Client and Server Mobility for WEB Applications [1)]

Yi-Hua Tsai[1], Jian-Jia Chen[1], Tei-Wei Kuo[1†], and Chi-Sheng Shih[1†]

[1] *Department of Computer Science and Information Engineering*

[†] *Graduate Institute of Networking and Multimedia*

National Taiwan University, Taipei, Taiwan, ROC

[1]*Email: {p90018, r90079, ktw, cshih}@csie.ntu.edu.tw*

Abstract:

As mobile devices and broadband networks are widely available, it is desirable to provide mobility for web services. A web service is mobile in the sense that, without interrupting the services, users are allowed to switch the browsing devices or the service providing servers could be changed as the users move. The mobility of users' browsing devices and the change of service providing servers are called client mobility *and* server mobility*, respectively. This paper develops an innovative approach, which does not require any third-party agents, for realizing client mobility and server mobility. Hence, the approach introduces less overhead and provides better backward compatibility for existing services. Only a plug-in module for the web browsing devices is needed to provide the client and server mobility services. The approach has been demonstrated using one of the popular web clients,* Konqueror*, and one of the popular web servers,* Apache*.*

Keywords: Mobility, WWW, Client Mobility, Server 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 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

[1)] This work is supported in parts by research grants from ROC National Science Council program NSC-93-2213-E-002-090, NSC-92-2213-E-002-091, and NSC-92-2213-E-002-093.

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 a web server farm. In the web server farm, to reduce the power consumption or the service charges, the load balancer [6, 9, 19] 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.

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. Lin et al. [13] proposed a mechanism for service sustainability of streaming audio/video with the considerations of terminal hand-offs. Due to the high sensitivity on the quality of services of audio/video, seamless and uninterrupted data access across heterogeneous devices in mobile and asynchronous computing environments are provided. Phan et al. [14] explored the issues of session transfers for uninterrupted data accessed across heterogeneous platforms. A middle-ware server was introduced as an agent for transferring information between servers and heterogeneous clients. With the assistance of the middle-ware server, the session transfer is transparent to users. Wang et al. [18] integrated the telephony and data services in the Internet for reducing the communication cost. Personal and service mobility are provided for transferring information between heterogeneous access networks or heterogeneous access devices. Beside information forwarding approaches, user mobility could also be achieved without helps from agents. Sultan et al. [8] proposed a non-agent-based approach by modifying the headers of TCP packets. Schulzrinne et al. [15] showed the feasibility of user mobility at the application layer by the Session Initiation Protocol (SIP).

There are little works taking considerations of the mobility of web applications. WWW servers communicate with clients by Hypertext Transfer Protocol (HTTP) [7]. 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 [12]. For some web pages, the contents for all connections are the same. Therefore, the state information for such web pages is not necessary. For such pages, client mobility and server mobility are both achieved without any modification of existing systems. However, some web applications do require the state information. For example, the e-Market systems, which provide buyers the facility for shopping via the Internet, must maintain a stateful *shopping cart* for each individual buyer. The goods picked by a buyer must be stored in a corresponding shopping cart so that the user selections can be aggregated before purchase. To maintain the state information, Kristol et al. [11] 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 [7]. 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. [10] 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 such an approach, the client mobility can be achieved by delivering the proper hypertext links when the users switch their browsing devices. To sustain the web services, a repository server for storing session information was added in [16]. The mobility was 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.

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. Difference handshake scenarios for switching client devices are presented and analyzed. A well-known browser Konqueror is selected as the implementation platform to realize client mobility. A module is inserted in Konqueror 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. A popular web server Apache is selected as the implementation platform to demonstrate our techniques. The overheads of the proposed methodologies and modifications to Konqueror and Apache are evaluated and reported. The results show that our technique does not degrade the performance since we do not introduce agents into transmissions of web pages.

The remainder of this paper is organized as follows. Section 2 presents the design approaches and implementations for client mobility; Section 3 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 2 and 3, respectively. Section 4 concludes the paper.

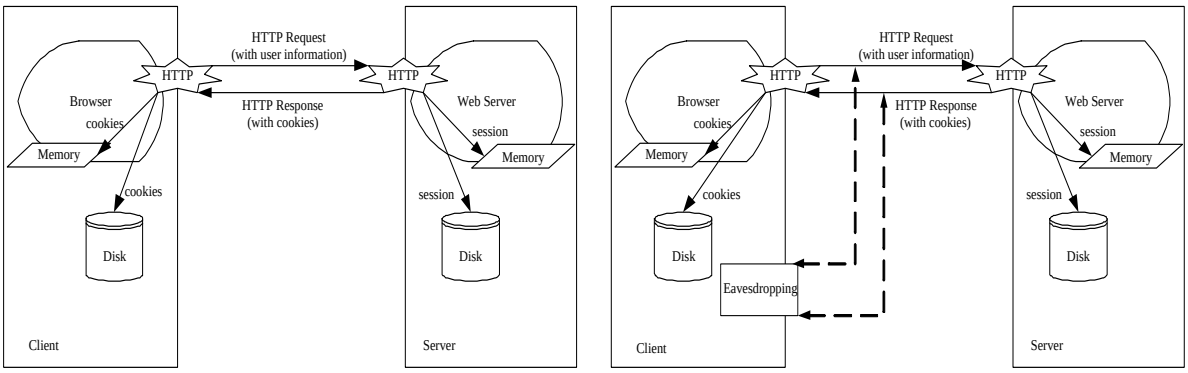
2 Client Mobility

We define *client mobility* as the capability of the switching of client devices without interrupting existing web sessions. In the following, we first discuss the alternatives of realizing the client mobility. How to implement the proposed technique in Konqueror follows.

2.1 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*/*destination* 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. Besides, the *original server*/*destination server* is defined as the web server before/after performing server mobility for a user connection.

Figure 1(a) illustrates the original HTTP procedures with cookies. 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



(a) Original HTTP (b) The eavesdropping approach

(c) The importing approach (d) The interception approach

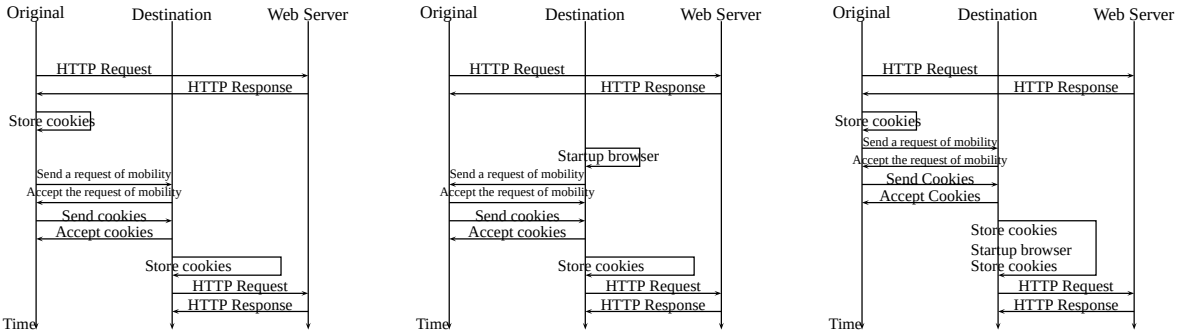
Figure 1: Approaches for achieving client mobility of web services

illustrated in Figure 1(b), Figure 1(c), Figure 1(d), respectively. In the *eavesdropping* approach, the original has the capability of monitoring the network packets that it receives. Hence, an additional daemon which monitors the packets between the original browser and web servers is added on the original. The cookies of HTTP sessions are captured and extracted by the added daemon. To handle the request of client mobility, the added component in the original delivers the extracted session information to the destination. Essentially, the added daemon is responsible for forwarding the cookies to the destination on behalf of the services initiated on the original. In the *importing* approach (Figure 1(c)), 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. In the *interception* approach (Figure 1(d)), a full-functioned browser is embedded in an *intercepting* process. The intercepting process handles all the network communications of the browser and forwards the alive cookies to the destination if exists.

For the eavesdropping approach, the added daemon in the original over-consumes system resources, including CPU and memory. The degradation of performance and the system overheads are significant compared to the other approaches. The implementation overhead of the interception approach is much more complicated than that of the others. It is because that the daemon acts like a proxy as a communication bridge between the Internet and the browser. Since cookies must be extracted, parsing the incoming web pages is required for this approach and causes additional overheads. In contrary, the importing approach does nothing until a request of client mobility occurs. With considerations of the performance overheads and the implementation overheads, the importing approach is concluded to be better than the other approaches. The importing approach is used in our design. The

	Eavesdropping	Importing	Interception
Development Difficulty	Middle	Low	High
System Overheads	High	Low	Middle
Browsing Overheads	High	Low	High
Architecture	Non-Agent-Based	Non-Agent-Based	Agent-Based

Table 1: Comparisons of the eavesdropping, importing and interception approaches for client mobility



(a) Transfer Now and Browse Now (b) Transfer Later and Browse Later (c)Transfer Now and Browse Later

Figure 2: Handshake procedures for client mobility

comparisons of our proposed approaches are summarized in Table 1. Beside the original, the destination must be also modified to receive the session information delivered from the original. A daemon in the destination is added to receive and store the delivered session information properly.

When the client device is switching, there are three handshaking scenarios between the original and destination browsers. In the scenarios, we only consider the case that the original browser is active. There is no need to migrate the sessions when the original browser is not on. If both of the original browser and the destination browser are active at the same time and the client mobility is activated on the original, the time-line of migrating sessions is illustrated in Figure 2(a), called *Transfer Now and Browse Now*. First, the user sends a client mobility request to migrate the session to a specified destination IP address. On the destination browser, the user verifies and accepts the session cookies delivered from the original browser. The added components for handling sessions in the destination browser store the received session information in the proper location on memory or local disk file systems. The second scenarios is the case when the client mobility is started on the destination. If the client mobility is activated on the destination, then the procedure of delivering sessions to the destination is shown in Figure 2(b), denoted as *Transfer Later and Browse Later*. In the original browser, a service daemon is installed and activated to receive requests of client mobility from the destination browser. To start a session migration, the user sends a request to migrate the session from the original to a specified address. After the service daemon on the original receives the request, the session cookies are delivered to the destination browser and placed properly. The last scenarios is the case when the destination browser is not active. In this case, the session information will be stored temporarily in some applications in the destination as shown in Figure 2(c), denoted as *Transfer Now and Browse Later*. The user first sends a request to deliver the session information to the destination IP address. When the user on the destination activates the web browser, the browser first queries the service daemon to obtain the session information and places the cookies properly. The session migration completes.

	Internet Explore	Konqueror	Mozilla
Persistent Cookies	Each file per cookie	A cookie file	A cookie file
Session Cookies	Main memory	Main memory	Main memory

Table 2: Locations of persistent and session cookies for some famous web browsers

2.2 Implementations of Client Mobility

We now describe how to realize client mobility on the web browser Konqueror. There are several popular web browsers, Internet Explore, Konqueror and Mozilla, etc. 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. However, there should be no problem of implementing our technique on the other browsers. The implementation is done on an Intel Pentium III PC's with the Red Hat Linux 8.0 distribution and a full KDE environment[1)].

In the following, we describe how to realize client mobility by utilizing the functions of KDE and Konqueror in detail. 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. Permanent file cookies are usually stored in regular text files on the local disk file system. Table 2 shows the locations of the cookie files resided for three popular web browsers, including Internet Explore, Konqueror and Mozilla. On the other hand, the session ID and the browsing state of a user are generally recorded in session cookies. Session cookies are usually stored in the main memory.

We now show how to extract the session cookies from the original browser and deliver these cookies to the destination to realize client mobility. 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 KCookiejar to obtain session cookies[2)]. However, for the sake of security consideration, KCookiejar 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 considerations of KCookiejar. Fortunately, KDE also provides another module KCookiesManagement, which is a Graphic User Interface (GUI) to manage cookies, including deletion and extraction of cookies. KCcookiesManagement returns all cookies including session cookies. 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 rewrite the DOM XML file of KCookiesManagement so as to extract all alive cookies and return these cookies to the web clients. The extracted cookies from KCcookiesManagement are stored in disks in our implementation. On the destination site, a DCOP client is added into the destination browser for inserting received cookies. We can add cookies into proper places by using the KCcookiejar module. In addition, we add a daemon in the destination to receive cookies which are delivered from the original.

[1)] KDE is a desktop environment for X-Windows, which is founded by the free edition of Qt library [4]. Konqueror is known as the default web browser for KDE, and is composed by a set of program modules, including KParts, the html rendering engine khtml, and so on [2]. In addition to the capability of web browsing, Konqueror is also a KDE shell which requests by the KDE module KIO Slaves and embeds the view of requests by the KDE module KPart.

[2)] DCOP is an interprocess communication service in KDE.

3 Server Mobility

Server mobility is defined as the capability of switching 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 follows.

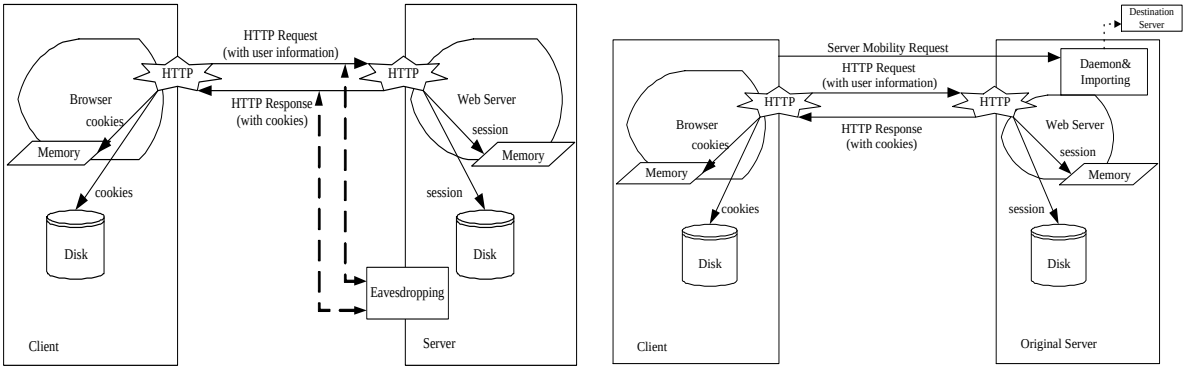
3.1 System Design for Realizing Server Mobility

In the paper, we only consider the web servers which are able to store session information and are homogeneous. The session information allows the server to keep track of the activities and state of the a client. Most modern web servers are able to store session information. The web servers have to be homogeneous because of the incompatibility on creating and storing web sessions on different web servers.

In general, a session for a web service 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. The session ID is embedded in the HTTP header as a session cookie as part of the response messages. As shown in Section 2.1, 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 a passport to access the information provided by the web server. When a web browser requests a web page, a unique session cookie is also embedded in the HTTP request header. The web server checks whether the session ID is valid and the corresponding user has the permission to access the requested web resources. 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 above discussion, the management of session cookies is the key design issue to realize server mobility.

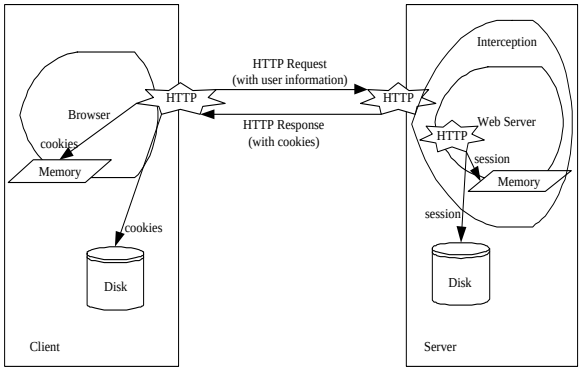
The basic idea of realizing server mobility is to properly deliver the corresponding session cookies from the original server to the destination server. There are three possible ways of storing the session cookies on the web servers: main memory, local disk file systems, and databases. When the session cookies are stored in the databases and the original and destination server may share one database, it becomes trivial to migrate the session from one server to another one. However, when the session cookies are stored in the main memory or local disk file systems, additional actions have to be taken on the web servers.

In the following, we discuss three alternatives for server mobility when web servers store session cookies in main memory or local disk file systems. They are *eavesdropping*, *importing*, and *intercepting* approach as shown in Figure 3(a), (b), and (c), respectively. In the *eavesdropping* approach, an additional daemon, which monitors the packets between browsers and the original server, is added to the original server. The cookies of HTTP sessions are captured and extracted by the daemon. When a request of server mobility arrives, the original server delivers the extracted session cookies to the destination server. An additional component on the original server might be needed to do so. In the *importing* approach shown in Figure 3(b), 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. In the *interception* approach shown in Figure 3(c), a full-functioned web server is embedded in an *intercepting* process. The process intercepts the network communication on the web server and forwards the session cookies to the destination server if exists. 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 in the destination server



(a) The eavesdropping approach

(b) The importing approach



(c) The interception approach

Figure 3: Approaches for realizing server mobility

	Eavesdropping	Importing	Interception
Development Difficulty	Middle	Low	High
System Overheads	Middle	Low	High
Accessing Performance	Slower	Same	Slower
Architecture	Agent-based	Non-Agent-based	Agent-based

Table 3: Comparisons of the eavesdropping, importing and interception approaches for server mobility

to receive and store the delivered session cookies properly.

The three alternatives have different impacts on run-time overhead and implementation overhead. In the eavesdropping approach, the added daemon in the original server over-consumes system resources including CPU and memory. The performance degradation is significant compared to the other approaches. The interception approach is much more complicated to implement than the others on implementations. Moreover, the daemon acts as a communication bridge between the web server and browsers. Since session cookies must be extracted, it is necessary to parse every packets that the server receives and, hence, causes remarkable overheads for sending web pages and receiving HTTP requests. In contrary, the importing approach does nothing until a server mobility request occurs. The performance degradation and implementation overhead of the importing overhead are much less than those of the other approaches. Therefore, we choose the importing approach to realize server mobility. The comparisons of the alternatives are summarized in Table 3.

3.2 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. Besides, 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 are hosted by Apache, reported by NetCraft Web Server Survey in October 2003 [17]. To create stateful web pages, we choose PHP, which is a popular web scripting language [3]. The setcookie() and getcookie() APIs are used to create and verify sessions 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 satisfy the requests must reply the request. The web client then selects a server with the shortest response time which is 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 deliver 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.

3.3 Implementation Remarks

The proposed mechanisms for client and server mobility have very limited overheads because only cookies are transmitted between clients and servers. The overheads are mostly resulted from the transmission of cookies. Although each cookie size is highly dependent upon its application or servers, a cookie size is no more than 4 KB and each browser has no more than 300 cookies [7]. For instance, the numbers of bytes for cookies are 674 and 1706 to serve a request of client mobility when Yahoo [5] and eBay [1] were adopted as example servers, respectively. We must point out that the overheads for the proposed mechanism are only paid once per moving, and no more overheads are needed once a client/server is moved (compared to the agent-based approaches).

Beside the limited overheads for the mobility implementations, the proposed mechanisms require little changes to existing systems. Compared to the agent-based approaches, no forwarding of data is needed for the proposed mechanisms. Once a moving is done (i.e., the completion of the transmission and refreshing of cookies and URL addresses), clients and servers operate as if no moving happens before. Although the proposed mechanisms do provide various advantages, compared to other work, there are limitations on the proposed mechanisms: For example, when client/server mobility involves the encoding of state information over hypertext links, the proposed mechanisms might not work well. Furthermore, cookies might not be accepted by some users/systems over different devices due to security considerations. It would impose serious impacts on the proposed mechanisms. If web servers always check up the session information and/or the IP address for each connection, then the proposed mechanisms might not work properly. The difficulty in achieving client mobility over Secure Sockets Layer (SSL) links is because of the challenges in the extraction of keys for the authentication in the existing browsers. Compared to the agent-based approaches, the proposed mechanisms tend to save network traffics but rely on some proper operations on cookies.

4 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 well-known 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] eBay Taiwan. http://www.tw.ebay.com.

[2] Konqueror - Web Browser, File Manager - and More. http://konqueror.kde.org.

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

[4] Qt overview. http://www.trolltech.com/products/qt/index.html.

[5] Yahoo Taiwan. http://www.yahoo.com.tw.

[6] 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*, March 1993.

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

[8] 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–.

[9] 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.

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

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

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

[13] 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.

[14] 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 15–, 2001.

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

[16] 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.

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

[18] 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, August 2000.

[19] 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.