Realizing Cleint and Server Mobility for WEB Applications

Yi-Hua Tsai, Jian-Jia Chen
Dept. of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan, ROC
Email: {p90018, r90079}@csie.ntu.edu.tw

Tei-Wei Kuo, Chi-Sheng Shih
Dept. of Computer Science and Information Engineering
Graduate Institute of Networking and Multimedia
National Taiwan University, Taipei, Taiwan, ROC
Email: {ktw, cshih}@csie.ntu.edu.tw

***Abstract*— Providing mobile web services has been a desirable feature for mobile users as the number of mobile devices increases and the broadband network services become popular. A web service is mobile in the sense that, the services sustain without interruption no matter the users switch to different browsing devices or the (logical or physical) web servers have been changed. The mobility of users' browsing devices and the change of service providing servers are called *client mobility* and *server mobility*, respectively. An innovative approach for realizing client mobility and server mobility is proposed in this paper. Our approach does not require any third-party agents and, hence, introduces less overhead and provides better backward compatibility for existing services. In order to provide the client and server mobility services, we only add plug-in modules for the web browsing clients and service providers. We demonstrate and implement our design on web client Konqueror and web server Apache.**

Keywords: Mobility, WWW, Client Mobility, Service Mobility.

1. Introduction

As mobile devices and broadband networks are widely available, it is desirable to provide personalized services for any devices and networks. Several kinds of mobile services such as service mobility, terminal mobility, and network mobility are defined for different scenarios or for different application domains. In the paper, we are concerned with providing the client and server mobility for web applications.

E-commerce is one example of client mobility and server mobility. Suppose a user is booking the airline tickets for a family trip in the Labor day holiday. However, the user has to leave his desk for an important meeting while the ticketing system processes the requests. It is very likely that the system has to abort the ticketing request as there is no response from the user to confirm the tickets. The result is that the user has to restart the ticketing process later (and may lost a good deal). Whereas, the user may carry his PDA, cellphone, or TabletPC as he walks away from his desk and continues the ticketing process. In order to do so, the service providing servers or the client devices have to maintain the connection across the devices. When the client mobility is available, the user can continue the ticketing process and does not need to login again as the user uses a different client browsing device. Moreover, when the user switches the client browsing device, the service provider may also need to switch the service providing server. For instance, the user may use the wired connection on his desktop computer but use the GPRS connection on his cellphone. It is very likely that different web servers are used for different network media. When the server mobility is available, the ticketing process will not be interrupted as the service provider migrates the service from one web server to another one.

Another example is the web server farm. In the web server farm, to reduce the power consumption or the service charges, the load balancer [16], [2], [5] or request dispatcher shall dynamically activate/de-activate the servers according to the workload demand. When the workload is greater than a certain threshold, the load balancer can create or activate additional web servers to shorten the average response times. On the other hand, when the workload is less than a certain threshold, the load balancer should bring down unnecessary web servers to reduce the power consumption or service charges. When a web server is activated or de-activated, the existing connections/services in this web server have to be migrated to another web server. It is not desirable to interrupt the services when the web server farm is reconfigured.

In this paper, we explore how to realize client and server mobility of web applications by maintaining the correct cookies and sessions for each connection. We first discuss different approaches for realizing client mobility. Our design adds a module for web clients to trigger the switches of client devices or web servers. Server mobility is then exploited for different approaches. Handshake scenarios for switching of web servers with and without client mobility are also studied. The overheads of the proposed methodologies and modifications to Konqueror and Apache are evaluated and reported. The results show that our design does not cause performance degradation.

The remainder of this paper is organized as follows. Related work of client and server mobility for web applications are presented in Section 2. Section 3 presents the design approaches and implementations for client mobility; Section 4 presents the design approaches and implementations for server mobility. How to realize client mobility and server mobility on Konqueror and Apache are also discussed in Section 3 and 4, respectively. Section 5 analyzes the proposed approach. Section 6 concludes the paper.

2. Related Work

World Wide Web (WWW) has been widely used either for commercial or personal purpose. WWW servers communicate with clients by Hypertext Transfer Protocol (HTTP) [3]. The system architecture for general web browsing is a client-server architecture, where each client request is transmitted to a service provider (server) directly. In order to balance the workload on a centralized web server to provide high availability, service providers might adopt a cluster architecture by grouping servers running a Web application simultaneously, appearing to the world as if it were a single server. The system distributes requests to different nodes within the server cluster, with the goal of optimizing system performance. This results in higher availability and scalability – necessities in an enterprise, Web-based application.

Our work is related to earlier works that are concerned with mobility of network connections. Examples are user mobility, service mobility, terminal mobility, and network mobility. Many results adopt information forwarding approaches with an agent-based architecture. For maintaining service sustainability of streaming audio/video, researchers in [9] and [10] proposed seamless mechanisms. With the assistance of the middle-ware server, the session transfer is transparent to users. Wang et al. [15] integrated the telephony and data services in the Internet for reducing the communication cost. Beside information forwarding approaches, user mobility could also be achieved without helps from agents. Researchers in [11] and [4] proposed a non-agent-based approach by modifying the headers of TCP packets or the application layer by the Session Initiation Protocol (SIP).

There are little works taking considerations of the mobility of web applications. HTTP was originated as a *stateless* protocol designed for short-lived services originally. A stateless protocol means that the state information for a connection is not maintained [8]. To maintain the state information, Kristol et al. [7] proposed a *session* mechanism by placing additional state information in the HTTP request and response messages. Each session is relatively short-lived and can be terminated either by the user or the issuing server. Such an idea was defined as *cookie* in the HTTP/1.1 specification [3]. The state management based on the cookie mechanism has become popular for the maintenance of the users' persistent or temporal information. With considerations of sustained services for web browsing, Iyengar et al. [6] proposed an agent-based mechanism which does not make use of cookie. Before the transmission of a web page, the corresponding web server modifies the hypertext links in the web page to embed the state information. When the client sends a request to the server, the state information is extracted from the request hypertext link as a session. In [12], a repository server for storing session information was added so that the mobility could be obtained by logining to the repository server to acquire the session information. Agent-based approaches result in transmission delay since a web page must be first transmitted to the repository server and then forwarded to the web client.

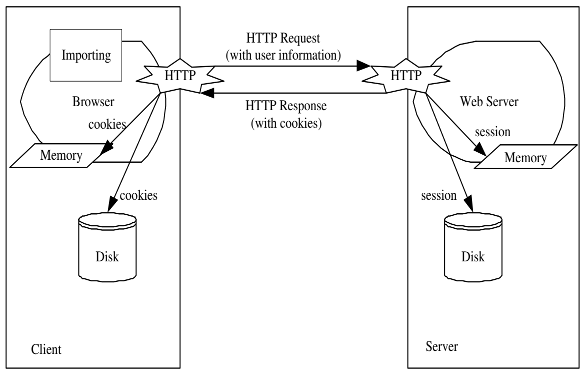

Figure. 1. The imported approach for achieving client mobility of web services

3. Client Mobility

We define *client mobility* as the capability of switching client devices without interrupting existing web sessions. In the following, we first discuss the alternatives of realizing the client mobility. The proposed approach and how to implement the approach follow.

A. System Design for Realizing Client Mobility

The major challenges of realizing client mobility are two-folds: performance overhead and backward compatibility. The imposed performance overhead should only occur when the client browsing device has been switched. In addition, the technique should have little impacts on existing services. As a result, the service providers do not need to re-build their web sites to provide client mobility. Before we step into the detail explanation, we define the terminologies used in the section. The *original node/destination node* for a request of client mobility is defined as the client browsing device before/after performing client mobility. The *original browser/destination browser* is the corresponding (client) web browser.

The client devices store cookies in the local disk file systems or memory and the server stores the session information in the memory or local disk file systems. There are three alternatives for realizing client mobility: *eavesdropping*, *importing*, and *interception*, as presented in [14]. With considerations of the performance overheads and the implementation overheads, the importing approach is concluded to be more efficient than the other approaches. The importing approach is used in our design. In the *importing* approach (Figure 1), additional components are added in the original browser. The components are responsible for delivering the cookies of alive HTTP sessions in the original browser to handle requests of client mobility. In short, it only delivers the cookie to the destination when the client browsing devices have been changed. Beside the original node, the destination node must be also modified to receive the session information delivered from the original node. A daemon in the destination node is added to receive and store the delivered session information properly.

B. Implementations of Client Mobility

We now describe how to realize client mobility on the web browser Konqueror. There are several popular web browsers, e.g., Internet Explore, Konqueror and Mozilla, we choose Konqueror as our implementation platform because of its complete functions, the capability of embedding services and plug-in components, and the open-source attractiveness. Besides, Konqueror is available for most modern operating systems, e.g., Linux, BSD, SunOS, UNIX, etc.

Cookies can be classified into two types, one is permanent file cookies and the other one is session cookies. In general, the preferences of users' browsing hobbies or the personal information are recorded as permanent file cookies. For Konqueror, permanent file cookies are stored in regular text files. On the other hand, the session ID and the browsing state of a user are recorded in session cookies, which are stored in the main memory for Konqueror.

In Konqueror, cookies are managed by the module KCookiejar. KCookiejar provides the functionality of adding, finding, and deleting cookies. Our implemented DCOP client is responsible for requesting the member function saveCookies() of KCookiejar to obtain session cookies[1]. However, for the sake of security consideration, the function saveCookies() skips session cookies for web-client requests. Although we can modify the source code of KCookiejar to return the session cookies for web-client requests, this breaks the secure mechanisms in KCookiejar. KDE also provides another module KCookiesManagement, which is a Graphic User Interface (GUI) to manage cookies, including deletion and extraction of cookies. KCookiesManagement returns all cookies including session cookies by referring to KCookiejar by DCOP communication. In addition, Konqueror provides a layout modification in a DOM XML files without modifying the source code or compiling the whole Konqueror project. Therefore, we could rewrite the DOM XML file of KCookiesManagement so as to extract all alive cookies and return these cookies to the web client. The extracted cookies from KCookiesManagement are stored in disks. After that, the stored cookies are then transmitted to the destination site by a TCP/IP sender, called KCookiesSender. On the destination site, a DCOP client, called KCookiesReceiver, is added to the destination browser for inserting received cookies. Depending on the handshake scenarios, i.e., *Transfer Now and Browse Now, Transfer Later and Browse Later, or Transfer Now and Browse Later* defined in [14], KCookiesSender might send the stored cookies to the KCookiesReceiver in an automic manner or after a request from the KCookiesSender. We can add cookies into proper places by using the KCookiejar module. In addition, we add a daemon in the destination node to receive cookies which are delivered from the original node. Essentially, the daemon receives TCP/IP packets. The block diagram for these components are shown in Figure 2.

[1]DCOP is an interprocess communication service in KDE.

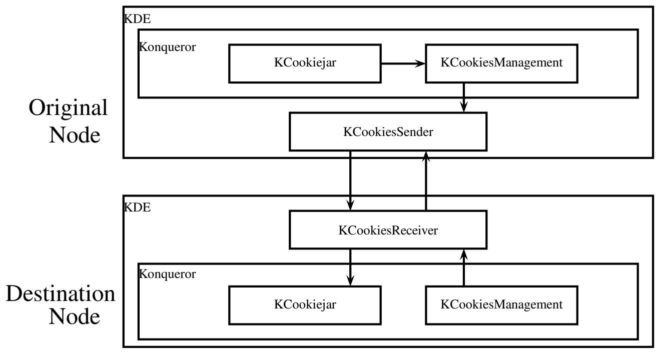

Figure. 2. The block diagram for client mobility

4. Server Mobility

Server mobility is defined as the capability of the switching of web servers without interrupting existing web sessions. In the following, we first discuss the alternatives of realizing the server mobility. How to implement the proposed technique in Apache as an example follows.

A. System Design for Realizing Server Mobility

In the following, we describe the concepts and design issues for web servers and web clients to provide sustainability of web applications when a user performs a request of server mobility. Most web servers keep track of the activities and the states of a client through a set of session data. In this paper, we only consider the web servers which are able to store session information and are homogeneous. In general, a session for a user is initiated by a login request. When a session starts, a session ID is first generated by the web server. The session ID is unique for all the sessions on one web server. After creating a new session, the session is put as a session cookie in the HTTP header in the response messages. As shown in Section 3-A, when the web browser receives the cookie, it stores the session cookie in a proper place for further handshaking process.

The issued session ID from a web server is used as an identity to access the information provided in the web server. While a web client requests a web page inside this web server, the unique session cookie is embedded in the HTTP header. The web server checks whether the corresponding user of the received session ID has the permission to access the requested web resources. The web client will then receive either the requested web page or an error page. If the session ID is valid and the user has the proper access permission, the web server returns the requested web pages. Otherwise, the web server returns an error page. According to the described properties of web servers and web browsers, the management of session cookies is the key component we have to take care for realization of server mobility.

The terminologies used in Section 3-A are adopted again for the following discussions. The *original server/destination*

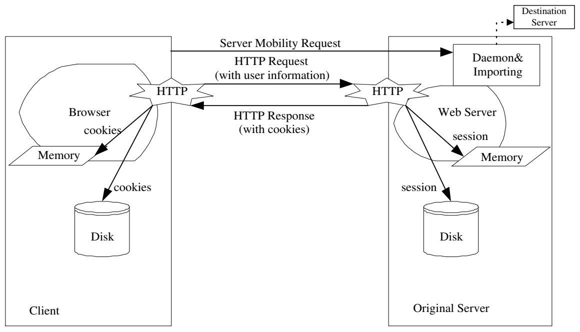

Figure. 3. The importing approach for realizing server mobility

server is defined as the web server before/after performing server mobility for a user connection.

The basic idea of realizing server mobility is to properly deliver the corresponding session cookies from the original server to the destination server. There are three possible ways of storing the session cookies on the web servers: main memory, local disk file systems, and databases. When the session cookies are stored in the databases, the original and destination server may share one database and it becomes trivial to migrate the session from one server to another one. However, the central database may become the performance bottleneck of the system. When the session cookies are stored in the main memory or local disk file systems, the system should be responsible for migrating the cookies to a different server. The problem is how to migrate cookies with minimal runtime overhead.

Three alternatives, *eavesdropping*, *importing*, and *intercepting* approaches, were presented in [14]. In the *importing* approach, shown in Figure 3, an additional process in the original server is added to deliver the session information of alive HTTP session cookies to handle a request of server mobility. It is also necessary to modify the destination server so as to receive the session cookies delivered from the original server. For instance, a daemon process can be added to the destination server to receive and store the delivered session cookies properly. The importing approach does nothing until a server mobility request occurs. The performance overheads and implementation overhead of the importing overhead is much less than that of the other approaches. Therefore, we choose the importing approach to realize server mobility.

In the following, the procedure of delivering sessions is presented, as shown in Figure 4, when requests of server mobility are activated by client browsers. First, a user initiates a request for server mobility in the client browser for a specified web service provider using a pop-up dialog. A daemon added to the original server accepts the request of server mobility and transmits the list of available service servers to the client. We must also add an additional component in the client browser for determining the new service provider (the destination server). The client browser then feedbacks the address of the determined destination server to the original server. The original server has to transmit session information to the

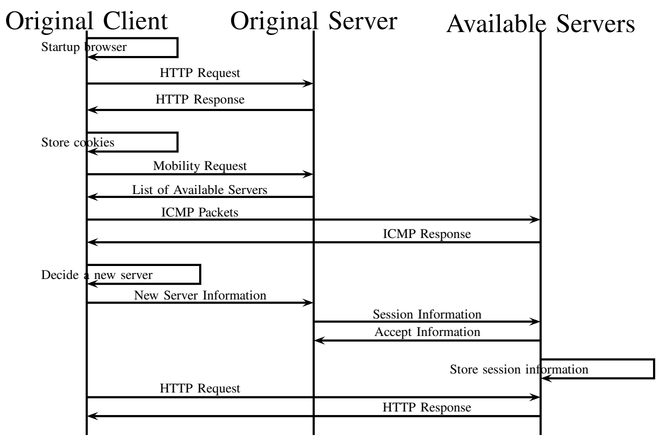

Figure. 4. A scenario for server mobility

destination server. An added daemon in the destination server must be created for putting the receiving session information in the proper memory or disk space.

B. Implementations of Server Mobility

We now describe how to realize server mobility on web server Apache. Apache is selected as the experimental platform because of its popularity and the open-source attractiveness. In addition, Apache is now available for most modern operating systems, e.g., Linux, BSD, SunOS, UNIX and Windows NT. More than 64% of the web sites on the Internet are hosted by Apache, reported by NetCraft Web Server Survey in October 2003 [13]. To create stateful web pages, we choose PHP, which is a popular web scripting language[1]. The setcookie() and getcookie() APIs are used to create and verify a session in PHP. With Apache and PHP, sessions are saved as regular files in local disk file systems.

Both client mobility and server mobility are activated by client users since web servers are designed to service requests passively. An external service daemon in the original server is implemented to accept server mobility requests from clients by TCP/IP modules. Any service providing servers which serve the requests must reply the request. The web client then selects a server having the shortest response time, measured by ICMP packets. The client then transmits information of the selected destination server to the original server by TCP/IP modules. While a server mobility request occurs, the added importing modules on the original server delivers the contents in the corresponding session cookie files to the destination server. We also add a TCP/IP service daemon in the destination server to receive the session information. The session information is then put in the local disk file systems on the destination server. The flowchart in the original server is shown in Figure 5.

a) Remark: Implementation for Other Web Browsers and Servers: Due to the popularity of Internet Explore (IE), we also try to implement our proposed approaches in IE. However, most key points are hidden in the kernel of Internet Explore. It might require more effort for tests and implementations.

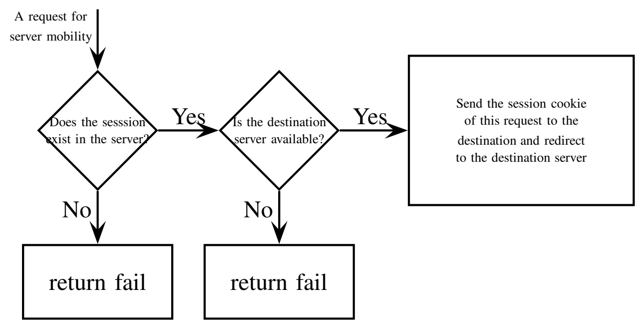
A request for server mobility
Does the sesssion exist in the server?
Yes
Is the destination server available?
Yes
Send the session cookie of this request to the destination and redirect to the destination server
No
No
return fail
return fail

Figure. 5. The flowchart for server mobility in the orginal server

Recently, Microsoft announced some methods to get and set cookies. Our design should also work for IE.

Microsoft also promoted her Internet Information Server (IIS) in recent years. In IIS, session data are stored and maintained in main memory. Although Active Server Page (ASP) provides a function to get session cookies, applying such a function must modify the applications developed by users.

5. Analysis

A. Performance Evaluation

Since only additional cookies and URL addresses are delivered for clients and servers, the overheads for data packets of client and server mobility are small. In our experiments, the overheads are within one second and are negligible compared to network delay in web services.

Table I shows the evaluation results of the overheads for performing client mobility. Yahoo and eBay were adopted for performance evaluations. The overheads were mainly due to the additional transmission of cookies and bookmarks. The size of cookies and bookmarks might be different with dependence on web applications or servers. It was specified on the specification of cookies at most 300 cookies for a browser, at most 4 kilobytes for per cookie and at most 20 cookies per server or domain. Therefore, the worst-case overheads for servicing a request of client mobility is no more than 82542 bytes with a simple calculation. The overheads of our implementations for server mobility are only dependent on the size of the transmitted session cookies from the original server. Therefore, the overheads for a request of server mobility are at most 4 kilobytes.

B. Remarks

In the following, we show the strength, weakness, opportunity, and threat for our proposed approaches and implementations in client mobility and server mobility. The analysis reveals that our approaches meet not only the growing convenience of mobility but also the prevailing usages of web services. In the following, we describe our analysis in the following order, the opportunity, strength, weakness, and threat. Table II summarizes our analysis

Convenience and time savings have been important issues for decades, a lot of work research how to promote the efficiency and performance for operations. In this paper, we provide the convenience of client and server mobility for web services in existing systems. The sustainability of web sessions during the switching of client devices and web servers provides users to enjoy an uninterrupted service on different environments. Since the concept of mobility has been accepted widely, tight connectivities between mobile services/devices and people’s lives provide opportunities for our implementations. Besides, web services have also become a major part of people’s life for information exchanges and product presentations in either commercial or personal use. Our implementations provide users sustained web services when users switch the operating device.

Our designs and implementations have several advantages for survival in the competitive information technology. Our approaches make as least modification to existing systems as possible without changing users’ habits. Therefore, it is possible for us to deliver our implementations along with the existing software as standing on the shoulder of giants. System overheads only take place on the moment of servicing a request of client or server mobility. After a request of mobility is accomplished, the behavior of clients and servers recovers as if nothing happens. Therefore, the overheads are almost negligible compared to applying the agent-based methodology. This is heavily due to the simplified system architectures for web browsing. Although most web sites realize uninterrupted services for web browsing by session cookies, it is noticeable that there also exists different approaches. For example in [6], the hypertext links might be modified to encode state information on the servers by filtering web pages before transmission so that the state information preserves, and the client mobility can be achieved by delivering the proper hypertext links for users when they switch into another terminal. Therefore, our implementations might not suitable for all web servers but for most web servers.

Our designs and implementations also have several disadvantages. Some users do not accept cookies due to the doubts about the insecurity issues in cookies. However, session cookies are still the most popular and most people accept the cookie approach. Our approach might not function well if web servers check the session information as well as the IP address of each connection with that of the initialized connection of a session. Since Secure Sockets Layer(SSL) is widely used for internet transactions, we shall also consider the service sustainability for SSL links. The hardness of achieving client mobility for SSL links is due to the hardness of extraction of the keys for authentication on the existing browsers.

The major threat we might meet comes from the insecure transmissions of session cookies. As mentioned above, it might be possible to provide sustained web services by different approaches instead of session cookies. The lifetime of our approach heavily depends upon the lifetime of the cookie mechanism. Although we provide simplicity for client mobility and server mobility, the functionality of our implementations is

Unit: bytes					
Web Server	Request	IP address	Cookies	Bookmarks	Total
Yahoo	7	15	674	539	1235
eBay	7	15	1706	521	2249

TABLE I

EVALUATIONS OF CLIENT MOBILITY FOR WEB SERVERS YAHOO AND EBAY

Strength	**Weakness**
Suitable for existing systems	Doubtful for cookies
Consistent in operations	Affected by the authenticate method in servers
Negligible in browsing overheads	Unsuitable for SSL links
Simplified in architectures	
Opportunity	**Threat**
Booming in the needs of mobility	Dependent on the lifetime of cookies
Popular of the Internet	Powerful for service providing in other designs

TABLE II

SUMMARY FOR THE STRENGTH, WEAKNESS, OPPORTUNITY AND THREAT

limited on client and server mobility. However, the agent-based systems might provide more functionalities with introductions of service agents (in either software or hardware) with a little overheads.

6. Conclusion

The purpose of this paper is to explore client mobility with little overheads and modifications to existing systems and infrastructures. We aim at a non-agent-based approach to sustain web sessions during the switching of client devices and web servers. Different approaches for achieving client mobility are studied, and the corresponding scenarios for the switching of client devices are presented. Server mobility is also exploited for different approaches. Scenarios for the switching of web servers are studied with and without client mobility. A popular browser Konqueror and a popular web server Apache are taken as case studies. A GUI module is inserted in Konqueror for the triggering of the switching of client devices and web servers. We measure the overheads of the proposed methodologies and also the modifications to Konqueror and Apache.

For future research, we shall extend the work to services of continuous streams for user mobility. More research work on intelligent management of servers would also be very rewarding.

References

[1] PHP: HyperText Preprocessor. http://www.php.net.

[2] R. Bunt, D. Eager, G. Oster, and C. Williamson. Achieving load balance and effective caching in clustered web servers. In *4th International Web Caching Workshop*, pages 159–169, March 1993.

[3] R. Fielding, J. Getys, J. Mogul, H. Frystyk, and T. Berners-Lee. IETF RFC 2068: Hypertext Transfer Protocol – HTTP/1.1, January 1997.

[4] L. I. Florin Sultan, Aniruddha Bohra. Service continuations: An operating system mechanism for dynamic migration of internet service sessions. In *22nd International Symposium in Reliable Distributed Systems (SRDS'03)*, pages 177–186, 2003.

[5] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-based scalable network services. In *ACM Symposium on Operating Systems Principles*, pages 78–91, October 1997.

[6] A. Iyengar. Dynamic argument embedding: Preserving state on the world wide web. *IEEE Internet Computing*, 1(2):50–56, 1997.

[7] D. Kristol and L. Montulli. IETF RFC 2109: HTTP state management mechanism, February 1997.

[8] J. F. Kurose and K. W. Ross. *Computer Networking*. Addison-Wesley Longman, Inc., 2001.

[9] J. Lin, G. Glazer, R. Guy, and R. Bagrodia. Fast asynchronous streaming handoff. In *Protocols and Systems for Interactive Distributed Multimedia*, volume 2515 of *Lecture Notes in Computer Science*, pages 274–287, 2002.

[10] T. Phan, K. Xu, R. Guy, and R. Bagrodia. Handoff of application sessions across time and space. In *Proceedings of the IEEE International Conference on Communications*, pages 1367–1372, 2001.

[11] H. Schulzrinne and E. Wedlund. Application-layer mobility using sip. *Mobile Computing and Communications Review*, 4(3):47–57, July 2000.

[12] H. Song, H. hua Chu, N. Islam, S. Kurakake, and M. Katagiri. Browser state repository service. In *Proceedings of the First International Conference on Pervasive Computing*, volume 2414 of *Lecture Notes in Computer Science*, pages 253–266, Zurich, August 2002. Springer-Verlag.

[13] The Apache Foundataion. Apache. http://www.apache.org.

[14] Y.-H. Tsai, J.-J. Chen, T.-W. Kuo, and C.-S. Shih. Cleint and server mobility for WEB applications. In *The Sixth International Conference on Information Integration and Web-based Applications & Services (IIWAS)*, pages 65–74, 2004.

[15] H. Wang, B. Raman, C. Chuah, R. Biswas, R. Gummadi, B. Hohlt, X. Hong, E. Kiciman, Z. Mao, J. Shih, L. Subramanian, B. Zhao, A. Joseph, and R. Katz. ICEBERG: An internet-core network architecture for integrated communications. 7(4):10–19, August 2000.

[16] K. J. C. Zornitza Genova. Challenges in URL switching for implementing globally distributed web sites. In *International Workshop on Parallel Processing*, pages 89–94, August 2000.