Available online at www.sciencedirect.com

SCIENCE DIRECT®

Information Sciences 176 (2006) 1241–1265

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

www.elsevier.com/locate/ins

Stateful session handoff for mobile WWW

Ming-Deng Hsieh [a], Tsan-Pin Wang [b,*], Ching-Sung Tsai [a], Chien-Chao Tseng [a]

[a] *Department of Computer Science and Information Engineering, National Chiao-Tung University, Taiwan*

[b] *Department of Computer Science and Information Engineering, Providence University, 200 Chung-Chi Road, Shalu, TaiChung 433, Taiwan*

Received 30 September 2003; received in revised form 27 January 2005; accepted 26 February 2005

Abstract

This paper proposes a web session handoff system that can hand over not only stateless but also stateful sessions between homogenous and heterogeneous user devices to enable uninterrupted and seamless web access. The proposed system adopts a proxy-based approach and an optional client-assisted scheme in order to track and hand over session information. In the proposed system, a session is registered at a User Agent Proxy (UAP) and then tracked by the UAP so that the session can be handed over from one device via the UAP to another device. In addition to session information tracked by a UAP, the UAP can hand over more comprehensive session information by using the client-assisted scheme. Compared with client-based approaches, our design has several advantages, such as less modification to user devices, practicability, and fault tolerance. We have implemented a UAP on a PC and client programs for both PC and PDA. The implementation can successfully hand over between PC and PDA a stateful session for online shopping applications.
© 2005 Elsevier Inc. All rights reserved.

* Corresponding author. Fax: +886 426324045.
E-mail addresses: mdhsieh@csie.nctu.edu.tw (M.-D. Hsieh), tpwang@pu.edu.tw (T.-P. Wang), tstsai@csie.nctu.edu.tw (C.-S. Tsai), cctseng@csie.nctu.edu.tw (C.-C. Tseng).

0020-0255/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2005.02.009

Keywords: Web session handoff; User agent proxy; Stateful session; Session information

1. Introduction

In recent years, advances in wireless technologies and mobile devices have made mobile Internet practically feasible. The new environment challenges researchers to support terminal mobility, personal mobility, and session mobility [1] for reliable and seamless Internet access. In the literature, there is much research on terminal mobility or personal mobility, but little on session mobility. Session mobility is a mixed problem of terminal and personal mobility, and has become increasingly important recently. For example, handing over an application session between user devices enables a user to resume his application when he changes device. Among current Internet applications, web browsing is the most popular one and contributes to most Internet traffic. Therefore, we focus on web session handoff in this paper. Web session handoff enables a user to resume browsing the last page that he viewed on a first device without traversing the hierarchy of a web site's content again after he changes to a second device. Besides, handing over history of a web session enables a user to review the pages that he just viewed on the first device. Due to these features, web session handoff becomes useful when a user's browsing is interrupted and he must leave his current device. For example, one may use a PC in his office to start browsing for preparing a suddenly-notified meeting and then can hand over his browsing session to a portable computer (a homogeneous device) or a PDA (a heterogeneous device) before he leaves his office or when he moves into a meeting room. Moreover, a session initially on a capability-limited mobile device can be handed over to a more powerful stationary device for the purpose of the user's convenience. For example, one can initially view and process stock information with a PDA or a smartphone and then hand over this session to a PC when it is available. In the literature, session handoff has been envisioned in first-aid and clinical domains [2]. As mobile Internet access becomes popular, there will be more useful and diverse scenarios about web session handoff in the future.

HTTP, the basic communication protocol of web browsing, is originally designed to support stateless sessions. However, supporting stateful web sessions is important and necessary for web servers, e.g., online shopping web sites, which should differentiate sessions initiated by different users. Therefore, many approaches have been proposed and extensively used by web servers to enable stateful web sessions. In order to hand over not only stateless sessions but also stateful sessions, a web session handoff protocol/system should be able to track and hand over session information including session state between users devices. For example, BSR [3] adopts a client-based approach that uses a substantially modified browser to save and hand over browser state to/from a BSR repository. However, using client-based approaches to hand over web sessions is not desirable since they require substantial modification to browsers. In client-based approaches, each browser for various types of user devices needs its own modification and program code. Besides, the inefficient browser-dependent modification is impractical and usually has incompatibility problem when inter-working with unmodified web proxies or web servers.

In this paper, we propose a web session handoff system that allows users to hand over not only stateless but also stateful sessions between homogeneous or heterogeneous user devices. Essentially, our design is a proxy-based approach that deploys a User Agent Proxy (UAP) between user devices and web servers to track and hand over web sessions. As a result, the proposed system requires less modification to user devices while being able to hand over much of session information. In our experience, proxy-based approaches are practical for web session handoff before a standardized or popular web session handoff protocol is widely recognized.

The remainder of this paper is organized as follows. Section 2 discusses previous related work in mobility and compares three kinds of approaches for web session handoff. Section 3 presents the system overview. Section 4 discusses ways to track sessions and addresses issues of session tracking in detail. Section 5 elaborates our session handoff protocol including ways to hand over session information. Section 6 shows our current implementation. Section 7 gives conclusions and future work.

2. Related work

2.1. Previous work

In the literature, much research on mobility focus on terminal mobility, e.g., Mobile IP [4] and its variants [5]. Mobile IP and many of its variants deal with delivering packets via the home agent and foreign agents to a mobile terminal. In [5], by looking-up the location of a mobile terminal and directly establishing a connection to the mobile terminal, packets can be delivered by a correspondent node without passing through the home agent or foreign agents of the mobile terminal. On the other hand, personal mobility becomes more and more important [6–12] because it realizes a convenient scenario that allows a user to use various devices and to switch between them. Netchaser [6] uses agent programs residing on user devices and servers to achieve personal mobility. MPA [7–9] aims to contact a user regardless of devices available at hand and usable communication protocols. The iProxy/iMobile system [10] is a cross-platform middleware that allows client applications on various devices with various underlying protocols to access web services. VHE [11] in UMTS aims to provide global service portability that enables a user to tailor his service set and to receive personalized services, regardless of his location, access network or device type. In general, most systems support personal mobility by using a performance enhancing middleware server [12] situated between mobile devices and correspondent nodes to perform protocol conversion, content adaptation, and content caching. These middleware servers can store and use information about users preference, devices capability, and personal service profiles to adapt contents for mobile devices accordingly.

There is little research on handing over between user devices application sessions. iMASH [2] implements a system to fulfill this concept and uses it in a special-purposed healthcare domain. Besides, web session handoff can be realized by using process migration, Virtual Network Computing (VNC) [13], or Remote Desktop provided by Microsoft Windows XP operating system. However, protocols for process migration are too complex and heavy to suit application handoff. Furthermore, VNC and Remote Desktop are intrinsically different from application handoff since they are designed for remote console that outputs the screen of a first device to a second device and sends user input to the first device for processing. The transmission of screen data is a weakness since it may consume much network bandwidth. Besides, they cannot work well between heterogeneous user devices.

To hand over web sessions, Browser State Repository (BSR) Service [3] has implemented a BSR repository server and a BSR plug-in for Microsoft Internet Explorer (IE) 5.0. The BSR plug-in of a first device takes a snapshot of a browser's state and saves the snapshot at a BSR repository server, and then a browser on a second device can retrieve the saved snapshot to resume the saved state and session. Note that the first device is hereafter referred to as *source device* since it is the source providing session information, while the second device is referred to as *target device* since it is the target to accept saved snapshot. In its current implementation, BSR service only supports Microsoft's desktop OS and does not show the ability to hand over web sessions between heterogeneous devices (though it envisions such scenarios). Besides, BSR is a client-based approach and therefore requires much effort for modification to user devices.

2.2. *Server-based, client-based, and proxy-based approaches*

A web session handoff system should be able to track and obtain session information from source devices so that session information can be handed over as much as possible and a session can be more completely resumed on the target device. There can be three alternatives to web session handoff: client-based, server-based, and proxy-based approaches. As described in [3], server-based approaches are restricted by the information that a client sends to a web server and are not compatible to web servers that do not support web session handoff. Client-based and proxy-based approaches can hand over more session information than server-based approaches, but require modification to clients and proxies respectively. Since these two kinds of approaches require no modification to web servers, session handoff can be performed transparently to web servers and therefore server compatibility problem need not be concerned.

The main advantage of client-based approaches is that complete and precise session information can be obtained since a modified client can locally monitor complete session information. However, each client-based approach is currently restricted to user devices that have specially modified browsers or specialized client programs for each approach. It is impractical and inefficient to modify all existing browser programs for all types of user devices unless a standardized or popular web session handoff protocol is widely recognized.

In proxy-based approaches, a proxy should record all HTTP requests and responses passing through the proxy and perform additional treatment to track session information. Proxy-based approaches have two advantages over client-based approaches:

1. *Practicability:* Proxy-based approaches have high practicability since they require less or no modification to user devices and therefore are easy to support more browsers and devices. Although client-based approaches can hand over complete and precise session information, they require in-depth knowledge about browsers and substantial modification to user devices. For example, the modification to user devices in BSR [3] is made directly to Microsoft's IE 5.0. In contrast, an unmodified user device in proxy-based approaches can still hand over a session without losing much session information. Besides, a user device in proxy-based approaches can be slightly modified without in-depth knowledge about browsers to hand over more session information. Thus client-based approaches cannot work properly when modified devices are unavailable to users. Besides, users may suffer from the problem that their favorite browsers do not have a modified version that supports web session handoff.
2. *Fault tolerance to unreachable source device:* During handoff, a source device may be unreachable due to lack of battery or wireless connection. In this case, a proxy-based approach can still hand over a session because this session has been tracked by a proxy while the user is browsing. As a result, session handoff in a proxy-based approach fails only when the tracking proxy is unreachable. On the contrary, client-based approaches store session information in source devices or outer repositories (e.g. BSR repository), and therefore session information will not be stored or handed over if source devices, outer repositories, or network connections to them fail. In summary, proxy-based approaches are tolerant to the failure due to unreachable

source devices. Thus, proxy-based approaches are more robust than the client-based approaches in terms of fault tolerance, especially in wireless and mobile environment.

On the contrary, compared with client-based approaches, proxy-based approaches have two disadvantages:

1. The additional treatment generates extra load to proxies.
2. Session information tracked by proxy-based approaches might be incomplete since some session information may not pass through a proxy and will not be tracked by the proxy.

To overcome these disadvantages, we adopt in our design a proxy-based approach with optional assistant information from user devices. The assistant information includes session information that cannot be tracked by a proxy, but can be collected by a slightly modified browser or an optionally installed client program in a user device. With the combination of proxy-based and client-assisted scheme, the proposed approach elaborated in the following sections outperforms the existing approaches.

3. System overview

Fig. 1 illustrates the system topology of our design. The topology is similar to the traditional web access topology that consists of user devices, proxies,

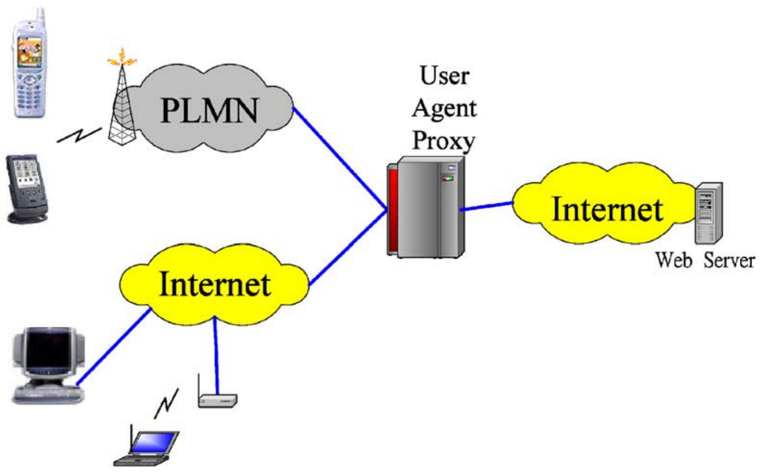


Fig. 1. System topology.

and web servers. In this topology, we introduce a special purpose web proxy, called User Agent Proxy (UAP), to replace a normal web proxy. In our design, user devices and the UAP are able to perform handoff-related functionalities in addition to traditional functionalities of web proxies. The UAP, situated between user devices and web servers, provides user authentication, session tracking, session handoff, and content adaptation functionalities for fulfilling web session handoff. As for user devices, they can be unmodified or slightly modified in comparison with normal user devices. The slight modification to user devices is to plug in a small client program that performs handoff-related functionalities such as registering sessions, collecting information for assisting the UAP in tracking sessions, and handing over sessions. User devices without modification still can perform session handoff as well, while the modification to user devices supports handoff of more session information, e.g., un-submitted user input. Intrinsically, the idea of this design is different from BSR service because a BSR repository server is not involved when a user device is browsing and does not provide functionalities of web caching and session tracking.

3.1. Proxy session handling

In our design, a UAP will track the session information of each proxy session for session handoff. A proxy session, in our definition, is different from a normal web session established between a user device and a web server. A proxy session starts when a browser process on a user device is authenticated with the UAP, remains when the user hands over his session to another device, and terminates when the authentication expires or the user logouts. According to this definition, a proxy session thus may contain multiple traditional sessions between several web servers in parallel.

In our design, processing of a proxy session is divided into three stages: registration, browsing and tracking, and session handoff.

3.1.1. Registration

In this stage, a user authenticates himself and his current device with a UAP to register his device location, profiles, and sessions. For the user's convenience, he is allowed registering a new session by using another browser process on the same or another device while keeping previously registered sessions.

3.1.2. Browsing and tracking

In this stage, an authenticated user device (can be a source device or a target device) sends browsing requests to the UAP, and then the UAP not only processes the browsing requests as a traditional proxy does but also tracks the proxy session. If the UAP cannot satisfy the requests with its cached objects, the UAP will forward the requests to original web servers. However the UAP needs to add user profiles as CC/PPex [14–16] headers in the requests sent by the browsers not supporting CCPP [14]. Furthermore, it needs to add in the requests sent from a target device or in the responses replied from web servers the session information, such as cookies, which has not been handed over to the target device. On the other hand, session tracking is performed when the user device and the UAP exchange browsing requests and responses. Therefore, no extra messages are exchanged between the user device and the UAP for session tracking. The basic treatment of session tracking is to record all HTTP requests from a registered browser and to track their corresponding responses for session information. Besides, the UAP can perform further treatment to learn more session information. The description of such further treatment about proxy session tracking will be elaborated in Section 4.

3.1.3. Session handoff

In this stage, the UAP, one or more source devices, and the target device cooperate to hand over the proxy session from a selected source device to the target device. First, the user switches to the target device and chooses to hand over a proxy session among those registered ones in the UAP. The UAP then combines necessary session information tracked at the UAP with assistant information collected at the source device (if available), and hands over the combined session information to the target device. Note that in this stage not all session information is handed over to the target device. Therefore, the UAP will hand over more session information to the target device when necessary. After handoff of the session information, the user can continue browsing with the same information that he viewed at the source device. The issues and protocol design of session handoff will be elaborated in Section 5.

3.2. UAP architecture

As shown in Fig. 2, UAP provides the following functional components:

(1) *User Authenticator:* User Authenticator authenticates source or target user devices and browser processes before proceeding to other UAP functionalities. It stores users' authentication information in *User DB*.
(2) *Profile Manager:* Profile Manager stores user profiles such as user preference, device capabilities, and locations of user devices in *User Profile DB*.
(3) *HTTP Proxy Agent:* HTTP Proxy Agent behaves like a traditional web proxy. It receives browsing requests from registered browsers and responses from web servers, asks Session Manager to deal with the requests and the responses, obtains the requested objects from web servers or *Content Cache*, and then replies user devices with objects adapted by *Content Adapter*.

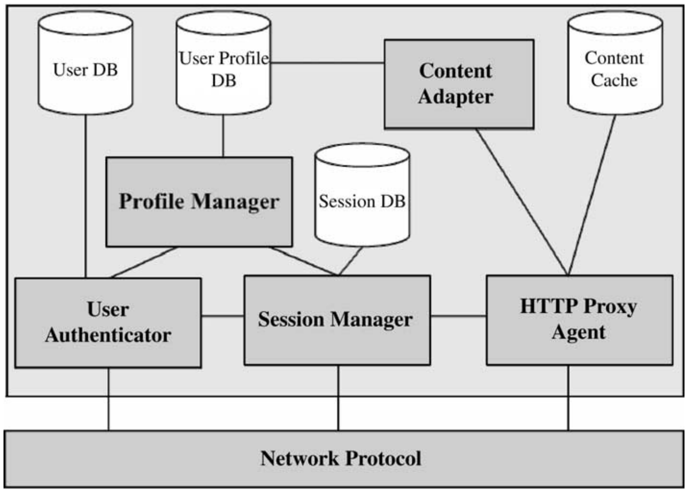


Fig. 2. UAP architecture.

(4) *Session Manager:* Session Manager tracks proxy sessions and adjusts requests and responses during browsing and tracking stage. In addition, Session Manager will be also involved during session handoff stage. During browsing and tracking stage, Session Manager analyzes requests and responses received by HTTP Proxy Agent to track and record session information in *Session DB*. Session Manger also adjusts requests and responses by adding necessary session information in the requests and responses from/to target devices. Moreover, if browsers do not support CCPP [14], Session Manger will add user profiles as CC/PPex [14–16] headers in the requests as well. On the other hand, during session handoff stage Session Manager inter-works with source devices and target devices to accomplish session handoff.

(5) *Content Adapter:* Content Adapter adapts web content to fit a user's profile during session handoff and browsing-and-tracking stages. If web servers can perform all the necessary adaptation, UAPs' functionality of content adaptation can be optional. Otherwise, multiple passes of adaptation by UAPs and web servers would be possible.

4. Proxy session tracking

This section discusses ways to track a proxy session with modified and unmodified user devices respectively and also addresses issues of session tracking in detail. Before going into proxy session tracking, we first clarify usage of the following terms in this paper:

Session information: It consists of any information about a session such as session history, HTTP cookies, web objects maintained by browsers, etc. Intrinsically, web session handoff means handoff of session information. Session information is called Browser State in BSR [3]. In this paper, we use the term session information to avoid confusion between session state and browser state.

Session state: Session state is a subset of session information. It is the information used by web servers for identifying or recording a stateful session, e.g., HTTP cookies and other information discussed in Section 4.2.

Proxy session tracking in our design includes three operations: session history tracking, session state handling, and un-submitted form fields tracking.

4.1. Session history tracking

Session history denotes the chronological sequence of pages browsed by a particular browser process. It can be accessed by a browser's "*back*" and "*forward*" commands/buttons. Handoff of session history enables users to traverse web pages in target devices in the original sequences as they have traversed in source devices. Therefore, it is the basis of session information.

The most intuitive way to obtain session history is from source devices. However, most browsers maintain session history in memory, which cannot be directly and easily accessed without in-depth knowledge about these browsers. Thus obtaining session history requires a modified browser or a program snooping a browser's session history. On the other hand, a UAP can track session history by recording requests and corresponding responses in the requested order. From the two solutions, we choose the latter in order to minimize modification to user devices.

The following sub-subsections further discuss three conditions about session history tracking and their respective treatments.

4.1.1. Multiple browser processes on a user device

For a user device executing multiple simultaneous browser processes, a UAP should be able to differentiate requests from different browser processes to separately track each proxy session. If persist connection is used (RFC2616 [17], Section 8.1), the simplest way to this differentiation is according to each request's source TCP port. However, a browser's connection to a UAP may occasionally terminate and the browser process will then use another TCP port to connect to the UAP. Thus the source port is not sufficient for differentiating requests. As a result, we propose a solution that requires modification to user

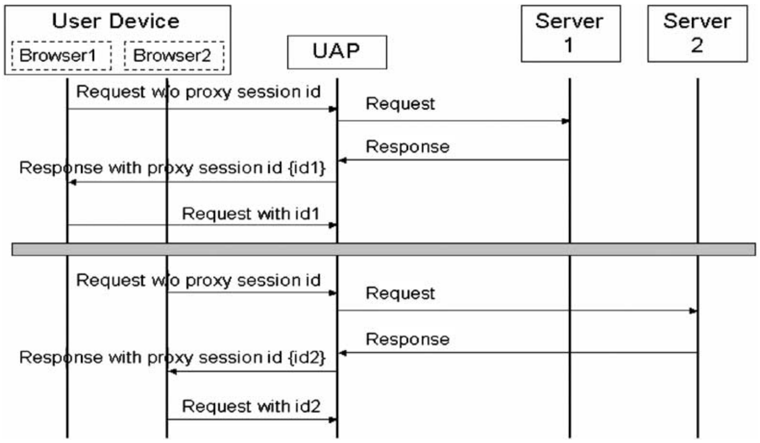

User Device
Browser1
Browser2
UAP
Server 1
Server 2
Request w/o proxy session id
Request
Response
Response with proxy session id {id1}
Request with id1
Request w/o proxy session id
Request
Response
Response with proxy session id {id2}
Request with id2

Fig. 3. Identification of proxy session.

devices and uses *session ids* to identify proxy sessions. Fig. 3 illustrates the flow of using session ids. Initially, browser1 on the user device sends the UAP its first request without proxy session id. Since this request contains no session id, the UAP assigns a unique proxy session id (id1) to the proxy session for this request. Afterward, upon receiving the first request's response from server1, the UAP adds in the response the session id as an extension HTTP header and relays this response to browser1. Browser1 will then tag all subsequent requests with this id so that the UAP can recognize these requests are sent from the same browser process. Afterward, when browser2 on the same device sends its first request without proxy session id to the UAP, the UAP will assign another unique proxy session id to the proxy session for the request. As a result, different browser processes are assigned different proxy session ids, and therefore proxy sessions can be easily differentiated. In our implementation, we make no modifications to the browsers. Instead, we plug in a client program to intercept and store session ids when a user device receives responses. Consequently, our client program appends session id in the requests when a user device sends requests.

4.1.2. HTML document containing other objects

HTML documents may contain other objects, e.g., frames or pictures. For such documents, a browser will send additional requests for the contained objects. In this case, the UAP should exclude these objects from session history. For example, after browsing three objects: A, B, and C sequentially, the session history should be "A, B, C" even though object B contains another two objects: B1 and B2. If a UAP has no information about B's structure and just records all requested objects into session history, the UAP will maintain

Table 1
Treatments for requests with a message body

Conditions	Impacts	
	UAP record message body	Treatment of documents
UAP can adapt content	No	Hand over adapted cached documents
UAP cannot adapt content; method is idempotent	Yes	Hand over re-requested documents from web servers
UAP cannot adapt content; method is not idempotent	No	Hand over un-adapted cached documents

an incorrect session history as "A, B, B1, B2, C". Unfortunately, to recognize the requested document's structure requires parsing of the document, which will increase load to the UAP [18]. To alleviate load of UAPs, our client program will send a UAP assistant information indicating whether a requested object should be recorded into session history.

4.1.3. A request with a message body

An HTTP request may contain a message body (RFC2616 [17], Section 4.3). The meaning and usage of the message body depend on the HTTP method and the requested URL of the request. The message body is necessary when the request will be sent to the original web server again. Therefore, the UAP should record the message body if the UAP will re-request content from the original server after session handoff. Due to the specification of HTTP [17] and CC/PP [14], such recording and re-requesting can occur in our system only when the UAP cannot perform content adaptation to fit for the target device and the HTTP method in the request is idempotent.[1] Table 1 summarizes the impact of a UAP's adaptation capability and HTTP methods on the treatment of documents and message body recording.

4.2. Session state treatment

Originally, HTTP is designed to support only stateless sessions. However, as web browsing become popular and is widely used for more aspect of information exchange, stateful sessions with session state persisting across multiple

[1] An HTTP request with non-idempotent HTTP method (RFC2616 [16], Section 9.1.2) should not be unintentionally re-requested to an original web server since re-performing such a request may result in unwanted update to server contents. According to the specification of HTTP 1.1 [16], the methods GET, HEAD, PUT, DELETE, OPTIONS, and TRACES are idempotent and the method POST is not idempotent. In other words, requests using POST method should not be unintentionally re-performed.

HTTP transactions become important and necessary. With stateful sessions, a web client can send requests with necessary session state so that a web server can identify a session and deal with these requests according to the session state.

Since a stateful session is established between a source device and a web server, the target device initially will not have the session state sent between the source device and the web server. A UAP thus must track session state so that the target device can continue the stateful session after session handoff. Among the approaches [19–22] proposed to support stateful sessions on the original HTTP, our design deals with session state information supported by three extensively used approaches that use hidden form fields, cookies, and URL rewriting. Note that using cookies [20] is different from the other two since transmission of cookies in HTTP is a special portion for stateful sessions under the definition in RFC2965 [20] while the other two approaches can serve for other usage as well as stateful sessions. The following subsubsections describe the three approaches and their respective treatment in our design.

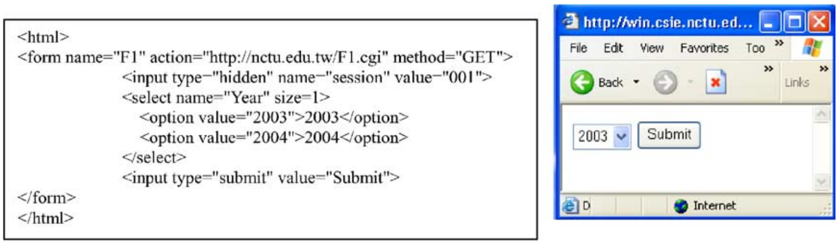
4.2.1. Hidden form fields and URL rewriting

To enable a stateful session, a web server can use hidden form fields (invisible in browser) [20] or URL rewriting [21] to embed session state in web documents. After browsing such web documents, a client thus can send browsing request with necessary session state included either in the requested URL or in the request's message body. Consequently, the web server can identify the session and deal with the request according to appropriate session state.

The following shows the detail of submitting a form containing hidden form fields or using URL rewriting and the corresponding treatment in our design:

(A.1) *A form using GET method and containing hidden form fields:* When submitting such a form, a client includes the hidden form fields as part of the requested URL. Consider submitting an example HTML document, which uses GET method and includes a form containing three form elements: the hidden field "session", the select field "Year", and the button "Submit", as shown in Fig. 4. The client will submit a request for http://nctu.edu.tw/F1.cgi?session=001&Year=2003. Note that the part "session=001" is the embedded session state.

(A.2) *Rewriting URL:* A web server can use URL rewriting to encode session state as part of a requested URL. URL rewriting is a server-side technique which translates the path part of a requested URL to another path. For the example in Fig. 4, if the web server translates its path "session001/F1.cgi" to "F1.cgi?session=001", the server can send a client a different HTML document that embeds session state in the form action field as "http://nctu.edu.tw/session001/F1.cgi" without using hidden form



```
<html>
<form name="F1" action="http://nctu.edu.tw/F1.cgi" method="GET">
                <input type="hidden" name="session" value="001">
                <select name="Year" size=1>
                   <option value="2003">2003</option>
                   <option value="2004">2004</option>
                </select>
                <input type="submit" value="Submit">
</form>
</html>
```

Fig. 4. Form example using GET method and hidden form fields.

field “session”. When submitting such a form, the client sends the server a request for “http://nctu.edu.tw/session001/F1.cgi?Year=2003” and the web server can automatically translate the requested URL to the real one “http://nctu.edu.tw/F1.cgi?session= 001&Year=2003”.

As a result, the UAP performs no special treatment and can automatically record session state embedded by URL rewriting or in hidden form fields of a form using GET method since the UAP should record requested URLs for tracking session history. Besides, the UAP performs no special action and can hand over session state embedded in these two ways when handing over a session’s history.

(B) *A form using POST method and containing hidden form fields:* When submitting such a form, a client appends the hidden form fields in the message body of the request. Consider the same example in Fig. 4, if the form method is POST, the client will submit a request for “http://nctu.edu.tw/F1.cgi” with a message body containing only one line “*session=001 & Year=2003*”.

As a result, tracking of session state embedded in this way needs the same treatment for a request’s message body as stated in Section 4.1.3. The handoff of recorded message body will be described in Section 5.3.

4.2.2. Cookies

To enable a stateful session by using cookies, a web server associates session state with cookies or encodes session state as cookies and then sends the cookies to a client. Afterward, the client will send requests containing these cookies so that the server can deal with the requests according to the session state.

Transmission of cookies in HTTP is a special portion for stateful sessions under the definition in RFC2965 [20]. In a stateful session, when answering a client’s request, a server will assign the client cookies which are associated with certain domains and some paths in these domains. The assigned cookies are then sent to the client by using “Set-Cookie2” or “Set-Cookie” HTTP header in the server’s response and are stored in the client. Afterward, when the client wishes to access objects in domains and paths which have been associated with client's stored cookies, the client will send requests containing the associated cookies in "Cookie" HTTP header in requests.

As a result, to track a stateful session using cookies, a UAP should handle "Set-Cookie2" or "Set-Cookie" headers in web servers' responses, record cookies in the Session DB, and hand over recorded cookies to the target device when handing over the session. However, sending all cookies of a session to the target device may be unnecessary and slow for a mobile device. As a result, in our design the UAP will not send any cookie to the target device upon session handoff. Instead, the target device obtains necessary cookies during browsing and tacking stage. In this stage, the target device sends requests without cookies, and the UAP should append cookies in these requests and the corresponding responses if necessary. The detail will be described in Section 5.3.

4.3. Un-submitted form fields tracking

Handoff of user input in un-submitted form fields provides users convenience to avoid tedious refilling when they hand over devices during filling forms. In normal browsing operation, values of form fields are sent to a UAP only after form submission and therefore the UAP has no way to receive and track user input in un-submitted form fields without assistance from modified user devices. In our design, we adopt a client program to snoop the maintained objects in browsers for the current-viewed HTML files, to find input fields, and to collect these fields' inputted values. The collected data will then be requested by the UAP during session handoff stage. The detail operation will be described in Section 5.1.

4.4. Violation of RFC's cache-control directives

In our design, UAPs may store responses and cookies for the purpose of session tracking. However, this storing operation may violate the specifications in RFC2616 [17] and RFC2695 [20] if a response is specified as non-cacheable by using "private", "no-cache", or "no-store" directives in "Cache-Control" header (RFC2616 [17], Section 14.9). These directives are originally used for preventing reveal of private information when proxies reply with cached data to subsequent requests from other user devices. Thus, in order to protect the private information such as personalized data and session state, a user device and a UAP should establish security association. Fortunately, there have been many application layer, network layer, and link layer approaches that can establish the required secure association [23–25]. Besides, a UAP must separately maintain private information and must not provide the private information of a user to other users. Moreover, the UAP should delete the private information when a proxy session terminates.

5. Stateful session handoff

Fig. 5 shows the message flow of our session handoff protocol. In this protocol, the participants at a user device can be an unmodified browser or our client program (if it has been installed in the user device). Initially, several source devices have ongoing registered sessions tracked by the UAP. At Step 1, the target device sends a handoff request to the UAP after registering to the UAP. At Step 2, the UAP replies a list of the user's registered sessions to the target device. At Step 3, the target device requests to hand over a selected session. If the source device is unreachable or without installing our client program, Step 4 will be skipped. Otherwise, at Step 4, from the selected source device the UAP obtains partial information of the selected session collected by our client program. At Step 5, the UAP sends the target device necessary session information that the UAP tracks as well as that collected by our client program. In the current state, our implementation only hands over the session history in order to avoid handing over unnecessary information that is not immediately used by the target device. If necessary, other part of the session information will be sent to the target device together with subsequent browsing responses. At Step 6, the target device sends the UAP a browsing request for a page in the session, e.g., the last-viewed page. At Steps 7, if necessary, the UAP adjusts the browsing request by adding the user profile as CC/PPex [14–16] header and session information containing the message body and cookies that are previously recorded by the UAP and associated with objects accessed in this browsing request. The adjusted request is then sent to the web server. At Step 8, the UAP receives a response from the web server. Steps 7 and 8 can

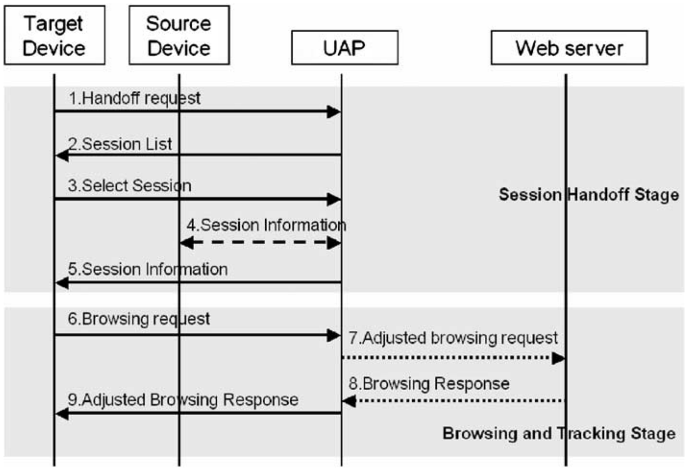


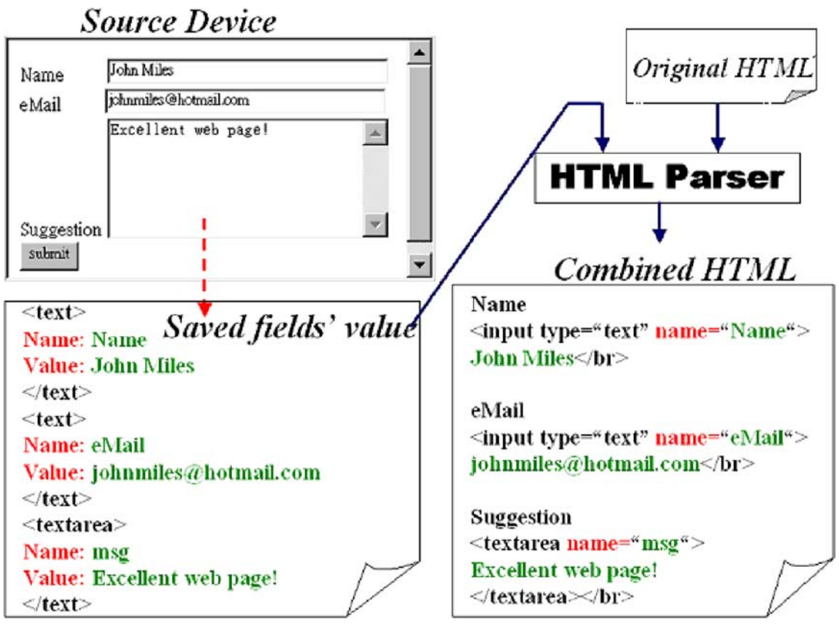
Fig. 5. Session handoff protocol message flow.

be optional if the UAP can perform content adaptation by using cached files. At Step 9, the UAP adjusts the response by appending necessary cookies and relays the adjusted response to the target device. Note that Steps 6–9 will repeat for every browsing request sent by the target device.

The following subsections describe handoff of user input in un-submitted form fields, web pages, recorded cookies, and recorded message bodies in detail.

5.1. Handoff of user input in un-submitted form fields

As stated in Section 4.3, user input in un-submitted form fields can be handed over only using modified user devices. In our design, our client program assists the UAP in handing over user input in un-submitted form fields. During browsing and tracking stage, the client program snoops the maintained objects in browsers and collects user input in un-submitted form fields as shown in the left of Fig. 6. The UAP then obtains the collected user input as part of session information during Step 4 of Fig. 5. Afterward, when the target device requests an HTML document containing un-submitted form fields (Step 6 of Fig. 5), the UAP combines un-submitted form fields into the HTML



```
John Miles
johnmiles@hotmail.com
Excellent web page!
```

```
<text>
Name: Name
Value: John Miles
</text>
<text>
Name: eMail
Value: johnmiles@hotmail.com
</text>
<textarea>
Name: msg
Value: Excellent web page!
</text>
```

```
Name
<input type="text" name="Name">
John Miles</br>

eMail
<input type="text" name="eMail">
johnmiles@hotmail.com</br>

Suggestion
<textarea name="msg">
Excellent web page!
</textarea></br>
```

Fig. 6. Combination of user input.

document before replying the target device (Step 9 of Fig. 5). The combination flow is shown in the right of Fig. 6. The UAP obtains the HTML document from the UAP's Content Cache or the original web server, parses the HTML document, and combines collected user input into corresponding fields according to each field's name. Note that the program code for snooping different browsers' maintained objects is browser-dependent. In the current state, we implement only the client programs for Microsoft's IE of desktop Windows and Pocket PC.

5.2. Handoff of a web page

Intuitively, a web page can be handed over in form of its content or URL. Handing over real content data is a convenient and fast choice since the browser on the target device is unnecessary to send another browsing request in addition to the handoff request. However, in this way our client program must save the content data of the page and objects contained in this page as local files, parse the page, change references of the contained objects in the page into locally saved files, and notify the browser to open the locally saved file of the page. As a result, the browser will display the locally saved file's name as the page's URL instead of the original network URL. Furthermore, the extra parsing and reference changing will generate extra load to user devices.

On the contrary, handing over URL is unnatural and slower since a browser needs to send additional requests to obtain real content data from the UAP. These requests result in extra messages exchanged between a user device and the UAP. However, this way makes the browser on a target device display the original network URL as the page's URL and frees the target device from the extra parsing and reference changing. Thus, our design adopts handoff of a web page in the form of its URL.

In our design, handoff of a web page occurs only at the end of session handoff stage, while the content of this page is requested during browsing and tracking stage. At Step 5 of Fig. 5, the UAP sends the target device the list of URLs in the session history and also redirects the target device to open the last-viewed page that we reasonably assume the user would like to view first after session handoff. With the redirection, the user device will automatically request the last-viewed page from the UAP without the user's instruction (at Step 6 of Fig. 5).

5.3. Handoff of recorded cookies and message bodies

As we have stated previously, to avoid handing over session information that is not immediately used, our session handoff protocol does not send a target device cookies and message bodies associated with a session and recorded by a UAP during session handoff stage. Instead, we send a target device session information associated with a particular object only when the target device requests this object.

The handoff of recorded cookies and message bodies of a session is performed during the browsing and tracking stage of Fig. 5. At Step 6, the target device sends a browsing request that may not contain the cookies and message body necessary for accessing the requested object. At Step 7, if necessary, the UAP appends in the browsing request the cookies and message body for accessing the requested object and forwards the adjusted browsing request to the web server. At Step 8, the web server replies the requested object to the UAP. Steps 7 and 8 can be optional if the UAP can perform content adaptation by using cached files. At Step 9, the UAP appends in the received response the cookies that it appends at Step 7 and that are associated with the object in the received response, and then forwards the adjusted response to the target device. Afterward, subsequent requests from the target device will contain these cookies. Therefore, the UAP need not append these cookies again until the session is handed over again. It should be noted that at Step 9 a recorded message body is not included in the adjusted response since the standard HTTP protocol cannot afford to send a message body in the reverse direction. Although the target device cannot obtain the message body, the target device is probably able to proceed to browse without the message body since the same browsing request of the target device can be satisfied by the content cache of the target device or the UAP.

5.4. Summary

Table 2 summarizes differences between BSR [3] and our approaches with and without modification to user devices. Our approach adopts a proxy-based approach which is inherently different to the client-based approach adopted by BSR. Besides the basic differences between proxy-based approaches and client-based approaches, our approach has two main advantages over BSR service:

Handoff Action: BSR service requires source devices to explicitly save browser state. In contrast, our design has the flexibility that a user can perform session handoff when he leaves his source device without performing a saving action.

History and Cookies Handoff: The style of handoff performed by BSR service requires transmission of much session information and page content from a source device to a BSR repository server and then from a BSR repository server to a target device. Such transmission may be unnecessary since a user may not re-visit all the web pages of a session. Although BSR service provides adjustable limit on the history size, such limitation may be impractical since a user may need different limits for different sessions. In order to

Table 2
Differences between BSR and our approaches

Item	Approach		
	BSR	Our approach	
		Without modification	With modification
Modification to user devices	Substantially modified browser	No	Install a small client program
Session registration	Unnecessary	By accessing UAP's web pages	By accessing UAP's web pages; or by the client program
Tracking session information	Unnecessary	By UAP; Unable to track un-submitted form fields	By UAP; Un-submitted form fields tracked by client program
Handoff action	Source saves to BSR repository and then target retrieves from it	Target retrieves from UAP	UAP retrieves un-submitted form fields from source; Target retrieves from UAP
History handoff	Directly loaded into browser	Sent as a list of web pages	Sent as a list of web pages
Web pages handoff	Content is directly loaded into browser	UAP redirects browser to open URLs	UAP sends URLs to client program and then browser opens the URLs
Cookie handoff	All cookies are sent to browser	UAP sends necessary cookies to browser during browsing and tracking step	UAP sends necessary cookies to browser during browsing and tracking step

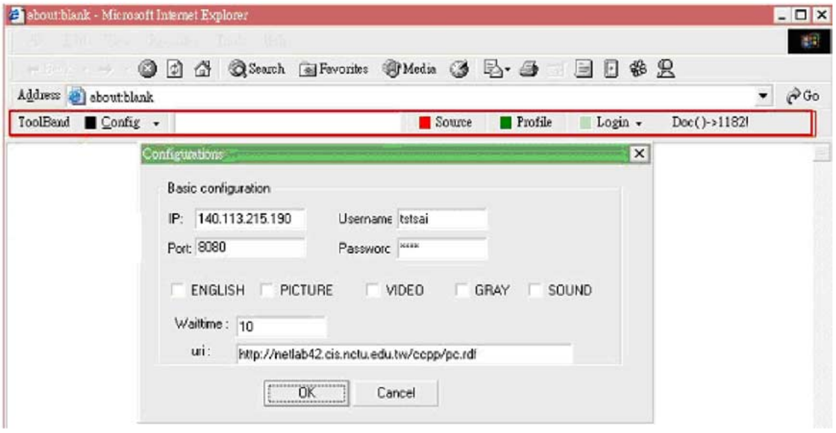
reduce network traffic, our protocol only hands over URLs of a session so that a user can follow the URLs to re-visit pages of the session.

6. Implementation and example

This section shows our implementation. We have implemented a UAP on a Microsoft's Windows 2000 platform and client programs for performing session registration and handoff for Microsoft's Windows PC and Pocket PC PDA. Besides, Our system has the flexibility that user devices without our client programs can perform session registration and handoff by accessing web pages on the UAP.

The client program for Microsoft's Windows PC is implemented as a plug-in in IE, which is the rectangle toolbar under the address bar in Fig. 7. Using the program, users can perform session registration and handoff directly in IE's main window. The configuration window of this program is shown in the center of Fig. 7. This window enables a user to input his username, password, IP address of a UAP, port number on the UAP for accessing the UAP's functionalities, basic user preferences for adaptive browsing (the options: ENGLISH, PICTURE, VIDEO, GRAY, SOUND, and Waiting-time), and URL of user profile.

The client program for Microsoft's Pocket PC PDA provides the same functionalities as a PC's client program except that the PDA client program is a standalone one instead of a plug-in. We do not show it for saving space.



Address: about:blank

Doc()->11821

Basic configuration

IP: 140.113.215.190 Username: tstsai

Port: 8080 Password: ****

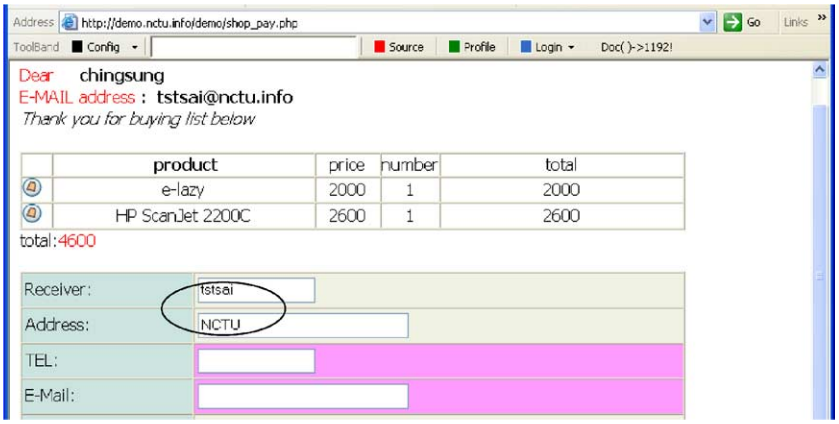
ENGLISH PICTURE VIDEO GRAY SOUND

Waittime: 10

url: http://netlab42.cis.nctu.edu.tw/ccpp/pc.rdf

Fig. 7. PC client program.

The following demonstrates handing over a session for online shopping by using the implemented system. As shown in Fig. 8, a user first uses a PC to visit an online shop, places his order, and inputs two form fields: Receiver and Address. Afterward, the user uses the client program on the PDA to hand over the session (Fig. 9(a)), and then the session resumes at the last-viewed page with the user input. That is, the user inputs in the inputted form fields are also handed over (Fig. 9(b)). The user then inputs another form field: TEL (Fig. 9(c)). Finally, the session is handed over back to the PC with the user input in the form field: TEL handed over (Fig. 10).


Address http://demo.nctu.info/demo/shop_pay.php Go Links
ToolBand Config Source Profile Login Doc( )->1192!

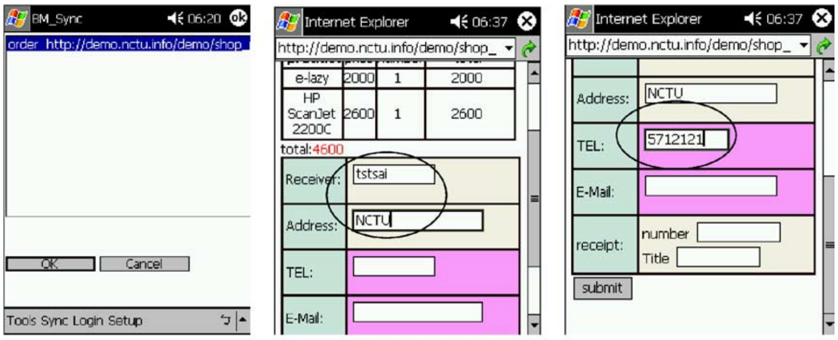
Dear chingsung
E-MAIL address : tstsai@nctu.info
Thank you for buying list below

	product	price	number	total
	e-lazy	2000	1	2000
	HP ScanJet 2200C	2600	1	2600

total:4600

Receiver:	tstsai
Address:	NCTU
TEL:	
E-Mail:	

Fig. 8. A session at a PC.



order http://demo.nctu.info/demo/shop

e-lazy	2000	1	2000
HP ScanJet 2200C	2600	1	2600

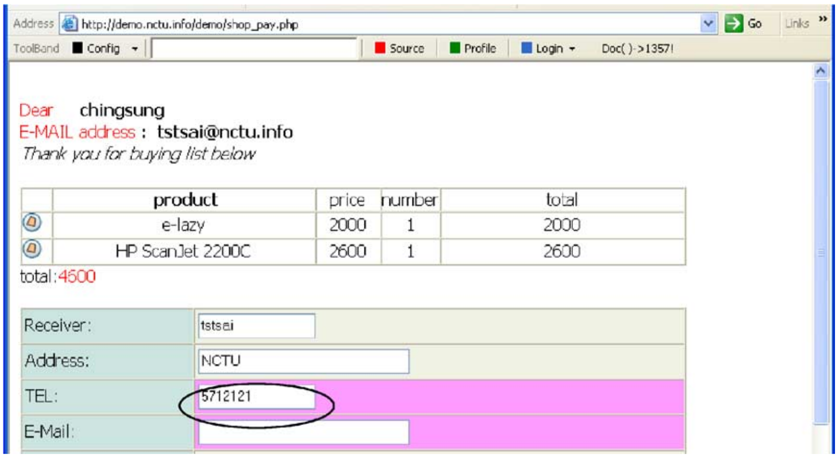
total:4600

Receiver:	tstsai
Address:	NCTU
TEL:	
E-Mail:	

Address:	NCTU
TEL:	5712121
E-Mail:	
receipt:	number Title

(a) (b) (c)

Fig. 9. Session handoff to PDA.



http://demo.nctu.info/demo/shop_pay.php

Dear chingsung
E-MAIL address : tstsai@nctu.info
Thank you for buying list below

	product	price	number	total
	e-lazy	2000	1	2000
	HP ScanJet 2200C	2600	1	2600

total:4500

Receiver:	tstsai
Address:	NCTU
TEL:	5712121
E-Mail:	

Fig. 10. Session handoff back to the PC.

7. Conclusion and future work

This paper proposes a web session handoff system that adopts a proxy-based approach with optional assistant information from user devices and discusses ways to track and hand over a variety of session information in detail. Using proxy-based approaches for session tracking and handoff has several advantages over using client-based approaches or server-based approaches. The advantages include less modification to user devices, practicability, and fault tolerance. During handoff, a source device may be unreachable due to lack of battery or wireless connection. In this case, a proxy-based approach can hand over a session because this session has been tracked by a proxy while the user is browsing. Besides, in order to hand over session information that can not be tracked by a proxy, we let the less modified user devices collect session information for assisting the proxy in session tracking. We have implemented a special proxy—UAP and client programs for PC and PDA to fulfill the session handoff system. The implementation can successfully hand over not only stateless but also stateful sessions between homogeneous or heterogeneous user devices.

There is still further improvement to our implementation. First, we will improve our client program to collect more assistant information so that the UAP can be more light-weighted and session information can be handed over completely. Besides, since session tracking generates additional processing and storage load to UAPs, we will perform a quantitative analysis on workload and storage space problem of UAPs. Furthermore, we will also apply and refer previous research in scalable hierarchical proxies to design a system that contains multiple co-operated UAPs.

Acknowledgement

This work was supported in part by the National Science Council, Republic of China, under Grants NSC 93-2219-E-009-001, NSC 93-2219-E-009-004, and NSC 93-2213-E-126-002.

References

[1] H. Schulzrinne, E. Wedlund, Application-layer mobility using SIP, in: Proceedings of IEEE Conference on Service Portability and Virtual Customer Environments, 2000, pp. 29–36.
[2] R. Bagrodia, T. Phan, R. Guy, A scalable distributed middleware service architecture to support mobile internet applications in wireless networks, Journal of Mobile Communication, Computation, and Information (WINET) 9 (4) (2003) 311–320.
[3] H. Song, H.-H Chu, N. Islam, S. Kurakake, M. Katagiri, Browser state repository service, in: Proceedings of International Conference on Pervasive Computing, 2002, pp. 253–266.
[4] C. Perkins, IP Mobility Support for IPv4, IETF Internet draft, June 2004.
[5] R. Jan, et al., Enhancing survivability of mobile internet access using mobile IP with location registers, IEEE Infocom 1 (1999) 3.
[6] A. Di Stefano, C. Santoro, NetChaser: agent support for personal mobility, IEEE Internet Computing 4 (2) (2000) 74–79.
[7] M. Roussopoulos, et al., Person-level routing in the mobile people architecture, in: Proceedings of the USENIX Symposium on Internet Technologies and Systems, 1999, pp. 165–176.
[8] Appenzeller et al., The mobile people architecture, Technical Report CSL-TR-99-777, Stanford University, 1999.
[9] R. Liscano et al., Integrating multi-modal messages across heterogeneous networks, in: Proceedings of ENM-97 In conjunction with the ICC-97, 1997, pp. 45–53.
[10] C.-H. Herman et al., iMobile: a proxy-based platform for mobile services, in: Proceedings of the first workshop on Wireless mobile internet, 2001, pp. 3–10.
[11] A.-P. Kanerva et al., P920 deliverable 1: VHE concept description, scenarios and protocols, 2000.
[12] K. Raatikainen, Middleware for future mobile networks, in: Proceedings of IEEE International Conference on 3G Wireless and Beyond, 2001, pp. 722–727.
[13] T. Richardson, et al., Virtual network computing, IEEE Internet Computing 2 (1) (1998) 33–38.
[14] M. Nilsson, J. Hjelm, H. Ohto, Composite capabilities/preference profiles: requirements and architecture, W3C Working Draft, 2000. Available from <http://www.w3.org/TR/2000/WD-CCPP-ra-20000721/>.
[15] F. Reynolds, C. Woodrow, H. Ohto, Composite capability/preference profiles (CC/PP): structure and vocabularies, W3C Working Draft, 2003. Available from <http://www.w3.org/TR/2003/WD-CCPP-struct-vocab-20030325/>.
[16] H. Ohto, J. Hjelm, CC/PP exchange protocol based on HTTP extension framework, W3C Note, 1999. Available from <http://www.w3.org/TR/NOTE-CCPPexchange>.
[17] R. Fielding et al., Hypertext transfer protocol—HTTP/1.1, IETF RFC2616, 1999.
[18] T.C. Sung, Inter-device handoff for WWW service, Master Thesis, Department of CSIE of National Chiao-Tung University, 2002.
[19] K. Moore, N. Freed, Use of HTTP state management, IETF RFC2964, 2000.
[20] D. Kristol, L. Montulli, HTTP state management mechanism, IETF RFC2965, 2000.

[21] J. Newmarch, HTTP session management, Electronic Commerce Technical Issues. Available from <http://jan.netcomp.monash.edu.au/ecommerce/session.html>.
[22] Netscape Support Documentation, Persistent client state—HTTP cookies. Available from: <http://wp.netscape.com/newsref/std/cookie_spec.html>.
[23] S. Kent, R. Atkinson, Security architecture for the internet protocol, IETF RFC2401, 1998.
[24] T. Dierks, C. Allen, The TLS Protocol Version 1.0, IETF RFC2246, 1999.
[25] E. Rescorla, A. Schiffman, The secure hypertext transfer protocol, IETF RFC2660, 1999.