# Method to Prevent Caching of JavaScript and CSS Files by ISP and at Browser End

**Prof. Ramesh Solanki[1] Bharti Ludhwani[2] Neeraj Chawla[3]**
[1]Assistant Professor
[1,2,3]Department of MCA
[1,2,3]VESIT, University of Mumbai, Mumbai – 400074, India

*Abstract*— In modern times most of the application developed are web based. Development and maintenance process of these applications involves a lot of trials and errors at the frontend scripting side. Which is quite obvious, but many a times this development process is done on a client server environment. The problem that arises in this scenario is, most of the script files are cached by browser and Internet Service Provider (ISP). This is a feature provided by almost all browsers and many ISPs, to accelerate the load time of the pages and reduce the network load. However for developers this may turn out to be an irritating experience. As they keep on making changes to the scripts and sync with or upload it to the server. They are unable to see the changes because, of the caching of the script files by browser and ISP. Proposed system addresses this issue by applying a mechanism at the server side to break through this caching mechanism at both the level.

***Key words:*** ISP, Caching, PHP, Server Side Scripting

## I. INTRODUCTION

In current times there is a trend in which developers and companies around the world take the entire software experience and put it online on the internet, and call it cloud. Most of the applications developed in recent time are web based applications. The very reason of this trend is ease of deployment, widespread availability and real-time update of the system. Also at the developer end its quite easy make changes to the application at real-time because the entire application is developed at multiple layers e.g. frontend and backend, in general.

Having a closer look at the front end, this deals with the user-interface and user-experience and many other validations at the client (browser) side. This development process sometimes involves working in a real-time client-server environment. Most of the developers and maintenance engineer work on script files and sync it with the server or upload it into the server. Till this part of the development things seems to be very simple and easy. What follows next is quite obvious but may turn out to be a bit painful. Now the output is to be seen on a browser, by hitting the URL of the server. As described earlier, script files are cached by both browsers and ISPs. This may turn into an irritating experience, because the files cached at browsers and ISPs are loaded every time you load the web pages. So basically all the script files are loaded from either browser or ISP cache. This surely accelerates the page loading speed, but the latest changes made to the file don't get reflected on the page for obvious reasons. This leads to a confusion many a times that, whether or not you are working on correct file. Or in many cases, whether the code written by you is complete and correct.

All files cached by browser or the ISP cache server is cached based on their name. The proposed system uses a technique which makes use of the of the server side scripting language to dynamically change the name of the file every time the file is modified. The proposed system makes use of PHP as the server side scripting language. However the technique used is a language independent concept and can be implemented using almost any language which can be run on a web server. This paper will focus on the technique proposed and not on the language of implementation. The only reason of selecting PHP as the server side scripting language for illustration is that it is simple and is interpreted on the fly and secondly it's just a matter of choice. Authors of this paper preferred PHP but readers may use any other language of their choice.

The rest of the paper is organized as follows. In section 2 we describe how the proposed system will work. In section 3 we will illustrate with example how the technique will work. Section 4 will be the concluding section of the paper and will put some light on the future work.

## II. AN OVERVIEW OF THE WORKING OF THE PROPOSED SYSTEM

This proposed system is based on the general idea that caching of any file at the browser side or by ISP cache servers is based on the file name. To be more specific, if a file having a name suppose for example 'script.js' is referred in your page. Browser of the ISP caches the file based on its name and not by its date and time of modification. So next time when you reload the page and a request for the same file is made to the server. It is either loaded from the browser cache or the ISP cache. Both the browser and ISP side caching is based on the assumption that these files don't change very frequently and can be reutilized from local cache for a certain period of time.

The above assumption is right in general scenario, where you do not deploy a new build of the project or make changes to the project every now and then. But in a development environment, it is quite a different scenario. Developers make changes to the file quite as often as every minute or even probably in seconds. And they hit the server for the output just like any real time environment. Now there arise a problem, as the cache at both browser and ISP level is obsolete, but since both level of caching do not consider last modification date. The file that is loaded onto the browser does not reflect the changes made recently.

Now in this scenario, if the caching is done at the browser side. User has the option to clear the cache and reload the page. Which is not very convenient but a firm solution. But if the file is cached at ISP cache server, User is left with not much options but to wait for a while till the cache at the ISP expires. As the time may vary for different ISPs. This waiting time may also go up to an hour. Which is not feasible for developers. Figure 1 illustrates the flow of current scenario.

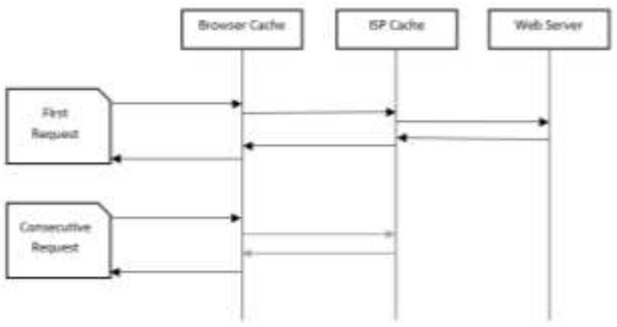


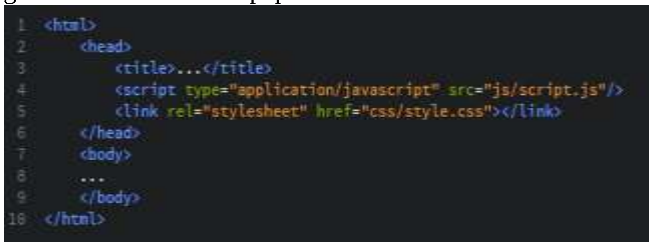
Fig. 1: Caching by browser and ISPs

Proposed system is based on the concept that files are indexed based on their name and changing the name of the file will simply break through the caching mechanism. However, changing the name of the files manually whenever they are modified is not feasible because, firstly, it will become very tedious and annoying for the developers and secondly, the same file is referred many other pages. Changing the file name manually may lead to a broken script error in all the other files where it was referred.

The proposed system does the job of renaming the files based on their last modified date. This modification of file name is dynamic and at runtime. And if applied properly to all the pages in the entire web site/application it will also make sure that your application pages do not face the broken script issue. The idea is quite simple. Writing a script using the server side scripting language that is basically a function which will replace the original script and link tags in your page. This function will take one parameter, i.e. the file name with its complete path for reference. Now every time the pages will be loaded the function will generate the script and link tags based on the last modified timestamp of the respective files (elaborated explanation of the same is given in the next section). And of course there will be a file maintained by this server side script. To keep track of the last modified date time of the files. This mechanism simply adds the timestamp to the file name. Which is going to be different every time the file is modified and hence the filename will change only when it is modified.

## III. WORKING MECHANISM AND FLOW OF THE SYSTEM

Having already described the fundamental idea behind the working of the proposed system. Let's dig in more into the what-&-how of the system. And how things will work technically in the system. This section will illustrate the working flow of the system and also the mechanism on which it has been designed and implemented. PHP will be the language used for any code illustrations and example given here after in this paper.



```
<html>
    <head>
        <title>...</title>
        <script type="application/javascript" src="js/script.js"/>
        <link rel="stylesheet" href="css/style.css"></link>
    </head>
    <body>
    ...
    </body>
</html>
```

Fig. 2: Code snippet of simple html file.

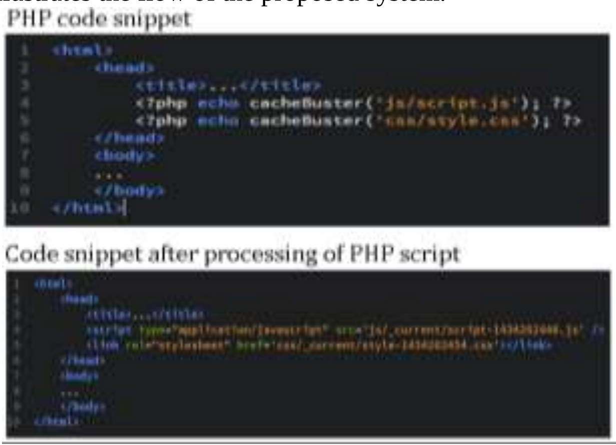
Figure 2 shows a code snippet which contains a script tag and a link tag. Both linking to a JavaScript and CSS file respectively. The proposed system uses a PHP function instead of these tags. The function will be called when a request for the page is made to the web server. Now here arises two scenario wiz, when the request is made for the first time and when request is not the first one to hit the server but the file is modified. Section 3.1 and 3.2 explain each of the scenarios in detail. Looking at the above code snippet in Figure 2, The JavaScript file is at '*js/*' directory of web root and CSS file is in '*css/*' directory.

### *A. Flow of System If the Request Is Made For the First Time*

When the first request is made the function searches for '*_timestamp.txt*' (this file is created by the code itself to keep track of the last modified date) file in both '*js/*' and '*css/*' directory. Since it is the first request file is not font in both the directory. A new file with the same name is created in both the directories. This file contains the last modified timestamp of each file in the directory. After creating this file in each of the directory, the code snippet then proceed further and created a directory in each of the above mentioned directory. Name of the directory being '*_current/*'. These directories contains the files with timestamp appended to their name. i.e. '*script.js*' becomes '*script-1434262454.js*' where the digits are the last modified timestamp in UNIX epoch time format. Basically the same file that were present in '*js/*' and '*css/*' directories are copied to their respective '*_current/*' directories by adding timestamp to their name. Since this timestamp is the last modified timestamp of the file. Whenever the file is modified the timestamp changes and so does the name.

### *B. Flow of System If the Request Is Not For the First Time*

Now if the request is an nth request i.e. not the first request. Then the script checks each file's last modified timestamp against the previously stored timestamp in the '*_timestamp.txt*' file. If it's the same then their respective tags are created referring to the respective file in the '*_current/*' folders and the script looks like the one shown in Figure 3. If the last modified timestamp defers then a copy of same file is created in the '*_current/*' directory replacing the old one. Only difference being that the file name is changed so as to reflect the last modified timestamp in its name. After this the corresponding changes are made to the '*_timestamp.txt*' file for further requests. Finally the respective tags are created in the html content so that the newly created file gets loaded at the browser side. Figure 4 illustrates the flow of the proposed system.



PHP code snippet

```
<html>
    <head>
        <title>...</title>
        <?php echo cacheBuster('js/script.js'); ?>
        <?php echo cacheBuster('css/style.css'); ?>
    </head>
    <body>
    ...
    </body>
</html>
```

Code snippet after processing of PHP script

```
<html>
    <head>
        <title>...</title>
        <script type="application/javascript" src='js/_current/script-[illegible].js' />
        <link rel="stylesheet" href='css/_current/style-[illegible].css'></link>
    </head>
    <body>
    ...
    </body>
</html>
```

Fig. 3: Code snippet after applying the proposed system

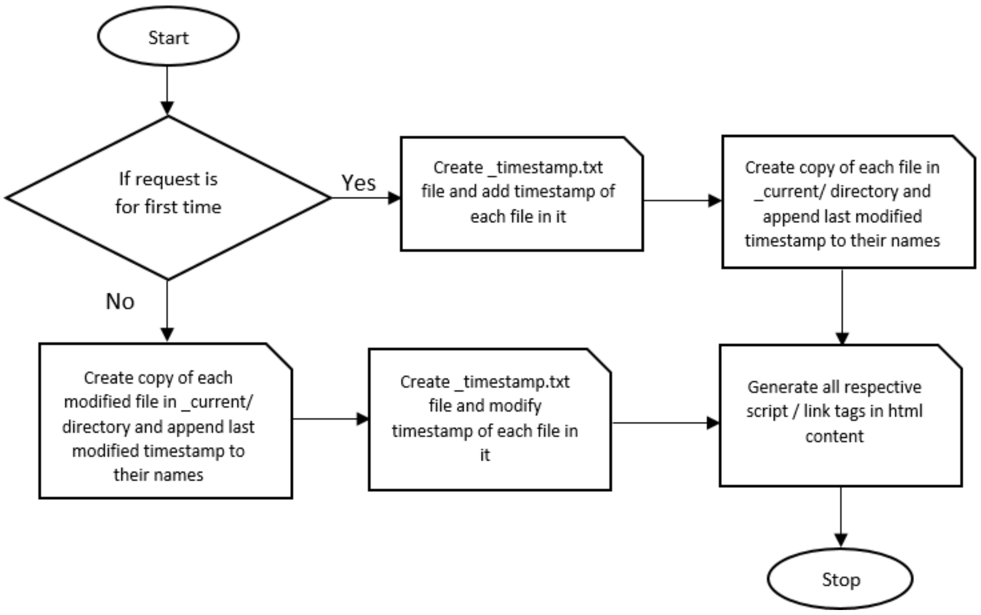

Start
If request is for first time
Yes
No
Create _timestamp.txt file and add timestamp of each file in it
Create copy of each file in _current/ directory and append last modified timestamp to their names
Create copy of each modified file in _current/ directory and append last modified timestamp to their names
Create _timestamp.txt file and modify timestamp of each file in it
Generate all respective script / link tags in html content
Stop

Fig. 4: Flow diagram of the proposed system

So basically, whenever the file is modified, a new copy of the file is created in the '*_current/*' directory with a change in the name. This not only bursts the cache system when changes are made to the files. But also ensures that if files are untouched. Taking full advantage of the cache system. Because the file name is not changed if it is not modified, hence the request can be satisfied from the cache. That also makes sense to the concept of caching system.

## IV. Conclusion and Future Work

The idea proposed in this paper handles and manages the caching system at the browser end and also at the ISP level. It works as an addition to the current caching system by bringing in the last modified timestamp to the file name to ensure that only the file that are not modified after the last hit made on them are taken from the cache. And rest all files are freshly loaded from the server.

Later development in this project would include a mechanism for combining all the JavaScript files and CSS file into one file of its kind i.e. all JavaScript file be combined into one '*.js*' file and all CSS file be combined to one single '*.css*' file. This would not only reduce the load time consumed by server to make an individual load request to file I/O system. But also reduce the call made by browser to load each script file.

## References

[1] Documentation of web caching [Online]. Available: https://www.mnot.net/cache_docs/
[2] Documentation on browser side caching [Online]. Available: http://gtmetrix.com/leverage-browser-caching.html
[3] Documentation on browser side caching [Online]. Available: http://refreshyourcache.com/en/cache/
[4] Documentation on proxy caching [Online]. Available: http://www.squid-cache.org/
[5] Documentation on proxy caching [Online]. Available: https://docs.trafficserver.apache.org/en/latest/admin/http-proxy-caching.en.html