Algorithms for Supporting Disconnected Write Operations for Wireless Web Access in Mobile Client-Server Environments

Ing-Ray Chen, *Member*, *IEEE Computer Society*, Ngoc Anh Phan, and
I-Ling Yen, *Member*, *IEEE Computer Society*

Abstract—In a wireless mobile client-server environment, a mobile user may voluntarily disconnect itself from the Web server to save its battery life and avoid high communication prices. To allow Web pages to be updated while the mobile user is disconnected from the Web server, updates can be staged in the mobile host and propagated back to the Web server upon reconnection. In this paper, we analyze algorithms for supporting disconnected write operations for wireless Web access and develop a performance model to identify the optimal length of the disconnection period under which the cost of update propagation is minimized. The analysis result is particularly applicable to Web applications which allow wireless mobile users to modify Web contents while on the move. We show how the result can be applied to real-time Web applications such that the mobile user can determine the longest disconnection period such that it can still propagate updates to the server before the deadline so that a minimum communication cost is incurred.

Index Terms—Wireless mobile systems, Web access, disconnected operations, performance analysis, coherency interval, mobile client-server systems.

1 Introduction

A mobile host (MH) performing Web accessing can voluntarily disconnect itself from the server to save its battery life and avoid high communication prices in wireless networks. Before disconnection, the MH can prefetch into its local cache frequently used Web pages of various formats [12] based on the MH's specification or the past Web access history. During disconnection, the MH accesses the Web using only these prefetched pages. In addition to supporting read-only Web browsing, write operations to these prefetched pages can be performed by recording new values into a log. When the MH is reconnected to the system, the operational log entries can be reintegrated back to the Web server to resolve conflicts with updates performed at other sites. In the literature, the three phases of supporting disconnected operations are termed *hoarding*, *disconnection*, and *reintegration* [4], [10], respectively.

Over the past few years, various schemes have been proposed to support disconnected Web access in wireless mobile environments in these three phases [1], [2], [3], [4], [6], [7], [9]. However, most existing schemes assume read-only Web browsing during disconnection. Of particular interest is the eNetwork Web Express system [2] which uses a coherency interval associated with a Web page to specify how often a cached Web page should be checked for changes, e.g., one day for text pages and no limit for images. Since different coherency intervals apply to individual Web pages, it makes the notion of "a disconnection period" practically nonexistent when more than one page are cached in the MH. To support Web page write/create operations during disconnection, the Caubweb project [9], on the other hand, proposes modifying contemporary Web browers/servers. The basic idea is to support disconnected updates via a HTTP (Hypertext Transfer Protocol) client proxy running on the MH side to cache staging updates while disconnected, and a PUT script running on the HTTP server to accept PUT requests from the HTTP client proxy upon reconnection. ARTour Web Express [1] and ROVER [5] both allow staging updates to be incrementally and asynchronously flushed back to the server to support intermittent or weak connections. The BAYOU system [14] proposes the notion of application-specific conflict resolution to allow application-specific merge algorithms to be applied when update conflicts are detected. Coda [7] also provides mechanisms to resolve update conflicts upon reintegration to support disconnected operations on files.

None of these systems above addresses the issue of *when* a disconnected MH should be reconnected to the server. Our work aims at addressing the issue of *how* and *when* staging updates should be propagated from a disconnected MH for wireless Web access in mobile client-server environments. We integrate the concept of *coherency interval* for supporting disconnected Web browsing [2] with the concept of *versioning and locking* as advocated by WEBDAV/IETF[1] [15], [16] for supporting disconnected write operations for wireless Web access. We first describe *how* these concepts can be integrated in designing update propagation algorithms applicable in

- *I.-R. Chen and N.A. Phan are with the Computer Science Department, Virginia Tech, Northern Virginia Center, 7054 Haycock Road, Falls Church, VA 22043. E-mail: {irchen, nphan}@vt.edu.*
- *I.-L. Yen is with the Department of Computer Science, University of Texas at Dallas, PO Box 830688, MS EC31, Richardson, TX 75083-0688. E-mail: ilyen@utdallas.edu.*

Manuscript received 23 Feb. 2001; revised 10 Nov. 2001; accepted 25 Feb. 2002.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number 11-022002.

1. WEBDAV stands for Web distributed authoring and versioning; it is a standard currently being defined by a working group under the Internet Engineering Task Force (IETF) to specify HTTP enhancements to support collaborative authoring on the Web.

the reintegration phase. We then develop analytical models with the goal of identifying *when* a disconnected MH should be reconnected to the server (after the prefetch phase) to propagate a set a Web pages so that the system's performance in terms of the communication cost is optimized during the reintegration phase based on these algorithms.

The rest of the paper is organized as follows: Section 2 states the system assumptions by means of a system model. Section 3 discusses three update propagation algorithms for supporting disconnected write operations in wireless mobile environments and derives cost expressions associated with these algorithm as a function of the disconnection period, along with the derivation of the optimal disconnection period for minimizing the communication cost for update propagation. Section 4 shows some numerical data identifying the length of the disconnection period under which the system performance is optimized and provides physical interpretations of the analysis results. Section 5 presents the simulation validation and discusses the sensitivity of the analysis result with respect to probability distributions. The applicability of the algorithms developed is discussed in Section 6. In particular, we exemplify how the analysis result can be applied to mobile client-server Web applications with a real-time deadline requirement. Finally, Section 7 concludes the paper and outlines possible future research areas deriving from this work.

2 System Model

2.1 Assumptions and Parameters

We assume that a number of Web pages will be prefetched and stored in the MH's cache during the prefetching phase. Some of these pages are read-only, while others may be updated by the MH during the disconnection phase if needed to facilitate distributed Web content authoring. We assume that a prefetching policy exists to determine which pages are to be prefetched, e.g., based on a prediction algorithm [3].

We assume that, for each Web cached page, the MH also obtains from the Web server some update history information in terms of a general parameter, i.e., the update rate of that Web page by all users of the system. This information can be collected by the Web server by monitoring the update history of the Web page. For a cached Web page i, we denote this parameter as λ_i^w. In addition, we assume that the MH has some idea of how often it is going to perform updates on each cached Web page. For each cached Web page i, we call this parameter λ_i. For read-only cached Web pages, $\lambda_i = 0$.

We assume that the Web server is located somewhere in the fixed network and is not moved during a Web session. The MH communicates with the Web server via expensive wireless links (to the base station) and, thus, will voluntarily disconnect itself from the Web server after the prefetch phase to avoid expensive communication prices and also to save the battery power. When the MH reconnects to the Web server, it enters into the reintegration phase during which it propagates all staging updates performed during the disconnection phase to the server to resolve update conflicts. During the reintegration phase, we assume that the MH uses an enhanced HTTP algorithm to connect to the server such that only one TCP/IP connection is required by making use of server/client proxies [9], instead of requiring a TCP/IP connection for every Web page access. For each modified page, the MH submits the differences between the original Web page prefetched from the server and the latest version that it modifies, as well as the version number associated with the original version, to the Web server. The Web server checks to see if the page has been modified during the MH's disconnection period by comparing the version number of the Web page it currently keeps with the version number submitted to it by the MH. If they are the same, the Web server will accept the update request and modifies the Web page accordingly based on the differences received from the MH; otherwise, the update request of the Web page is rejected.

If an update request (by means of a PUT request) is rejected, we assume that the MH will stay connected and will issue a request to write lock the Web page.[2] Differences relative to the MH's original version before updating are sent to the MH, so the MH can regenerate a new version. Then, the MH will apply an application-specific merge algorithm such as the UNIX diff3 program to resolve the update differences. After the update is done, the MH will propagate the changes to the Web server by means of differencing again and will then release the lock. It is assumed that the MH will stay online performing the merge operation and update propagation in this algorithm, i.e., the case in which the MH locks the page and then goes away will not occur. If the MH is forced to be disconnected because of environment changes (e.g., due to roaming), we assume that the lock will be broken by the server after a timeout period. This can be implemented by the server by attaching a timestamp to the lock and releasing the lock after a timeout period expires. The MH upon reconnection will discover that it does not own the lock anymore and will retry again by repeating part of the update propagation algorithm. This extra time overhead involved in this extreme case is not considered in this paper due to its small probability.

We assume that a MH communicates with the Web server via an intelligent server gateway located on the fixed network, e.g., it can be just the base station. Further, the communication time on the fixed network is negligibly small compared with that on the wireless link. This assumption is justified for future high-speed wired networks. While it is possible that optimization schemes based on caching, transcoding, and differencing may be used by the server gateway to minimize the volume of data sent over the wireless network, we will assume that two general cost parameters, T_1 and T_2, suffice for our analysis: T_1 is the one-way communication cost of transmitting a packet carrying the differences along the wireless link between the MH and the server gateway; and T_2 is the one-way communication cost of transmitting a simple acknowledgement/reply packet. These two parameters can be estimated by knowing more specific parameter values of the wireless network under consideration. Let s_r be the average size of a simple acknowledgement/reply packet. Let s_o be the average size of a Web page. Let p_m be the average fraction of any Web page being modified. Let B be the bandwidth of the wireless channel through which the MH communicates with the fixed network. Assume that the communication time in the fixed network is relatively small compared with that in the wireless network. Then, T_1 and T_2 can be estimated as

2. The implication of applying locks is that it helps push updates a little quicker at the expense of causing other clients possibly a longer lock waiting time. We assume that the effect of the latter can be ignored as the probability of concurrent update propagations upon reconnection is low.

$$T_1 = \frac{p_m s_o}{B}, \quad (1)$$

$$T_2 = \frac{s_r}{B}. \quad (2)$$

From the MH's perspective, T_1 accounts for the time for transmitting the PUT update request that carries the version number of the original cached page and the differences between the latest version and the original prefetched version. From the server's perspective, T_2 accounts for the time for transmitting a reply from the Web server to the MH.

For notational conveniences, let $T = T_1 + T_2$, representing the round-trip communication cost for propagating a Web page that has been updated by the MH, but has not been updated by the server at the reconnection time. Normally, $T_1 >> T_2$, so $T \approx T_1$.

2.2 Performance Metric

The objective of the paper is to design and analyze algorithms for supporting write operations in mobile client-server environments and to identify the best reconnection time for propagating updates so as to optimize the system performance. We use the following metric as the basis of our design and analysis: The total communication (or reconnection) time between the MH and the Web server in the reintegration phase during which all updates are propagated to the Web server based on versioning and locking mechanisms supported by the Web server. A main goal of this paper is to find the best disconnection period so that the total communication (or reconnection) time for propagating updates is minimized, thereby saving the power consumption of the MH. For real-time Web applications, in addition to the above goal, we must also satisfy the requirement that the deadline not be missed. We will illustrate how our analysis result can be applied to real-time applications with such a requirement in Section 6.

Here, we should emphasize that we intentionally exclude (from the communication time metric above) certain costs that will always incur irrespective of when the MH is reconnected to the server, e.g., the connection setup time, the server processing time, etc. since adding such cost terms to the cost objective function doesn't affect the outcome of the analysis. Also, while it is possible that the MH can employ heuristics to possibly make more intelligent decisions about when it should reconnect to the server to adapt to resource changes (e.g., wireless bandwidth and server load changes), we will not consider this possibility in the paper.

3 Update Propagation Algorithms

We assume the following update propagation protocol is used when the MH propagates a modified Web page to the Web server during the reintegration phase. The MH sends the differences between the latest version and the original version, and also the version number of the original version in a PUT request packet. This step takes T_1 time. There are two possible cases following this step:

1. If the Web page has not been updated by other users, the Web server will send an ACK reply packet to the MH to accept the PUT request. In this case, the communication cost for the MH to update the modified Web page to the server is exactly $T_1 + T_2$ or simply T. This case is illustrated in the top part of Fig. 1.
2. If the Web page has been updated by other users and thus two versions do not match, the Web server sends a rejection reply packet to the MH (which takes T_2 time). When the MH is informed, it sends a lock request packet to the Web server to lock the Web page (which takes another T_2 time). On receiving the lock request packet, the Web server acts accordingly and also sends the new up-to-date version to the requesting MH via differencing (which take T_1 time). After the MH applies a merge algorithm (possibly with some manual inspection if not done automatically) to resolve the update conflict based on the new version received (which takes D_m time to be defined below), it sends another request packet (which takes T_1 time to transmit), carrying the differences to the Web server along with a command to release the lock, for which the Web server acts accordingly and afterward sends a reply packet to the MH to complete the update process (which takes another T_2 time). The overall communication cost in the latter scenario is $3T_1 + 3T_2 + D_m = 3T + D_m$, where D_m stands for the average time taken to resolve the update conflict by the MH. This case is illustrated in the bottom part of Fig. 1.

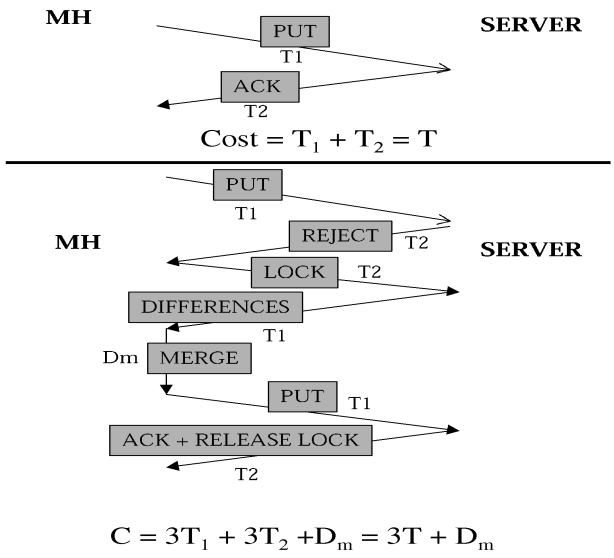

Fig. 1. Nonforced Update Protocol.

Many online Web applications necessitate *forced updates* when the MH is reconnected to the server, i.e., the MH must propagate its updates to the server at an appropriate reconnection time so as not to miss the deadline requirement. This real-time requirement calls for a special handling protocol for those pages that have not been updated by the MH at the time of reconnection and also mandates that the MH be reconnected to the server at an appropriate time prior to the real-time deadline to account for the time needed to propagate updates.

We assume that if the MH must perform a forced update on a cached Web page which has not been updated by the MH at the reconnection time, it will first issue an inquiry packet carrying the Web page's version number to the server to check if the cached page is still valid (an operation

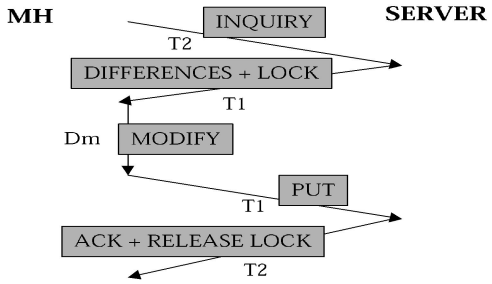

Fig. 2. Case 1 of Forced Update Protocol.

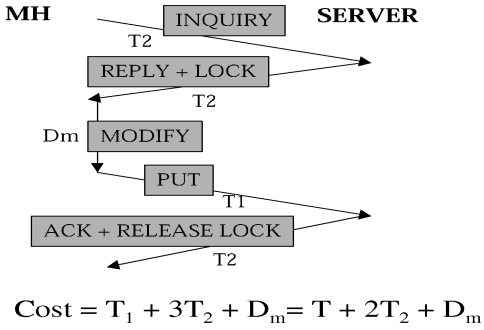

Fig. 3. Case 2 of Forced Update Protocol.

which takes T_2 time). Then, one of the following two cases will occur:

1. **Case 1**. If the server has updated the Web page, the server will send the differences to the MH so that the MH can perform its forced update operation based on the new version received. This conditional step takes T_1 time.
2. **Case 2**. If the server has not updated the Web page, the server will simply send a reply packet to the MH indicating that the MH's cached page is still valid. This conditional step takes T_2 time.

In either case, the particular Web page will be locked to prevent other users from updating it. The MH subsequently modifies the Web page (which takes D_m time) while it is online and propagates the differences to the server (which takes T_1 time). The server then will send an ACK to the MH and release the lock (which takes T_2 time). Summarizing above, Case 1 will take a total communication cost of $2T_1 + 2T_2 + D_m = 2T + D_m$ (see Fig. 2), while Case 2 will take a total communication cost of $T_1 + 3T_2 + D_m = T + 2T_2 + D_m$ (see Fig. 3).

Table 1 summarizes the parameters which will be used in the paper. Suppose that the length of the disconnection period is L. Upon reconnection, the MH will attempt to propagate the update of a modified Web page. The PUT request will be denied, however, if the Web server has modified the Web page during L. Thus, r_i, the probability that an update request for page i is rejected by the server upon reconnection after the MH has been disconnected for a period of L, is given by

$$r_i = Prob\{server\ update\ time\ to\ page\ i \leq L\} = F_i^s(L), \quad (3)$$

where $F_i^s(t)$ is the cumulative distribution function (CDF) of the time t that the server updates page i. For the case that updates to page i arrive at the system as a Poisson process, with rates λ_i and λ_i^w by the MH and by the world, respectively, then,

$$r_i = 1 - e^{-(\lambda_i^w - \lambda_i)L}. \quad (4)$$

Since we are interested in estimating the cost of propagating updates to the Web server during the reintegration phase, we like to know the probability that a Web page has been modified by the MH at the end of the disconnection phase. Suppose that the length of the disconnection phase is L again. Then, p_i, the probability that page i is updated by the MH within a period of L, is simply given by

$$p_i = Prob\{MH\ update\ time\ to\ page\ i \leq L\} = F_i^{MH}(L), \quad (5)$$

where $F_i^{MH}(t)$ is the CDF of the time t that the MH updates page i. For the case that updates to page i arrive at the system as a Poisson process with rate λ_i by the MH,

$$p_i = 1 - e^{-\lambda_i L}. \quad (6)$$

Below, we consider three update propagation algorithms. The first is for a special case in which the MH caches only a single Web page. The second considers the case where the MH has multiple cached Web pages but it propagates one Web page update at a time to the server. The third model also considers multiple cached Web pages; however, the MH propagates all the updates to the Web server in "batch" using as few message exchanges as possible in order to reduce the reconnection time. The third case requires the server and the MH to pack and unpack individual Web page updates embedded in a message.

3.1 Single-Page Update Propagation Algorithm

We first consider the case in which the MH prefetches only a single cached Web page. Here, we apply the concept of coherency interval to determine the best disconnection period for the MH. Recall that under the coherency interval method for supporting read-only Web browsing, an access to a cached page is allowed to proceed as long as the access time is within the interval. If the access time is out of the interval, then the MH must reconnect to the server and fetch a new copy if it is different from the cached copy. In the previous work [2], the length of the interval is heuristic in nature, defined arbitrarily either by the system or by the user.

We modify the above algorithm to support write operations on a single Web page, say page i, as follows:

1. If the time of modifying the cached page by the MH is less than L (relative to the beginning of the disconnection phase), we allow the modification to proceed by recording the differences between the latest version and the previous version in the MH's local cache/log without propagating the update to the server. All subsequent updates to the cached page, if any, are processed locally until the time reaches L, at which point the MH is reconnected to the Web server and updates are propagated by following the nonforced update protocol described earlier. Since at time L, the probability that the PUT request is rejected by the server is r_i, the

TABLE 1
Parameter List

λ_i	update rate for web page i by the MH
λ_i^w	update rate for web page i by the world (including by the MH)
s_o	average size of a cached web page
s_r	average size of an acknowledgement/reply packet
p_m	fraction of a web page being modified by the MH
B	bandwidth of a wireless channel
T_1	average one-way wireless communication time to propagate differences of two versions of a web page between the MH and its server gateway
T_2	average one-way wireless communication time to send an ACK/reply between the MH and its server gateway
T	$T_1 + T_2$
D_m	average time to execute a merge algorithm to resolve update conflicts
L	length of the disconnection period
r_i	probability that page i is updated by the server within a time period of L
p_i	probability that page i is updated by the MH within a time period of L
C	total cost for propagating updates to the web server upon reconnection

communication cost of update propagation is $3T + D_m$ with probability r_i or T with probability $1 - r_i$.

2. If the time of modifying the cached page by the MH is greater than L, that is, the MH has not updated the page by the time L is reached, a forced update is performed at L, i.e., the MH is forced to perform an update at time L. The MH at time L does not know if the cached page is valid or not. To avoid performing the forced update operation on an out-of-date version, it sends a message carrying the version number of its cached page to the server, for which the server responds by locking the page and, if the page has been updated during L, also sends the difference to the MH.

   The communication time in this case thus depends on whether the cached page has been updated by the server or not at time L. Following the forced update protocol discussed earlier, it can be obtained by considering the following two subcases:

   a. The cached page has been updated by others during L and, therefore, a fresh copy must be retrieved from the server before a forced update by the MH can be performed. In this case, the communication time is $2T + D_m$.
   b. The cached page is still valid, i.e., not being updated by others during L. In this case, the communication time is $T + 2T_2 + D_m$.

   Note that the probabilities of the above two subcases are r_i and $1 - r_i$, respectively.

Summarizing all of the above, the average communication time involved in propagating the update from the MH to the server for a single Web page upon reconnection is thus given by

$$C_i = p_i \times [r_i \times (3T + D_m) + (1 - r_i)T] + (1 - p_i) \times [r_i \times (2T + D_m) + (1 - r_i)(T + 2T_2 + D_m)]. \quad (7)$$

By taking the differentiation of C_i with respect to L to zero, we can determine the best L value (representing the disconnection period) that minimizes the total reconnection time. The expressions of r_i and p_i depend on the underlying assumption regarding the update interarrival time distribution. For the case that updates to page i arrive at the system as a Poisson process, with rates λ_i and λ_i^w by the MH and by the world, respectively, we plug in the expressions for r_i and p_i given in (4) and (6) into (7) and take the derivative of the resulting C_i with respect to L to zero, yielding

$$(2T + D_m)(\lambda_i^w - \lambda_i)e^{-(\lambda_i^w - \lambda_i)L} + T\lambda_i e^{-\lambda_i L} - (T + 2T_2 + D_m)\lambda_i^w e^{-\lambda_i^w L} = 0. \quad (8)$$

The best L that minimizes the communication cost must satisfy the above equation.

Note that the best L exists as long as $\lambda_i < \lambda_w$. For the extreme condition that the MH is the only process updating the Web page such that $\lambda_i = \lambda_w$, the above equation reduces to

$$-(2T_2 + D_m)\lambda_i e^{-\lambda_i L} = 0$$

which means that under the extreme case $\lambda_i^w = \lambda_i$, the best L would be at ∞. This is because at $L = \infty$, the probability of Web page update $p_i = 1$ and, thus, $C_i = T + 2T_2 + D_m - p_i(2T_2 + D_m) = T$ is at its minimum with no forced-update cost. We observe that, in this extreme case, there exists a time point, $L_{subopt} < \infty$, beyond which the cost is nearly the same as the minimum cost at $L_{opt} = \infty$. Thus, if $\lambda_i^w \approx \lambda_i$ (i.e., the MH dominates the Web page update), in order to avoid an extremely large disconnection period in this case, the MH should connect to the system at L_{subopt} such that the cost is close to the minimum (whose exact value is determined by (7)). In addition, if the application has a real-time deadline, say t_R, then the MH should connect to the system such that the deadline is not violated. This latter case will be addressed in Section 6.

3.2 Multiple-Page Update Propagation Algorithm 1

We now consider the case in which the MH prefetches a set of cached Web pages, say, based on a prediction algorithm. All updates occurring before L are again staged and later propagated back to the server at time L relative to the beginning of the disconnection phase. Moreover, all Web pages not updated during L are also forcibly updated,

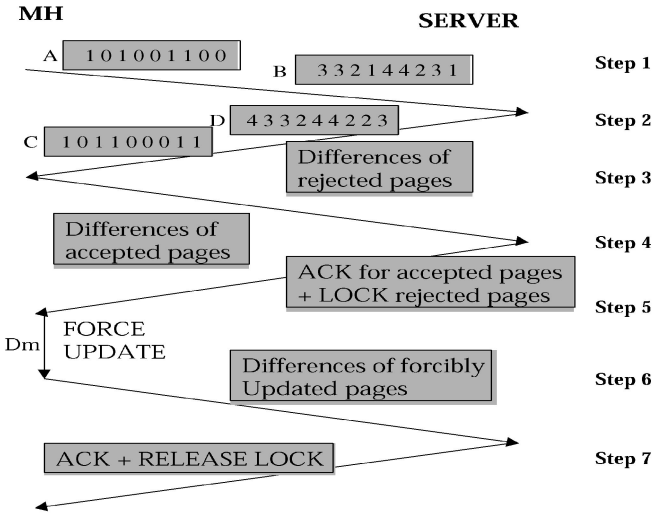

Fig. 4. Multiple-Page Update Propagation Algorithm 2.

following the algorithm described in the previous section. Again, we are interested in identifying the best disconnection period that will optimize the system performance. In this section, we consider the case in which modified Web pages are propagated to the server one at a time. In the next section, we will consider the case in which all modified Web pages are propagated back to the server in "batch" to further save the communication cost.

Without loss of generality, consider a particular Web page, say i, in a set of N prefetched pages. Again, let λ_i and λ_i^w be the update rates on page i by the MH and by the world, respectively. Since the MH propagates one page at a time to the server upon reconnection at time L, the total reconnection time needed for the MH to propagate updates of all N pages to the server is simply given by

$$C_N^{case\ 1} = \sum_{i=1}^{N} C_i, \tag{9}$$

where C_i is given by (7), standing for the reconnection time needed for propagating updates to page i. Again by taking the differentiation of $C_N^{case\ 1}$ to zero, we can find the best L that minimizes $C_N^{case\ 1}$. The derivation is similar to that in Section 3.1 and is not repeated here.

3.3 Multiple-Page Update Propagation Algorithm 2

Here, we also deal with the case in which the MH prefetches a set of Web pages before disconnection. However, when the MH reconnects at time L, the server and the MH execute a more sophisticated algorithm to propagate updates in batch in order to shorten the reconnection time. Fig. 4 illustrates the following seven steps taken to execute the algorithm:

1. The MH sends to the server a bit vector A (carrying 0 or 1 values) indicating which cached pages have been updated by the MH, and also a value vector B (carring integer values) indicating the original version numbers associated with the cached Web pages prefetched into the MH before disconnection. This is done by sending a single message from the MH to the server. The time needed for this step is T_2.
2. The server decides accepting or rejecting page updates based on vectors A and B received from the MH and the current version numbers associated with the requested pages stored in the server. It then sends a bit vector C indicating which pages can be accepted (for which the value is 0) and which pages are to be rejected (for which the value is 1). Those pages not updated by the MH have their corresponding values also marked with 1 in vector C. The server also sends a separate version number vector D indicating the current version numbers of the pages stored in the server. All the above information are embedded in a single message sent from the server to the MH. The time for this step is also T_2.
3. For those pages not accepted by the server, the server locks those pages to prevent them from being updated by other users in the system. Simultaneously, the server sends to the MH the differences of those pages rejected by the server (because the server has updated those pages) and also those pages that have not been updated by the MH but have been updated by the server. The time for this step is
$$\sum_{i=1}^{N} r_i T_1$$
because, on average, $\sum_{i=1}^{N} r_i$ pages (out of N) are updated by the server during $[0, L]$, for which the server must send the differences to the MH so that the MH can perform forced updates on the newest version.
4. When the MH receives vectors C and D, it knows immediately which pages are accepted by the server. Those accepted pages (i.e., those updated by the MH but not updated by the server during $[0, L]$) are then sent to the server by means of differencing. The time for this step is
$$\sum_{i=1}^{N} p_i(1 - r_i)T_1$$
because on average $\sum_{i=1}^{N} p_i(1 - r_i)$ pages are updated by the MH but not updated by the server (thus are accepted by the server) during $[0, L]$.
5. The server sends an acknowledgement to the MH after it receives and processes the update propagation for those accepted pages. The time for this step is T_2.
6. The MH then performs forced updates for those pages not having been accepted by the server, including a) pages updated by the MH and also updated by the server; b) pages not updated by the MH but updated by the server; and c) pages not updated by the MH and not updated by the server. After the MH receives differences for those pages updated by the server during $[0, L]$, it performs merge/update operations on all of the above three types of pages. Then, it sends differences of the newest versions of all these pages to the server. The time for this step is given by
$$\sum_{i=1}^{N} r_i(D_m + T_1) + \sum_{i=1}^{N} (1 - r_i)(1 - p_i)(D_m + T_1),$$

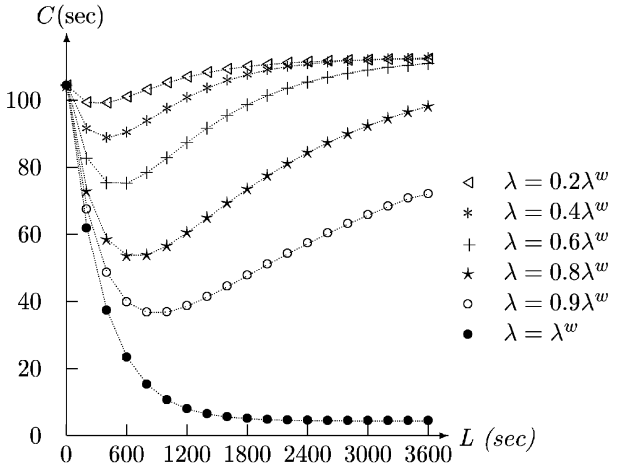

Fig. 5. Effect of λ/λ^w ratio on Single-Page Update Propagation Algorithm.

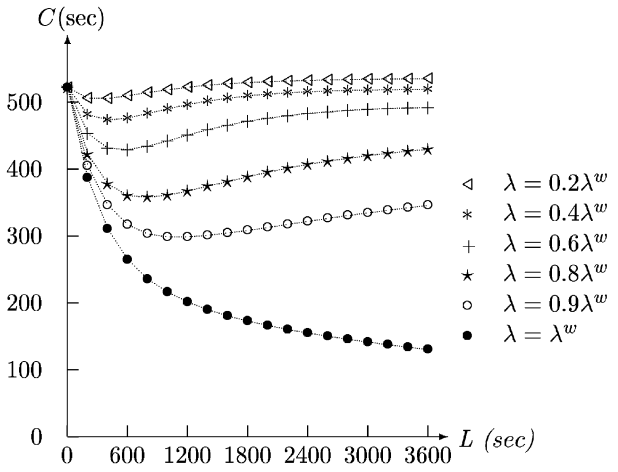

Fig. 6. Effect of λ/λ^w ratio on Multiple-Page Update Propagation Algorithm 1.

where the first term accounts for the time taken to generate and propagate updates for the first two types of pages (i.e., those updated by the server during $[0, L]$) based on the versions received from the server; the second term accounts for the time taken to generate and propagate updates for the third type of pages (i.e., those not updated by either the MH or the server during $[0, L]$).

7. The server sends an acknowledgement to the MH after it receives and processes the update propagation for those pages being forcibly updated by the MH in the previous step. All locks are also released in this step. The time for this step is T_2.

The total reconnection time to propagate updates of all cached pages is therefore equal to the sum of all the individual times in steps 1 to 7 above, i.e.,

$$C_N^{case\ 2} = 4T_2 + \sum_{i=1}^{N}[r_i T_1 + p_i(1-r_i)T_1 + r_i(D_m + T_1) + (1-r_i)(1-p_i)(D_m + T_1)]. \quad (10)$$

For the case that updates to page i arrive at the system as a Poisson process, with rates λ_i and λ_i^w by the MH and by the world, respectively, the best L value that minimizes the communication cost will satisfy the following equation:

$$\sum_{i=1}^{N}\left[(T_1 + D_m)(\lambda_i^w - \lambda_i)e^{-(\lambda_i^w - \lambda_i)L} - D_m\lambda_i^w e^{-\lambda_i^w L}\right] = 0. \quad (11)$$

Again, in the extreme case that $\lambda_i^w = \lambda_i$ and thus $r_i = 0$ and $p_i = 1$ for all i, $C_N^{case\ 2}$ is at its minimum, i.e., $4T_2 + NT_1$, when $L_{opt} = \infty$. Thus, if $\lambda_i^w \approx \lambda_i$ for all is (i.e., the MH dominates the Web page update activities for the set of pages prefetched), then similar to the arguments discussed in Section 3.1, there exists a L_{subopt} beyond which the cost (as determined by (10)) is nearly the same as the minimum cost at $L_{opt} = \infty$. To avoid an arbitrarily long disconnection period, the MH should reconnect to the system at L_{subopt}. Similarly, if a real-time deadline exists, then the MH should connect to the system such that the real-time requirement is not violated. Section 6 will deal with the latter case.

4 Numeric Example

In this section, we present numerical data based on the analytical result derived in Section 3 for the case in which the update interarrival time to a Web page is exponentially distributed. Later, in Section 5, we validate the analytical via simulation and analyze the sensitivity of the result with respect to the probability distribution of the update interarrival time by considering other general types of distributions.

Three cases to analyze the effect of different parameters $(\lambda, \lambda^w, D_m, p_m, s_o)$ on the optimal disconnection period (L) are reported:

1. the effect of the ratio between the MH update rate and the world update rate (λ/λ^w);
2. the effect of the average time to execute a merge algorithm (D_m);
3. the effect of the size of differences between two versions $(p_m s_o)$.

Other cases can be found in [11]. For each case, we obtain analytical results under three algorithms: single-page

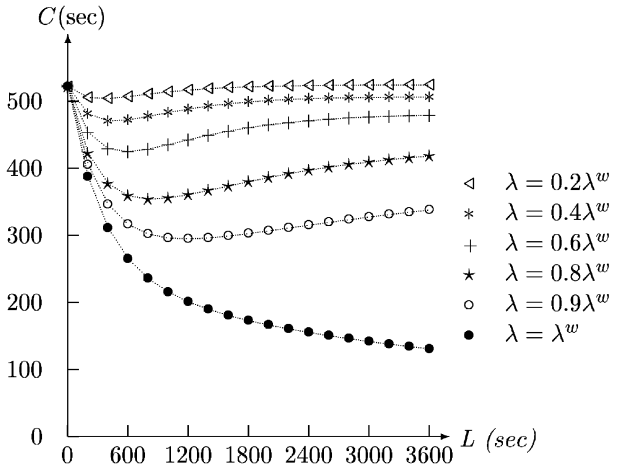

Fig. 7. Effect of λ/λ^w ratio on Multiple-Page Update Propagation Algorithm 2.

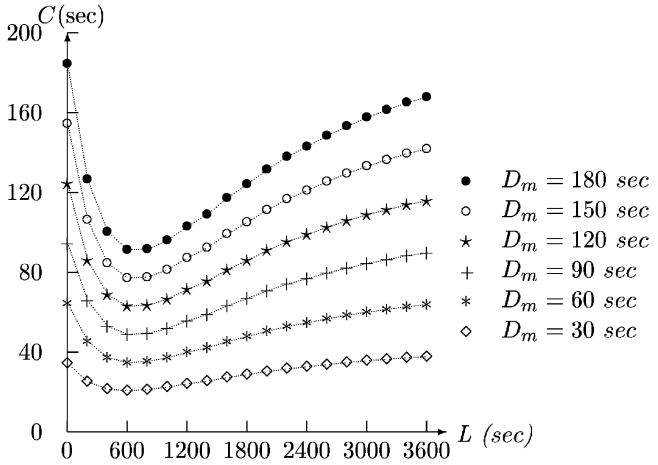

Fig. 8. Effect of D_m on Single-Page Update Propagation Algorithm.

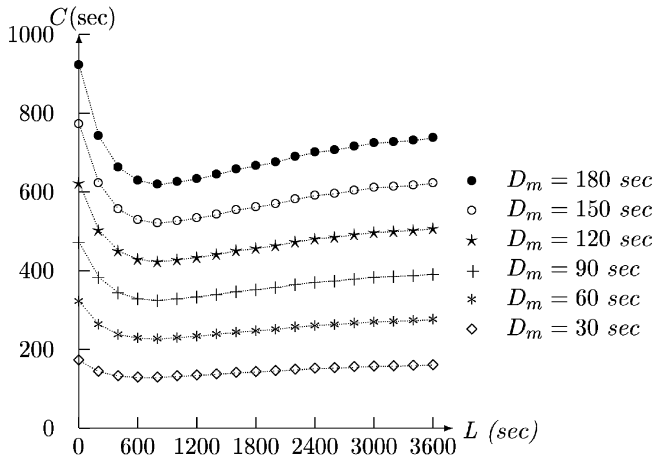

Fig. 9. Effect on D_m on Multiple-Page Update Propagation Algorithm 1.

update propagation (Section 3.1), multiple-page update propagation algorithm 1 (Section 3.2), and multiple page update propagation algorithm 2 (Section 3.3). We provide a physical interpretation of these results.

4.1 Effect of λ/λ^w Ratio

To demonstrate the effect of λ/λ^w ratio, we fix $T_1 \approx 5$ seconds, $T_2 \approx 0$ second, $D_m = 100$ seconds. We arbitrarily select a λ^w value and vary the λ value from $0.2\lambda^w$ to λ^w.

For the single-page update propagation algorithm, we consider the case where the MH updates a Web page for which the world will update it with a rate $\lambda^w = 10$ updates/hour (in Fig. 5). For the multiple-page update propagation under both algorithms, we consider the case where the MH updates five Web pages (an arbitrary selection) for which the corresponding λ^w values are arbitrarily selected in the range of [0, 15] updates/hour (in Figs. 6 and 7, respectively).

We observe that as the MH's update rate λ varies from $0.2\lambda^w$ (the top curve) to λ^w (the bottom curve), the curves get deeper. This means that when the MH's update rate is closer to the world update rate, the MH can save more communication cost by propagating updates at the optimal disconnection point L. An exception occurs when λ equals λ^w, which means that the page is updated only by the MH. In this extreme case, there is essentially no optimal disconnection period, i.e., the optimal disconnection period equals infinity. In this special case, the MH can connect to the system at L_{subopt} (say 1,500) at which the cost is very near to the optimal cost. Last, we also observe that the optimal disconnection point shifts toward right as we go from the top curve to the bottom curve. This indicates that the reconnection time interval (L) is shorter when the MH's update rate is far below the world update rate, and it is longer when the MH's update rate is close to the world update rate.

4.2 Effect of the Time to Perform Merge Operations

To demonstrate the effect of the time to perform the merge operation, we selectively fix the λ/λ^w ratio to 0.8, $T_1 \approx 5$ seconds, $T_2 \approx 0.1$, and vary the D_m values in the range of 30 to 180 seconds. Figs. 8, 9, and 10 show the results.

We observe that as the time to perform merge operations varies from 30 seconds (the bottom curve) to 180 seconds (the top curve), the curves get deeper as the D_m value gets larger for all algorithms. This means that the MH can save more communication cost by propagating updates at the optimal disconnection point when the merge operation requires more time to perform.

4.3 Effect of the Size of Differences between Two Versions

To demonstrate the effect of the size of the differences between two versions, we selectively fix λ/λ^w to 0.8 and the D_m value to 100 seconds. We vary the fraction of the Web page modified by the MH (p_m values) in the range of $[10\%, 50\%]$, and the average size of a Web page (s_o values) in the range of [50 KB, 250 KB]. In this case, the size of the version differences ($p_m s_o$ values) will vary from 5 KB to 25 KB. Figs. 11, 12, and 13 show the results.

As the size of the version differences varies from 5 KB (the bottom curve) to 25 KB (the top curve), we observe that, for all cases, the curves get deeper as the product of p_m and s_o gets larger. This means that the MH can save more communication cost by propagating updates at the optimal disconnection point when it modifies a larger portion of the Web pages or a larger Web page.

When comparing multiple-page update propagation algorithms 1 and 2, we observe that the MH can save

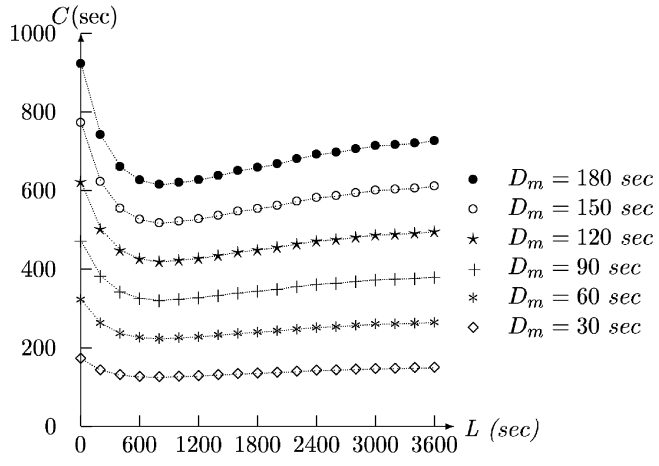

Fig. 10. Effect on D_m on Multiple-Page Update Propagation Algorithm 2.

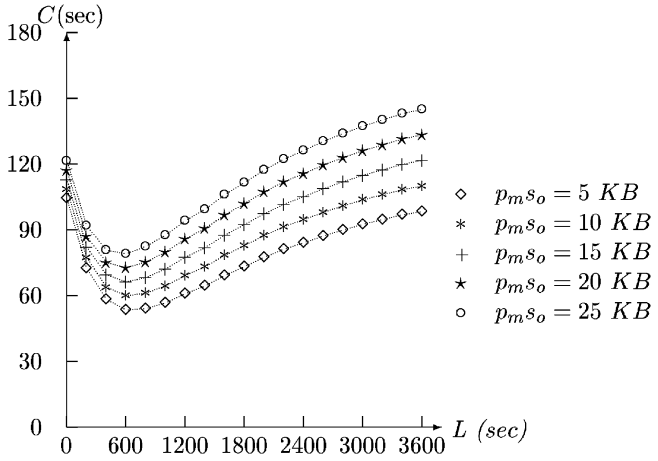

Fig. 11. Effect of $p_m s_o$ on Single-Page Update Propagation Algorithm.

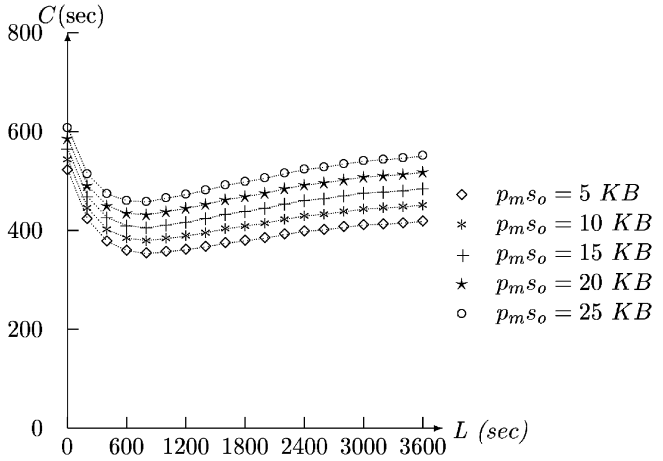

Fig. 13. Effect of $p_m s_o$ on Multiple-Page Update Propagation Algorithm 2.

more communication cost by propagating updates under algorithm 2. Algorithm 2 is especially effective when the MH modifies a large portion of the Web pages or a large Web page.

5 Simulation

All analytical results presented in Sections 4.1-4.3 are validated via simulation. A simulation model was developed based on smpl [8] running on a SUN Ultra 10 machine with Solaris. To simulate update events, we randomly generate the time when an update will occur to page i by the MH (T_i^c) and by the server (T_i^s) based on the probability distribution of the update interarrival time. We first report the simulation results obtained based on exponential distributions to validate results obtained in Section 4. Then, we report the sensitivity of the results obtained with respect to the underlying probability distribution by considering other types of time distributions, viz., Erlang, hyperexponential, normal, and uniform. For each value of L in the x-coordinate, we generate a value of C in the y-coordinate as follows: We compare L with T_i^c and T_i^s and use the update propagation algorithms described earlier in Section 3 to determine C. The number of states that page i could be in is given by $p(3,2) = 3!/(3-2)! = 6$ as there are six distinct ways to arrange L, T_i^c, and T_i^s (see Table 2 for illustration).

According to our protocol when both the MH and server (i.e., other users) have updated a Web page before the disconnection period, the order in which who updates the page first does not affect the total communication cost. This also applies to the case when both the MH and the server have not updated a Web page before the disconnection period. Therefore, for each Web page, there are four remaining possible states:

State 1. The MH has updated the page but the server has not updated the page, i.e., $T_i^s > L > T_i^c$.

State 2. The MH has updated the page and the server has also updated the page, i.e., $L > T_i^c, T_i^s$.

State 3. The MH has not updated the page but the server has updated the page, i.e., $T_i^c > L > T_i^s$.

State 4. The MH has not updated the page and the server also has not updated the page, i.e., $T_i^c, T_i^s > L$.

Table 3 shows the possible states and the associated communication costs (C) for the single-page update propagation algorithm.

If the MH updates N pages, the total number of possible states for these N pages altogether is given by 4^N. Tables 4 and 5 show examples of calculating C for a five-page update propagation case under multiple-page update propagation algorithms 1 and 2, respectively.

We use the batch means analysis method to obtain the average communication cost (C) for a given L value with 5 percent accuracy and 95 percent confidence levels. To collect data and get the average value of a batch run, we run

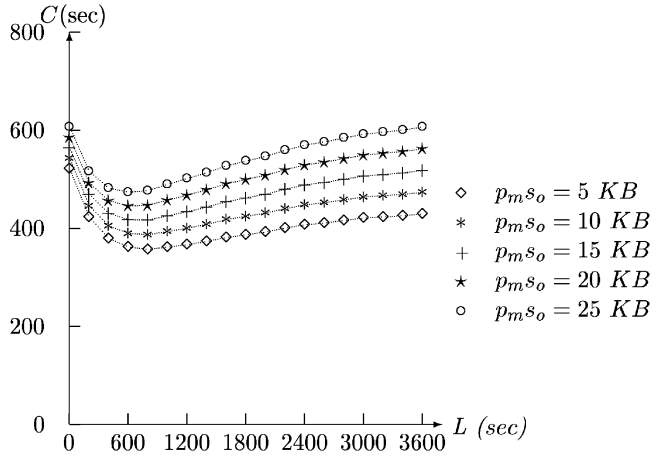

Fig. 12. Effect of $p_m s_o$ on Multiple-Page Update Propagation Algorithm 1.

TABLE 2
Possible State of a Web Page

State 1: $T_i^s > L > T_i^c$
State 2: $L > T_i^c > T_i^s$
State 3: $L > T_i^s > T_i^c$
State 4: $T_i^c > L > T_i^s$
State 5: $T_i^c > T_i^s > L$
State 6: $T_i^s > T_i^c > L$

TABLE 3
Communication Cost under Single-Page Update Propagation

State	Cost
1	$T_1 + T_2$
2	$3T_1 + 3T_2 + D_m$
3	$2T_1 + 2T_2 + D_m$
4	$T_1 + 3T_2 + D_m$

2,000 simulation runs in one batch run, where each simulation run returns a C value. After k batch runs, we compute the sample mean and the sample variance to determine if the accuracy and confidence levels have been achieved. If not, we run another batch run. We use $k = 10$ in the simulation. The sample mean is given by

$$\overline{C} = \frac{\sum_{i=1}^{k} C_i}{k}.$$

The sample variance is given by:

$$S^2 = \frac{(\sum_{i=1}^{k} C_i^2) - k * \overline{C}^2}{k-1}. \quad (12)$$

The confidence level is determined as

$$prob[\overline{C} - H \leq \mu \leq \overline{C} + H] = 1 - \alpha, \quad (13)$$

where H is given by

$$H = t_{\frac{\alpha}{2};k-1} * \frac{S}{\sqrt[2]{k}}. \quad (14)$$

The accuracy level is determined as

$$\frac{H}{C} \leq \beta. \quad (15)$$

The parameters used in the simulation model are shown in Table 6.

For all data generated by our simulation based on exponential distributions for the generation of update interarrival times, we observed virtually the same curves shown earlier in Figs. 5, 6, 7, 8, 9, 10, 11, 12, and 13. The difference between simulated and analytical data is less than 0.1 percent. We show an example of the simulation results for the effect of λ/λ^w on the single-page update propagation algorithm in Fig. 14. This figure is virtually identical to Fig. 5 obtained earlier from the analytical results.

We tested the sensitivity of the results obtained with respect to the probability distribution of the update interarrival time by considering other types of probability distributions including Erlang,[3] hyperexponential,[4] normal, and uniform. Tables 7 and 8 report the cost (C) obtained versus L for the case in which $\lambda_i = 0.5\lambda_i^w$ under different distributions for the single page update case. Other cases display similar sensitivity results and are not shown here. As we can see, although different distributions may slightly affect the exact position of L_{opt}, the shape of the curve is relatively insensitive to the underlying probability distribution used. All data curves exhibit virtually the same shape with the optimal L_{opt} in the range of [400,600]. Thus, we conclude that the analysis results reported in the paper are valid and are not sensitive to the probability distribution used.

3. The Erlang distribution models a r-stage exponential center connected in a series structure such that it has the same mean interarrival time $x = 1/\lambda_i$ for the MH update and $x = 1/\lambda_i^w$ for the world update as that under the exponential distribution with the standard deviation s (square root of the variance) less than x. The number of stages is equal to $\lfloor (x/s)^2 \rfloor$.

4. The hyperexponential distribution models a 2-stage exponential center in a parallel structure such that the mean (x) is the same as that under the exponential distribution but the standard deviation (s) is higher than the mean.

TABLE 4
An Example of Calculating the Communication Cost under Algorithm 1 for a 5-Page Update Propagation Case

Web page number	State	Communication cost
1	2	$3T_1 + 3T_2 + D_m$
2	4	$T_1 + 3T_2 + D_m$
3	3	$2T_1 + 2T_2 + D_m$
4	4	$T_1 + T_2 + D_m$
5	1	$T_1 + T_2$
Total communication cost		$8T_1 + 10T_2 + 4D_m$

6 Application

In this section, we show how the analysis results can be applied to mobile client-server Web applications. Examples include distributed calendar tools or meeting room schedulers as having been considered in BAYOU and ROVER mobile systems [4], evolving design document/form editing systems with several authors updating the same set of Web pages concurrently as described by WEBDAV [15], and online newspaper/journalism/sports services [17]. We consider the case in which these Web pages are updated by multiple users, some of which may be stationary while some of which may be mobile so as to collect the needed information on the go. Regardless of who is performing the update, there is a real-time deadline by which these Web pages must be updated. We denote this deadline by t_R whose magnitude depends on the application. For stock Web servicing applications, this deadline may be in the order of seconds. For less urgent applications such as distributed document editing, this deadline may be in the order of minutes or hours.

Let the optimal disconnection period be denoted as L_{opt} as a function of model parameters. If this optimal dis-

TABLE 5
An Example of Calculation the Communication Cost under Algorithm 2 for a 5-Page Update Propagation Case

Step	Cost
1	T_2
2	T_2
3	$2T_1$
4	T_1
5	T_2
6	$4T_1 + 4D_m$
7	T_2
Total communication cost	$7T_1 + 4T_2 + 4D_m$

TABLE 6
Parameters used in the Simulation

T_i^c	a randomly generated time interval over which an update will occur on the MH side
T_i^s	a randomly generated time interval over which an update will occur on the server side
C_i	average value of the communication cost in a batch run
$\overline{C}$	sample mean of the communication cost in k batch runs
k	number of batch runs ($k = 10$)
μ	true mean of the communication cost
H	the difference between the true mean and the sample mean
$1 - \alpha$	confidence level (95%)
β	accuracy level (5%)
$t_{\frac{\alpha}{2};k-1}$	t-distribution based on $\alpha/2$ and $k - 1$

connection time plus its associated update propagation time is less than the deadline, then the MH should reconnect to the server at L_{opt}, to reduce the communication cost and power consumption; otherwise, it should reconnect to the server at the time point less than L_{opt} such that the deadline is equal to the sum of the selected disconnection time (less than the optimal point) and the associated update propagation time (as a result of selecting the disconnection time) so as to avoid deadline violation. In all numerical results presented so far, as shown in Figs. 5, 6, 7, 8, 9, 10, 11, 12, and 13, we have used x-y diagrams showing the relationship between the selected disconnection time period L (the x-coordinate) versus the associated reconnection time C (the y-coordinate) needed for update propagation based on that selection, thus giving an estimate of the reconnection time needed for update propagation when given a disconnection period. Specifically, let (L, C) denote any point in a x-y diagram and let (L_{opt}, C_{min}) denote the optimal L point at which the cost is minimum. For real-time applications with a deadline of t_R, if $L_{opt} + C_{min} < t_R$, then L_{opt} is the disconnection time period of choice; otherwise, we select the largest $L < L_{opt}$ such that $L + C = t_R$. Of course, for non-real-time applications, we always select L_{opt} as the disconnection time period to reduce the communication time.

To illustrate how our earlier presented result can be applied to real-time Web applications with a deadline, consider that the data displayed in Fig. 15 are for a real-time Web application for which $t_R = 25$ minutes or 12,500 seconds and $\lambda = 0.5\lambda^w$ since two users are updating the same set of Web pages simultaneously with virtually the same rate. Then, from the figure we see that the MH should propagate its updates at $L_{opt} = 850$ since $L_{opt} + C_{min} = 850 + 350$ seconds which is less than 25 minutes. However, if on the other hand, the real-time deadline is 15 minutes, then the MH should disconnect itself from the server (while it performs updates) for only approximately 540 seconds because with that disconnection time period, the anticipated update propagation time is 360 seconds as predicted from the diagram, i.e., the point has the coordinate (540,360) such that $360 + 540 = 900$ seconds $= 15$ minutes.

7 Conclusions and Future Work

In this paper, we have discussed several algorithms for propagating updates made by the MH and analyzed their effect on performance in terms of the communication time needed to propagate updates, including the time needed to detect and resolve conflict if it happens. We have developed simple analytical arguments to relate the disconnection period of the MH with the associated communication time required based on the choice of disconnection period, taking into account some system conditions such as the update rates of the MH and the world, the time to communicate between the MH and server based on differencing/locking, and the time to resolve Web page conflicts, etc. The end result is a simple $L - C$ diagram which may be applied at the runtime by the MH to determine the longest time to stay disconnected (L) so as to minimize the communication time (C) needed to propagate updates.

Our analysis shows that there exists an optimal L_{opt} value under which C is minimized. Moreover, this optimal disconnection period increases as the mobile user update rate increases relative to the world update rate. At one extreme where only the mobile user performs updates, the communication cost is minimum at time equals ∞. In this case, the MH can connect to the system at a time point at which the communication cost is near the minimum or is at an acceptable level as identified by the cost expression derived in this paper. At the other extreme where the world's update rate is much higher than that of the MH, the communication cost is minimum at time equals 0 since the world is going to update the Web pages very often and the

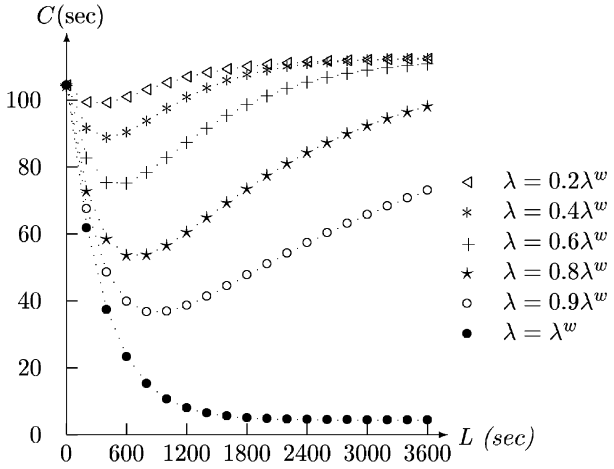

Fig. 14. Simulation data for the Effect of λ/λ^w ratio on Single-Page Update Propagation Algorithm.

TABLE 7
Cost (C) versus L Data Chart I with $\lambda_i = 0.5\lambda_i^w$ under Various Distributions

C (cost)	Analytical	Exp(x)	Hyper $(x, s = 1.5x)$	Hyper $(x, s = 1.3x)$	Hyper $(x, s = 1.1x)$	Erlang $(x, s = 0.9x)$	Erlang $(x, s = 0.7x)$
$L = 0$	104.30	104.27	104.27	104.27	104.27	104.27	104.27
$L = 200$	87.22	87.27	84.53	85.12	86.30	87.27	87.27
$L = 400$	82.46	82.72	82.61	82.50	82.11	82.72	82.72
$L = 600$	83.53	83.38	85.67	84.83	84.09	83.38	83.38
$L = 800$	87.04	87.13	90.49	89.11	87.57	87.13	87.13
$L = 1000$	91.23	90.99	94.35	93.06	91.81	90.99	90.99
$L = 1200$	95.30	95.29	97.67	96.43	95.27	95.29	95.29
$L = 1400$	98.88	99.05	100.18	99.41	98.66	99.05	99.05
$L = 1600$	101.89	101.85	102.48	101.95	102.10	101.85	101.85
$L = 1800$	104.34	104.09	103.98	103.75	104.10	104.09	104.09
$L = 2000$	106.28	106.11	104.88	105.20	105.71	106.11	106.11

x: mean, s: standard deviation.

TABLE 8
Cost (C) versus L Data Chart II with $\lambda_i = 0.5\lambda_i^w$ under Various Distributions

C (cost)	Erlang $(x, s = 0.5x)$	Normal $(x, s = 1.5x)$	Normal $(x, s = x)$	Normal $(x, s = 0.5x)$	Uniform $(a = x/2, b = 3x/2)$	Uniform $(a = 0, b = 2x)$
$L = 0$	104.27	86.66	91.52	102.18	104.27	104.25
$L = 200$	101.80	84.54	87.57	97.88	104.27	93.09
$L = 400$	89.79	83.18	84.24	89.97	99.23	85.67
$L = 600$	82.44	82.18	82.33	83.39	84.04	82.79
$L = 800$	85.79	82.82	83.09	83.89	84.21	82.74
$L = 1000$	95.23	85.26	86.83	93.39	101.33	87.87
$L = 1200$	102.53	87.18	91.29	103.44	112.80	97.32
$L = 1400$	107.60	89.32	95.88	109.54	112.80	109.83
$L = 1600$	110.41	93.45	101.51	111.93	112.80	112.80
$L = 1800$	111.73	97.93	105.88	112.63	112.80	112.80
$L = 2000$	112.31	101.14	108.70	112.78	112.80	112.80

x: mean, s: standard deviation.

benefit of disconnecting from the Web server is lost since the cost penalty for detecting and resolving conflicts dominates. In this case, it is better that the MH stays connected to the server, locks the Web pages, and propagates updates to the server while it is connected.

For those cases where the MH's update rate is a fraction of that of the world, we observed that there exists a finite value of L_{opt} under which the communication cost for update propagation is minimized. We also observed similar curves with other sets of parameter values as long as the time to execute the merge operation to resolve update conflicts D_m is not too small, i.e., more than 10 seconds, with all other parameter values fixed. When D_m is small, the optimal disconnection time interval, i.e., L_{opt} approaches ∞ because the penalty of rejection is virtually zero. In practice, D_m is large relative to other parameter values since a manual inspection may be required even with the help of merging tools when the MH resolves update conflicts.

One application of the analysis result as we have demonstrated in this paper is that for applications with a real-time deadline. We can use the result presented in the $L - C$ diagram to determine the time at which the MH should reconnect to the server so that the total time, including the disconnection time and the algorithm execution time for update propagation, is less than the deadline. The needed $L - C$ diagrams can be generated at the static time to cover a possible range of parameter values. Such results can be stored in a table in the MH which can then perform a table lookup at the runtime to adapt to environment changes at the runtime to reflect network conditions such as channel bandwidth. The (L, C) diagram

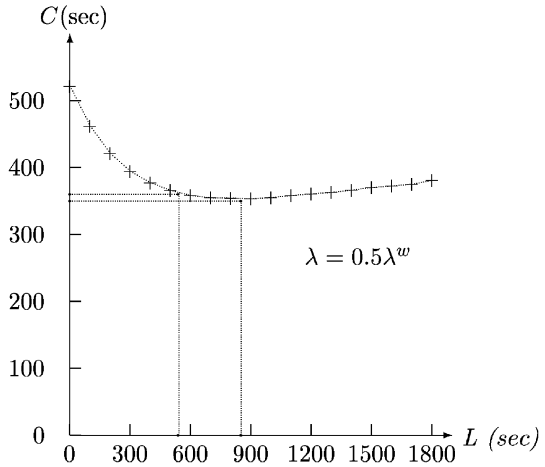

Fig. 15. Applying to real-time Web applications with deadline requirements.

generated from our analysis also provides information regarding "anticipated" arrivals of disconnected mobile users. This information can be very helpful to wireless mobile networks as the system allocates wireless channels to users.

There are some possible future research areas which can be extended from this work. We can investigate how a channel allocation manager in the mobile network could use the (L, C) diagram information to decide when a mobile user should be reconnected to the network so that the mobile user will stay connected with the least amount of time to propagate updates. We can further develop channel scheduling algorithms for optimal allocation of wireless channels taking into account both random and anticipated arrivals of mobile users. We can apply the result obtained from the paper to building real-time Web applications used in wireless mobile environments so that the MH can always select the longest disconnection time possible to reduce the communication cost without violating the real-time requirement and also to save its valuable battery power.

Also, Web applications which are not real time do not require mobile users to stay connected for propagating updates. In this case, The mobile user may just send the request to the server and then disconnect immediately. Later, it can reconnect to the server to check if the server has accepted/rejected the request. In the future, we can modify our analysis model to account for this possibility. Finally, as most Web applications are read-only applications, the read/write probability ratio may affect the performance of the overall system because, as Web pages are locked for update propagation, read operations cannot be served. Therefore, it is possible to develop algorithms taking into account the effect of the read/write ratio factor to optimize the overall system performance.

ACKNOWLEDGMENTS

This research is partially supported by a US National Science Foundation grant #9987586 and by a Microsoft Research grant.

REFERENCES

[1] H. Chang et al., "Web Browsing in a Wireless Environment: Disconnected and Asynchronous Operation in ARTour Web Express," *Proc. Third ACM/IEEE Conf. Mobile Computing and Networking (MobiCom '97),* pp. 260-269, Sept. 1997.
[2] R. Floyd, R. Housel, and C. Tait, "Mobile Web Access Using eNetwork Web Express," *IEEE Personal Comm.,* vol. 5, no. 5, pp. 47-52, Oct. 1998.
[3] Z. Jiang and L. Kleinrock, "Web Prefetching in a Mobile Environment," *IEEE Personal Comm.,* vol. 5, no. 5, pp. 25-34, Oct. 1998.
[4] J. Jing, A.S. Helal, and A. Elmagarmid, "Client-Server Computing in Mobile Environments," *ACM Computing Survey,* vol. 31, no. 2, pp. 117-157, June 1999.
[5] A.D. Joseph, J.A. Tauber, and M.F. Kaasheok, "Mobile Computing with the Rover Tool-Kit," *IEEE Trans. Computers,* vol. 46, no. 3, pp. 337-352, Mar. 1997.
[6] M.F. Kaashoek, T. Pinckney, and J.A. Tauber, "Dynamic Documents: Mobile Wireless Access to the WWW," *IEEE Workshop Mobile Computing Systems and Applications,* pp. 179-184, Dec. 1994.
[7] J.J. Kistler and M. Satyanarayanan, "Disconnected Operation in the Coda File System," *ACM Trans. Computer Systems,* vol. 10, no. 1, pp. 3-25, Feb. 1992.
[8] M.H. MacDougall, *Simulating Computer Systems.* MIT Press, 1987.
[9] M.S. Mazer and C.L. Brooks, "Writing the Web while Disconnected," *IEEE Personal Comm.,* vol. 5, no. 5, pp. 35-41, Oct. 1998.
[10] E. Pitoura and G. Samaras, *Data Management for Mobile Computing.* Kluwer Academic Publishers, 1998.
[11] N.A. Phan, "Performance Analysis of Algorithms for Supporting Disconnected Write Operations in Wireless Web Environments," master's thesis, Dept. of Computer Science, Virginia Polytechnic Inst. and State Univ., Dec. 1999.
[12] S. Saha, M. Jamtgaard, and J. Villasenor, "Bringing the Wireless Internet to Mobile Devices," *Computer,* vol. 34, no. 6, pp. 54-58, June 2001.
[13] A.S. Tanenbaum, *Computer Networks,* third ed. Prentice Hall, 1996.
[14] D.B. Terry et al., "Managing Update Conflicts in Bayou: A Weakly Connected Replicated Storage System," *ACM SIGOPS Operating Systems Rev.,* vol. 29, no. 5, pp. 172-182, Dec. 1995.
[15] IEFT WEBDAV Working Group, http://www.ics.uci.edu/pub/ietf/webdav/, 2002.
[16] E.J. Whitehead Jr. and M. Wiggins, "WEBDAV: IETF Standard for Collaborative Authoring on the Web," *IEEE Internet Computing,* vol. 2, no. 5, pp. 34-40, Sept. 1998.
[17] The Washington Post, Online Journalism, Sept. 5, 1999.

Ing-Ray Chen received the BS degree from the National Taiwan University, Taipei, Taiwan, and the MS and PhD degrees in computer science from the University of Houston, University Park, Houston, Texas. He is currently an associate professor in the Department of Computer Science at Virginia Tech. His research interests include mobile computing, multimedia, distributed systems, and reliability and performance analysis. Dr. Chen has served on the program committee of various conferences. He also served as program co-chair of the 2000 IEEE Symposium on Application-Specific Systems and Software Engineering Technology and program chair of the 2002 IEEE International Conference on Tools with Artificial Intelligence. Dr. Chen currently serves as an associate editor for *IEEE Transactions on Knowledge and Data Engineering* and *The Computer Journal*. He is a member of the IEEE Computer Society and ACM.

Ngoc Anh Phan received the BS degree from Moscow Technical University of Communication and Computer Science in 1997 and a MS degree in computer science from Virginia Polytechnic Institute and State University (Virginia Tech) in 1999. She is currently a PhD student at Virginia Tech and a senior software engineer at America Online Inc. Ms. Phan's research interests include wireless communications, distributed systems, and mobile computing. Her PhD research work continues in the subject of supporting disconnected operations in mobile client-server environments. Some extensions to her previous MS thesis work includes developing algorithms to minimize the tuning time during update propagations using cache invalidation and data broadcasting, and applying these algorithms to mobile object systems.

I-Ling Yen received the BS degree from Tsing-Hua University, Taiwan, and the MS and PhD degrees in computer science from the University of Houston. She is currently an associate professor of computer science at the University of Texas at Dallas. Dr. Yen's research interests are in distributed systems, fault-tolerant computing, self-stabilization algorithms, and security. She has served as program co-chair for the 1997 IEEE High Assurance Systems Engineering Workshop, the 1999 IEEE Symposium on Application-Specific Systems and Software Engineering Technology, and the 1999 Annual IEEE International Conference on Computer Software and Applications Conference. Dr. Yen is a member of the IEEE Computer Society.