Enhanced Client Polling with Multilevel Pre-Fetching Algorithm for Wireless Networks

Nazrul Muhaimin Ahmad and Tan Kim Geok

Abstract: **The implementation of client polling as a weak cache coherence mechanism has two major drawbacks: Firstly, the cache may return a stale copy if the object is changed in the origin server while the cached copy is considered valid. Secondly, the cache can invalidate a cached copy that is still valid in the server. Therefore, we propose a multilevel pre-fetching (MLP) in conjunction with the client polling to refine these drawbacks. MLP is introduced to improve the level of freshness among the cached objects. The simulation results presented in this paper show that the proposed MLP significantly minimizes the number of stale objects and reduces the invalidation messages sent out to the server, i.e., increase the cache HIT rate.**

Index Terms: **Cache coherence, cache effectiveness, client polling, pre-fetching, wireless networks, world wide web.**

I. INTRODUCTION

The world wide web (WWW) has become among the largest source of Internet traffic, but it performs poorly in wireless networks [1]–[3]. Wireless network is basically known as narrow bandwidth, highly variable transmission delays and sudden disconnection. Naturally, the presence of web cache between client and servers has been considered a good solution to refine these limitations [4]–[6]. Web caching is a mechanism that stores web objects for convenient future access [7]. It obviously reduces bandwidth consumption, server load and latency of responses [8]. However, this web caching may bring side effects to the clients due to the absence of cache coherence, which enforces clients to receive stale web pages [7], [9]. This is due to the fact that if the object in the remote server is changed while the client keeps accessing an out-of-date copy from its local cache, then stale object would be accessed. How to keep the cached object consistent with the original server changes is a real challenge. Therefore, cache coherence mechanisms are implemented to ensure that cached copies are frequently updated to keep consistency with the original data.

Basically, there are two cache coherence techniques [7]–[9]: Weak cache coherence and strong cache coherence. The weak cache coherence is the model in which the access time is significantly reduced, but a stale document might be returned to the client [9]. However, with the fact that the nature of wireless network where the bandwidth is limited and expensive, it is too costly for the clients to keep refreshing the web browser every time they received the stale documents. Therefore, if the clients have strict requirement on the freshness of the documents or a stale copy is not tolerable, then a strong cache coherence mechanism is necessary [10]. Strong cache coherence enforces the freshness of the document all the time at the expense of more control messages over network. However, the flooding of control messages substantially consumes bandwidth and adds to the server loads, not to mention that the client might experience longer response time [7].

Many caching systems apply weak cache coherence, believing that methods such as time-to-live (TTL) and client polling are sufficient and most appropriate for web caching [9], [11]. In either case, modification to the web object before its TTL expires or between two successive polls causes the web cache to return stale object. On top of that, weak cache coherence mechanisms are easily deployed and supported by HTTP [10]–[12]. Based on these facts, we decided to investigate the possibility of deploying weak cache coherence mechanism particularly client polling in the web cache for wireless web services. Our motivation is to find the solution on how mechanism could be improved to alleviate stale web objects forwarded to the clients by the web cache.

In this paper, we ameliorate the drawbacks of client polling by introducing a cache refreshment algorithm, multilevel pre-fetching (MLP). We argue that today's web cache that is supported by client polling must be augmented, by improving its responsiveness towards easing the stale cached objects in order to make caching more effective. Therefore, the MLP algorithm is proposed in conjunction with client polling to enhance the frequency of updating cached objects. It is responsible to validate the cached objects without waiting for the requests from the clients. It is executed whenever the cache-server channel is under utilized. The MLP algorithm has been tested and evaluated by replaying web trace over wireless local area network. The simulation results have shown that the MLP records a couple of advantages over the conventional client polling mechanism. It reduces the number of stale objects that could be forwarded to the client. Also, it has been observed that the MLP increases the number of cache HITs and considerably improves the access latency for retrieving web objects from the web cache.

II. CLIENT POLLING

A client polling (CP) [11] is a typical example of weak cache coherence. It means the client (web cache for this context) periodically checks back with the server to determine if the cached objects are still fresh. The cache issues an if-modified-since (IMS) request with the last-modified response header value (indicating last modification time of the object). The server then checks to see if the object has changed since the timestamp. If so, a 200 HTTP response code is sent along with fresh object.

Manuscript received March 28, 2005; approved for publication by Sudhir Dixit, Division II Editor, September 3, 2006.

N. M. Ahmad is with the Faculty of Information Science and Technology, Multimedia University, Malaysia, email: nazrul.muhaimin@mmu.edu.my.

T. K. Geok is with the Faculty of Engineering and Technology, Multimedia University, Malaysia, email: kgtan@mmu.edu.my.

1229-2370/07/$10.00 © 2007 KICS

When a new HTTP GET request arrives at web cache,

1. Web cache imposes a lifetime attributes to a web object.
   Age = current time – object last modified time
2. If the update threshold is set to x%, then
   Object age = Age * x%
3. Web cache checks the time since last validation
   Check time = current time – object last accessed time
4. Web cache validates the object,
   if Object age ≥ Check time
   Object lifetime is not expired
   Else
   Object lifetime is expired

Fig. 1. Pseudocode of Alex protocol.

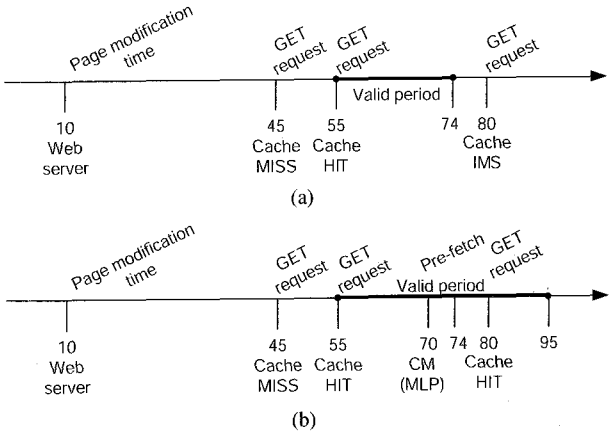


Fig. 2. CP on one web object with and without MLP. All calculations are based on $\theta = 30\%$: (a) CP, (b) proposed MLP.

Otherwise, the server returns a code of 304 (document not modified). Alex file transfer protocol (FTP) cache [14], a form of CP, uses an update threshold, θ, to determine how frequent the cache to poll the server. The θ is expressed as a percentage of the object's age. The age is the time since last modified time of the object. An object is invalidated when the time since the last validation exceeds the θ times the object's age [11]. Whenever a client issues GET request to the cache, the validation procedure is invoked as illustrated in Fig. 1. An invalidated object causes the cache to send an IMS to the server to check whether a file transfer is necessary. Otherwise, the cache sends the valid object to the client.

CP is designed on the assumption that young web objects are modified frequently than old objects. Hence, the web cache needs to poll less frequently for older objects. However, the older objects may be stale and there is likelihood those objects could be forwarded to the clients [11]. In brief, the implementation of CP as a cache coherence mechanism has two major drawbacks: Firstly, the cache may return a stale copy if the object is changed in the origin server while the cached copy is considered valid. Secondly, the web cache can invalidate a cached copy that is still valid in the original server. Therefore, we propose a MLP algorithm to improve the level of freshness among the cached objects to refine these drawbacks.

III. PROPOSED MULTILEVEL PRE-FETCHING

A number of studies found in the literature have focused on the cache coherence, cache replacement and pre-fetching issues [8]. However, a limited research has been done on cache refreshment methods [12], [13]. Without a suitable cache refreshment policy, the probability of cache HIT will decrease over time, leading to an increase in access delay. We considered the refreshment policy that is quite similar to our proposed idea. Cohen and Kaplan [12] proposed policies for caches to proactively validate selected cached objects as they become stale, and thus allow for more client requests to be processed locally. The decision of which objects to renew upon expiration varies between policies and is guided by natural properties of requests history of each object such as TTL values, popularity and recency of previous request. In this paper, we propose a different approach of web cache refreshment, which validates the cached objects based on client polling algorithm.

To illustrate the concept of MLP, consider one web object as shown in Fig. 2 whose last modification time was at 10 second. The respective web object was fetched from web server at 45 second. For the subsequent GET requests, web cache invokes Alex protocol to determine whether the cached object will be constituted as cache HIT or cache IMS. As illustrated in Fig. 2(a), if GET request is issued outside the validity period of the previous GET request, web cache treats that request as a cache IMS. Thus, web cache sends IMS to web server. Such operation induces extra processing time to client's perceived latency. However, if MLP successfully pre-fetches the respective cached object within the validity period, the outcome of the next GET request is changed from cache IMS to cache HIT as shown in Fig. 2(b). This is because MLP potentially extends the validity period of the cached object. As a result of that, MLP reduces IMS bytes sent out to web server for the GET requests received by web cache. Additionally, if the object has been modified in the web server, MLP assures the fresh object delivery to the client as the response to the GET request that constituted cache HIT.

MLP is designed to operate independently from the normal operation of client polling. The refreshment method is performed as "offline" validation between web servers and web cache. Thus, the MLP does not add processing overhead to the client's perceived latency. The proposed MLP can be easily integrated with client polling without much modification. Fig. 3 shows the enhanced CP with MLP algorithm architecture. Basically, there are two major components added to the conventional CP mechanism: Bandwidth manager (BWM) and cache manager (CM).

A. Bandwidth Manager

Bandwidth manager (BWM) operates as a front-end of the web cache and it is assigned to take care of traffic monitoring especially on file transfer activities at cache-server channel. In general, BWM monitors, computes the link utilization, and estimates the available bandwidth. Then, BWM is responsible to trigger the CM whenever the traffic condition is good enough to

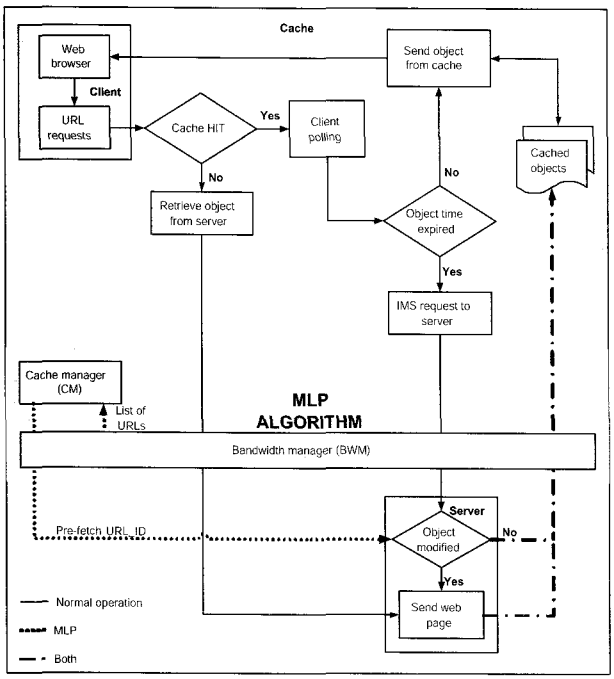


Fig. 3. MLP algorithm in conjunction with CP.

perform pre-fetching operations. BWM keeps the records of all cache's entries and prepares a list of URLs that recently fetched from the server. It uses a heap as the data structure to store the cached object's header information. For performance reasons, the BWM does not store the real object in it; only the object's header is stored. For each cached object, the following information is kept in the heap: The URL_ID, object's size, the entry time, and the last modification time. Whenever a new web object is cached, it is assigned with an index number by BWM, which allows it to be inserted sequentially after the previous cached objects. The BWM will append the object's header information at the end of heap. Therefore, the list of URLs is expected to grow from time to time depending on the new requests from the clients. If there is an IMS request sent by the web cache to server, the BWM will necessarily update the information in the heap if the web cache received a response code 200. Before the list of URLs is forwarded to CM for pre-fetching operations, an available bandwidth is measured. If available bandwidth is lower than the specified threshold (MLP used bandwidth threshold of 80%), then the BWM will not convey the list of URLs to CM. Therefore, the pre-fetching operations will be suspended until the bandwidth is under utilized again.

B. Cache Manager

The cache manager (CM) is responsible to monitor and validate the freshness of the cached objects in the web cache. Once the list of URLs arrives at CM, the list of pre-fetching sequences are created and stored in a tree like data structure, pre-fetch tree. Every sub-node in a tree corresponds to each cached object in the list of URLs. Then, CM checks all objects' validity in the same way used in CP. If the object's age exceeds the validation time, the cached object is considered valid. Later, CM uses this heap to pre-fetch the objects that are still valid by sending IMS control message. If the object is modified in the server, it will be sent to the web cache, otherwise response code 304 is relayed back to web cache. The modified object is stored in the web cache and object's header information is copied in order to update the list of URLs maintained by BWM. In the case of invalid objects, the web cache only treats them whenever there is a request from the client. When the object expires, the web cache will immediately send IMS to remote server. Thus, the web cache always assures the freshness of object at the cost of client's perceived latency.

C. Implementation of Proposed Algorithm

The MLP invokes a timer to schedule the events as follows. After every t seconds, the BWM initiates the computation on the bandwidth utilization of the web cache. BWM estimates the network traffic generated by the web cache and message bytes sent by the web server to the web cache over specific period of time. It is used to determine whether the bandwidth is sufficient enough to perform pre-fetching operations. Whenever the link utilization exceeds 20% of the available bandwidth, all pre-fetching activities will be suspended. Whenever the bandwidth is considered available, BWM forwards the list of URLs to CM for pre-fetching at regular interval, *MLP_interval*, i.e., mt seconds. However, if the bandwidth is unavailable, BWM will delay the forwarding until it computes the link utilization in the next t seconds.

Upon receipt of a list of URLs from the BWM, CM starts to create pre-fetch tree. Pre-fetch tree consists of aggregate of multilevel sub-nodes. CM assigns each sub-node in the pre-fetch tree with one of URL_ID in the list of URLs. In the next step, CM traverses the pre-fetch tree by visiting sub-node by sub-node to validate the object's freshness based on the information obtained in the list of URLs. At each *prefetch_interval*, CM invokes Alex protocol to determine whether object's freshness is valid. If the object is valid, CM pre-fetches the respective URL_ID in the web server by sending IMS. Otherwise, the sub-node is terminated and CM proceeds to the next sub-node in the upcoming *prefetch_interval*. CM repeats the same procedure until it reaches the last sub-node in the pre-fetch tree.

The CM expects the arrival of the next list of URLs after *MLP_interval* or more, and accordingly it creates a new pre-fetch tree, even though the existing tree still performs the pre-fetching process. Thus, there is possibility of several trees to be run concurrently. The idea to have this configuration is to cope with the fact that dynamic objects are frequently modified in the web server. Hence, web cache needs to validate the consistency of the cached objects regularly to make sure those objects are far from expiry. By lowering down the *MLP_interval*, CM more aggressively performs the pre-fetching operations in order to keep both web cache and web server synchronized, i.e., CM regularly updates all the cached objects listed in the list of URLs. However, frequent pre-fetching comes with the expense of overloading more traffic on the channel. Therefore, BWM is assigned to monitor these activities and once the channel is congested, BWM signals the CM to terminate all the existing pre-fetch trees. By eliminating the trees, the bandwidth is completely reserved for normal operation. Once the bandwidth is available again, the pre-fetching process will resume.

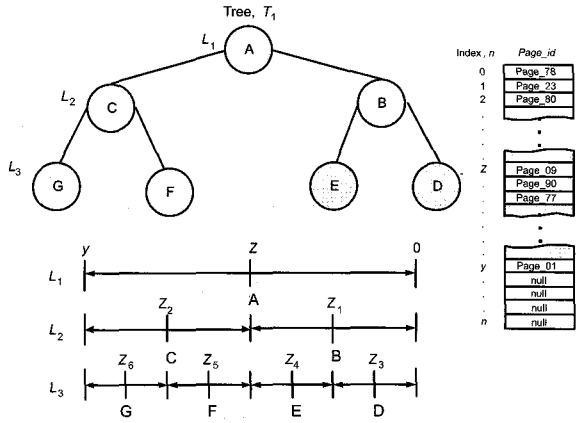


Fig. 4. Construction of multilevel sub-nodes.

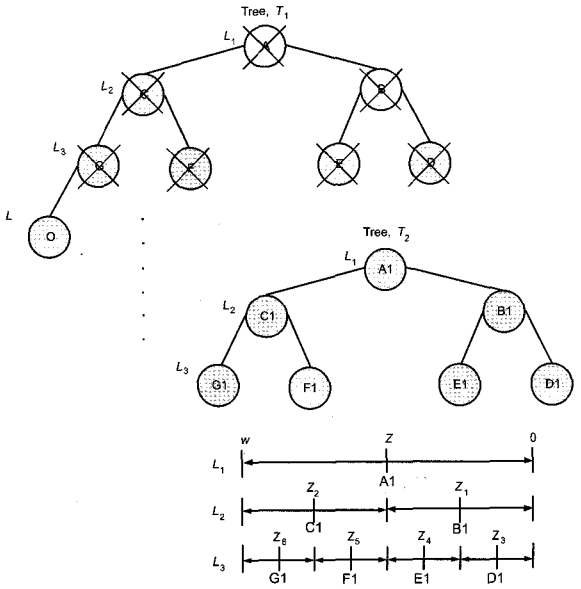


Fig. 5. Creation of T_2 in concurrent with T_1.

D. Construction of Multilevel Sub-Nodes

This section discusses the construction of pre-fetch tree for MLP. The pre-fetch tree is constructed as follows. Once the CM receives a list of URLs from the BWM, the first sub-node is generated as a root and it is assigned with a random index number from the list of URLs. For instance, as shown in Fig. 4, the sub-node A is created at level L_1 of the first pre-fetch tree T_1 and it is assigned with a random index number Z which lies between 0 and y, where y is the last index number in the list of URLs. This index number, Z, corresponds to the URL_ID of the cached object, which is in this case referred to Page_09. Similarly, the second sub-node B is generated at level L_2 and assigned with any index number within the range from 0 to Z. This process is repeated until it covers the entire cached object's header information stored in the list of URLs. After the pre-fetch tree is completely built, CM waits until the next *prefetch_interval* to perform pre-fetching operations. At *prefetch_interval*, starts with the root, CM picks URL_ID of the sub-node and it searches the corresponding object's header information in the list of URLs. Then, CM checks the object's freshness by invoking Alex protocol. If the object is valid, CM pre-fetches the respective URL_ID from the web server. Otherwise, the sub-node will be terminated. In the upcoming *prefetch_interval*, CM traverses to the next sub-node in the pre-fetch tree. CM repeats the same procedure until it visits the last sub-node of the tree.

Meanwhile, the CM can start creating another pre-fetch tree whenever it receives a new list of URLs from BWM. Fig. 5 illustrates the construction of the second pre-fetch tree T_2 at the next *MLP_interval*. For the second tree, CM expects the total number of cached objects in the list of URLs equal or more than the first list. List of URLs contains more cached objects if there is new GET requests from the clients that constituted cache MISS within the first and second *MLP_interval*, and so forth. In brief, CM validates one sub-node on every *prefetch_interval*. With a short *prefetch_interval*, CM visits sub-node by sub-node in the pre-fetch tree at faster rate.

IV. SIMULATION ENVIRONMENT

The simulations were performed on network simulator (NS2) version 2.1b9a [15]. NS2 was chosen over other simulation tools because it is a widely accepted discrete event simulator and actively used for wired and wireless simulations. The simulation model consists of 20 wireless nodes connected to a single web cache at the edge of wireless network. The web cache is attached to the web server via a Fast Ethernet link of 100 Mbps. The wireless nodes emulate 914 MHz Lucent WaveLAN DSSS radio interfaces for MAC 802.11. The simulation used the value encoded in NS2 corresponding to the physical specification of 914 MHz Lucent WaveLAN. All the experiments have been carried out by using trace-driven simulation to examine cache coherence mechanism behavior. Trace-driven simulation consists of replaying trace collected from the real web server through a simulator. The experiments used one web trace from Internet Traffic Archives [16]. The trace is EPA: The EPA WWW server located at Research Triangle Park, NC [17], and it contains a day's worth of all HTTP requests to the EPA WWW server. A summary of the trace appears in Table 1.

Table 1. EPA WWW web server trace.

Parameter	Value
Number of GET requests	33,285
Number of URL	3,741
Minimum page size	33 bytes
Average page size	20.7 kbytes
Maximum page size	4.59 Mbytes
Static page modification interval	10,000 seconds
Dynamic page modification interval	60 seconds

In this simulation, we considered all HTTP GET requests such that 200 and 304 response codes were returned to the client. Since the trace does not contain page modification information, we assigned every web object with a bimodal page modification model [9]. Furthermore, we allocated 10% of the objects to be dynamic, i.e., modified frequently, and the rest of the objects to be static, i.e., less frequently changed. Percentages of static and dynamic objects are assumed based on the observation reported

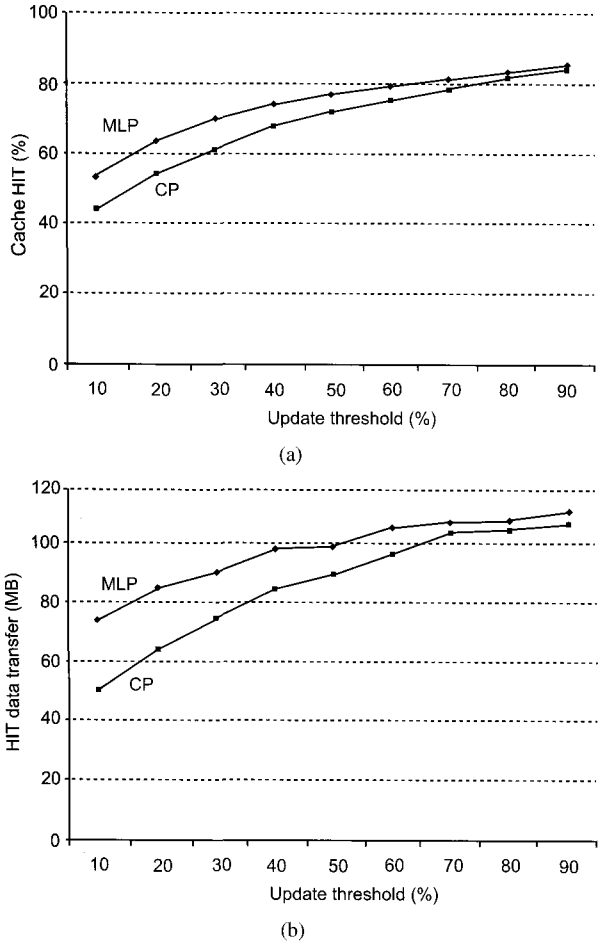


Fig. 6. Impact of MLP on cache HIT: (a) Cache HIT rate, (b) cache HIT data transfer.

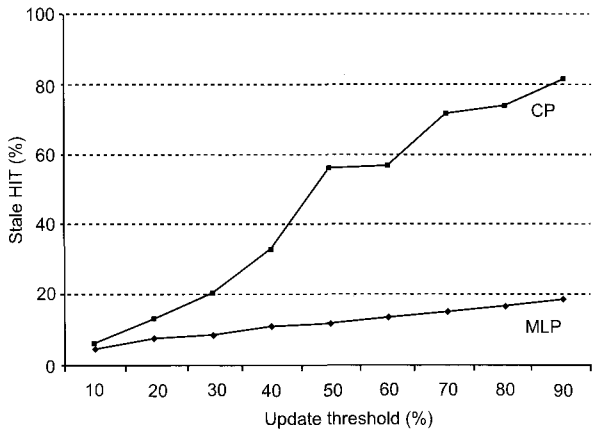


Fig. 7. Effect of MLP in alleviation of stale HIT.

in [11] and [18]. Since the underlying premise of this paper is to investigate the freshness of the cache HIT, we assumed that the cache to have an infinite storage capacity and none of the cache replacement policies are considered. However, we will explore both issues in the future work. Web cache receives requests from a number of wireless clients. Cache HIT is serviced using locally cached object whereas a cache MISS is simulated by fetching the object from the web server. The web cache deploys cache coherence mechanism to ensure the consistency of cached object with that stored on the server. Specifically, we used CP that is based on Alex protocol. We ran CP with or without cache refreshment algorithm, MLP, by varying θ in order to evaluate the impact of object's age on the effectiveness of CP.

The preceding section describes the design of MLP algorithm. There are two parameters associated with MLP: *MLP_interval* and *prefetch_interval*. We find that the performance of MLP algorithm depends on the value of the parameters chosen. Moreover, the performance is also much affected by other circumstances such as page next modification time for dynamic and static web objects in the web server. Therefore, with the lack of information on page next modification time, we decided to arbitrary assume the value of the parameters. Hence, the updating process of the cached objects is scheduled every 500 seconds (*MLP_interval* of 500) and CM performs pre-fetching operation at a frequency of 10 seconds (*prefetch_interval* of 10). On the other hand, BWM is assigned to monitor the cache-server channel every 100 seconds.

Responsiveness of web cache in web services plays a critical role in determining client satisfaction. Therefore, we concentrate on the quality of web cache's cache HIT rate. Based on this metric, we then investigate the percentage of cache HIT that turned out to be stale when the object is disseminated to the client, i.e., stale HIT rate. Finally, we measure how much bandwidth is used on cache-server channel during a specific time period to observe the impact of MLP.

V. RESULTS AND ANALYSIS

A. Impact of MLP on Cache HIT

This section explores the impact of MLP on the performance of cache HIT. Fig. 6(a) presents the experimental results of cache HIT rate for CP and MLP as the θ varies from 10% to 90%. On average, MLP increases the number of cache HITs recorded by CP by 6%. MLP reasonably keeps the cached objects up to date, i.e., it preserves the object's age far from expiry. Therefore, more GET requests are served by web cache locally, which increases the number of cache HITs. The efficiency of MLP is further exhibited in Fig. 6(b). It shows the data transfer bytes between web cache and clients. With the higher number of cache HITs, MLP improves the communication between clients and the intermediary node by terminating more GET requests at the web cache. Thus, most of the time, MLP helps the clients to retrieve the web objects at minimal download time since the objects are available more nearer to them. Therefore, MLP accounted not only higher cache HIT rate, but also the reduction in the client's perceived latency. At low values of θ, MLP records an improvement of approximately 20 MB.

B. Effect of MLP in Alleviation of Stale HIT

This is a vital performance metric of the study, which shows the positive effect of the MLP algorithm on improving the freshness of the cached objects. As indicated in Fig. 7, CP initially records low percentage of stale HIT at low values of θ. How-

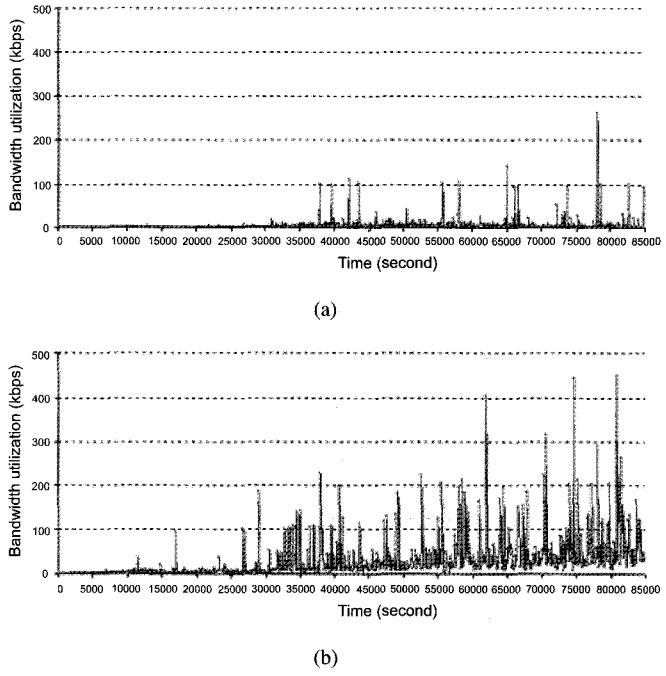


Fig. 8. Bandwidth utilization of the web cache for $\theta = 10\%$: (a) CP-average utilization $= 5.21$ kbps, (b) MLP-average utilization $= 31.66$ kbps.

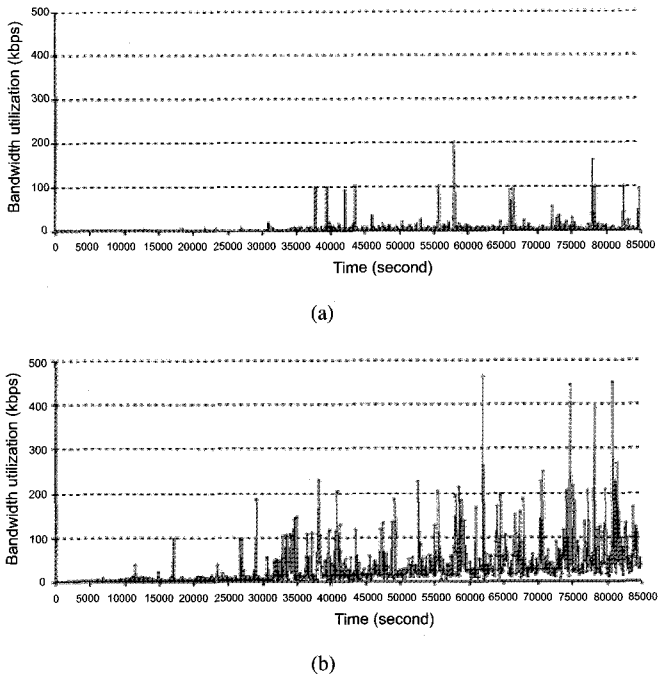


Fig. 9. Bandwidth utilization of the web cache for $\theta = 30\%$: (a) CP-average utilization $= 4.53$ kbps, (b) MLP-average utilization $= 31.58$ kbps.

ever, the rate starts to increase as the θ increases. The introduction of MLP reasonably improves the stale HIT rate, which is kept lower than 20% for all values of θ. We assume that the acceptable level of stale HIT rate is 10%. Note that in the CP case, to achieve the target, the web cache needs to tune the θ less than 20%. Conversely, MLP allows the web cache to have flexibility in choosing much higher θ than CP. Web cache can select θ of 40%, nearly two orders of magnitude more than CP in order to achieve same acceptable level of stale HIT rate. The advantage of having much higher θ is to allow web cache to satisfy more GET requests locally, i.e., to increase the cache HIT rate. In summary, MLP helps the web cache to refresh the outdated cached objects. Thus, it provides a high probability for the web cache to deliver fresh objects to the clients.

C. MLP Bandwidth Utilization

Figs. 8 and 9 illustrate time-series plots of the aggregate bandwidth consumed by web cache particularly on cache-server channel over EPA web trace. In analyzing the impact of MLP on network traffic, this section compares the bandwidth utilization of CP and MLP at two different values of θ, 10% and 30%. And, it considers all the network traffic such as control message bytes generated by web cache and all HTTP responses received from web server. MLP relatively causes more bytes to be transferred over the channel. In particular, pre-fetching operation needs to send IMS control message bytes to the web server for validation process. As responses to this, web server forwards response code of 200 together with the data bytes, or response code of 304 (not-modified), which relatively smaller in size. Such activities add more loads on the channel and if the bandwidth is used excessively, there is likelihood of MLP to interrupt the normal operation of the web cache or even causes the HTTP connection to be aborted. Thus, it significantly degrades the performance of the web cache. Therefore, there is a need to deploy traffic-controlled mechanism like BWM to observe the pre-fetching operation. Once the bandwidth consumed by web cache exceeds 20% of the available bandwidth, all pre-fetching will be suspended.

In interpreting Figs. 8 and 9, this section begins with the investigation on the bandwidth utilization of CP. On average, CP with θ of 10% records much higher bandwidth consumption than other θ values. This is due to the fact that at low θ, CP actively sends out more IMS control messages to the web server for consistency checking. This is because of the object's age is relatively short at this θ. Hence, it can be deduced that the bandwidth utilization of CP is directly related to the choice of θ value. The lower the value of θ, the more bandwidth is consumed for CP operations. In contrast, MLP substantially increases the usage of cache-server channel bandwidth. Likewise in the normal case, the amount of the used bandwidth is hardly affected by the percentage of the object's age. However, for the MLP, as θ increases, the bandwidth consumption also depicts significant increment. With the longer object's age, more cached objects are considered valid and thus eligible to be validated and if necessary to be pre-fetched from the web server. In summary, the significant growth on the bandwidth utilization is still considered a fractional usage; MLP indicates approximately 0.16% of the used bandwidth as compared to the available bandwidth that is reserved for pre-fetching operations, i.e., 20% of the cache-server channel capacity. If it is necessary to reduce the excessive freshness tests on the origin web servers, the construction of the pre-fetch tree can be scheduled for irregular hours or for

off-peak hours.

VI. CONCLUSIONS

In this paper, we have proposed MLP algorithm for wireless web services aimed at easing the stale cached object from being forwarded to the clients. In this scheme, MLP assists web cache to decide on which cached objects need to be validated or if necessary to be pre-fetched from the remote servers, and assures the returned cached objects to the client are not stale. From the performance analysis in this paper, we draw the following conclusions.

Firstly, since the validity of the cached objects is extended, the proposed algorithm records an increase in the total number of cache HITs. This situation reduces the access latency by making the requested objects available closer to clients. With the expensive rate of access and limited accessing bandwidth, a cache HIT promises a fair reduction in client's perceived latency. Secondly, the higher numbers of cache HIT does not necessarily assure forwarded objects by the web cache to the client are fresh. But, the proposed algorithm does improve the level of freshness among the cached objects. Hence, a higher number of fresh objects could be returned to the clients. Thirdly, the proposed algorithm pre-fetches the cached objects that are likely stale in the web cache. Hence, it refreshes the stale cached objects, which in turn helps web cache to reduce the number of "if-modified-since" sent out to the web server for validating the requested cached objects from the clients. Finally, the proposed MLP introduces a slight increase in the control message bytes, the cache-server channel bandwidth consumption and system performance of the web cache such as CPU overhead and memory resources, due to pre-fetching process. However, the increment and the effects are minimal as compared to the available resources of the web cache and the total channel bandwidth.

REFERENCES

[1] T. B. Fleming, S. F. Midkiff, and N. J. Davis, "Improving the performance of the world wide web over wireless networks," in *Proc. IEEE GLOBECOM'97*, 1997, pp. 1937–1942.

[2] H. Balakrishnan and R. H. Katz, "Explicit loss notification and wireless web performance," in *Proc. IEEE GLOBECOM Internet Mini-Conf.*, Sydney, Australia, 1998.

[3] Y. Tian, K. Xu, and N. Ansari, "TCP in wireless environment: Problems and solutions," *IEEE Commun. Mag.*, vol. 43, no. 3, pp. S27–S32, 2005.

[4] G. Cao, "A scalable low latency cache invalidation strategy for mobile environments," *IEEE Trans. Knowl. and Data Eng.*, vol. 15, no. 5, pp. 1251–1265, 2003.

[5] A. Kahol, S. Khurana, S. K. S. Gupta, and P. K. Srimani, "A startegy to manage cache consistency in a distributed mobile wireless environment," *IEEE Trans. Para. and Dist. Sys.*, vol. 12, no. 7, pp. 686–700, 2001.

[6] Z. Wang, S. K. Das, H. Che, and M. Kumar, "A scalable asycnhronous cache consistency scheme for mobile environment," *IEEE Trans. Para. and Dist. Sys.*, vol. 15, no. 11, pp. 983–995, 2004.

[7] L. Y. Cao and M. T Ozsu, "Evaluation of strong consistency web caching techniques," in *World Wide Web: Internet and Web Information Systems*, vol. 5, no. 2, Kluwer Academic Publishers, 2002, pp. 95–123.

[8] J. Wang, "Survey of web caching schemes for the Internet," *ACM Computer Commun. Rev.*, vol. 27, no. 5, pp. 36–46, 1999.

[9] P. Cao and C. Liu, "Maintaining strong cache consistency in the world wide web," *IEEE Trans. Computers*, vol. 47, no. 4, pp. 445–457, 1998.

[10] F. Doswell, M. Abrams, and S. Varadarajan, "The effectiveness of cache coherence implemented on the web," in *Proc. Workshop on Caching, Coherence, and Consistency*, 2001.

[11] J. Gwertzman and M. Seltzer, "World-wide web cache consistency," in *Proc. USENIX Tech. Conf.*, 1996, pp. 141–152.

[12] E. Cohen and H. Kaplan, "Refreshment policies for web content caches," in *Proc. 20th Annual Joint Conf. IEEE Computer and Commun. Soc.*, 2001, pp. 1398–1406.

[13] E. Cohen and H. Kaplan, "Aging through cascaded caches: Performance issues in the distribution of web content," in *Proc. SIGCOMM 2001*, 2001.

[14] V. Cate, "Alex—A global file system," in *Proc. USENIX File System Workshop*, 1992, pp. 1–11.

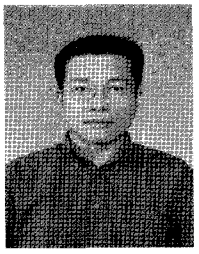
[15] Virtual InterNetwork Testbed-VINT, (1997). Network simulator (NS-2). [Online]. Available: http://www.isi.edu/nsnam/ns

[16] ACM SIGCOMM, (2000) The Internet traffic archive. [Online]. Available: http://ita.ee.lbl.gov/html/traces.html

[17] L. Bottomley, (1995) EPA-HTTP—A day of HTTP logs from the EPA WWW server. [Online]. Available: http://ita.ee.lbl.gov/html/contrib/EPA-HTTP.html

[18] V. Duvvuri, P. Shenoy, and R. Tewari "Adaptive leases: A strong consistency mechanism for the world wide web," *IEEE Trans. Knowl. and Data Eng.*, vol. 15, no. 5, pp. 1266–1276, 2003.

Nazrul Muhaimin Ahmad received B.Eng. degree in Electronics and Communications from the University of York, UK, in 1999, and M.Sc. in Information Technology from Multimedia University, Malaysia, in 2006. He is currently a Ph.D. student in telecommunication engineering and also an academic staff at Multimedia University, Malaysia. His major research interests are in multi-carrier technologies, mobile link adaptation policies, network security, and mobile content delivery.



Tan Kim Geok received the B.E., M.E., and Ph.D. degrees all in electrical engineering from University of Technology Malaysia, in 1995, 1997, and 2000 respectively. He has been Senior R&D engineer in EPCOS Singapore in 2000. In 2001–2003, he joined DoCoMo Euro-Labs in Munich, Germany. He is currently academic staff in Multimedia University. His research interests include radio propagation for outdoor and indoor, RFID, multi-user detection technique for multi-carrier technologies, and A-GPS.