from the request of the web browser, then SSWO will take decision based on the client's preference.
During regular transmission key content will get higher priority and will be sent before other contents.

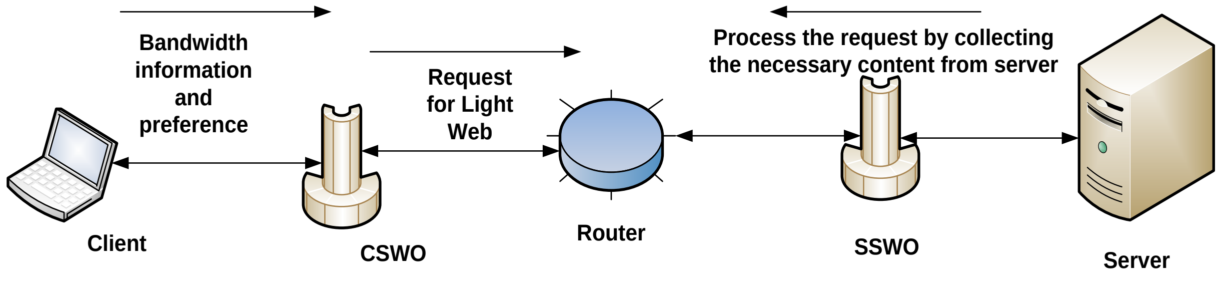

Bandwidth information and preference
Request for Light Web
Process the request by collecting the necessary content from server
Client
CSWO
Router
SSWO
Server

Figure 43: "LiteWeb" request is being served by server

3.4.4. Scenario analysis

With the help of LiteWeb technology, the necessary content could be served faster with low bandwidth. Based on the user's preference rest of the content could be served later. Here the web developers have to distinguish the key contents from others.

Traditional technology takes about 1 minute to download 2800 Kb contents, where key contents are only 250Kb. with the bandwidth of 50Kbps, key contents will take only 5 seconds to be transferred from server to client. The clients do not have to wait to browse this website.

This LiteWeb technology will also save the data limit of user. Some internet providers restrict the data limit or offer good bandwidth with low price.

By using the LiteWeb technology, the user may save their data usages.

“Responsive design” web site designs are getting popular that change the appearance of website based on the screen resolution/dimension. The goal of “LiteWeb” is a responsive design that will change the web content (image/script) based on the availability of internet bandwidth.

3.4.5. Challenges

The look of expected website is the main challenges for “LiteWeb”.

1. Inconsistent Layout could be major problem for “LiteWeb”. Because many scripts and images will not be served from server.
   Responsive web design with proper information of height and width of multimedia contents is a solution for this challenge.
2. Possibilities of missing information. Mostly images will not be included as Key-content. To solve this problem alternative text (“alt” attribute of “img” tag, used to add alternative information about that images) could be very useful.
   On request image, **could** be another option. Beside alternative information there will be button to download that image.

3.4.6. Advantages

This “LiteWeb” will reduce the web traffic that will help the client to download necessary content faster within limited bandwidth.

1. Key contents will get more priority, so the clients do not have to download unnecessary scripts or images.
2. The LiteWeb will also reduce the web traffic.

Chapter 4
Validation

To validate the proposed solution a prototype system has been designed based on java. Where the server sends response to the client according to the proposed architecture.

4.1. Prototype of Cache Tree

A prototype has been designed to demonstrate the working procedure of "Cache Tree" based on java (Appendix I). Where, there are 12 contents on the server. Client's required those contents to render a page. During the first request the client will download all of those contents.

```
From Client: Have to update items(id): 0 1 2 3 4 5 6 7 8 9 10 11
From Client: Adding Content: 0
From Client: Adding Content: 1
From Client: Adding Content: 2
From Client: Adding Content: 3
From Client: Adding Content: 4
From Client: Adding Content: 5
From Client: Adding Content: 6
From Client: Adding Content: 7
From Client: Adding Content: 8
From Client: Adding Content: 9
From Client: Adding Content: 10
From Client: Adding Content: 11
```

Figure 44: Client is Downloading content from the server (First time)

There is a heap on the server side and it works as "Cache Tree". This heap contains the content id and timestamp.

```
Heap Content: Id: 9 Time: 1
Heap Content: Id: 10 Time: 1
Heap Content: Id: 11 Time: 1
Heap Content: Id: 6 Time: 4
Heap Content: Id: 8 Time: 2
Heap Content: Id: 5 Time: 5
Heap Content: Id: 4 Time: 6
Heap Content: Id: 0 Time: 10
Heap Content: Id: 3 Time: 7
Heap Content: Id: 2 Time: 8
Heap Content: Id: 7 Time: 3
Heap Content: Id: 1 Time: 9
```

Figure 45: Heap data structure has been used to construct this Cache Tree

If there are no updates on the server side, then it will send a negative response to the client.

```
From Client: Nothing to do, updated Till now
```

Figure 46: There is no updated content on the server

Meanwhile, two new contents have been added on the server and two other contents have been updated. At the same time the cache tree has to update accordingly. The cache tree will help the server to find out the new content at minimum possible time and inform the client.

```
Heap Content: Id: 12 Time: -1
Heap Content: Id: 7 Time: -1
Heap Content: Id: 13 Time: -1
Heap Content: Id: 6 Time: 4
Heap Content: Id: 10 Time: 1
Heap Content: Id: 5 Time: -1
Heap Content: Id: 9 Time: 1
Heap Content: Id: 0 Time: 10
Heap Content: Id: 3 Time: 7
Heap Content: Id: 2 Time: 8
Heap Content: Id: 8 Time: 2
Heap Content: Id: 1 Time: 9
Heap Content: Id: 11 Time: 1
Heap Content: Id: 4 Time: 6
```

Figure 47: Updated “Cache Tree” of Server

After that, if client request for those content. The server will inform the client that there are 4 new updates.

```
From Client: Have to update items (id):  5 7 12 13
```

Figure 48: Server informed the client about new contents

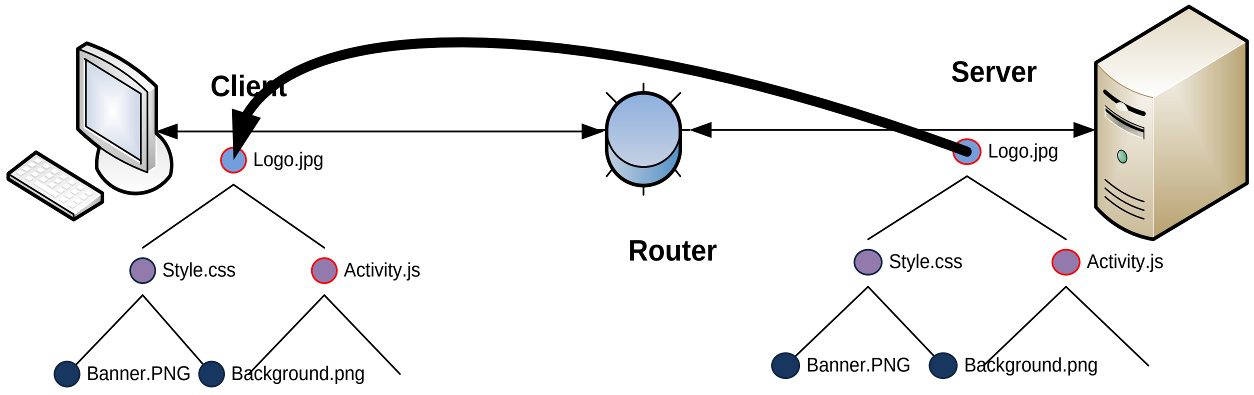


Figure 49: New contents have been served from server to client

After that the client will collect those contents form server.

```
From Client: Updating Content ID: 5
From Client: Updating Content ID: 7
From Client: Adding Content: 12
From Client: Adding Content: 13
```

Figure 50: Client collect the new updated content form server

If the modification date is same or older than no data will be send. A negative response will be send. (12 different request and response messages have been resolved by only one request and response message. Here it also saves the server processing time)

4.2. Prototype of Key Content

A prototype has been designed to demonstrate the working procedure of “Key Content”. To identify the key content an extra information has been added with content properties. If the client request for “Key contents” due to bandwidth limitation, then the server will send only the key contents related with the requested page.

```
number of content :5
Total Data: 3800
File name: logo.jpg || Size: 250  || Modification Date: 5
File name: style.css || Size: 50  || Modification Date: 5
File name: image1.jpg || Size: 500  || Modification Date: 5
File name: image2.jpg || Size: 1000  || Modification Date: 5
File name: image3.jpg || Size: 2000  || Modification Date: 5
```

Figure 51: Regular Request served by Server

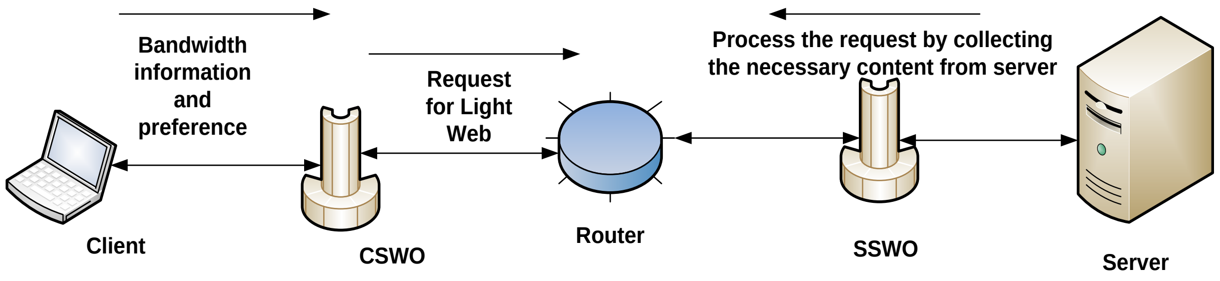

Bandwidth information and preference
Request for Light Web
Process the request by collecting the necessary content from server
Client
CSWO
Router
SSWO
Server

Figure 52: “LiteWeb” request is being served by server

Here the server sends 5 contents and the total size of 5 contents are 3800 Bytes. If the client request for only “key contents” then the server will send only 2 contents.

```
number of content :2
Total Data: 750
File name: logo.jpg || Size: 250 || Modification Date: 5
File name: image1.jpg || Size: 500  || Modification Date: 5
```

Figure 53: Key Contents Severed by the server

Here, this procedure saved 3050 bytes and it will load five times faster than regular system.

Conclusion and Future Work

The goal of web optimizer is to minimize the intermediate web traffic that will help to browse web content faster. There are few excising architectures that work to improve the web browsing experience. Among them Web Cache works to minimize the number of downloadable contents and Content Delivery Network (CDN) works to minimize the distance between client and server. The working procedures of web cache and CDN have been discussed on chapter 2. Where web cache requires freshness check for each cached contents and it causes extra web traffic.

Redundant web content on internet is another scope of improvement. A scenario has been discussed on chapter 1, where redundant content exit on intermediate path in between server and local network. Bandwidth limitation on client’s end is one of the major challenges of present web technology.

A new architecture named as “Web Optimizer” has been proposed to improve the present scenario of web. This architecture includes Server Side Web Optimizer (SSWO) and Client Side Web Optimizer (CSWO) as transparent proxy server. This SSWO and CSWO helps to minimize web traffic by using “Cache Tree”, “Cache Broadcasting” and “Cache Sharing”. To serve the key contents to the clients “LiteWeb” have been introduced.

Cache Tree helps to synchronize cached content of browser with server’s contents based on the modification date. Only new or updated contents will be downloaded from server. Only one check is required to be confirmed that there are no downloadable contents on the server, where the number of cached contents does not matter.

Cache Broadcasting ensures that multiple clients of a network do not have sent same request to the server for required content individually. The server will broadcast new cacheable contents to all known CSWO. Then CSWO is responsible to distribute or serve it to the client.

Cache Sharing tries to collect required contents from local network. Here CSWO maintain a log of public cacheable contents. If any client of that network requires any file, then CSWO check its log and if that content is stored on another computer of same network then the CSWO establish a peer to peer connection with another. Then the host of that content acts as server and send the response. Here CSWO is responsible to check the freshness of those contents according to its log file. LiteWeb helps to serve only the key content based on the request of client. This LiteWeb helps to browse website with low bandwidth.

The proposed “Web optimizer” will help to browse internet faster by minimizing the web traffic. To validate this proposed system a prototype has been designed based on java. But the actual environment is different form prototype. Our future plan is to implement this proposed architecture within real application.

Reference

[1] Mukesh Dawar, Charanjit Singh , “Study on web Caching Architecture: A Survey”, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 11, November 2013.

[2] Valeria Cardellini , Michele Cola janni, Philip S. Yu , “Dynamic Load Balancing on Web-server Systems”, IEEE Internet Computing, vol. 3, no. 3, pp. 28-39.

[3] “Optimizing content efficiency”, url: https://developers.google.com. Accessed on : 20January 2016.

[4] “ How to control cache”, http://www.web-caching.com/ , Accessed on : 22January 2016

[5] “ Caching tutorial”, https://www.mnot.net , Accessed on : 29 February 2016.

[6] “Caching in HTTP”, https://www.w3.org/Protocols/ , Accessed on : 2 February 2016.

[7] P. Rodriguez, C. Spanner, E.W. Biersack, “Analysis of Web caching architectures: hierarchical and distributed caching”, IEEE/ACM Transactions on Networking (Volume: 9, Issue: 4, Aug 2001)

[8] R.L.R. Mattson and S. Ghosh, "HTTP(P2P): a transaction based (HTTP) peer-to-peer protocol for the dissemination of Web-objects in congested networks", Cluster Computing and the Grid 2004. CCGrid 2004. IEEE International Symposium on, pp. 174-181, 2004.

[9] Troy A. Johnson , Patrick Seeling , “CacheConnect: On-device proxy and web cache for performance increases”, Consumer Communications & Networking Conference (CCNC), 2016 13th IEEE Annual.

[10] "Cisco visual networking index: Global mobile data traffic forecast update 2014–2019", Feb. 2015.

[11] T. Miyoshi, “Consideration about the cache server optimization and stable network access”, Knowledge-Based Intelligent Electronic Systems, 2008.

[12] Jin Li. “On peer-to-peer (P2P) content delivery,” Peer-to-Peer Networking and Applications, vol. 1, pp. 45-63, Jan. 2000.

[13] Jia Wang, “A Survey of Web Caching Schemes for the Internet”, Cornell Network Research Group (C/NRG), 2009.

[14] Hai Jiang, Jun Li, Zhongcheng Li , Xiangyu Bai , "Performance Evaluation of Content Distribution in Hybrid CDN-P2P Network "; Second International Conference on Future Generation Communication and Networking 2008.