IMPLEMENTATION OF EFFICIENT LOAD BALANCING USING PROXY SERVERS AND RANDOMIZED ALGORITHM

A PROJECT REPORT

Submitted By

KANNAMAL.A.	**71202104016**
PRABHA PRIYA DHARSINI.A.	**71202104029**
SRIHARI.D.	**71202104063**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE OF ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY,
COIMBATORE

ANNA UNIVERSITY: CHENNAI 600025

APRIL 2006

BONAFIDE CERTIFICATE

ANNA UNIVERSITY: CHENNAI 600025

BONAFIDE CERTIFICATE

Certified that this project report "**IMPLEMENTATION OF EFFICIENT LOAD BALANCING USING PROXY SERVERS AND RANDOMIZED ALGORITHM**" is the bonafide work of "**A.KANNAMAL(71202104016), A.PRABHA PRIYADHARSINI (71202104029), D.SRIHARI(71202104063)**",who carried out the project work under my supervision.

SIGNATURE
Dr.S.Thangasamy
DEAN OF THE DEPARTMENT

Department of Computer
Science and Engg,
Kumaraguru College Of
Technology,
Chinnavedampatti P.O.,
Coimbatore-641006

SIGNATURE
Ms.P.Devaki
SUPERVISOR,
Assistant Professor
Department of Computer
Science and Engg,
Kumaraguru College of
Technology,
ChinnavedampattiP.O.,
Coimbatore-641006

Submitted for the Viva-voce Examination held on 2.5.06

Internal Examiner

External Examiner

EVALUATION CERTIFICATE

ANNA UNIVERSITY: CHENNAI 600 025

EVALUATION CERTIFICATE

College : **KUMARAGURU COLLEGE OF TECHNOLOGY**
Branch : **COMPUTER SCIENCE AND ENGINEERING**
Semester : **EIGHT(08)**

S.No	Name of the Student	Title of the Project	Name of Supervisor
1.	KANNAMAL.A	Implementation Of Efficient Load Balancing Using Proxy Servers and Randomized Algorithm	Mrs.P.Devaki, Asst Professor
2.	PRABHA PRIYADHARSINI.A		
3.	SRIHARI.D		

The report of the project work submitted by the above students in partial fulfillment for the award of **BACHELOR OF ENGINEERING** degree in **COMPUTER SCIENCE AND ENGINEERING** of **ANNA UNIVERSITY** were evaluated and confirmed to be the report of the work done by the above students and then evaluated.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION

DECLARATION

We hereby declare that the project entitled "**IMPLEMENTATION OF EFFICIENT LOAD BALANCING USING PROXY SERVERS AND RANDOMIZED ALGORITHM** ", is a record of original work done by us and to the best of our knowledge, a similar work has not been submitted to Anna university or any other institution, for fulfillment of the requirement of the course study.

This report is submitted in partial fulfillment of the requirements for the award of the Degree of Bachelor of Computer Science and Engineering of Anna University, Chennai.

Place: Coimbatore

Date:

(KANNAMAL.A)

A. Prabhapriadh

(PRABHA PRIYADHARSINI.A)

D. Srihari

(SRIHARI.D)

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

We would like to thank our principal, **Dr.K.K.Padmanabhan, B.Sc(Engg), M.Tech., Ph.D.,** for the facilities provided to carry out this project.

We are very much indebted to **Dr.S.Thangasamy, B.E. (Hons), and Ph.D.,** Dean of the Department of Computer Science And Engineering, for his support towards us and for his gracious permission to undertake this project.

We deem it a pleasure and privilege to feel indebted to our guide
Mrs. P. Devaki, M.S. Assistant Professor, who is always a constant source of inspiration and encouragement that helped us in the successful completion of the project.

We would like to thank all the **staff members** of Computer Science department and all those who have directly or indirectly assisted us in successfully completing this project.

We extend our special thanks to our **family and friends** for all their support and help rendered to us.

ABSTRACT

ABSTRACT

The enormous popularity of the World Wide Web in recent years has caused a tremendous increase in network traffic due to HTTP requests. Since the majority of web documents are static, caching them at various network points provides a natural way of reducing traffic. Hence we develop a proxy server using JAVA which caches the web pages and provide faster responses to the clients.

A key component of a cache is its replacement policy, which is a decision rule for evicting a page currently in the cache to make room for a new page. Here we propose a randomized algorithm for approximating any existing web cache replacement scheme.

CONTENTS

TABLE OF CONTENTS

INTRODUCTION

1. INTRODUCTION

1.1OVERVIEW

Our project provides an intelligent environment containing a number of ready-made options like cache, log file, error checking, connection pooling, etc. These ready-made tools may be any of the GUI components that are available in the Java AWT package. By using this utility, Administrator can control and maintain the whole network.

1.2 EXISTING SYSTEM AND ITS LIMITATIONS

The existing system uses processor cache to cache the web pages in the proxy server. Also the eviction policy used for this system is LRU (Least Recently Used) whose utility function assigns to each page a value which is the time since the page's last use. It worked well only for processor cache. The drawback in using processor cache is that there can be only few selected number of users, limited number of the web page access, if exceeds the slow retrieval timing. Also the processor cache needs multiple copies to provide responses to multiple clients.

Besides these, the major limitations of the existing policy (LRU) are as follows,

- Requires data structure to be implemented.
- Data structure requires a priority queue to be implemented.
- Data structure needs to be constantly updated even when there is no eviction

1.3 PROPOSED SYSTEM AND ITS ADVANTAGES

We use web cache to cache the web pages which has several advantages. A web cache is one which resides between the web server and the client and watches requests for web pages. The eviction policy we propose here is "Randomized Algorithm" which do not need any data structure to support the eviction decisions. The utility function of this algorithm takes into account not only the recency of use of a web document, but also its size, cost of fetching, and frequency of use which is expected to perform significantly better.

The advantages of this proposed system are:

- The usage of web cache reduces latency, traffic and also the load on web servers.
- The system helps to access internet from many systems using a single internet connection.
- Caching helps in reducing the bandwidth demands on a local network's connection to the Internet.
- Avoid the need for data structures
- Since JAVA is a platform independent language, the system can run in any platform
- Speed and security are also the additional features of JAVA

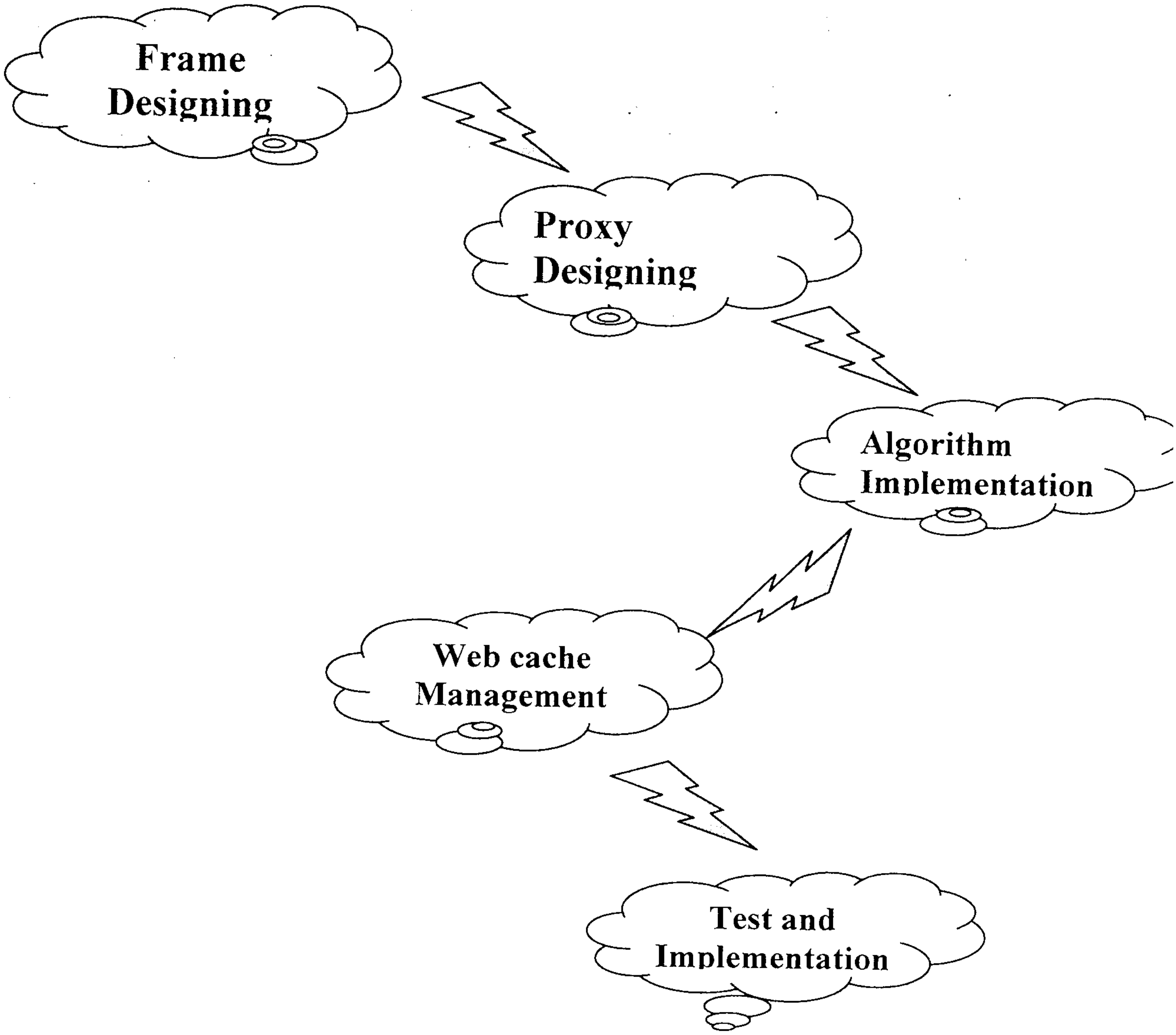
SYSTEM DESIGN

2. SYSTEM DESIGN

The system here opens the connection between the proxy server and various clients and provides quick responses to the clients. The focus is on the identification of the modules and how the modules should be interconnected.

2.1 OVERVIEW OF THE MODULES

The project is broadly viewed as composure of five modules:



1. Frame Designing

In frame designing, we design a form that contains various menus available for the administrator of proxy server to perform various actions. The various menus available are

- ***Admin***

This menu helps the administrator of proxy server to switch on the server (start server), switch off the server (stop server), empty the cache in proxy server (clear cache), empty the images present in the cache of proxy server (clear images).

- ***View***

View menu helps the administrator of proxy server to view various requests from the client and also the responses provided by them. The files present in the cache of proxy server can also be viewed including the images.

- ***Set up***(The proxy settings)

The configuration of the proxy server is set up and stored here. There are two tabs available namely networking and logging. Under networking, we can store the port number of the proxy server, etc. Under logging, the storage place of access log and error log files can be specified.

- ***Help and About*** (Help topics / About the proxy)

The details about the version of the proxy and the help topics are given.

2. Proxy Designing

The internal coding of the working of the proxy is carried out here. Here we assume that initially the cache of proxy server is empty. Also the TCP/IP connection between the proxy server and the clients is established.

The process is

- Initially, when there is a request for a web page from the client, the proxy server first connects to the World Wide Web. Then the requested page is downloaded from the web and client is responded. Also a copy of that web page is stored in the local cache of proxy server.
- Then the next request from the client is processed. We check whether the requested web page is present in the local cache. If present, the web page is checked for its updation and the updated web page is responded to the client and also the page in local cache is updated.
- If the web page requested by the client is not present in the local cache of the proxy server, then the page is downloaded from the web world. Thus the client's request is responded and a copy of that web page is put in proxy's local cache if there is enough space in it.
- If there is no enough space in the local cache, then the proxy server uses the randomized algorithm to evict a web page and create a room for the new page.

- The algorithm works as follows,
    - pick N documents at random from cache
    - evict the least useful document
    - retain the next M least useful document
    - pick N-M documents at random from cache
    - append to the M previously retained
    - evict the least useful document of N samples
    - retain the M next least useful documents

- The sample algorithm is,

```
If (eviction)
 {
            if(first_iteration)
             {
                    sample(N);
                    evict_least_useful;
                    keep_least_useful(M);
            }
             else
            {
                     sample(N-M);
                    evict_least_useful;
                    keep_least_useful(M);
            }
}
```

3. Algorithm Implementation

Here we implement the above said algorithm to perform the eviction function.

4. Web Cache Management

In the web cache management we are maintaining two folders one is the text storage folder and another one is to store the images.

5. Testing and implementation

The testing is done in the server with the clients of about 20. It runs successfully with the clients and it could be implemented for more number of clients as mentioned before.

FEATURES

3. FEATURES OF THE PROJECT

- Helps for the fast accessing of the internet by many systems using a single internet connection.
- Caching helps in reducing the bandwidth demands on a local network's connection to the Internet. The single internet connection and same bandwidth is shared between all the clients where this proxy is shared.
- Though there is a single internet connection for as many as 2000 system approx. The speed of downloading or uploading from the client from/ to the web world will be same for all the client systems.
- Load balancing is a special feature. Since the web page is cached in the local memory, the responses for number of requests are provided from the local cache. Hence the load is balanced and the server will not be down.
- We could setup the limit for the clients in terms of the web pages and the timing. (ie) For the web pages we could assign a specific number of pages like 3000 pages could only be used in the given span of 2 hours that is set for the clients.
- Another feature is that without disturbing the privacy of the client we could interfere in their request and we could have control over them even with out the firewall protection.
- The access of the specified web page of the client would even be denied from this server if the site is not suitable in the group.

PROGRAMMING ENVIRONMENT

4. PROGRAMMING ENVIRONMENT

The hardware and software configurations that were used to develop this system are mentioned below.

4.1 SOFTWARE REQUIREMENTS

The software required for this project is the JAVA programming language (jdk1.4) installed in the system.

4.2 HARDWARE REQUIREMENTS

The minimum requirements for the project:

CPU Type	:	Pentium II
Cache Memory	:	512 K
Co-Processor	:	Installed.
CPU Clock	:	300 MHz
Hard Disk	:	4.3 GB
RAM Memory	:	64 MB
Operating System	:	Windows 9x
Monitor	:	14" Samtron color monitor

DETAILED DESIGN

5. DETAILED DESIGN

Java makes it easy to connect to other computers, using classes from *java.net* package. We first write client programs that connect to the server, which is a program that performs a task useful to its clients, and then write our own servers. A web server provides files such as *web pages, java applets, and images.*

To write client programs, we first use java classes that enable us to hide the details of the interaction between client and server. We then try classes that let us customize the connection, and present classes that give us full control of the communication, but require us then to know the details of the request and response commands.

The simplest clients connect to a server using a URL object, or for more flexibility, a URLConnection talks to a server using a specific protocol. We introduce HTTP, the Hypertext Transfer Protocol used to connect to a web server. Understanding HTTP allows us to customize a connection using URLConnection methods, and to write our own simple web client and web server using *socket* and *ServerSocket* objects. This will enable us to develop and use our own protocols for clients and servers to communicate.

5.1 USING A URL TO CONNECT

Computers use protocols to communicate. A client sends requests using the commands by the protocol in the order specified in the protocol, and the server responds similarly. The URL class encapsulates several popular protocols, handling their details thereby making it easier for java programmers to make network connections to display a page, retrieve a file, or get mail for example.

5.2 CLIENT REQUEST AND SERVER RESPONSE

An HTTP client sends a **request** to the server in which the first line has the form

Method used *Identifier for the resource* *Protocol version*

A sample client request will be of the form

```
Get /~artg/TryURL.java HTTP/1.0
User-Agent: Java1.3.0
Host: www.cecs.csulb.edu:80
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

The description of the fields in the request are

FIELD	DESCRIPTION
User-agent	Indicates that Java 1.3.0 is running our client.
Host	Identifies the server.
Accept	Specifies the type of files that the client is prepared to accept. Each type has a preference associated with it given by the value of q(for "quality"). This value ranges from a low of 0 to the default of 1. The three types *text/html, image/gif, and image/jpeg* have the highest preference (the default of q=1 is not shown). If the server cannot send these types, then the client will accept any type, d4noted by *, or any subtype of any type, denoted by */*. These later generic types have preferences of q=2.
Connection	Specifies the types of connection. Here keep-alive expresses the client's wish to keep the connection alive for multiple requests.

An HTTP server responds to a responds to a request with a **status line** followed by various **response headers.** The status line has the form

HTTP version　　*Status Code*　　*Reason*

A sample HTTP server response is of the form

```
Status line: HTTP/ 1.0 200 Document follows
Response headers:
        Date: Mon,07Dec 1998 21:12:05 GMT
        Server: NCSA/1.4.2
        Content-type: text/plain
        Last-modified: Wed, 11 feb 1998 19:19:01 GMT
        Content-length: 439
```

The description of the fields in the response is,

FIELD	DESCRIPTION
Date	Gives the day and Greenwich Mean Time
Server	Names the web server used
Content-type	Describes the content. (Here text/plain for a Java program)
Content-length	Number of bytes in the file

ADVANTAGES

6. ADVANTAGES

The main advantages of the project are as follows

- Saves memory

  Here system memory is considerably saved as the already existing as already viewed pages are stored in a separate directory. On re-viewing of the same page by another client will consume the same memory where the data is stored first (ie) same text frame and images

- Saves processing power

  The processing supremacy over the client request is reduced as if the same page is viewed again by any the client, the page is displayed from the local area where it is stored locally.

- Reduces network traffic

  The passage control of the web pages to/ from the clients are stored in the separate directory so as said above the traffic is reduced if the often repeated web pages are requested by the clients as the request is responded with the help the directory data retrieval.

- Reduces Latency time

  Latency time (System's processing time) is reduced by usage of the local storage directory.

APPLICATIONS

7. APPLICATIONS

The various applications for this proxy are

- ✓ Large networks
- ✓ Load Balancing

The large networks means a server with a proxy server with clients as many as 2000-4000. This developed proxy is capable of having control over the clients specified.

The load balancing is the corresponding effect of having specified number of web-pages to be viewed at the given period of time.

FUTURE ENHANCEMENTS

8. FUTURE ENHANCEMENTS

In this project we developed a single proxy server for the whole LAN. We can have one more proxy server that will start providing responses once the running proxy server fails.

Now we have developed a proxy server that acts as client only for web server. We can make it work as a sub-proxy for the proxy server that is already working.

CONCLUSION

9. CONCLUSION

We have developed a **PROXY SERVER**, which runs with mentioned features, which inherently helps speeder browsing of web pages with use of randomized algorithms.

This server is successfully implemented with a few numbers of clients but it could be implemented for more of them. As mentioned before it is more reliable, more advantageous than the existing one which uses the old Data structures concept. So we conclude that this system application is executable under any platform and with any number of clients too.

REFERENCES

10.REFERENCES

- Balaji Prabhakar and Konstantinos Psounis,"Efficient Randomized Web-Cache Replacement Schemes Using Samples From Past Eviction Time", IEEE/ACM transactions on networking, vol, 10 no.4, August 2002
- Balaji Prabhakar, D.Engler and Konstantinos Psounis "A Randomized Cache Replacement Scheme Approximating URL" in proc.34th Annu,c onf International Sciences and Systems, Princeton, NJ, March 2000
- Art Gittleman, *Internet Programming with JAVA2 Platform,* 3rd ed, Tata Mcgraw-Hill, 2000
- Patrick Naughton and Herbert Schildt, *JAVA2- The Complete Reference,* 3rd Edition , Tata Mcgraw- Hill,1999
- www.google.com
- www.ieee.org

APPENDICES

11.APPENDICES

11.1 SAMPLE CODES

PROXY SERVER CREATION

```
import java.lang.*;
import java.net.*;
import java.io.*;
import java.util.*;
/** ProxyServer
 */
public class ProxyServer implements Runnable {
   String  server_ip      = null;     // ip this server binds to, (JDK 1.1
only)
   int     server_port    = 80;        // port this server listens on
   int     max_connections = 8;         // maximum connection allowed
   private Thread myThread;              // thread for this object
   private ServerSocket ss;            // server socket
   private boolean server_running = false;
   private boolean server_exit    = false;
   Object  console;
```

```
    final static String  localhost  = "127.0.0.1";
    final static boolean debug_mode = false;   // for debugging

   ProxyServer(Object p, String ip, int port) {
       console      = p;
       server_ip   = ip;
       server_port = port;
       myThread     = new Thread(this);
       myThread.setDaemon(true);
       myThread.start();
   }
   /**      constructor
    */
   ProxyServer(Object p, int port) {
      this(p, localhost, port);
   }
   /* test if the server is running
    */
   public boolean isServerRunning() {
      return server_running;
   }
   /* start server
    */
   public synchronized boolean startServer() {
      if( server_running ) return true; // check if server is running
already
      try {          if( server_ip == null )
```

```
        ss = new ServerSocket(server_port, max_connections);
      else
        ss = new ServerSocket(server_port, max_connections,
                    InetAddress.getByName(server_ip));
        ss.setSoTimeout(1000); // set timeout to 1 second
    }
    catch(IOException e) {
      ss = null;
      if(debug_mode) System.out.println("Error: Can't set up server
socket on " + server_ip + ":" + server_port + "\n" + e );
      return false;
    }
    System.out.println("Server Started on " + server_ip + ", port: "+
server_port);
    server_running = true;
    return true;
  }
  /**    stop server
   */
  public synchronized boolean stopServer() {
    server_running = false;
    try {
      ss.close();
    }
    catch(IOException e) {
      if(debug_mode) System.out.println("Can't close server
socket");
```

```
            }
            ss = null;
            if(debug_mode) System.out.println("Server socket closed.");
            return true;
        }


        /**  set server_exit flag
         */
        public void serverExit() {
            server_exit = true;
        }
       /* run
         */
        public void run() {
            while( !server_exit ) {
                if( server_running ) {
                    try {
                        Socket s = ss.accept();
                        new ProxyConnection( console, this, s );
                        ((ServerInterface) console).updateHTTPCounter();
                    }
                    catch(IOException e) { }
                }
                else {
                    try{ myThread.sleep(500); } catch (InterruptedException e)
{ } ;
                } // end of if( server_running )
```

```
      } // end of while( true )
    } // end of void run()
  } // end of ProxyServer
```

PROXY CONNECTION

```
import java.net.*;
import java.io.*;
import java.util.*;


/** ProxyConnection handles connection from client and opens a
    new connection to the remote server
*/
  ProxyConnection(Object c, Object p,  Socket s ) {
      console = c;
      parent  = p;
      sock_c  = s;
      setPriority( NORM_PRIORITY + 1 );
      httpconfig = ((ServerInterface)console).getHTTPConfiguration();
      cachepool  = ((ServerInterface)console).getProxyCachePool();
      using_cache = cachepool.isCacheEnabled();
      _RECV_TIMEOUT =
Integer.parseInt(httpconfig.getProperty("network.receive_timeout")) *
1000;
   proxy_addr  = (String) httpconfig.getProperty("network.proxy.ip");
      if(proxy_addr != null) proxy_addr = proxy_addr.trim();
```

```
        try {          proxy_port  = Integer.parseInt((String)
 httpconfig.getProperty("network.proxy.port"));
        }
        catch(NumberFormatException ne) {
          proxy_port = 0;
        }
 String showcontent_str = (String)
 httpconfig.getProperty("network.showcontent");
  If(showcontent_str != null) show_content =
 showcontent_str.equals("true") ? true : false;
  if(proxy_addr != null && !proxy_addr.equals("") && proxy_port >=
 0) using_proxy = true;
  try {
        sock_c.setSoTimeout(_RECV_TIMEOUT); // set socket time out
          start();
        }
        catch(SocketException se) {
        System.out.println(se);
        }
        catch(OutOfMemoryError oe) {
          Runtime r = Runtime.getRuntime();
          System.out.println("Out of Memory!!\nFree memory is: "+
r.freeMemory());
          System.out.println("Total memory is: "+ r.totalMemory());
          System.gc();
          try { Thread.sleep(5000L); } catch(InterruptedException ie) {
}
```

```
    }
  }
  private boolean closeSock(Socket s) {
    try { s.close(); } catch(IOException err) { return false; }
    return true;
  }



/* run
 */
 public void run() {
    StringBuffer sb = new StringBuffer();
    boolean keep_alive  = true;
    boolean read_finish = false;
    String buf = "";
    String key, value;
    URL url    = null;
    InputStream   in_c  = null; // input from client socket
    InputStream   in_s  = null; // input from server socket
    OutputStream  out_c = null; // output to client socket
    OutputStream  out_s = null; // output to server socket
    try {
      in_c  = sock_c.getInputStream();
      out_c = sock_c.getOutputStream();
    } catch (IOException ie) {
           ((ServerInterface) console).logError(tracer.getSource(),
 ie.toString());
           closeSock(sock_c);
           return;
        }



       /*  Read Client Request From Socket InputStream in_c
        */
       byte onebyte[] = new byte[1];
       boolean EOH = false; // flag End Of Header
       try {
          while(!EOH) {
             if(in_c.read(onebyte) <= 0) {
                EOH = true;
                continue;
             }
             else
                sb.append(new String(onebyte, "8859_1"));
                     if(sb.toString().endsWith(_CRLF2) ||
        sb.toString().endsWith(_LF2)) {
                EOH = true;
             }
          }
      } catch (IOException ie) {
         ((ServerInterface) console).logError(tracer.getSource(),
ie.toString());
```

```
            closeSock(sock_c);
            return;
        }
        c_header = new HTTPRequestHeader(sb.toString());
        String server_adpt = ""; // address and port
        String server_addr = "";
        String doc       = "/";
        int   server_port = 80;
        int   loc = 0;
        if(c_header.URI == null || !c_header.URI.startsWith("http://")) {
            closeSock(sock_c);
            return;
        }
        ((ServerInterface)
console).logAccess(_LOG_LEVEL_NORMAL,  tracer.getSource(),
c_header.getPrimeHeader());
        ((ServerInterface)
console).logAccess(_LOG_LEVEL_MINIMAL, tracer.getSource(),
sb.toString());
        if((loc = c_header.URI.indexOf("/", 8)) <= 0) {
            server_adpt = c_header.URI.substring(7);
        }
        else {
            server_adpt = c_header.URI.substring(7, loc);
            doc = c_header.URI.substring(loc);
        }
        if((loc = server_adpt.indexOf(":")) <= 0) {
            server_addr = server_adpt;
            server_port = 80;
        }
        else {
            server_addr = server_adpt.substring(0, loc);
            server_port = Integer.parseInt(server_adpt.substring(loc+1));
        }

        /**          Check if request is in cache(for GET only)
         **/
        if(using_cache && c_header.Method.equals("GET")) {
            pc = cachepool.getCache(c_header.URI);
            if(pc == null) {
                // Not in Cache
            }
            else {
                // get header and content from cache
                // then send it back to client
                byte  header[]  = pc.getHeader();
                byte  content[] = pc.getContent();
                try {
                    out_c.write(header); //, 0, header.length);
                    //System.out.println("Using Cache " + c_header.URI);
                    //System.out.println("Header:["+ new String(header)+"] "
+ content.length);
                    if(content != null) {
                        out_c.write(content); //, 0, content.length);
                }                  out_c.flush();
              }
              catch(IOException e) { System.out.println(e.toString()); }
              closeSock(sock_c);  // close connection to client
              return;
          }
      }


      /*          Open a socket connection to remote server
       */
      byte resp[] = new byte[2048];
      int  content_len = 0;
      int  total_len   = 0;
      int  len = 0;
      content_len = c_header.getContentLength(); // get content length
from client request
      EOH = false; // flag End Of Header
      try {
         if(using_proxy) { // connect to another proxy server if using
proxy
            sock_s = new Socket(proxy_addr, proxy_port);
            sock_s.setSoTimeout(_RECV_TIMEOUT); // set socket
time out
            in_s = sock_s.getInputStream();
            out_s = sock_s.getOutputStream();
         }
         else { // connect directly to remote web server
```

```
            // open a socket to connect to server
            sock_s = new Socket(server_addr, server_port);
            sock_s.setSoTimeout(_RECV_TIMEOUT); // set socket
time out
            // System.out.println("connected to : " + server_addr + " „ "
+ server_port );
            in_s  = sock_s.getInputStream();
            out_s = sock_s.getOutputStream();
            // create the message to send to server
            sb = new StringBuffer();
            sb.append(c_header.Method).append("
").append(doc).append("
").append(c_header.Version).append(_CRLF);
            Hashtable ht = c_header.getHeaderFields();
            for (Enumeration enu = ht.keys(); enu.hasMoreElements() ;)
               {
               key   = (String) enu.nextElement();
               value = (String) ht.get(key);
               if(key.equalsIgnoreCase("Proxy-Connection")) continue;
// bypass the proxy connection row
               sb.append(key).append(":
").append(value).append(_CRLF);
            }
            sb.append(_CRLF);
         }
         // System.out.println("sending : \n" + sb.toString());
         // write header message to server
```

```
        out_s.write(sb.toString().getBytes(), 0, sb.toString().length());
        out_s.flush();
        // write content body to server if there is any
        if(content_len > 0) {
           while(content_len > total_len) {
              if((len = in_c.read(resp)) <= 0) break;
              total_len += len;

              // determine if need to show content
              if(show_content) {
                 if(byteArrayOperator.isPrintable(resp, len)) {
                    ((ServerInterface) console).showClientRequest(new
String(resp, 0, len));
                 }
              }
              out_s.write(resp, 0, len);
              out_s.flush();
           }
        }
        sb = new StringBuffer();
        /**
        Read Server Response Header
        **/
        while(!EOH) {
           if(in_s.read(onebyte) < 0) {
              EOH = true;
              continue;
```

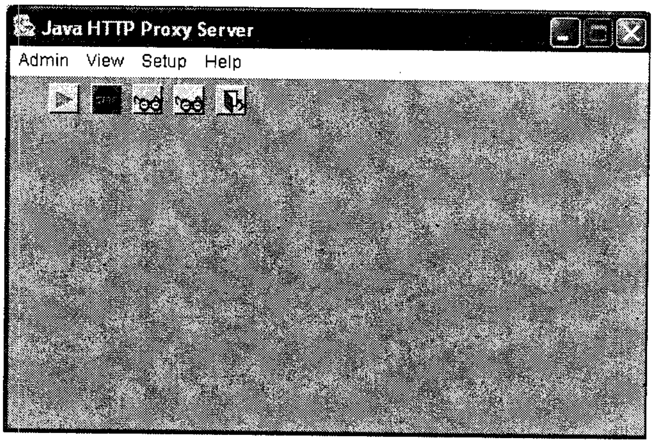
```
                    }           else
                        sb.append(new String(onebyte, "8859_1"));
                    if(sb.toString().endsWith(_CRLF2) ||
sb.toString().endsWith(_LF2)) {
                        EOH = true;
                    }
                }
                // send server response header
                out_c.write(sb.toString().getBytes(), 0, sb.toString().length());
                out_c.flush();
                // read server response body
                s_header = new HTTPResponseHeader(sb.toString());
                // logging
                ((ServerInterface)
console).logAccess(_LOG_LEVEL_NORMAL,  tracer.getSource(),
s_header.getPrimeHeader());
                ((ServerInterface)
console).logAccess(_LOG_LEVEL_MINIMAL, tracer.getSource(),
sb.toString());
                content_len = 0;
                total_len   = 0;
                len = 0;
            /*          Read Server Response Content
                */
                content_len = s_header.getContentLength();
                byte resp_content[] = null;
                if(content_len > 0) {
```

```
                resp_content    = new byte[content_len];
                int last_byte = 0;
                while(content_len > total_len) {
                    if((len = in_s.read(resp)) <= 0) break;
                    if(using_cache || show_content) {
byteArrayOperator.copy(resp_content, total_len, resp, len);
                    }
                    total_len += len;
                    out_c.write(resp, 0, len);
                    out_c.flush();
                }
            }
            else {  // if content length is not set
                StringBuffer resp_sb = new StringBuffer();
                while((len = in_s.read(resp)) >= 0) {
                    out_c.write(resp, 0, len);
                    out_c.flush();
                    if(using_cache || show_content) { // save content if using
cache
                        resp_sb.append(new String(resp, 0, len, "8859_2"));
                    }
                }
                if(using_cache || show_content) resp_content =
resp_sb.toString().getBytes("8859_2");
            }
            // determine if need to show content
            if(show_content) {
```

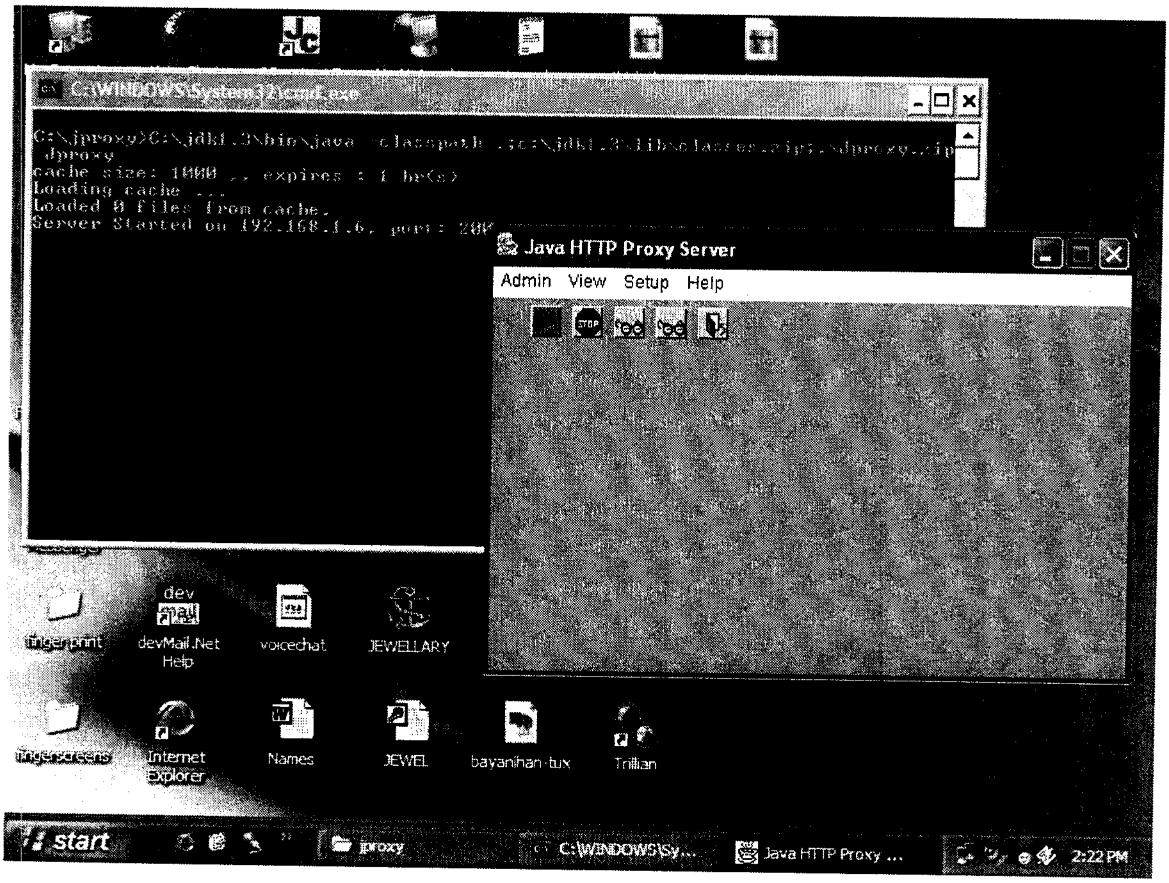
```
          if(byteArrayOperator.isPrintable(resp_content)) {
            ((ServerInterface) console).showServerResponse(new
String(resp_content));
          }}

    /*  Add Content to Cache
        */
        if(using_cache && c_header.Method.equals("GET")) {
          // add url, header, content to cachepool
          cachepool.setCache(c_header.URI, s_header.toString(),
resp_content);
        }
      }
      catch(SocketException se) {
        ((ServerInterface) console).logError("Error: SocketException,
", c_header.URI);
        ((ServerInterface) console).logError(tracer.getSource(),
se.toString());
        System.out.println(c_header.URI + " " + se);
      }
   catch(IOException ie) {
        ((ServerInterface) console).logError("Error: IOException, ",
c_header.URI);
        ((ServerInterface) console).logError(tracer.getSource(),
ie.toString());
        // ie.printStackTrace();
        System.out.println(c_header.URI + " " + ie);
        }       closeSock(sock_c);      // close connection to client
    } // end of run()
} // end of ProxyConnection
```

11.2 SCREEN SHOTS

SAMPLE FRAME

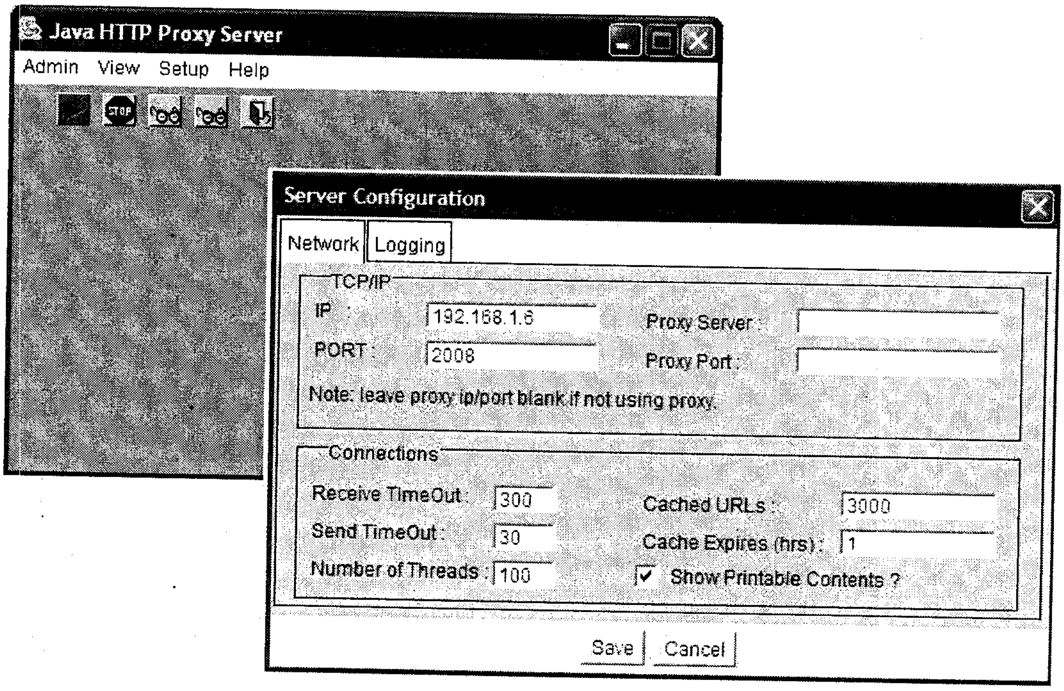


SERVER STARTED…..



```
C:\jproxy>C:\jdk1.3\bin\java -classpath .;c:\jdk1.3\lib\classes.zip;.\jproxy.zip
 Jproxy
cache size: 1000 .. expires : 1 hr(s)
Loading cache ...
Loaded 0 files from cache.
Server Started on 192.168.1.6, port: 200
```

SET UP..



IP : 192.168.1.6

PORT : 2008

Proxy Server :

Proxy Port :

Note: leave proxy ip/port blank if not using proxy.

Receive TimeOut : 300

Send TimeOut : 30

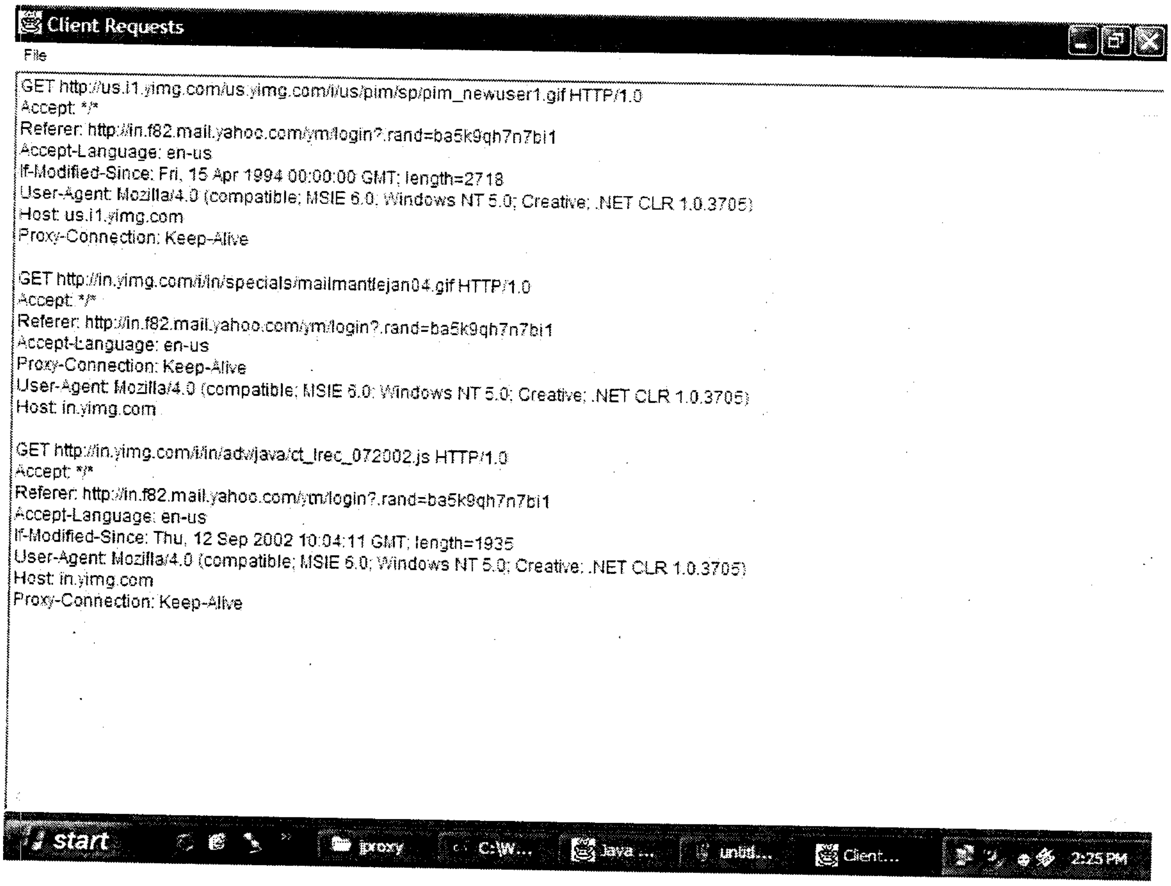
Number of Threads : 100

Cached URLs : 3000

Cache Expires (hrs) : 1

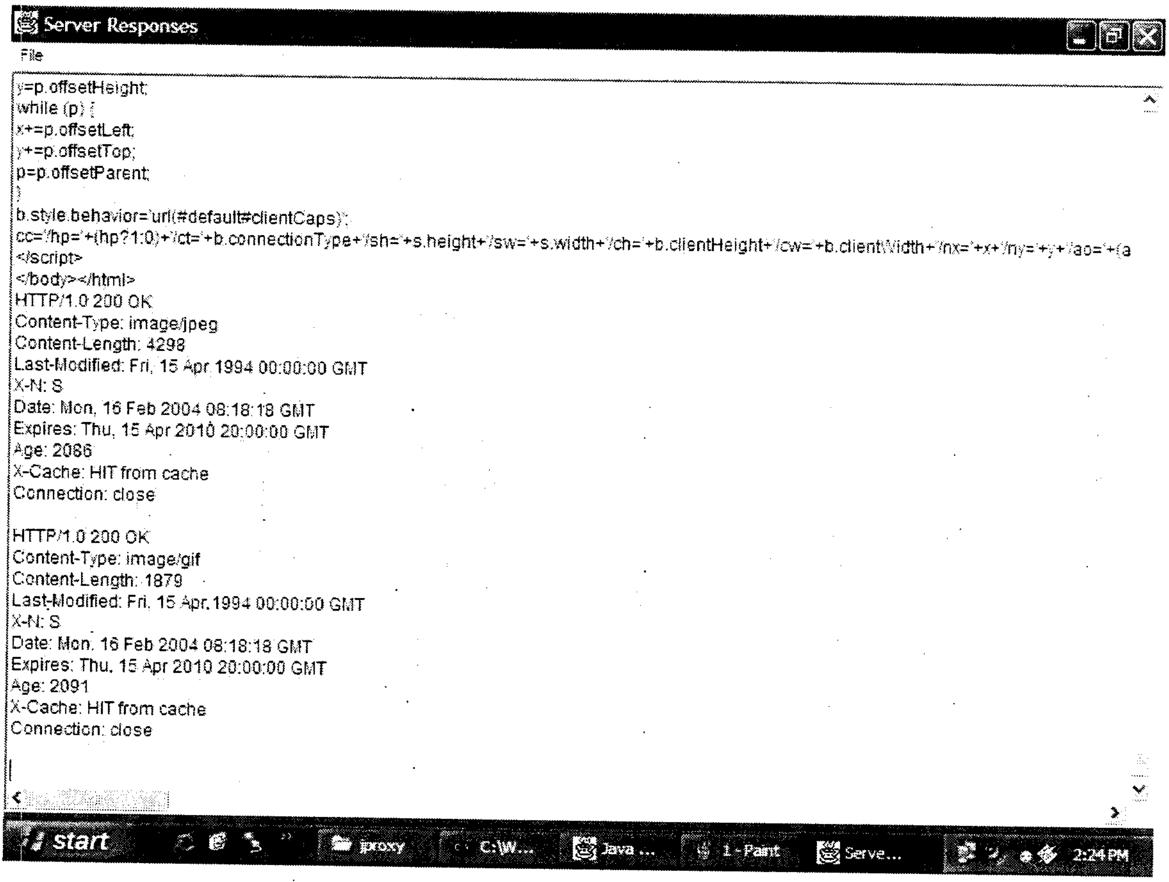
☑ Show Printable Contents ?

CLIENT REQUESTS..



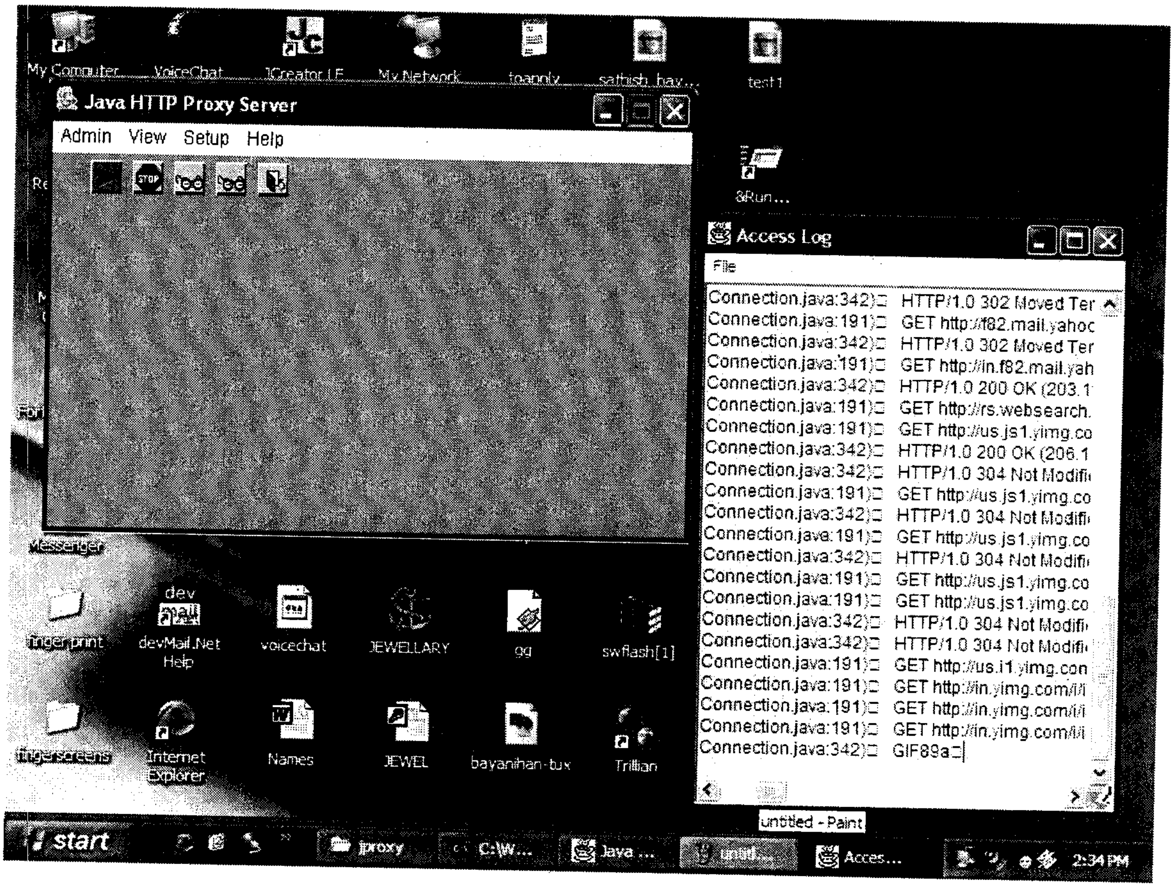
```
GET http://us.i1.yimg.com/us.yimg.com/i/us/pim/sp/pim_newuser1.gif HTTP/1.0
Accept: */*
Referer: http://in.f82.mail.yahoo.com/ym/login?.rand=ba5k9qh7n7bi1
Accept-Language: en-us
If-Modified-Since: Fri, 15 Apr 1994 00:00:00 GMT; length=2718
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; Creative; .NET CLR 1.0.3705)
Host: us.i1.yimg.com
Proxy-Connection: Keep-Alive

GET http://in.yimg.com/i/in/specials/mailmantlejan04.gif HTTP/1.0
Accept: */*
Referer: http://in.f82.mail.yahoo.com/ym/login?.rand=ba5k9qh7n7bi1
Accept-Language: en-us
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; Creative; .NET CLR 1.0.3705)
Host: in.yimg.com

GET http://in.yimg.com/i/in/adv/java/ct_lrec_072002.js HTTP/1.0
Accept: */*
Referer: http://in.f82.mail.yahoo.com/ym/login?.rand=ba5k9qh7n7bi1
Accept-Language: en-us
If-Modified-Since: Thu, 12 Sep 2002 10:04:11 GMT; length=1935
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; Creative; .NET CLR 1.0.3705)
Host: in.yimg.com
Proxy-Connection: Keep-Alive
```

SERVER RESPONSES…



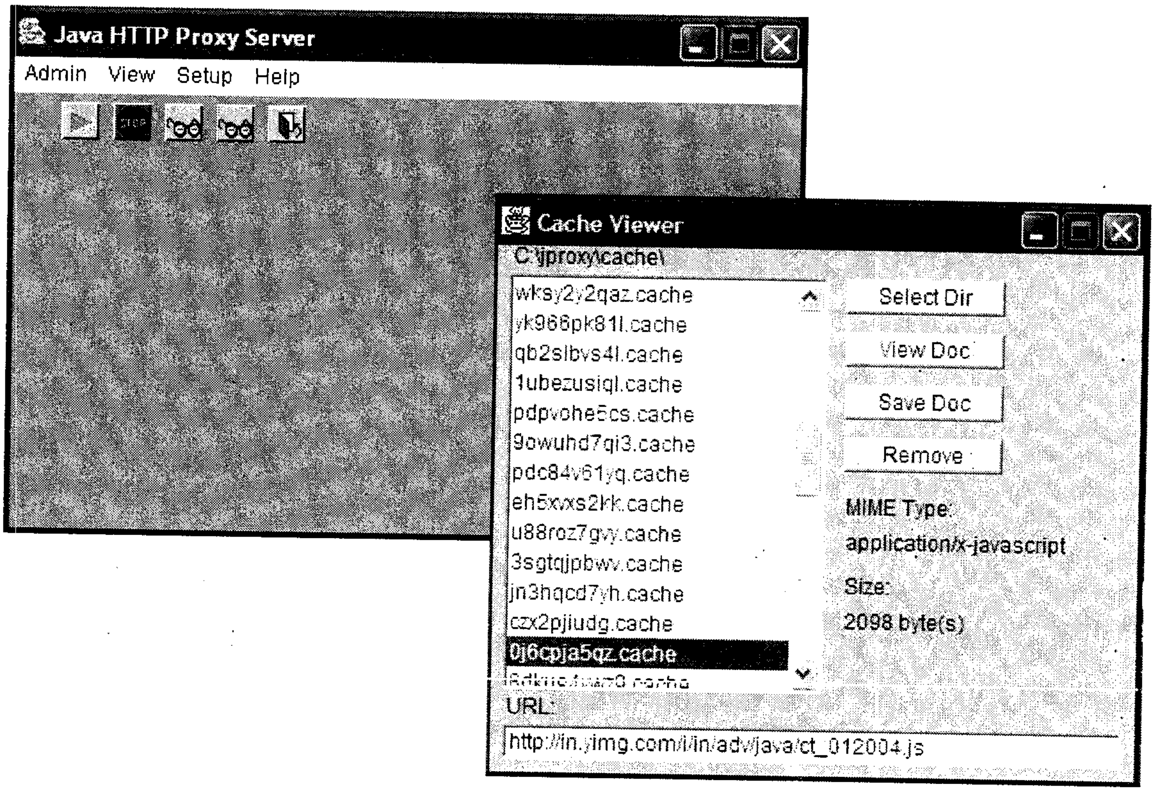
```
y=p.offsetHeight;
while (p) {
x+=p.offsetLeft;
y+=p.offsetTop;
p=p.offsetParent;
}
b.style.behavior='url(#default#clientCaps)';
cc='/hp='+(hp?1:0)+'/ct='+b.connectionType+'/sh='+s.height+'/sw='+s.width+'/ch='+b.clientHeight+'/cw='+b.clientWidth+'/nx='+x+'/ny='+y+'/ao='+(a
</script>
</body></html>
HTTP/1.0 200 OK
Content-Type: image/jpeg
Content-Length: 4298
Last-Modified: Fri, 15 Apr 1994 00:00:00 GMT
X-N: S
Date: Mon, 16 Feb 2004 08:18:18 GMT
Expires: Thu, 15 Apr 2010 20:00:00 GMT
Age: 2086
X-Cache: HIT from cache
Connection: close

HTTP/1.0 200 OK
Content-Type: image/gif
Content-Length: 1879
Last-Modified: Fri, 15 Apr 1994 00:00:00 GMT
X-N: S
Date: Mon, 16 Feb 2004 08:18:18 GMT
Expires: Thu, 15 Apr 2010 20:00:00 GMT
Age: 2091
X-Cache: HIT from cache
Connection: close
```

VIEW- ACCESS LOG…



```
Connection.java:342)□  HTTP/1.0 302 Moved Ter
Connection.java:191)□  GET http://f82.mail.yahoc
Connection.java:342)□  HTTP/1.0 302 Moved Ter
Connection.java:191)□  GET http://in.f82.mail.yah
Connection.java:342)□  HTTP/1.0 200 OK (203.1
Connection.java:191)□  GET http://rs.websearch.
Connection.java:191)□  GET http://us.js1.yimg.co
Connection.java:342)□  HTTP/1.0 200 OK (206.1
Connection.java:342)□  HTTP/1.0 304 Not Modifi
Connection.java:191)□  GET http://us.js1.yimg.co
Connection.java:342)□  HTTP/1.0 304 Not Modifi
Connection.java:191)□  GET http://us.js1.yimg.co
Connection.java:342)□  HTTP/1.0 304 Not Modifi
Connection.java:191)□  GET http://us.js1.yimg.co
Connection.java:191)□  GET http://us.js1.yimg.co
Connection.java:342)□  HTTP/1.0 304 Not Modifi
Connection.java:342)□  HTTP/1.0 304 Not Modifi
Connection.java:191)□  GET http://us.i1.yimg.con
Connection.java:191)□  GET http://in.yimg.com/i/i
Connection.java:191)□  GET http://in.yimg.com/i/i
Connection.java:191)□  GET http://in.yimg.com/i/i
Connection.java:342)□  GIF89a□
```

VIEW- WEB CACHE…



C:\jproxy\cache\

wksy2y2qaz.cache
yk966pk81l.cache
qb2slbvs4l.cache
1ubezusiql.cache
pdpvohe5cs.cache
9owuhd7qi3.cache
pdc84v61yq.cache
eh5xwxs2kk.cache
u88roz7gvy.cache
3sgtqjpbwv.cache
jn3hqcd7yh.cache
czx2pjiudg.cache
0j6cpja5qz.cache

MIME Type:
application/x-javascript

Size:
2098 byte(s)

URL:
http://in.yimg.com/i/in/adv/java/ct_012004.js