JOURNAL OF KOREA MULTIMEDIA SOCIETY, VOL. 7, NO. 12, DECEMBER 2004(pp. 1754-1764)

An Effective Pre-refresh Mechanism for Embedded Web Browser of Mobile Handheld Devices

Huaqiang Li[†], Young-Hak Kim[††], Tae-Hyong Kim[†††]

ABSTRACT

Lately mobile handheld devices such as Personal Digital Assistant (PDA) and cellular phones are getting more popular for personal web surfing. However, today most mobile handheld devices have relatively poor web browsing capability due to their low performance so their users have to suffer longer communication latency than those of desktop Personal Computers (PCs). In this paper, we propose an effective pre-refresh mechanism for embedded web browser of mobile handheld devices to reduce this problem. The proposed mechanism uses the idle time to pre-refresh the expired web objects in an embedded web browser's cache memory. It increases the utilization of Central Processing Unit (CPU) power and network bandwidth during the idle time and consequently reduces the client's latency and web browsing cost. An experiment was done using a simulator designed by us to evaluate the efficacy of the proposed mechanism. The experiment result demonstrates that it has a good performance to make web surfing faster.

Keywords: Pre-refresh mechanism, embedded web browser, mobile handheld devices

1. INTRODUCTION

Recent years, as World-Wide Web (WWW) technologies and mobile handheld devices have had a very fast development, mobile handheld devices such as Personal Digital Assistant (PDA) and cellular phones are much more popular for personal web surfing. Most current mobile handheld devices have been integrated with embedded web browser for surfing the internet, and users can access the internet from anywhere at any time. Furthermore, in the future, the mobile handheld devices will greatly replace desktop Personal Computers (PCs) for accessing the internet.

※ Corresponding Author : Tae-Hyong Kim, Address : (730-701) 1, Yangho-dong, Gumi, Gyeongbuk, Korea, TEL : +82-54-478-7528, FAX : +82-54-478-7539
E-mail : taehyong@kumoh.ac.kr
Receipt date : Feb. 9, 2004, Approval date : May 31, 2004
[†] China R&D Center, LG Electronics
(E-mail : huaqiang@lge.com)
[††] Kumoh National Institute of Technology
(E-mail : kimyh@kumoh.ac.kr)
[†††] Kumoh National Institute of Technology
※ This paper was supported in 2003 by Research Fund, Kumoh National Institute of Technology.

However, today most mobile handheld devices have relatively poor web browsing capability because of the slow bandwidth of wireless connection, the slow speed of Central Processing Unit (CPU), the low capacity of Random Access Memory (RAM) and flash Read Only Memory (ROM), the small display size of Liquid Crystal Display (LCD), the short life-cycle of battery and the deformity of embedded web browser. Therefore, users have to suffer longer internet communication latency than those of desktop PCs. In addition, embedded web browsers have the relatively poor web browsing ability because of the defects of the hardware used in mobile handheld devices. The deformity of most embedded web browsers not only increases the internet communication latency but also induces the poor vision effect of internet surfing.

These hardware and embedded web browser's problems bring long internet communication latency when users access the internet. To tackle this problem, we propose an effective pre-refresh mechanism for embedded web browsers of such

mobile handheld devices. In this paper we first summarize the existing technologies for reducing client latency, and indicate their advantages and disadvantages, then we propose an effective pre-refresh mechanism for mobile handheld device's embedded web browser to reduce the client communication latency and make user's web surfing faster. The experiment was done using a simulator designed by us and we analyze the experiment results to evaluate the ability of reducing client communication latency of this mechanism.

The structure of this paper is as follows. In section 2, the existing technologies are explained for reducing client communication latency, and we indicate their advantages and disadvantages. The proposed mechanism is explained in section 3 with its approach and basic assumptions. Section 4 explains the simulation process and performance analysis: the performance metrics, the implementation method of the simulator, and the analysis of the simulation results. Finally the conclusions and the future work are given in section 5.

2. RELATED WORK

Ever since WWW emerged, reducing client latency has been one of the primary concerns of the Internet research and development community. A lot of techniques were proposed for reducing client latency as follows.

Prefetching between caching proxies and browsers[1] is a well-known technique for reducing latency for modem users. Because the low modem bandwidth is a primary contributor to client latency, this approach relies on the proxy to predict which cached documents a user might reference next, and takes advantage of the idle time between user requests to push or pull the documents to the user. The authors said this approach can reduce user perceived latency up to 23.4%. But this approach is not supported by most current Internet Service Providers (ISPs) and Local Area Network (LAN) proxies, because it is very difficulty to realize and imposes a big process burden for proxies.

Large browser cache is another technique to reduce client latency[1]. It increases the hit ratio and reduces network traffic. Infinite browser cache (almost no replacement occurs) is supposed in the experiment[1]. The result shows it reduces 4.1% client latency. But this approach is absolutely not fit for mobile handheld devices. Taking PDA as an example, general desktop PC's browser cache has the a default size of 5~8MB[1]. But most current PDA products have the system memory from 8MB to 32MB and Flash ROM from 2MB to 32MB. It is impossible to allocate a big browser cache to PDA.

Delta compression technique[2] only transfers modified web pages between the proxy and client. That is, if an old copy of the modified page exists in the browser cache, the proxy only sends the difference between the latest version and the old version. The authors said this approach can eliminate 88% of bytes in transfers of modified objects[2]. But until now, this technique is not an official standard for an extension of HyperText Transfer Protocol (HTTP)[3]. Most current web servers, ISPs, proxies and clients do not support it. So mobile handheld device which uses wireless modem for web surfing can not get benefits from delta compression technique.

Application-level compression to HyperText Markup Language (HTML) document was investigated in the literature[2,4]. It has suggested that HTML texts can be first compressed, and then transferred from one end to another. HTTP/1.1 supports application-level compression via the *transfer-encoding* tag[5]. This technique can eliminate 25% of bytes in transfers of HTML documents according to the average number reported [2]. But this technique needs CPU overhead for compression and decompression. It is not practical for slow speed CPU used by mobile handheld device. It will aggravate the burden of CPU and memory of mobile handheld devices during web

surfing and may result in more latency.

3. THE PROPOSED MECHANISM

We have summarized the current technologies for reducing client latency and indicated their advantages and disadvantages in section 2. Because those techniques, however, are not fit for reducing client latency for mobile handheld devices, a new effective pre-refresh mechanism is required for mobile handheld device's embedded web browser.

3.1 The Approach

After having plentifully observed users' web browsing, we found they usually have a common habit in their web surfing. It means every user has his some favorite web addresses. For example, somebody likes Yahoo, Google, Microsoft, etc. From the beginning of browsing, users always open one of his favorite web pages and find the information which he wants. During a long period, users always mainly concentrate on some favorite web pages. After a long period, the internet web objects (please refer to Table 1) saved in a web browser cache are those which the user always browses. So the favorite web objects of most users are saved in the web browser's cache.

Furthermore, most internet web objects saved in web browser's cache have the expiry time defined by an HTTP Header, *Expires*. The expiry time of internet web objects means their lifetime. After that time, if the user has a request to this web object, the browser has to refresh or reload this web object. The web object's expiry mechanism is mainly used to judge if the web objects are fresh or stale, in other words, if it is necessary to send a request to the remote web server for reloading and validating these web objects. Some web objects are not assigned the expiry time. In this case some browsers will calculate a heuristic expiry time for those web objects with their HTTP

Table 1. Internet web object type

Internet Web Object Type
Microsoft HTML Document 5.0
Jpeg, gif, jpg, png, ico, xbm,
Cabinet
Cascading Style Sheet Document
Common Gateway Interface (CGI)
.htc file
HTML@TOP
INC file
Javascript
Microsoft Powerpoint, Word document and other MS-DOS Application
PHP, PHP3, ASP, PHTML, SHTML, XML Document
Shockwave Flash Object
Coockie
Others

headers, *Last Modified Time* and *Max-age*. Let's see an example, when you open the 'Temporary Internet Files' folder in the Microsoft Windows, you will see every web object saved in this cache has eight attributes, one of which is *Expiration Date*. You can see certain expiry time for some web objects. But note that the time in expiration date attribute is Greenwich Mean Time (GMT), not local time[6].

We have observed 7279 web objects cached in 'Temporary Internet Files' folder. Image objects such as gif, jpeg, and jpg have generally the expiry time from one month to a year, the Cascading Style Sheet (CSS) documents from several days to one year, the Javascript files from several hours to one year, and the HTML document objects from several hours to several days. Especially, most all-around commercial web sites continuously refresh their web pages. For example, "www.chosun.com" is the web address of a famous Korean news web site; the web pages of this web site are refreshed many times for an hour. For both mobile handheld device's users and desktop PC's users, whenever users open the browser and surf the internet, almost all the HTML document objects are saved and many other objects in the browser cache are expired. And for every request,

a browser mainly executes the process shown in Fig. 1[7]. For every expired web object in the cache, the browser will use the HTTP headers *Last-Modified* and *Etag* to validate this object in the web server[6]. If that object is not changed, the browser will send it directly to the user; otherwise, the browser will refresh that object.

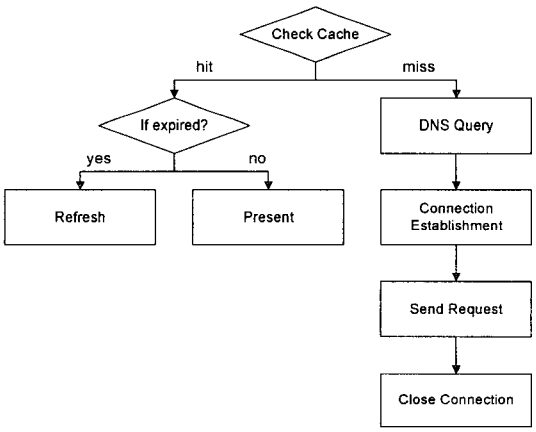


Fig. 1. Steps for every request in a browser.

In addition, during the user's reading web documents, there is much idle time wasted. This is a huge waste because the mobile handheld device's users should pay according to the time in use when they surf the internet. Furthermore, since most mobile handheld devices have relatively poor hardware performance and weak embedded web browser, the user has to spend a lot of time impatiently waiting for web pages to come up on the screen. Therefore, we use the idle time to pre-refresh the expired web objects in an embedded web browser's cache. It means the embedded web browser will automatically refresh the expired web objects in the cache during the idle time. After the user reads some documents, if he or she requests the web objects which have been already pre-refreshed in the cache, the web objects will be displayed very fast to the user. Furthermore this mechanism will increase the utilization of CPU power and network bandwidth during the idle time, so it will consequently reduce the client's latency and web browsing cost.

The proposed pre-refresh mechanism helps mobile handheld device's embedded web browser automatically detect the idle time during user's web surfing and automatically refresh the expired web objects in the browser cache during the idle time. The refresh sequence of the expired web objects abides by the predefined rule.

3.2 Basic Assumptions

Before explanation of the details of the pre-refresh mechanism, some assumptions have to be made. The proposed mechanism is effective if and only if these assumptions are satisfied. The basic assumptions on the pre-refresh mechanism are:

- Users have the idle time between the requests, because users often read some parts of one web page's document before jumping to the next one. This idle time varies greatly from several seconds to several minutes, which depends on the user and the size of web page's documents.
- The browser can predict which web pages a user will access in the near future based on the number of references for every cached object observed from the past period. We will make a rule to automatically pre-refresh the expired web objects with the high access possibility during the idle time.
- The browser has cache memory that holds the user's favorite web pages. Furthermore, since most mobile handheld devices have small flash ROM, it can not allocate much memory capacity for embedded web browser's cache, and most commercial web sites make their web pages so big; for example, the full content of Yahoo main page is 128KB including HTML documents and other web objects such as gif, jpg, Javascript, Flash, etc. It is impossible for most embedded web browsers to support and cache all the web objects. But the HTML document can be cached in any embedded web browser. So we assume only HTML doc-

uments are pre-refreshed. It is reasonable because the text information in the HTML documents is the most important for users in general. Furthermore, if an HTML document is pre-refreshed, other web objects embedded in this HTML document may be also refreshed together.

Fig. 2 shows the cumulative distribution of user idle time in the UC Berkeley (UCB) traces[1]. In the literature[1], the authors estimated the idle time by calculating the difference between the estimated end-of-transmission of one request to the start of the next request from the same user. Fig. 2 also includes the idle time distribution observed from two other LAN web traces, DEC[8] and Pisa[9]. The curves are similar to each other and show that there is inherent idle time between user requests; about 40% of the requests are preceded by 2 to 128 seconds of idle time, indicating plenty of pre-refresh opportunities.

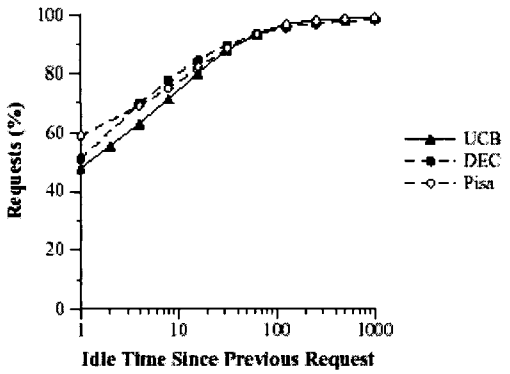


Fig. 2. Cumulative distribution of idle time between user requests[1].

3.3 The Pre-refresh Mechanism

The basic pre-refresh mechanism is as follows. First, an attribute named *reference number* is allocated to the cached HTML document object in the embedded browser's cache. *Reference number* means the number of reference for every cached HTML document in the past period. In most web browser caches, every cached web object has several attributes. For example, Microsoft Internet Explorer's cache has 8 attributes: size, name, internet address, type, expiration date, last modified date, last accessed date, and last validated date. We add *reference number* attribute to the web browser cache. This new attribute is most important in our design, because pre-refresh, sort, and deletion of the cached HTML documents are decided by this attribute. We will use this attribute as a primary index to sort the cached HTML documents in the embedded web browser. If a HTML document object is cached for the first time, we assign 1 to its *reference number*. Later, if the user requests this HTML document again and this HTML document is still in the cache, we increase its value by 1. We believe, after a long period, among cached HTML documents the user's favorite web pages will have high *reference number*. Such cached HTML documents have the high access frequencies, and possibly they will be requested again in the near future.

Second, the attribute *reference number* is used as the primary index, and the attribute *size* as the second index to sort all cached HTML document objects. The attribute *size* is a default attribute in most browsers' cache and indicates the size of the cached HTML document. During sorting, we use *reference number* to arrange the cached HTML documents in descending order. When *reference number* is same, the one which has relatively small *size* value will be sorted ahead. The reason *size* is used as the second index is, we think, cached HTML documents with smaller size have higher possibilities to be successfully pre-refreshed during the idle time. Accordingly, when several cached HTML documents have the same *reference number*, we select the one with the smallest *size* to be pre-refreshed during the idle time.

Third, the Least Frequently Used (LFU) removal policy is used with the attributes *reference number* and *size* to remove HTML documents when the cache is saturated. Because most mobile handheld devices have smaller cache than desktop PCs, the cache can be often saturated and then

some web objects should be removed to make room for the coming one. Among many kinds of removal policies for the network cache[10], we use the LFU policy because it is reasonable to remove the document with the lowest *reference number*. As a result, LFU algorithm can remove the cached HTML documents using this primary key. When several cached HTML documents have the same *reference number*, we will choose one with the biggest *size* among them, because removing big-sized documents can decrease the number of removal operations. To sum up, the HTML documents with low *reference number* value and big *size* value will be removed continuously until the cache has enough room for the incoming HTML document.

Last, a background process is provided to pre-refresh the expired HTML documents in the cache during the idle time. This pre-refresh background process runs while the browser is open. In a general situation, when a user opens the embedded web browser, he will input one web page's address in the browser's address edit box, and then request it. In this case, the pre-refresh background process will start when this request is finished and the browser become idle. In another situation, if a user opens his embedded web browser and left it there, there is no request. In this case, after the user opens his browser, the pre-refresh background process will start when the browser become idle. Then, how can we decide the browser became idle? Through plentiful observing, we found some condition as follows. After a user opens a web page, he will look through its contents and search for the information he wants. For example, when a user opens the Yahoo main web page, he will look through the page. When finding out the information he wants, he will click the hyperlink of that information to see its detailed contents. In this case, the time used for looking through a web page or seeing the detailed contents of a web page will generally exceed 10 seconds. If a user is very familiar with a web page, after opening the web page, he will probably open another hyperlink soon for the information he wants. For example, a user who is familiar to Yahoo web page often directly clicks the mail box hyperlink to see his new mails. In this case, the time from opening a web page to opening another hyperlink in the web page will not exceed 10 seconds.

According to this observation, we decide the idle time judgment condition as 10 seconds. It means if a user has no request within 10 seconds, this pre-refresh background process will refresh the expired cached HTML documents continuously according to the order decided in the second step. Another problem we have to mention is that during pre-refreshing, if the user has a request, the pre-refresh process should be stopped unless the request is for the object that is being pre-refreshed. Because there are generally a lot of expired HTML documents in the cache, continuous pre-refreshing those expired HTML documents needs much time; after a user finishes reading some contents, he may have another request. Therefore, we have to stop the pre-refresh process in order not to affect the user's web surfing. Fig. 3 shows the flowchart representation of the proposed pre-refresh mechanism.

4. SIMULATION AND PERFORMANCE ANALYSIS

4.1 Performance Metrics

In order to evaluate the proposed mechanism, we decided two performance metrics as follows.

- **Request savings**: the number of times that user requests hit the pre-refreshed HTML documents in the browser cache *under* ***unexpired*** *condition*, in percentage of the total number of user requests during a browsing time of an experimental browser.
- **Wasted bandwidth**: the number of times that user requests hit the pre-refreshed HTML documents in the browser cache *under* ***expired***

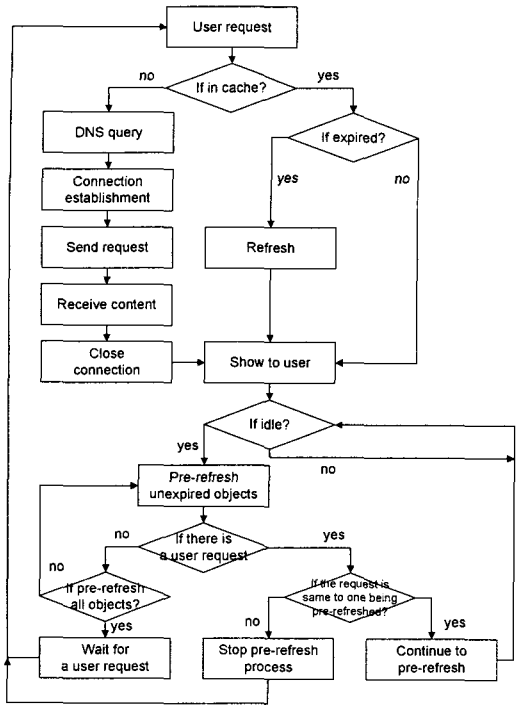


Fig. 3. Flowchart of the pre-refresh mechanism.

condition, in percentage of the total number of user requests during a browsing time of an experimental browser.

Between two metrics, *request savings* is the main concern and the primary evaluation goal of the proposed pre-refreshed mechanism. It will prove if the pre-refreshed mechanism has a good performance for reducing communication latency for users. If more pre-refreshed HTML documents are hit by the user requests under unexpired condition, the proposed mechanism would work better. *Wasted bandwidth* can be tolerated, because it only uses the idle time's network bandwidth and CPU process ability. Without the proposed pre-refreshed mechanism, these network bandwidth and CPU process ability would be also wasted.

4.2 Simulation Environment

For performance evaluation of the proposed mechanism, we first had tried to modify the open source embedded web browser to implement the mechanism. We found two open source embedded web browser: *ViewML* and *Mozilla*. However, *ViewML*[11] has no cache and no detailed documents for its structure and source code and *Mozilla*[12] is so big that most current mobile handheld devices do not have the ability to use it. Therefore, we construct a simulator designed by ourselves to implement the proposed mechanism.

Many application development tools such as Microsoft Visual C++ and Microsoft Visual Basic can develop web browsers. We use Microsoft Visual Basic 6.0 to build an experimental web browser, and the pre-refresh function and performance evaluation functions are implemented in that browser. Actually it is not for applications for mobile handheld devices. However we think it does not matter to our experiment, because the proposed mechanism mainly concentrates on the browser cache and the pre-refresh function. This simulator has 512KB cache and runs in the notebook computer with 133MHz Intel Pentium CPU and 10Mbps Ethernet LAN to emulate a mobile handheld device. Since this computer's CPU speed and network speed is very close to the hardware performance of mobile handheld devices (especially close to PDA), we expect that the result with real mobile handheld devices will be very similar to our experiment result. Fig. 4 shows a snapshot of the experimental web browser. With this experimental web browser, we will do some web browsing enough time for the proposed method to be effective.

4.3 The Implementation of the Simulator

The experimental browser for simulation uses the cache directory of Microsoft Internet Explorer named 'Temporary Internet Files' in Microsoft Windows 98. Although this cache directory does not contain *reference number* attribute designed for the cached HTML documents and we can not modify it to contain this attribute because it is good cache enough to simplify our experiment.

In order to implement the proposed mechanism,

Fig. 4. A sample image of the experimental web browser.

we created two separate access tables and used them with 'Temporary Internet Files'. Furthermore these two tables contain the fields for performance analysis, so we can use these fields to get the performance metrics. Fig. 5 and 6 show the sample images about these two tables. 'Pageattributes' access table shown in Fig. 6 contains *access_count* field, which corresponds to reference number of the proposed mechanism. *Hit_rate* and *miss_rate* fields are the number of times pre-refreshed documents were hit by user requests under unexpired and expired condition respectively. If_pre_refreshed field indicates if the document was pre-refreshed

number: 表

total_request	total_size
16	322.08

Fig. 5. The sample image of the 'number' access table.

to calculate hit_rate and miss_rate. 'Number' access table shown in Fig. 7 is used to obtain the performance metrics. Total request and total size fields means the total number of user requests and cached documents respectively.

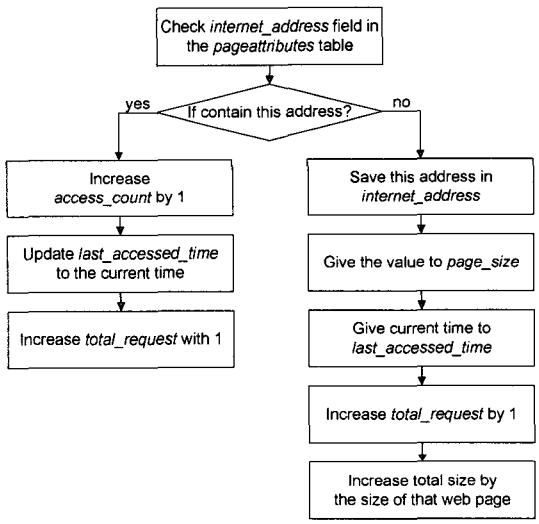


Fig. 7. Flowchart of execution step for manipulating table fields.

When a user inputs an internet address in the address box of the simple browser or he clicks one hyperlink in a web page (we assume this internet address is correct), the browser first downloads the requested web page to the user, then executes the process shown in Fig. 7 to modify the value of some fields in the tables.

There are some assumptions in manipulating table fields. For simplicity, we used the average

pageattributes: 表

internet_address	access_count	page_size	hit_rate	miss_rate	last_accessed_time	if_pre_refreshed
http://cafe.daum.net/	1	20.13	0	0	02-11-11 13:11:32	-1
http://kumoh.kumoh.ac.kr/home2000/	1	20.13	0	0	02-11-11 13:12:00	-1
http://news.sohu.com/	1	20.13	0	0	02-11-11 13:11:25	-1
http://www.chosun.com/	2	20.13	0	1	02-11-11 13:09:56	0
http://www.cnn.com/	1	20.13	0	0	02-11-11 13:11:59	-1
http://www.daum.net/	2	20.13	1	0	02-11-11 13:10:07	-1
http://www.embedded.com/	1	20.13	0	0	02-11-11 13:11:45	-1
http://www.google.co.kr/	1	20.13	0	0	02-11-11 13:11:42	-1
http://www.linuxdevices.com/	1	20.13	0	0	02-11-11 13:11:41	-1
http://www.microsoft.com/	1	20.13	0	0	02-11-11 13:11:40	-1
http://www.microwindows.org/	1	20.13	0	0	02-11-11 13:11:34	-1
http://www.sina.com/	1	20.13	0	0	02-11-11 13:13:14	0
http://www.sohu.com/	1	20.13	0	0	02-11-11 13:11:29	-1
http://www.yahoo.com/	1	20.13	0	0	02-11-11 13:12:01	-1

Fig. 6. The sample image of the 'pageattributes' access table.

size of HTML documents instead their actual size in our experiment. The total size of 1058 HTML documents is 21.3MB. So the average size is 20.13KB. We used this value as the value of the field page_size for every cached HTML document. This assumption is thought to be sound because the difference in the size of documents is trivial in our experience. In addition, we found the size of main web pages in most commercial web site does not change too much. So we decided to use the fixed value of page_size in this simulation; without this simplification, the simulation would produce a better result.

We designed cache memory of 512KB and use the Least Frequent Used (LFU) removal algorithm to remove cached HTML documents. When every request is finished, we compute the value of total_size field, and check its size. When total_size exceeds 512KB, the program first sorts all internet addresses using *access_count* field in the descending order, and deletes one location at the end of sort. Because of the assumption on the size of cached HTML document, we did not use *page_size* field as the second index for sorting. However, in the real browser environment, it must be used.

In the pre-refresh function, 10 seconds was used as the idle time judge condition as we mentioned in Section 3.2. In programming, we use three 'timer controls' for starting and stopping the pre-refresh function. When the browser finishes processing the user request, the user will begin to read the web document. During the idle time, the browser executes the process shown in Fig. 8.

During pre-refreshing, the browser first selects one internet address with the maximum access_count and the minimum page_size and checks if it is fresh. If it expires, then it will be pre-refreshed. After one pre-refresh process is finished, the corresponding last_accessed_time will be modified to the current time and the corresponding if_pre_refreshed is modified to true. If there is no user request, the next pre-refresh process will start.

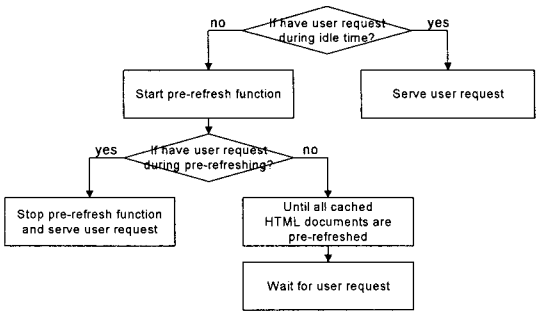


Fig. 8. Flowchart of execution step for pre-refresh function.

Finally, the browser manipulates *hit_rate* and *miss_rate* fields as follows. When a user inputs an internet address in the address box of the browser or he clicks one hyperlink in a web page, if the requested web page has been already cached ago, the browser will do the process shown in Fig. 9.

The flowchart shows that the browser first checks if the cached HTML document was pre-refreshed before a user's request. Then it checks if the document was expired. If no, we will increase hit_rate by 1 and present it to the user; otherwise, we will increase miss_rate by 1 and refresh it.

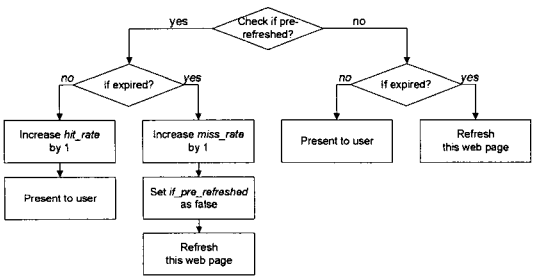


Fig. 9. Flowchart for manipulating *hit_rate* and *miss_rate* fields.

4.4 Performance Analysis and Evaluation

After the experiment there were 1,058 HTML documents in the cache and their total size was 21.3MB. The result of performance metrics shows that 'request savings' has the value 23.16%, 'wasted bandwidth' has the value 47.35%. That is to say, 23.16% of user requests could be responded

instantly without being refreshed owing to the proposed pre-refresh method. If we assume x percent of user requests can be responded instantly owing to the cache in the existing web browser, $100-x$ (>23.16) percent of user request have to be refreshed in that browser. However, with the proposed method, since additional x percent of user requests can be responded instantly by nature, $x+23.16$ percent of user requests can be responded instantly in total and $76.84-x$ (> miss rate, namely 47.35) percent of user request still have to be refreshed. Since refresh time of web documents is much larger than reading time of web documents in the cache, total service times of user requests with the existing method and the proposed method are approximately $(100-x)t_r$ and $(76.84-x)t_r$ respectively, where t_r is the mean refresh time of a web document. Therefore, we can say that the service time of user requests with the proposed method is 23.16%~41.82% less than that with the existing method because the value of x ranges from 0 to 29.49 due to the above conditions. This result could be improved if we use larger memory because *hit_rate* would be increased with larger memory.

The value of 'wasted bandwidth' is somewhat high but it does not matter. As we explained in Section 4.1, without the proposed method, these network bandwidth and CPU process ability may be also wasted during the idle time. In order to decrease this metric value we have to reduce the idle time judgment condition to less than 10 seconds. Such an adjustment of the idle time judgment condition would probably increase the other metric, 'request savings'. However it would also increase the processing time of the proposed method and thus may disturb instant serving user requests. Therefore, we think that 10 second is a reasonable condition for the idle time judgment. A possible problem of the proposed method is that the processing of the proposed pre-refresh method might interfere with smooth operation of some background processes when those background processes utilize most of CPU computing power. However, that condition is not likely to happen because mobile handheld device's users practically do not want such heavy background processes to bother web browsing.

5. CONCLUSIONS AND FUTURE WORK

Today, more and more internet population use mobile handheld devices to access the World Wide Web. But users spend a lot of time impatiently waiting for web pages to come up on screen. In this paper, we proposed an effective pre-refresh mechanism for mobile handheld device's embedded web browser. The proposed mechanism uses the idle time of users' reading web pages to pre-refresh cached HTML documents in the embedded web browser. The simulation shows that it considerably reduces the latency for users during web surfing.

Although this mechanism has been aimed primarily at mobile handheld device's embedded web browser, we believe that it may also be appropriate for desktop PC's Browser. The following work will be done for further study.

- This mechanism will be practically implemented in the open source embedded web browser in order to evaluate its performance in the real environment.
- Other good performance metrics may be designed and more traces will be used to evaluate the performance of the proposed mechanism more accurately.
- Other techniques may be merged with the proposed mechanism to more reduce latency for users.

6. REFERENCES

[1] Li Fan, Pei Cao, and Quinn Jacobson, Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Perfromance, *International ACM Conference on Measurement and Modeling of Computer System*, pp. 178-

187, 1999.

[2] Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy, Potential benefits of delta encoding and data for http, *In Proc. of ACM SIGCOMM'97*, Aug. 1997.

[3] Jeffrey C. Mogul, What is HTTP Delta Encoding, *Available at* "http://webreference.com/internet/software/servers/http/deltaencoding/intro/", 2002.

[4] Henrik Frystyk Nielsen, Jim Gettys, Anselm BairdSmith, Eric Prudhommeaux, Hakon Wium Lie, and Chris Lilley, Network performance effects of http/1.1, *In Proc. of ACM SIGCOMM'97*, Aug. 1997.

[5] R.Fielding, J.Gettys, J.Mogul, H.Frystyk, and T.Berners-Lee, Network performance effects of http/1.1, Hypertext Transfer Protocol-HTTP/1.1, *RFC 2068*, Jan 1997.

[6] Mark Nottingham, Caching Tutorial for Web Authors and Webmasters, *Available at* "http://www.mnot.net/cache/docs/", Feb. 17, 2002.

[7] Md. Ahsan Habib and Marc Abrams, Analysis of Sources of Latency in Downloading Web Pages, *In Proc. of WebNet 2000*, Oct. 2000.

[8] T. M. Kroeger, J. Mogul, and C. Maltzahn, Digital's Web Proxy Traces, *Available at* "hftp://ftp.digital.com/pub/DEC/traces/proxy/webtraces.html", Aug. 1996.

[9] Luigi Rizzo, Web Proxy Traces. Available at "http://info.iet.unipi.it/~luigi/proxytraces/", May 1997.

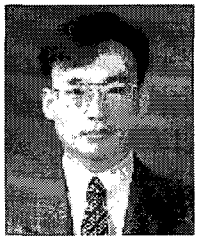
[10] S. Williams, M. Abrams, C. Stanbridge, G. Abdulla, and E. Fox, Removal Policies in Network Caches for World-Wide Web Documents, *In Proc. of ACM SIGCOMM'99*, 1999.

[11] Century Software Embedded Technologies, ViewML Brochure, *Available at* "http://embedded.centurysoftware.com/documents/index.php/#cendocs", 2001.

[12] The Mozilla Organization, Introduction to Mozilla - A Manual for First Time Users, *Available at* "http://downloads. mozdev.org/mozmanual/en/mozmanual-version-1.0.pdf".

Huaqiang Li

He received a B. S. degree in computer science and engineering from Harbin Institute of Technology, China in 2001, and an M. S. degree in computer engineering from Kumoh National Institute of Technology (KIT), Korea in 2003. He is currently an engineer on mobile cellular phone systems at China R&D Center of LG electronics in Peking. His main research interest is embedded and mobile software.



Young-Hak Kim

He received a B. S. degree in electronic engineering from KIT in 1984, and M. S. and Ph. D. degrees in computer science from Sogang University, Korea in 1989 and 1997 respectively. He was a full-time lecturer in the Dept. of Computer Science at Republic of Korea naval Academy from 1989 to 1997, and in the School of Multimedia at Yosu National University from 1998 to 1999. He returned to KIT in 1999 and is currently an associate professor in the School of Computer and Software Engineering (SCSE) at KIT. His current research interests are parallel algorithms, and distributed and parallel processing.

Tae-Hyong Kim

He received B.S. and M.S. degrees in electronic engineering from Yonsei University, Korea in 1992 and in 1995 respectively, and a Ph.D. degree in electrical and electronic engineering from the same university in 2001. He was a postdoctoral fellow at the School of Information Technology and Engineering at the University of Ottawa from 2001 to 2002. He is currently an assistant professor in the SCSE at KIT. His current research interests are communication software and protocol engineering, and wireless networks and their security.