# AN EFFICIENT SYSTEM FOR COLLABORATIVE CACHING AND MEASURING DELAY

**[1]Mohd Husain, [2]Ayushi Prakash, [3]Ravi Kant Yadav**

*[1]Department of C.S.E, MGIMT, Lucknow, (India)*

*[2]Department of C.S.E, Dr. KNMIET, Ghaziabad, (India)*

*[3]S & A, BWMT, Ghaziabad, (India)*

## ABSTRACT

*The Collaborative caching and browsing have been proposed for www clients for reducing network traffic and improving access latency of the document, which allows sharing and coordination of cached data among clients, is a potential technique to improve the data access performance and availability. Collaborative caching of web documents at client end has been shown to be an effective technique for reducing web traffic and improving access latencies. It allows a group of clients to retrieve cached documents from within the group while collaborating through a user interface to access items indicative of group behavior. However, variable data sizes, frequent data updates, limited client resources, insufficient wireless bandwidth and client's mobility make cache management faces many challenges. An experimental result of the system is evaluated and presented in terms of latency in loading a sample set of web elements.*

***Keywords*: Browsing, Collaborative, Latency, Network Traffic, Proxy.**

## I INTRODUCTION

The enormous growth of WWW based services has made the network traffic a major concern over the last few years. It is common to encounter long delays in web document retrieval due to slow connections, network congestions, remote server overloading etc. Most WWW clients use memory and disk caches for speeding up accesses to frequently used web documents [1]. In collaborative caching schemes, read access to cached document is given to the cooperating group of users. Such integration creates an illusion of a single cache of much bigger size. Since all users of the group use the same integrated cache, it has an added advantage that the probability of the next accessed document being found in the cache is high. Collaborative caching schemes provide a better hit ratio than individual caching schemes. It allows the users to cache and surf collaboratively, and proof-of-concept modules were developed. In this work, we restate the design [1], include additions such as a design for the user interface framework, and present the implementation. For the implementation, the target group was a student user base with reasonable computing power and memory, running a variety of operating systems. The implementation aims to solve

issues commonly faced by the target, such as network latencies introduced onto the network due inadequate resources dedicated to any centralized caching proxy. As a typical situation where the System will be put into use, suppose a group of students need to research a topic on the Internet. If they could utilize each other's cache, minimizing retrievals from the Internet, while working together in their research in real time, they would have the ability to surf as a single entity. It allows these users to utilize their geographic proximity to make surfing efficient, and facilitates community behavior when they collaborate through the user interface. We introduce modules to allow clients to access caches created during surfing, and a collaboration framework which allows effective collaboration through a browser-based interface, with flexibilities to support different methods of collaboration.

It can be used to explore improvements in group research efficiencies when users are familiar others and know what other members are looking, while also being able to browse through those links with minimum delays.

## II BACKGROUND AND APPROACHES

Software and hardware which facilitate computer supported cooperative work allow users to perform tasks such that efforts supplement or complement to accomplish a common goal. Many approaches to the challenge of collaborative caching have been explored. Systems such as Squirrel [3], based on peer-to-peer overlay networks such as Pastry, CAN, Chord or Tapestry [4, 5, 6, 7] have been explored. Collaboration systems have utilized Distributed Hash Tables (DHTs) similar to those used for peer-2-peer file sharing networks. For example, Dermi[8] has been used to create a framework to build collaborative applications based on DHT peer-to-peer overlay networks[9]. DHT overlays, too, come in many flavors. For example, the Kademlia protocol compares node IDs of participating nodes to file hashes, iteratively traverses links till a source node is reached [10]. Others, such as Broose[11], have also been explored and utilize the De-Bruijn topology, as does Kademlia. These systems eliminate single points of failure by allowing the creation of distributed hash tables which are utilized to route requests as the network is traversed. Though this might allow for increases in reliability, a centralized query system may offer advantages with regards to latency, especially for our target, due to efficiencies offered by a direct look-up, and the fact that the target is of the order of small research groups. The approach used herein routes a request to the internet or one of its peers, as advised by a central query system. It utilizes standards-complaint caching methodologies to give a consistent and robust caching mechanism. HTTP 1.1 supports numerous caching features which allow elements to be stored and retrieved in a reliable and efficient manner. Caching allows us to optimize the number of requests directly serviced by a web server, and increase speeds by serving common elements quickly from more local locations. Caches under HTTP 1.1 allow for caching control such that the client is informed of non-transparent behavior, and the server can control many aspects of caching, beginning from whether or not the page should be cached at all.

### 2.1 Caching

HTTP 1.1 provides caching such that users and originating servers are allowed to control caching mechanisms and features to judge factors such as freshness of a cached entity, such as date and age response headers from a cache.

Validation of cached responses from the origin server when freshness criteria are not met has also been provided for, through cache validators such as entity tags or fields such as the last modified time. HTTP 1.1 provides for caching and delivery of byte sub-ranges of an entity, if associated cache validators match. Caches under HTTP 1.1 allow for caching control such that the client is informed of non-transparent behavior, and the server can control aspects of caching, beginning from whether or not the page should be cached at all.

### 2.2 Collaboration

The area of collaboration has been explored in many directions. Collaborative search mechanisms allow users to compose queries and view results together, or present systems which suggest queries. Collaborative search in multimedia repositories has given rise to systems which use algorithmic mediation to influence results on the basis of relevance indexes and weights such as the number of results visited amongst a certain result set. Systems such as Colab allow for controlling resources accessed by a user through the implementation of access rules [13], while others use scouts to search pages visited by users, develop user profiles and suggest materials [14]. More recent systems include plug-ins which allow users to tag or rate pages, or guide each other as they browse[15].

## III DESIGN

Each user computer or device carries a module which provides basic services needed for our system. All requests and replies by and to the browser are channeled to through this module. When a request from a browser is made (User Y), the module contacts a central server, which maintains a URL-IP map in a database, and requests address identifier of a computer which has the particular resource in its cache. If another computer X is found, the cached copy of that piece of information in X is returned to the user Y. If no computer with that particular resource is found, then the request is directed to the internet for a fresh fetch of data. An entry for this particular resource is made in the central server so that the local cached copy of this particular resource is available to other users from now on. A table is also maintained which contains the list of online users IPs.

When a user goes offline, the user is removed from the table of online users. In case a match is found in the URL-IP map and the user is offline, then the request is redirected to the actual web server rather than the other user. The central server analyzes the URL-IP map, as well as information such as the number of times a URL is accessed, to provide facts such as the most popular resources, or the most recently visited resources. Colloration Framework software on the central server puts these facts into pages which are loaded by each client. Depending upon options selected at the central server as part of the collaboration framework, a number of methods to present these facts are possible. Restating the design, the Figure 1 provides a diagrammatic representation of the different steps of page retrieval. Corresponding to the number labels in Figure 1

1). Client 1 requests a page with the Server Module as proxy

2). Server Module queries the MySQL server to determine forwarding IP - either an IP belonging to the Internet infras-tructure or another client (Client 2 in this sample case)

3). Server module uses Client 2 as proxy, requests page requested by Client 1

4). Client 2 requests page from Web Server.

5). Web server and Client 2 complete all transactions necessary for page retrieval.

6). Client 2 completes all transactions necessary to satisfy theserver module's request in (3) with cached data and freshly fetched data.

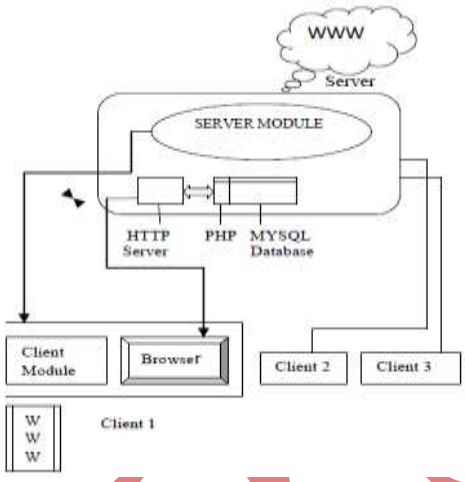


**Fig 1 Retrieval Model**

7). Server module completes all transactions necessary to satisfy Client 1's request in (1)

8). Client 1 requests collaboration sidebar page from HTTP server on server module.

9). All the data required for each widget in the collaboration sidebar is retrieved from the MySQL server using PHP. Pages are created with relevant javascript elements using PHP.

10). HTTP Server completes all transactions necessary to satisfy Client 1's request in (8).

### 3.1 System Architecture

The system runs on a two tier model:

- Client Module
- Central Server Module

The basic structure of both modules is as follows.

**3.1.1 Client Module Structure:** The client module acts as an intermediary between the user generated requests and the central server. The module provides the following services:

- Connection Initiation Ability - connect to a remote server socket.
- Connection Acceptance Ability - accept connections under certain conditions, as stipulated by protocol.
- Caching - Data accessed is cached at the user's computer.

The data is stored locally in the cache and is maintained with a URL-filename map, which allows the system to retrieve data related to a specific URL. Data in the cache is maintained till an expiry date (as set in the meta-data by

the website administrator), beyond which it is purged. If the cache size limit is crossed, older data is purged. The corresponding entries in the central server are also cleaned. All html standard cache directives are obeyed at all caching levels.

Administration - The user has a control over the system like cache size, setting options regarding resources which should be cached.

**3.1.2 Central Server Structure:** The central server interacts with the client modules, and provides a variety of services to clients. It creates and manages entries in a URL-IP map which it maintains for redirection of requests received by it. The central server must provide an ability to add an entry when a search request returns a null set. Also, it must provide for an entry deletion service which allows modules to delete entries in case of connection failures and cache purges, among others. Software at the central server, part of the collaboration framework backend, is responsible for analyzing the URL-IP map to generate relevant facts, such as popular searches, or most recent links visited, as requests occur. It generates these facts by running code associated with different widgets. These are loadable by a user's browser.

**3.1.3 Client-Server Communication:** Client server communication for search or deletion queries may be carried out using a simple protocol implemented over the established TCP connection. Connections for retrieval of cached data should be done through a reliable connection to prevent data corruption. Thus, HTTP over TCP would be the preferred protocol for such a transfer.

Collaboration Framework: The Collaboration Framework divides the user interface presented to the user into widgets - independent sets of UI elements and associated traffic analysis code at the backend - which may be included on their own into the user interface presented. Content is pulled by each client from the central server as a mash up of these widgets, or from servers residing on themselves or other clients. A widget based user interface is used to present the user with a variety of analysis, while providing for additional services such as page tagging or rating. Widgets which mine URLs and other factors such as referers to build user profiles from the stored information and enable content targeting can also be developed. Data from the widget user interface may be sent to elements in the framework back-end to allow for features such as page or html element tagging. The framework allows us to enrich research activities by generating content containing facts such as most recent links visited by others or popular links.

## IV IMPLEMENTATION

Both Client module and Central server module shared a common set of functionalities which are provided by a general proxy server. Hence a proxy server was used as a common base with separate modifications for each module.

Functionalities provided by the base proxy used are as follows:

- Connection Initiation
- Connection Acceptance

- Caching Indexing
- Proxy Chaining
- Script Injection

### 4.1 Modification - Client Module

The base proxy was modified to differentiate between requests coming from the user's browser and request coming from the Central Server. The client module redirects the requests selectively based on the decision made in the above statement.

### 4.2 Modification - Server Module

4.2.1 **Database Integration**: A MySQL server is storing all the necessary data. A persistent connection to the database made as soon as the server module starts. When a client requests a particular URL, the server runs a SQL search query on the MySQL database through the connection object. The database server traverses the URL - IP table to find a match and return the IP of the client if match is found (returns null, if match not found). The Persistent Connection object with the MySQL database is used to issue a query to the database.

4.2.2 **Database Updating:** INSERT and UPDATE SQL queries are used to modify the IP - URL table whenever a client requests for a web resource.

4.2.3 **Database Maintenance**: Whenever the client loses some data (in case of cache corruption or data expiry or cache purging), the central proxy server updates it's IP - URL table accordingly.

### 4.3 Redirection Scheme

On an incoming request, the central server runs the Search function to retrieve the IP of the client module having the particular resource. - If match is found, the central server chains itself to that particular client module proxy which fetches the data from its cache. - If match is not found, the central server redirects the request to the Internet for a fresh fetch of data. In either case, the IP-URL table is updated.

### 4.4 Collaboration

The collaboration framework provides a flexible widget framework to create and implement User Interfaces and data mining for collaboration. Widgets are implemented as PHP scripts which are written to generate or store certain HTML-Javascript files. These widgets analyze stored data such as the URL -IP database, or a referer database. Other widgets may include input fields for users to enter data, such as tags for some URLs, etc. Currently, the popular URLs widget analyzes a table containing counts for URL occurance in the server module database, using SQL queries via PHP. Data created from this is presented as part of a mash up HTML page created dynamically. At the client, a firefox plugin - Greasemonkey - adds elements to all web pages rendered in the client browser. These elements retrieve the mash up HTML page from the HTTP server on the central server. Thus, the Popular URLs

widget is called as part of the PHP for this HTML page. As part of the Popular URLs widget, a list containing the most popular sites visited is shown to the user. Depending upon the additional widget generated pages listed for asynchronous retrieval on this page, additional HTML may be loaded, thus generating a mash up of data presented in a variety of ways. The PHP widget scripts may be run periodically by an independent PHP instance, which would execute a master script, thus running all widget scripts which require execution as time-bound jobs. The power and flexibility offered by PHP may exploit to create a variety of widgets which can be included with minimal additional code.

## V EVALUATION

It is tested and evaluated on a network of 4 clients and one central server. The clients were run on Windows XP machines, while the server was run on a Virtual Machine (VMWare) on top of a Windows XP Host. A set of 6 HTTP resources was requested from 4 nodes, one at a time, from each fresh node, that is, from nodes which have never requested these materials before.

**X: Time for complete page load, Y: Time elapsed before first byte is received**

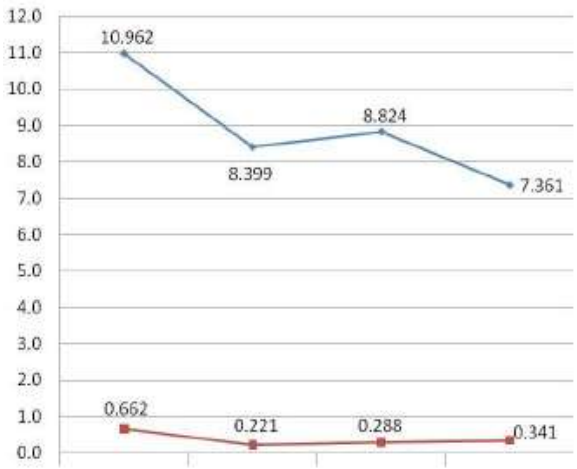


**Fig 2**

| | 1st Req. | 2 nd Req. | 3 rd Req. | 4 th Req |
|---|---|---|---|---|
| **X** | 10.962 | 8.399 | 8.824 | 7.361 |
| **Y** | 0.662 | 0.221 | 0.288 | 0.341 |

**Table 1: Loading times- Text Heavy Page of Size 1.62 MB**

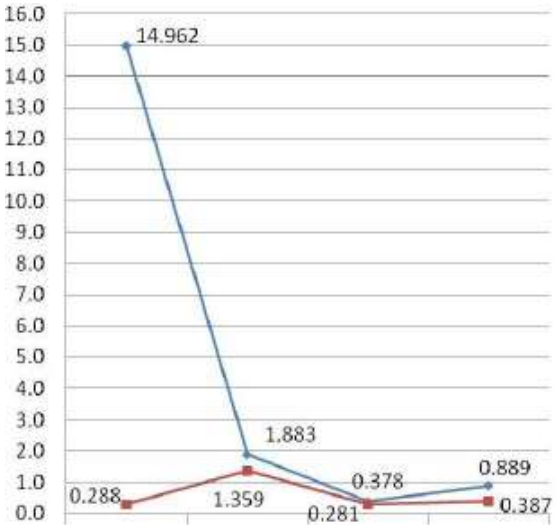


**Fig: 3**

| | 1st Req. | 2 nd Req. | 3 rd Req. | 4 th Req |
|---|---|---|---|---|
| **X** | 14.962 | 1.883 | 0.378 | 0.889 |
| **Y** | 0.288 | 1.359 | 0.281 | 0.387 |

**Table 2: Loading times- Image of Size 361 KB**

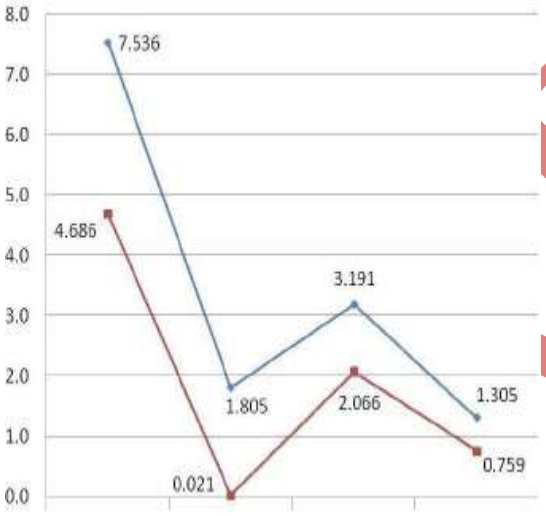


**Fig: 4**

| | 1st Req. | 2 nd Req. | 3 rd Req. | 4 th Req |
|---|---|---|---|---|
| **X** | 7.536 | 1.805 | 3.191 | 1.305 |
| **Y** | 4.686 | 0.021 | 2.066 | 0.759 |

**Table 3: Loading times- Image of Size 918 KB**

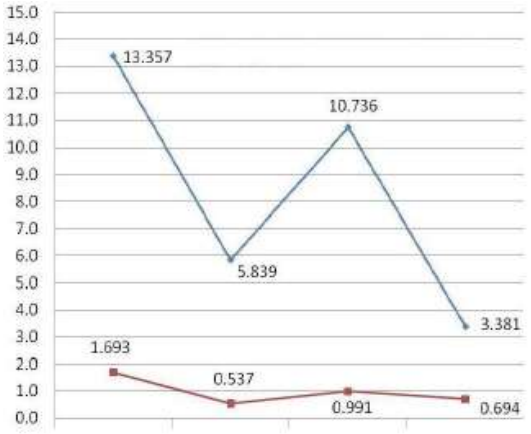


**Fig: 5**

| | 1st Req. | 2 nd Req. | 3 rd Req. | 4 th Req |
|---|---|---|---|---|
| **X** | 13.357 | 5.839 | 10.736 | 3.381 |
| **Y** | 1.693 | 0.537 | 0.991 | 0.694 |

**Table 4: Loading times- Image of Size 1.8 MB**

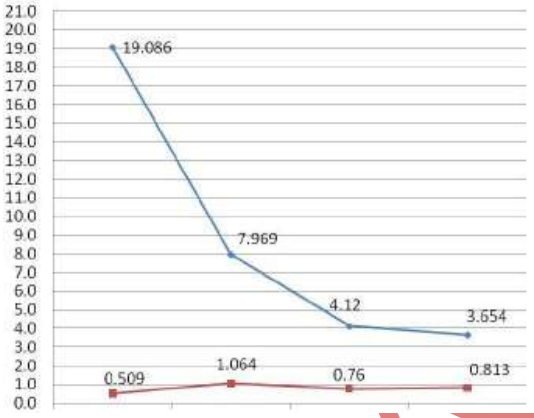


**Fig: 6**

| | 1st Req. | 2 nd Req. | 3 rd Req. | 4 th Req |
|---|---|---|---|---|
| **X** | 19.086 | 7.789 | 4.120 | 3.654 |
| **Y** | 0.509 | 1.064 | 0.76 | 0.813 |

**Table 5: Loading times- Image of Size 3.42 MB**

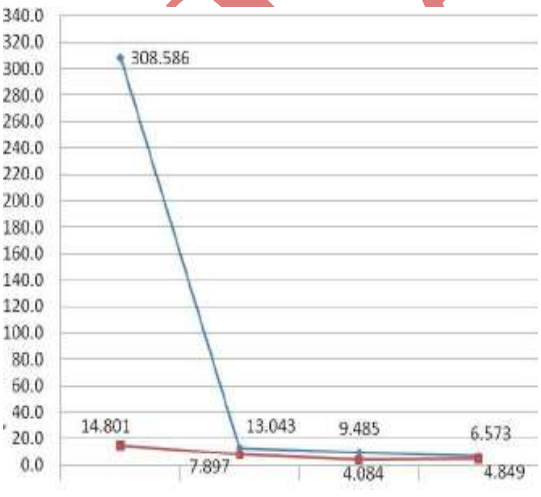


**Fig: 7**

| | 1st Req. | 2 nd Req. | 3 rd Req. | 4 th Req |
|---|---|---|---|---|
| **X** | 308.586 | 13.043 | 9.485 | 6.573 |
| **Y** | 14.801 | 7.897 | 4.084 | 4.849 |

**Table 6: Loading times- Image of Size 8.57 MB**

HTTP resources were one text heavy HTML page, 4 images and 1 PDF file. The images were of varying sizes and resolution. The HTML Page size was 1.62 MB (including all the images and other resources on the page). Image sizes varied from 361 KB to 3.42 MB. PDF file was 8.57 MB in size. The Firefox browser was used on each of the clients, client module as its proxy server on each computer. A firefox add-on - Life-of-Request info - was used to check loading time of each HTTP resource. 2 parameters - time elapsed before the first byte was received from the server, and time elapsed before complete loading were noted for each HTTP Resource. Please refer to the graphs and tables for this data. Internet link bandwidth savings may simply be estimated as

$$S = ( N_{req} - 1 ) X Scached$$

Where, $N_{req}$ is the total number of requests issued for the resource, and Scached is the size of the cached resource.

## VI OBSERVATIONS

The observations have been presented as a set of 6 graphs with accompanying tables. A general trend of decreasing latencies in complete page loading times was found for the HTTP resources tested. Certain aberrations, possibly due to background processing by additional services present on the test computers, were observed.

## VII CONCLUSION

This paper presents a system for collaborative caching, in its ability to allow users access to each other's cache, as well as enabling collaboration during browsing. It is implemented in Java, PHP, JavaScript, SQL, Shell Scripts. It retrieves additional content along with user-requested web materials as part of the collaboration framework. It is evaluated by measuring latencies in loading a set of HTML pages, images and a PDF document, over successive requests from a fresh node. A general trend of decreasing latencies was found.

## REFERENCES

[1] Marc Abrams et al, Caching Proxies: Limitations and Potential,Computer Sc Dept,Virginia Tech, Blacksburg, VA 24061-0106 USA.

[2] M.Bedekar,P. Gupta and S. Chatterjee- A Distributed Caching system International Journal of Knowledge Engineering, Bioinfo Publications, Vol. 1, Issue 1, pp. 01-04,2010.

[3] R.M. Baecker,Others. Reading in Human –Computer interaction: toward the year 2000. Morgan Kaufmann Publishers, 1995.

[4] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: a decentralized peer-to-peer web cache In Proceedings of the twenty-first annual symposium on Principles of distributed computing (PODC '02). ACM, New York, NY, USA, pp. 213-222, 2002.

[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '01). ACM, New York, NY,USA, pp. 161-172, 2001.

[6] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg (Middleware '01), Rachid Guerraoui (Ed.). Springer-Verlag, London, UK, pp. 329-350, 2001.

[7] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to- peer lookup service for internet applications (SIGCOMM'01). ACM, New York, NY, USA, pp. 149-160, 2001.

[8] B. Y. Zhao, J.D. Kubiatowicz, and A. D. Joseph. Tapestry: an Infrastructure for Fault-Tolerant Wide-Area Location and Routing. Technical Report. University of California at Berkeley, Berkeley, CA, USA, 2001.

[9] C.P. Gavalda, P.G. Lopez, A.F.G. Skarmeta, Dermi: a new distributed hash table-based middleware framework, Internet Computing, IEEE, vol.8, no.3, pp. 74- 84, 2004.

[10] C. Pairot, P. Garcia, R. Mondejar, A.F.G. Skarmeta, Building wide-area collaborative applications on top of structured peer-to-peer overlays, 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise, pp. 350- 355, 2005.

[11] P.Maymounkov and D. Mazi Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In Revised Papers from the First International Workshop on Peer-to-Peer Systems (IPTPS '01), Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron (Eds.). Springer-Verlag, London, UK, pp.53-65, 2002.

[12] Anh-Tuan Gai, Laurent Viennot, Broose: A Practical Distributed Hashtable Based on the De- Bruijn Topology, Fourth International Conference on Peer-to-Peer Computing (P2P'04), pp.167-164, 2004.

[13] RFC 2616 IETF (Internet Engineering Task Force. Hypertext Transfer Protocol - HTTP/1.1, ed. R. Fielding, J. Gettys, J. Mogul, et.

[14] Guillermo de Jesus Hoyos-Rivera, Roberta Lima-Gomes, and JeanPierre Courtiat A Flexible Architecture for Collaborative Browsing. In Proceedings of the 11th IEEE International Workshops on Enabling Technologies:Infrastructure for Collaborative Enterprises (WETICE '02). IEEE Computer Society, Washington, DC, USA, pp. 164-169, 2002.