

# Project Report

for

## Proxy Server - Web Cache



Group Members  
Mehdi Raza Rajani - 16k3904  
Moazzam Maqsood – 16k3868  
(Students of Section - F)

Course Instructor  
Dr. Jawad Shamsi

May 10, 2019

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Project Motivation . . . . .	3
1.2	Proposed Solution . . . . .	3
1.3	Proxy Server . . . . .	3
1.4	Web Cache . . . . .	4
1.5	Project Objective . . . . .	5
<b>2</b>	<b>Methodology</b>	<b>6</b>
2.1	Fetching HTML File from URL . . . . .	6
2.2	TCP Connection . . . . .	7
2.3	JAVA SWING Component . . . . .	7
2.4	Basic Structure of Proxy Server . . . . .	7
<b>3</b>	<b>Different Strategies to manage cache</b>	<b>9</b>
3.1	FIFO . . . . .	9
3.2	LIFO . . . . .	10
3.3	LRU . . . . .	11
3.4	MRU . . . . .	12
3.5	Random . . . . .	13
<b>4</b>	<b>Results</b>	<b>14</b>
4.1	Fetching Web File with and without Cache . . . . .	15
4.2	Cache Hit Ratio with different Algorithms . . . . .	15
<b>5</b>	<b>Conclusion</b>	<b>17</b>

### **Abstract**

A proxy server is a server that sits between a client application, such as a Web browser, and a real server. It intercepts all requests to the real server to see if it can fulfil the requests itself. If not, it forwards the request to the real server. Web proxy cache plays a vital role in reducing the server loads, client request latencies and network traffic. Web Proxy Server includes the concept of Web Cache, if requests are made for the first time it is saved into the cache else it processed returned to the client without sending it to servers.

Different strategies are used in web cache like LIFO, FIFO, MRU, LRU etc. The objective of this project is to implement a server-side proxy server which can be used to reduce the network traffic and analyse which of the above strategies better Cache Hit Rate.

After the analysis, we concluded that LRU (Least Recently Used) produces greater Cache Hit Rate but it is not still efficient so we have to come up with some other approaches which are discussed in conclusion portion. Web Cache also reduces the retrieval time of the web page and reduce the network traffic.

We first studied the research paper “Design and Implementation of Server Side Web Proxy Caching Algorithm” by Dr.K.Ramu and Dr.R.Sugumar and then implemented the concept.

# 1 Introduction

## 1.1 Project Motivation

When a user enters a URL on a web page it is first sent to DNS Server to get the IP Address of the requesting Server. Then a TCP connection is developed between the Client (Web Page) and Server. Then the HTTP request has been initiated from Client and Server respond to that request by sending the HTML file and after acknowledgement, the TCP connection is closed. Consider a person is requesting a

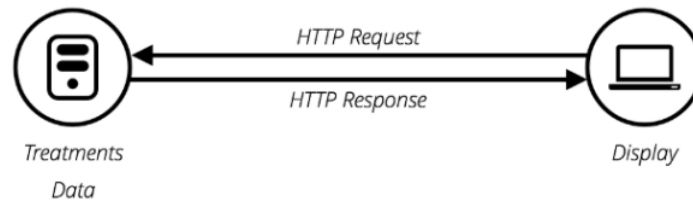


Figure 1: HTTP Request

URL to frequently such as the Home Page of Browser, so this request made again and again which add traffic to the network.

## 1.2 Proposed Solution

So if we add a Server in between the Client and Web Server which stores the frequently requested file and sends the file back to Client and if the file is not found at the intermediary server it initiates the request to get the Web File and saves the frequent file after retrieval and send back to Client. This reduces the Network traffic and also retrieval time is significantly is reduced.

The above-mentioned intermediate Server is known as Proxy Server.

## 1.3 Proxy Server

A proxy server is a dedicated computer or a software system running on a computer that acts as an intermediary between an endpoint device, such as a computer, and another server from which a user or client is requesting a service. The proxy server may exist in the same machine as a firewall server or it may be on a separate server, which forwards requests through the firewall.

### 1.3.1 Purpose of a Proxy Server

Proxy servers have two main purposes:

- Improve Performance
- Filter Requests

### 1.3.2 Improve Performance

Proxy servers can dramatically improve performance for groups of users. This is because it saves the results of all requests for a certain amount of time. Consider the case where both user X and user Y access the World Wide Web through a proxy server. First user X requests a certain Web page, which we'll call Page 1. Sometime later, user Y requests the same page. Instead of forwarding the request to the Web server where Page 1 resides, which can be a time-consuming operation, the proxy server simply returns the Page 1 that it already fetched for user X. Since the proxy server is often on the same network as the user, this is a much faster operation. Real proxy servers support hundreds or thousands of users.

### 1.3.3 Filter Requests

Proxy servers can also be used to filter requests. For example, a company might use a proxy server to prevent its employees from accessing a specific set of Web sites.

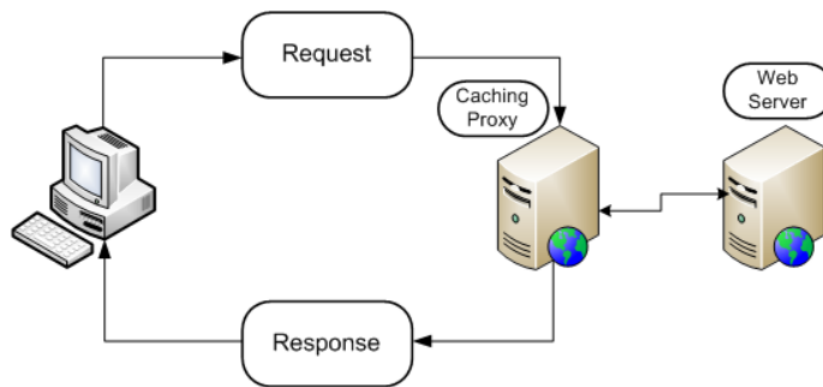


Figure 2: Proxy Server

We just focus on Improving the Performance by using Web Caches.

## 1.4 Web Cache

A Web proxy cache is a type of cache that stores and delivers frequently accessed websites, images and/or objects on the Internet. It is designed to help in delivering Internet-based data and objects more quickly to end users and also to free up bandwidth. Also, A web cache is an information technology for the temporary storage (caching) of Web documents, such as web pages, images, and other types of Web multimedia, to reduce server lag. A web cache system stores copies of documents passing through it; subsequent requests may be satisfied from the cache if certain conditions are met.

A Web proxy cache is also known as a Proxy Cache or HTTP Cache.

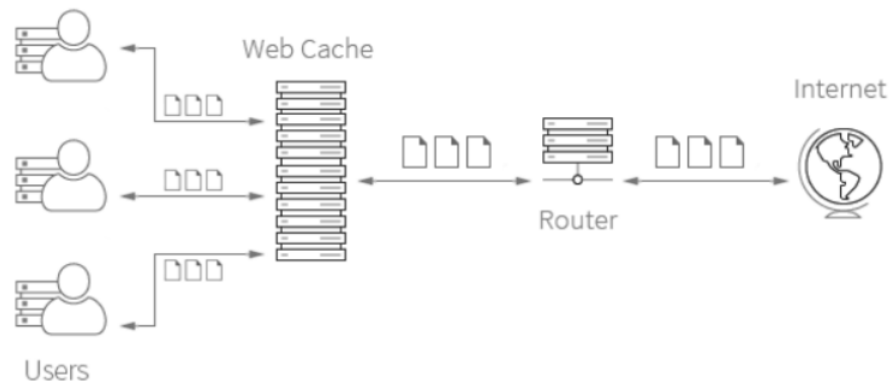


Figure 3: Web Cache

This project analyzes various proxy cache replacement algorithms including LFU, LRU and MRU.

## 1.5 Project Objective

The objective of this project is to design and implementation of a server-side proxy caching algorithm which can be used to reduce the network traffic and bandwidth. storage (caching) of web documents i.e. HTML pages, to reduce bandwidth usage, server load, and perceived lag. A web cache stores copies of documents passing through it; subsequent requests may be satisfied from the cache if certain conditions are met.

## 2 Methodology

### 2.1 Fetching HTML File from URL

The HTML file was fetched from the Server using “java.net.HttpURLConnection” library. First URL was given to the function, then a “GET” request was initiated and User Agent was mentioned. The response was received in DataInputStream. Following code-snippet was used to complete the task.



```
body = new ArrayList<>();
String USER_AGENT="Mozilla/5.0";
try {
    URL obj=new URL(this.url);
    HttpURLConnection con=(HttpURLConnection) obj.openConnection();
    con.setRequestMethod("GET");
    con.setRequestProperty("USER-AGENT",USER_AGENT);
    int responsecode=con.getResponseCode();
    try (BufferedReader in1 = new BufferedReader(
        new InputStreamReader(con.getInputStream(),"UTF-8"))) {
        String inputline;
        while((inputline=in1.readLine())!=null)
            body.add(inputline);
    }

} catch (MalformedURLException e) {
    this.body.add("MalformedURLException: " + e.getMessage());
} catch (UnknownHostException e) {
    this.body.add("UnknownHostException: " + e.getMessage());
} catch (IOException ex) {
    this.body.add("IOException: " + ex.getMessage());
}
```

Figure 4: Fetching HTML File from URL

The above code can give three kinds of Exceptions:

1. MalformedURLException:

This exception is thrown when a built-in URL class encounters the invalid URL. This exception is useful when a user mistakenly inputs the invalid URL. This exception will make our debugging fast and will let us know if the URLs which are entered are valid or not.

2. UnknownHostException:

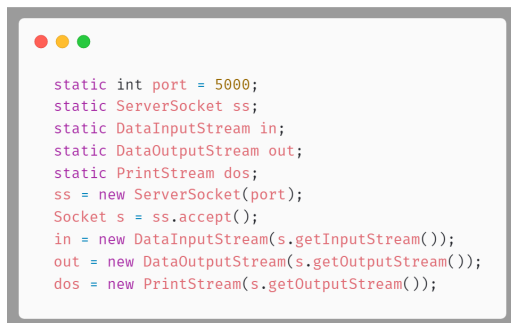
This exception is thrown in a variety of scenarios in which a remote connection is failed due to invalid or unknown hosts which includes IP, URL, URI etc. This exception is important for the TCP connection because to successfully communicate with the server we need a valid and known host,

### 3. IOException:

This exception has occurred when IO operations failed for some reason. It is also a checked exception which means that your program has to handle. This exception is thrown explicitly by any class which uses IO exception.

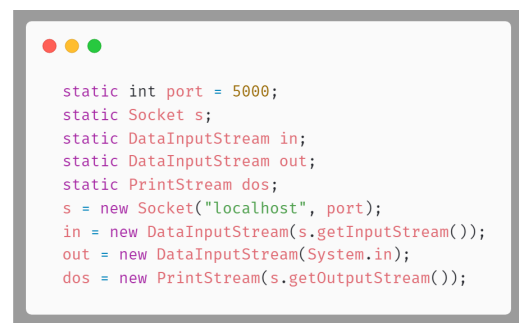
## 2.2 TCP Connection

We have used TCP connection instead of UDP because we need a reliable and successful communication between client and server. As we are saving the new responses in the cache we need to make sure that no information is lost and or else every time the same URL is requested will be responded with the same defected response. Also by making a TCP connection, we can easily send the HTML file as the reply in the connection of URL one. We used JAVA Socket and ServerSocket to make the connections.



```
static int port = 5000;
static ServerSocket ss;
static DataInputStream in;
static DataOutputStream out;
static PrintStream dos;
ss = new ServerSocket(port);
Socket s = ss.accept();
in = new DataInputStream(s.getInputStream());
out = new DataOutputStream(s.getOutputStream());
dos = new PrintStream(s.getOutputStream());
```

Figure 5: Server End



```
static int port = 5000;
static Socket s;
static DataInputStream in;
static DataInputStream out;
static PrintStream dos;
s = new Socket("localhost", port);
in = new DataInputStream(s.getInputStream());
out = new DataInputStream(System.in);
dos = new PrintStream(s.getOutputStream());
```

Figure 6: Client End

## 2.3 JAVA SWING Component

We have used a swing component in JAVA for GUI to give all this code a presentable frontend. The project includes a window for the client and one for the proxy server.

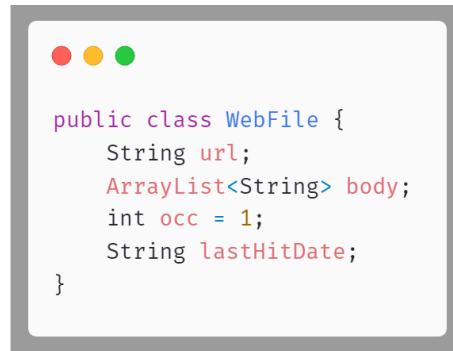
Client window is to enter a URL and get the look of the response client will be getting in the text area below.

The proxy server window will get all the responses client will be getting on the requests, and will be showing all those responses which caches will be holding. This window will also tell us the cache hit rate and cache miss rate.

## 2.4 Basic Structure of Proxy Server

There is a cache of a web file (Queue, Stack, Deque, List and etc.) and a Hash Map which is used to map the URL to the index in the cache. We first check that the map contains the URL if Yes then we modify the Web File, increase Cache Hit Rate and show the file. If the file does not exist then we check that Cache has space, in case of no space we remove an entry (if needed) and a new entry is fetched and Cache Miss Rate is increased.





```
public class WebFile {  
    String url;  
    ArrayList<String> body;  
    int occ = 1;  
    String lastHitDate;  
}
```

Figure 7: WebFile Class



```
LinkedList<WebFile> cache = new LinkedList<>(); // depends upon strategy  
Map<String,Integer> map = new HashMap<>();  
  
public WebFile add(String URL){  
    if (map.containsKey(URL)){  
        Stack<WebFile> tempStack = new Stack<>();  
        if (map.get(URL) != 0)  
            for (int i = 0; i < map.get(URL) - 1; i++)  
                tempStack.add(cache.remove()); // depends upon strategy  
        WebFile current = cache.remove(); // depends upon strategy  
        while (!tempStack.isEmpty())  
            cache.addFirst(tempStack.pop());  
        current.occ++;  
        DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss",Locale.US);  
        current.lastHitDate = dateFormat.format(new Date());  
        cache.addLast(current);  
        map.put(URL, cache.size()-1);  
        cacheHit++;  
        return current;  
    }  
    if (cache.size() == cacheCapacity)  
        map.remove(cache.remove().url); // depends upon strategy  
    WebFile newFile = new WebFile(URL);  
    if (!newFile.body.handleException()){  
        cache.add(new WebFile(URL)); // depends upon strategy  
        map.put(URL,cache.size()-1);  
        cacheMiss++;  
    }  
    return newFile;  
}
```

Figure 8: Basic Structure

### 3 Different Strategies to manage cache

#### 3.1 FIFO

FIFO is the strategy what usually queue uses, the idea is the same but implemented on caches. As soon as cache is out of space. The FIFO strategy will remove the first response which was saved in the cache and add the new response in place of this cache. FIFO cache hit and miss are totally dependant on the user trend as it is simple policy and might remove the cache which is to come next. This solution is not optimal but as the removal is to be done on top management is much easier and faster.

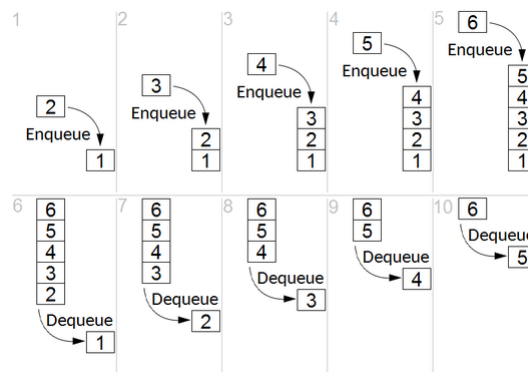


Figure 9: FIFO Logic

```
Queue<WebFile> cache = new LinkedList<>();
Map <String,Integer> map = new HashMap<>();

public WebFile add(String URL){
    if (map.containsKey(URL)){
        Stack<WebFile> tempStack = new Stack<>();
        if (map.get(URL) != 0)
            for (int i = 0; i < map.get(URL) - 1; i++)
                tempStack.add(cache.remove());
        WebFile current = cache.remove();
        while (!tempStack.isEmpty())
            cache.add(tempStack.pop());
        current.occ++;
        DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss",Locale.US);
        current.lastHitDate = dateFormat.format(new Date());
        cache.add(current);
        map.put(URL, cache.size()-1);
        cacheHit++;
        return current;
    }
    if (cache.size() == cacheCapacity)
        map.remove(cache.remove().url);
    WebFile newFile = new WebFile(URL);
    if (!newFile.body.handleException()){
        cache.addLast(new WebFile(URL));
        map.put(URL, cache.size()-1);
        cacheMiss++;
    }
    return newFile;
}
```

Figure 10: FIFO Code Snippet

### 3.2 LIFO

LIFO is last in first out this concept is used in stack usually this exactly the same concept but implemented on the cache. The last saved response is the first to be removed as soon as the cache is out of space. LIFO does not need to reassemble the responses as the pushing and popping are from the fixed place. Removing is much faster and easier as the deletion is from the fixed point but this strategy is not the best strategy in most of the scenarios.

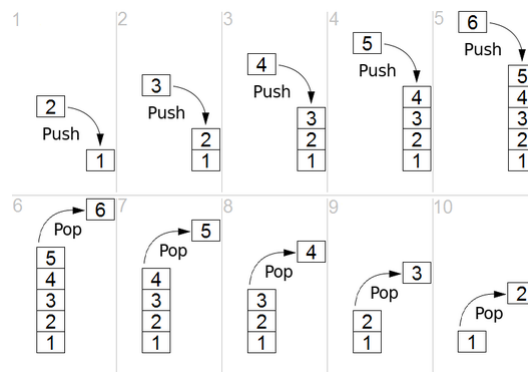


Figure 11: LIFO Logic

```
Stack<WebFile> cache = new Stack<>();
Map<String,Integer> map = new HashMap<>();

public WebFile add(String URL){
    if (map.containsKey(URL)){
        Stack<WebFile> tempStack = new Stack<>();
        if (map.get(URL) != 0)
            for (int i = 0; i < map.get(URL) - 1; i++)
                tempStack.add(cache.pop());
        WebFile current = cache.pop();
        while (!tempStack.isEmpty())
            cache.add(tempStack.pop());
        current.occ++;
        DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss",Locale.US);
        current.lastHitDate = dateFormat.format(new Date());
        cache.add(current);
        map.put(URL, cache.size()-1);
        cacheHit++;
        return current;
    }
    if (cache.size() == cacheCapacity)
        map.remove(cache.pop().url);
    WebFile newFile = new WebFile(URL);
    if (!newFile.body.handleException()){
        cache.addLast(new WebFile(URL));
        map.put(URL,cache.size()-1);
        cacheMiss++;
        return newFile;
    }
}
```

Figure 12: LIFO Code Snippet

### 3.3 LRU

Least Recently Used is another strategy for Web Cache. In this strategy, the proxy server has to keep check and balance on each response and count the occurrence of these responses. The removal of response when the cache is full is on the basis of least used response new response will be replaced with the least used response. After the removal and insertion of a new entry proxy server have to reassemble the cache. This strategy seems to be the best one as the least used is response is deleted and frequently used response is still in the cache.

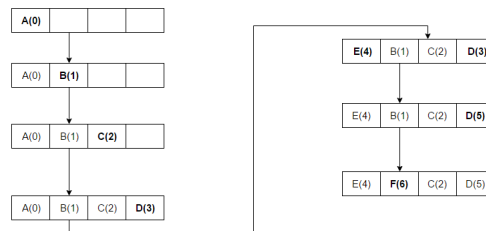


Figure 13: LRU Logic

```

Deque<WebFile> cache = new LinkedList<>();
Map <String,Integer> map = new HashMap<>();

public WebFile add(String URL){
    if (map.containsKey(URL)){
        Stack<WebFile> tempStack = new Stack<>();
        if (map.get(URL) != 0)
            for (int i = 0; i < map.get(URL) - 1; i++)
                tempStack.add(cache.removeFirst());
        WebFile current = cache.removeFirst();
        while (!tempStack.isEmpty())
            cache.addFirst(tempStack.pop());
        current.occ++;
        DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss",Locale.US);
        current.lastHitDate = dateFormat.format(new Date());
        cache.addLast(current);
        map.put(URL, cache.size()-1);
        cacheHit++;
        return current;
    }
    if (cache.size() == cacheCapacity)
        map.remove(cache.removeFirst().url);
    WebFile newFile = new WebFile(URL);
    if (!newFile.body.handleException()){
        cache.addLast(new WebFile(URL));
        map.put(URL,cache.size()-1);
        cacheMiss++;
    }
    return newFile;
}
  
```

Figure 14: LRU Code Snippet

### 3.4 MRU

Most Recently used is vice-versa of LRU. A proxy server has to increment the particular responses and as soon as the cache gets full it. replaces the new response with the most recently used. The cache has to maintain itself with the most used order. This strategy seems to be the worst strategy as it removes the requests which are made the most, and it will contribute the least in reducing traffic.

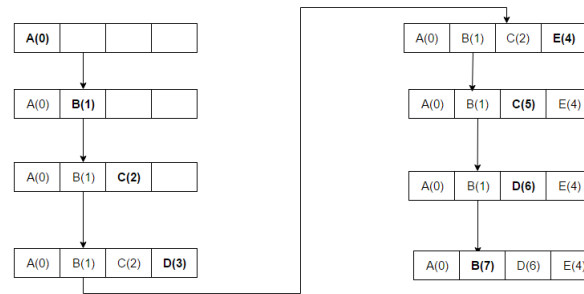


Figure 15: MRU Logic

```
Deque<WebFile> cache = new LinkedList<>();
Map <String,Integer> map = new HashMap<>();

public WebFile add(String URL){
    if (map.containsKey(URL)){
        Stack<WebFile> tempStack = new Stack<>();
        if (map.get(URL) != 0)
            for (int i = 0; i < map.get(URL) - 1; i++)
                tempStack.add(cache.removeLast());
        WebFile current = cache.removeLast();
        while (!tempStack.isEmpty())
            cache.addFirst(tempStack.pop());
        current.occ++;
        DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss", Locale.US);
        current.lastHitDate = dateFormat.format(new Date());
        cache.addLast(current);
        map.put(URL, cache.size()-1);
        cacheHit++;
        return current;
    }
    if (cache.size() == cacheCapacity)
        map.remove(cache.removeLast().url);
    WebFile newFile = new WebFile(URL);
    if (!newFile.body.handleException()){
        cache.addLast(new WebFile(URL));
        map.put(URL, cache.size()-1);
        cacheMiss++;
    }
    return newFile;
}
```

Figure 16: MRU Code Snippet

### 3.5 Random

In this, strategy proxy server will have a random function which will randomly choose from the cache table when the cache is full and remove it to make space for the new response. This strategy saves memory and time as it doesn't have to keep check and balance on each response and count the number of its occurrence. Randomly picking can be sometimes the best and sometimes least helpful we never know.

## 4 Results

We used following 20 website to analysis the result for the following two categories.

1. <https://www.google.com/>
2. <https://mail.yahoo.com/>
3. <https://www.youtube.com/>
4. <https://medium.com/>
5. <https://translate.google.com/>
6. <https://www.wikipedia.org/>
7. <https://whatis.techtarget.com/>
8. <https://www.webopedia.com/>
9. <https://www.cisco.com/>
10. <https://www.wireshark.org/>
11. <http://slate.nu.edu.pk/portal>
12. <http://nu.edu.pk/>
13. <http://flexstudent.nu.edu.pk/Login>
14. <http://acm.khi.nu.edu.pk/>
15. <http://khi.nu.edu.pk/>
16. <http://developersday.pk/>
17. <https://github.com/Rajani1998/>
18. <https://www.behance.net/>
19. <https://www.coursera.org/>
20. <https://www.edx.org/>

## 4.1 Fetching Web File with and without Cache

We categorized the above 20 websites into 5 classes and then calculated the loading time which is as following:

HTML FILE	Average Time Reqd to fetch with Cache	Average Time Reqd to fetch without Cache
Incorrect URL	0.93 sec	0.71 sec
Small web file without graphics	1.12 sec	0.71 sec
Small web file with graphics	1.45 sec	0.72 sec
Medium web file	1.73 sec	0.73 sec
Large web file	1.9 sec	0.75 sec
Large web file with high graphics	2.2 sec	0.78 sec

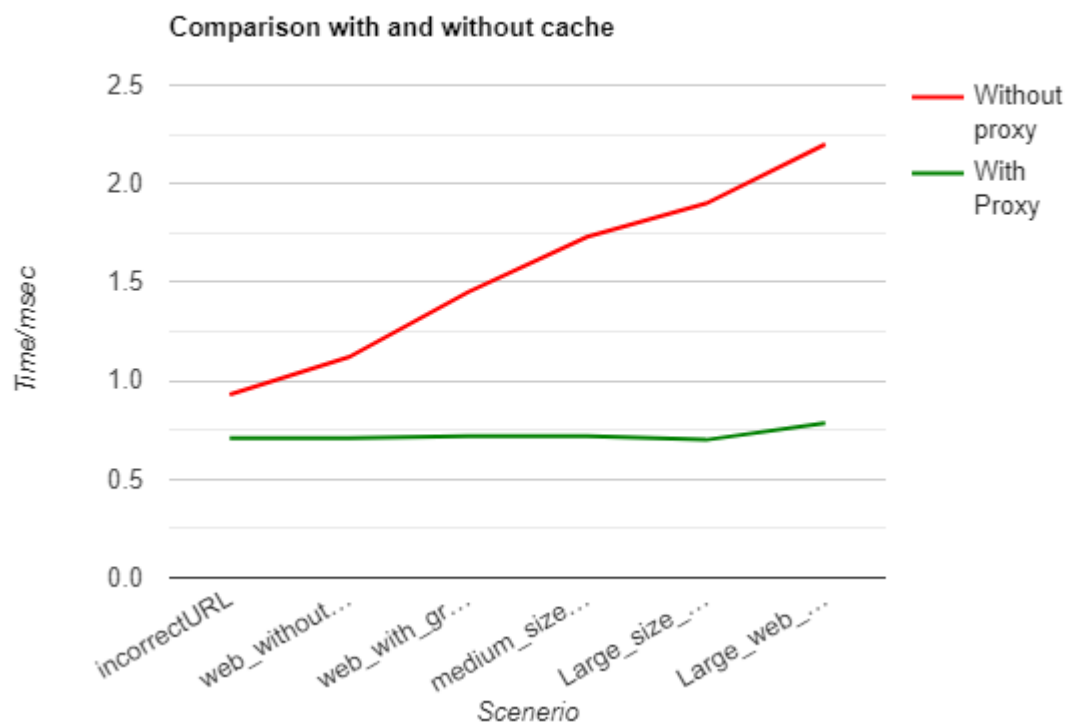


Figure 17: Average time with and without cache

## 4.2 Cache Hit Ratio with different Algorithms

We randomly entered the web pages from above mentioned data for 35 times and set the cache size to 5. Following are the Cache Hit and Cache Miss results after 35 iterations.



Algorithm	Cache Hit Rate	Cache Miss Rate
FIFO	0.54	0.46
LIFO	0.34	0.66
LRU	0.74	0.26
MRU	0.41	0.59
Random	0.58	0.42

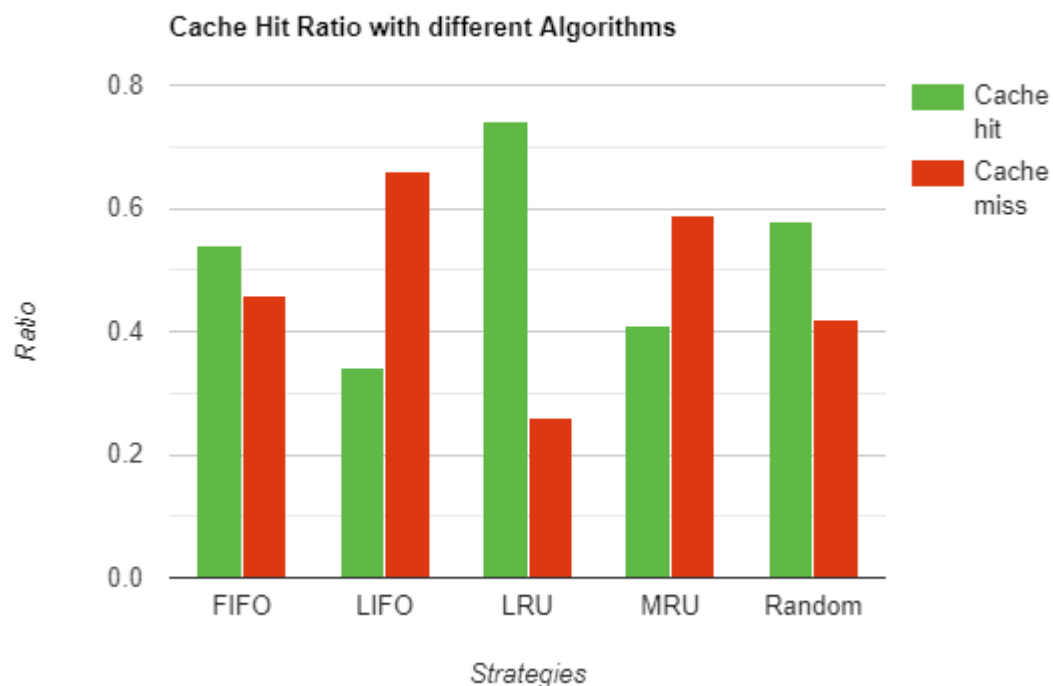


Figure 18: Cache Hit and Miss Analysis

From the above analysis, we concluded that

- FIFO is good when there is a stable and reoccurring surfing pattern because when a user request for a page it will remain in the cache until cache is not filled.
- LIFO and MRU are good for Hot Topics Searches only because they remain in demand for a short period of time and is not requested afterwards. The performance of LIFO is much better than MRU.
- LRU is the best fit for a normal search pattern and it is according to the temporal locality.
- Random gives a very diversified result, sometimes it performs better than LRU but sometimes it is worst then MRU because of it just guess the removing web file on the fly.

## 5 Conclusion

A web cache is one of the optimal solution for heavy traffic, server loads and latencies as requests don't have to route all over the server, again and again, it just has to see in cache table and look for the responses.

We have limited resources because of which we cannot cache every response and have to limit the size of cache too. Least recently use strategy for cache management is the best in most of the cases as the user requests that particular request least often.

We can also improve it by using multiple level cache in which the caches are a break into further levels. This reduces the miss penalty as it skips the ranges of responses which it might not belong. And have to look only in few responses.

In future, we can develop one more strategy by learning the trend of the user and predict the users request so we can remove other than that particular response. It will increase the cache hit and reduces miss hits. Another strategy could be of multi-level cache which can significantly affect the retrieval rate.