A Strategy to Manage Cache Consistency in a Distributed Mobile Wireless Environment [*]

Anurag Kahol, Sumit Khurana, Sandeep K. S. Gupta, and Pradip K. Srimani
Department of Computer Science
Colorado State University
Ft. Collins, CO 80523

Abstract

Mobile computing environments are characterized by slow wireless links and relatively underprivileged hosts with limited battery powers, predisposed to frequent disconnections. Caching data at the mobile hosts (MHs) in a wireless network helps alleviate problems associated with slow, limited bandwidth wireless links, by reducing latency and conserving bandwidth. Battery power is conserved by reducing the number of up-link requests. A mobile computing environment is a distributed system, thus when data at the server changes, the client hosts must be made aware of this fact in order for them to invalidate their cache otherwise the host would continue to answer queries with the cached values returning incorrect data. The nature of the physical medium coupled with the fact that disconnections from the network are very frequent in mobile computing environments demand a cache invalidation strategy with minimum possible overheads.

In this paper, we present a new cache maintenance scheme, called AS. The objective of the proposed scheme is to minimize the overhead for the MHs to validate their cache upon reconnection, to allow stateless servers, and to minimize the bandwidth requirement. The general approach is i) to use asynchronous invalidation messages, and ii) to buffer invalidation messages from servers at the MH's Home Location Cache (HLC) while the MH is disconnected from the network and redeliver these invalidation messages to the MH when it gets reconnected to the network. Use of asynchronous invalidation messages minimizes access latency, buffering of invalidation messages minimizes the overhead of validating MH's cache after each disconnection and use of HLC off-loads the overhead of maintaining state of MH's cache from the servers. The MH can be disconnected from the server either voluntarily (e.g. switching off the laptop) or involuntarily (e.g. wireless link failure, handoff delay); we capture the effects of both by using a single parameter s: the percentage of time a mobile host is disconnected from the network.

We demonstrate the efficiency of our scheme through simulation and performance modeling. In particular, we show that the average data access latency and the number of up-link requests by a MH decrease by using the proposed strategy at the cost of using buffer space at the HLC. We provide analytical comparison between our proposed scheme and the existing scheme for cache management in a mobile environment [BI95]. Extensive experimental results are provided to compare the schemes in terms of performance metrics like latency, number of up-link requests etc. under both high and low rate of change of data at servers for various values of the parameter s. A mathematical model for the scheme is developed which matches closely with the simulation results.

Keywords: Caching, client-server computing, data consistency, mobile computing, performance analysis.

1 Introduction

Caching relevant data at the hosts is an effective tool to improve performance (query response time and throughput) in any distributed system. Important issues in designing an effective caching scheme include (1) what to cache (and when and for how long), (2) when and how to invalidate the cached items and at what granularity level, (3) data consistency provided to the user and at what cost. Most of these concerns, especially the consistency management of cached data are exacerbated for

[*]**Address for Correspondence:** Sandeep K. S. Gupta, Department of Computer Science, Colorado State University, Ft. Collins, CO 80523, Email: gupta@cs.colostate.edu, Voice: (970) 491-7323, Fax: (970) 491-2466

mobile computing environments. Mobile computing environments are characterized by slow wireless links (low bandwidth radio links) which are susceptible to frequent *disconnections* from the base station (server) and low battery power at the mobile clients (hosts) which necessitates the clients to minimize up-link queries as well as to voluntary *disconnect* from the network to conserve battery power. This unique feature of frequent disconnection adds a new dimension to the task of maintaining consistent cache at the mobile client since the underlying cache maintenance protocol should make optimal use of the limited bandwidth. In addition to being tolerant of disconnections, these protocols should be energy-efficient, and adaptive to varying quality-of-service provided by the wireless network. Various models have been suggested in literature to estimate the usefulness of caching in mobile environments under different cost models [SWH98, WH98, SWH94, HW93, HW94] with encouraging results. Our purpose in the present paper is to consider the problem of management of cache consistency in a mobile wireless environment, once the question of *what* to cache has been answered. Frequent voluntary and involuntary disconnection of clients from servers makes this a very challenging problem [ABGM90, AK92, IB94].

Efficient caching schemes for mobile environments should ideally take into account the following factors: data access pattern and update rates, communication/access costs, mobility pattern of the client, connectivity characteristics (disconnection pattern, available bandwidth etc.) and location dependence of the data. Validation checks [Mic], that are normally used in an wired environment is not at all suitable for mobile environments since it wastes the precious wireless bandwidth. Almost all the cache coherency schemes proposed for mobile environment are based on call-back (invalidation report) mechanism. A client may miss invalidation reports from the server if it is disconnected during the broadcast; in an attempt to solve this problem, Barbara and Imilienski [BI94] have developed a *periodic* broadcasting invalidation reports scheme.

In this paper we present a caching scheme for wireless networks which uses *asynchronous* invalidation reports (call-backs) to maintain cache consistency i.e. reports are broadcast by the server only when some data changes, and not periodically. Each mobile client (host) (MH) maintains its own *Home Location Cache (HLC)* to deal with the problem of disconnections. The HLC of an MH is maintained at a designated *home* Mobile Switching Station (MSS). It has an entry for each data item cached by the MH and needs to maintain only the time-stamp at which that data item was last invalidated[1]. At the cost of this extra memory overhead of maintaining an HLC, an MH can continue to use its cache even after prolonged periods of disconnection from the network. We show through a mathematical model and simulation studies that the hit-rate and access latency of this scheme is better than it's *synchronous* counterparts and very close to an optimal strategy which incurs no overhead of maintaining cache consistency.

The rest of the paper is organized as follows. Section 2 describes the system model and assumptions, an informal overview of the proposed scheme, a brief description of the related work, and a simple comparision with the exisiting caching schemes for mobile environment. Section 3 gives the formal description of our caching scheme. Performance analysis of the proposed scheme in terms of hit ratio and mean query delay is presented in Section 4. Section 5 presents comparative simulation results of the proposed scheme with some existing caching schemes. Finally, conclusions are drawn in Section 6.

2 Proposed Scheme & Comparison with Existing Schemes

2.1 System Model

The mobile computing environment considered in this paper is shown in Figure 1. In this environment, the mobile hosts (MHs) query the database servers that are connected to a static network. The mobile hosts communicate with the servers via wireless cellular network consisting of mobile switching stations (MSS) and base stations.

A mobile host can be in two modes: **awake** or **sleep**. When a mobile host is awake (connected to the server) it can receive messages. Hence this state includes both active and dozing CPU modes. A MH can be disconnected from the network either voluntarily or involuntarily. From the perspective of the mobile host's cache it is irrelevant whether the invalidation were delayed due to voluntary disconnection (e.g. switching off the laptop) or involuntary disconnection (e.g. wireless link failure, handoff delay). Hence, for our purpose, a disconnected client is in sleep mode; we use the term **wakeup** to indicate reconnection.

[1]The HLC model fits perfectly into existing architectures to support mobility in wireless networks (e.g. Mobile IP [Per96]) which also uses the concept of a *home agent* for each MH

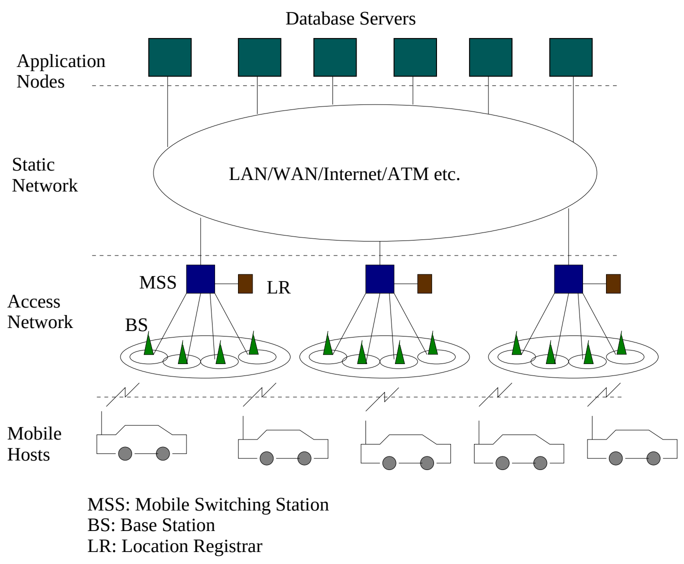


Figure 1: Mobile Computing Environment.

We consider the following computing scenario. The application program runs on the mobile host as a client process and communicates with the database server through messages, i.e. the client sends a **up-link request** (**query**) for the data it needs to the database server and the server responds by sending the requested data on the **down-link**. In order to minimize the number of up-link requests, the client caches a portion of the database in its local memory. The client-cached data is also referred to as *active* data [WN90]. Caching data at clients necessitates a protocol between the client and the database server to ensure that the client cache remains consistent with the shared database. The objective of the proposed scheme is to minimize the overhead for the MHs to validate their cache upon reconnection, to allow stateless servers, and to minimize the bandwidth requirement; the general approach is to buffer the invalidation messages at Home Location Cache (HLC) which is a static trusted host on the static network and acts on behalf of a mobile host.

Our caching scheme for the mobile environment is based on the following assumptions:

- Whenever any data item is updated anywhere in the network, an invalidation message is sent out to all MSS via the wired network; thus, when a mobile host MH is roaming, it gets the invalidation message if it is not disconnected (we assume no message is lost due to communication failure or otherwise in the wired network).
- An MH can detect whether or not it is connected to the network.
- An MH informs its HLC before it stores (or updates) any data item in its local cache.
- The static host, which is nearest to the MH and maintains the HLC of the MH, forwards the MH any invalidation it receives from the server.

2.2 An Informal Overview

The caching architecture is shown in Figure 2. A preassigned static host of any mobile host (MH) maintains its *home location cache* (HLC). If a mobile host is roaming, its HLC is duplicated at the MSS of its current cell. Thus, an MSS always maintains a HLC for each MH in its coverage area at any given time. Consider an MSS with N mobile hosts (MH_i, $1 \leq i \leq N$) at any given time. For any i, HLC_i for MH_i, as maintained in the MSS, keeps track of what data has been locally cached at MH_i (state information of the MH). In general, HLC_i is a list of records $(x, T, invalid\_flag)$ for each data item x locally cached at MH_i, where x is the identifier of a data item and T is the time-stamp of the last invalidation of x. The key feature of our scheme is that the invalidation reports are transmitted asynchronously and those reports are buffered at the MSS (in the HLCs of the mobile hosts) until an explicit acknowledgment is received from the specific MH. The $invalid\_flag$ (in the

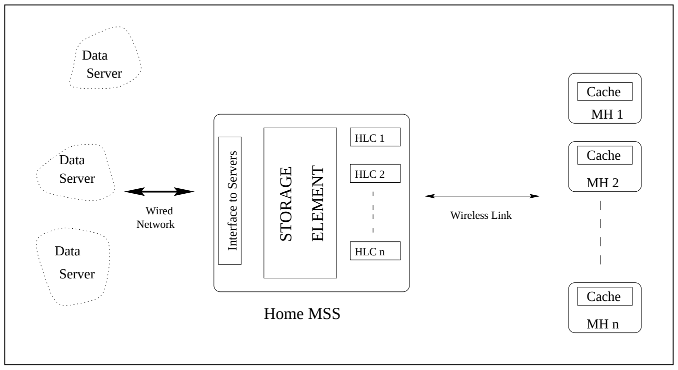


Figure 2: System Architecture.

HLC record for the specific data item) is set to TRUE for data items for which an invalidation has been sent to the MH but no acknowledgment has been received. Note that the time-stamp is the same as that provided by the server in its invalidation message.

Each MH maintains a local cache of data items which it frequently accesses. Before answering any queries from the application, it checks if the requested data is in a consistent state. We use call-backs from a MSS to achieve this goal. When a MSS receives an invalidation from a server, the MSS determines the set of MHs that are using the data by consulting the HLCs and sends an invalidation report to each of them. When a MH receives that invalidation message, it marks the particular data item in its local cache to be invalid. When an MH receives (from the application layer) a query for a data item, it checks the validity of the item in its local cache; if the item is valid, it satisfies the query from its local cache and saves on latency, bandwidth and battery power; otherwise, an up-link request to the MSS for the data item is required. The MSS make a request to the server for the data item on behalf of the MH. When the data item is received the MSS adds an entry to the HLC for the requested data item and forwards the data item to the MH. Note that the data item may or may not be cached at the MSS.

A mobile host alternates between active mode and sleep mode. In sleep mode a mobile client is unable to receive any invalidation messages sent to it by its HLC. We use the following time-stamp based scheme by which the MA can decide which invalidations it needs to retransmit to the mobile host. Each client maintains a time-stamp for its cache called the *cache time-stamp*. Cache time-stamp of a cache is the time-stamp of the last message received by the MH from its MA. The client includes the cache time-stamp in all its communications with the MA. The MA uses the cache time-stamp for two purposes:

1. To discard invalidations it no longer needs to keep, and
2. To decide the invalidations it needs to re-send to the client.

Upon receiving a message with time-stamp t, the MA discards any invalidation messages with time-stamp less than or equal to t from the MH's HLC. Further, it sends an invalidation report consisting of all the invalidation messages with time-stamp greater than t in the MH's HLC to the MH. When a MH wake-ups after a sleep, it sends a *probe* message to its HLC with its cache time-stamp. In response to this probe message the HLC sends it an invalidation report. This way an MH can determine which data items changed while it was disconnected. A MH defers all queries which it receives after waking up until it has received the invalidation report from its HLC. In this scheme we do not need to know the time at which the MH got disconnected and just by using cache time-stamp we can handle both wireless link failures and voluntary disconnections. Even if the MH wakes up and then immediately goes back to sleep before receiving the invalidation report, consistency of the cache is not compromised as it would use the same value of cache time-stamp in its probe message after waking-up and hence get the correct information in the invalidation report. Thus, arbitrary sleep patterns of the MH can be easily handled.

An Example: Consider the example scenario shown in Figure 3. Initially, the cache time-stamp of the MH is $t0$ and MH's cache has two data items with ids x and z. When MSS receives an invalidation message notifying it that x has changed at the server at time $t1$, it adds the invalidation message to MH's HLC and also forwards the invalidation message to the MH with (data-item id, time-stamp), i.e. $(x, t1)$. On receiving the invalidation message from the MSS, the MH updates its cache

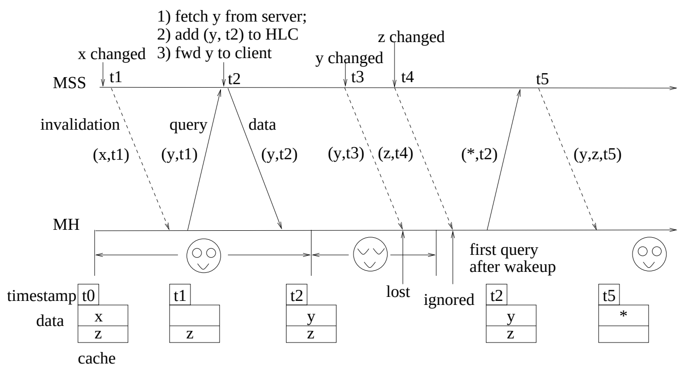


Figure 3: An Example Scenario.

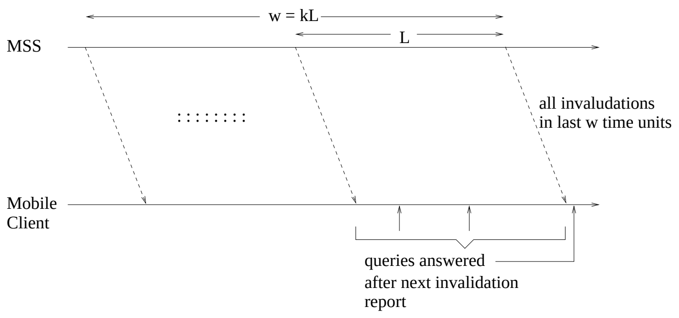


Figure 4: Barbara and Imilienski's TS/AT Scheme (L: broadcast latency; w: window size; $k > 1$ for the TS scheme and $k = 1$ for the AT scheme)

time-stamp to $t1$ and deletes data item x from its cache. Later when MH wants to access y it sends a data request with $(y, t1)$ to the MSS. In response to the data request, the MSS fetches and forwards data item associated with y to the MH and adds $(y, t2)$ to the MH's HLC, where $t2$ is the last updates time-stamp provided by the data server. The MH updates its time-stamp to $t2$ and adds y to its cache. Now suppose MH gets disconnected from the network and the invalidation message for y is lost due to this disconnection. When MH wakes up it ignores any invalidation messages it receives (until the first query), since later upon the first query after waking up it sends a probe message (invalidation check message) to the MSS. The MSS uses the time-stamp in this probe message to determine the invalidations missed by the MH and sends an invalidation report with all the missed invalidations by the MH. In this case, the MSS determines from MH's cache time-stamp $t2$ that MH has missed invalidation for y and z and so it resends them to the MH.

2.3 Related Works & Comparison

Existing schemes [BI94, LJ96, JEHA97] for maintaining cache consistency in the event of disconnections in a mobile environment are all based on callback mechanism. The basic problem is that the callback break (invalidation) messages get lost when the client is disconnected. As noted in introduction, Barbara and Imilienski [BI94] have developed a periodic broadcasting invalidation report scheme to address the problem. In this scheme, a server broadcasts invalidation reports every L time units that carry information about all data items that changed during the past $w = kL$ time units; a client still has to discard its entire cache if it is disconnected from the network and misses k consecutive reports and user queries generated during the past L time units are answered only after receiving an invalidation report. Several other schemes have been proposed with extend broadcasting invalidation reports scheme with respect to optimizing the size of invalidation [JEHA97] report, adjusting the periodicity of invalidation reports in accordance to the query rate and client disconnection time [HL98] and restricting broadcast to vicinity of the mobile client [LJ96]. Wu in [WYC96] has proposed an enhancement in which the mobile host sends back to the server the ids of all cached data items, along with their time-stamps after a long disconnection. The server identifies the changed data items and returns a validity report. This results in wastage of bandwidth and unnecessary up-link requests and still does not solve the problem for arbitrarily long sleeps, as the local cache has to be discarded after a disconnection of time greater than W. Jing in [JBEA95] has proposed a Bit-Sequence scheme where each data item in the database is represented by binary bit. The bit-sequence structure contains more update history information than the window w, but results in larger invalidation reports when only a few things have changed. The cache still has to be discarded if more than half of the items in the database have changed. Authors in [HL98] have suggested an adaptive algorithm which predominantly uses the TS and Bit-Sequence approaches but provides better tuning of the system according to the current invalidation and query rates.

The schemes based on *synchronous broadcasting invalidation reports* have the following characteristics: (1) They assume a stateless server and do not address the issue of mobility (except the work by Liu and Maguire [LJ96]); (2) The entire cache is invalid if the client is disconnected for a period larger than the period of the broadcast (or some multiple of it) (3) scalability for large database systems is not adequately addressed.

Coda file system [Sat90, S$^+$90, MS94b, MS94a] provides an alternate mechanism to provides support for disconnected operations on shared files in UNIX-like environment. Coda also uses two mechanisms for cache coherency. When the client is reachable from at least one server, callback mechanism is used. When the disconnection occurs, access to possibly stale data is permitted at a client for the sake of improving availability. Upon reconnection only those modification at the client are committed which do not cause any conflict. Balance between speed of validating cache (after a disconnection) and accuracy of invalidations is achieved by maintaining version time-stamps on volumes (a subtree in the file system hierarchy). However, validating the entire cache upon every reconnection may put a unnecessary burden on the client. Further, since Coda is a distributed file system it assumes a stateful server which may not be appropriate for other applications such as web caching. Also, the server has to keep cache state of each client and the client has to perform volume by volume validation check after each reconnection.

2.4 Comparison with Our Scheme

As has already been noted, all of the existing schemes are based on synchronous or periodic broadcast of invalidation reports. None of these works investigates the effect of using these schemes on an actual wireless network. All of the schemes are based on aggregating queries for a fixed period of time and then answering them after receiving an invalidation report. This provides very poor network utilization as there is no traffic for long periods of time followed by a very heavy burst. This also results in higher queuing delay for answering a query.

Unlike these strategies, all of which use synchronous invalidation reports and are based on the basic scheme of [BI94], our strategy is (A)synchronous and (S)tateful; we use the name **AS**. In order to compare our scheme with the sleepers and workaholics scheme of [BI94], we note the salient features of that scheme as follows:

- A server broadcasts invalidation reports every L time units which carry information about all data items that changed during the past $w = kL$ time units. Two variations of this basic scheme are suggested: (1) **TS**, where invalidation reports carry information about changes in data items over a larger window ($k > 1$), and (2) **AT** for which $k = 1$ (See Fig. 4).
- Clients maintain local caches and use the information in invalidation reports to update their caches.
- If a client is disconnected from the network and misses k consecutive reports, it discards its local cache.
- Queries generated during the past L time units are answered only after receiving an invalidation report.

We also consider an **Ideal Scheme** to compare our scheme to with the following characteristics: (1) whenever any data is changed at the MSS, the invalidation information is available to the clients instantaneously at no cost; (2) the above holds even when the MH is disconnected from the MSS, i.e., there is no overhead due to disconnections. Clearly, the ideal strategy is infeasible for any practical system; nevertheless, it provides useful reference points on the achievable hit-rate and delay. Table 1 gives a comparison of the salient features of the proposed scheme with those of the TS and AT schemes; quantitative bounds are shown in Table 2 for a system with N mobile hosts each of which are caching M data items. The improvement in performance is achieved at the cost of maintaining additional buffer space at the MSS; since memory is cheap especially at the stationary MSS, performance benefits outweigh this relatively minor cost.

TS/AT [BI95]	**AS** (proposed)
Server is stateless (no information about client cache is maintained)	Server is stateful (HLC maintained)
Invalidation reports sent regardless of whether clients have any data in cache.	Invalidation report broadcast only if any client has valid data in cache.
Invalidation reports sent periodically at rate L (synchronous)	Invalidation reports sent as and when data changes (asynchronous)
Average cache access latency is $L/2$ plus network queuing delay	Latency is governed only by the queuing delay on the network.
Traffic on the network is bursty as queries are aggregated for a period of time L.	Queries are answered as they are generated
Cache restored for sleep limited to a maximum duration of w(TS) or L(AT)	Arbitrary sleep patterns can be supported
Mobility is supported by assuming a replication of data across all stationary nodes (not scalable)	Mobility can be transparently supported by using a mobility aware network layer e.g. mobile IP

Table 1: Comparison of salient features of TS/AT and AS schemes.

Parameter	TS [BI95]	Ideal	AS (Proposed)
Maximum sleep time supported	w	∞	∞
Up-link overhead on wakeup	0	0	0 or 1
Cache access Latency	0 to L	0	0
Buffer space required at MSS	$\mathcal{O}(N)$	-	$\mathcal{O}(MN)$

Table 2: Comparison of cache invalidation strategies.

3 Formal Description of the Proposed Scheme

3.1 Data Structures

Every data object has an unique identifier. The letters x, y, and z will be used to denote data identifiers. We will use the notation $Data_x$ to denote the data associated with a data item with identifier x.

The following data structures are maintained at each MH:

- t_s : Time stamp of the last invalidation report or data received by the MH from its home MSS.
- *cache* : Data cache. An item in the data cache is of the form $(x, Data_x, Valid\_flag)$. The data $Data_x$ can be considered valid only when the $Valid\_flag$ is TRUE.
- $First\_Request$: A flag set to TRUE when an MH wakes up and is yet to make its first query after awakening. The flag is set to FALSE once the the first request after waking up is made.
- $First\_Waiting$: A flag set to TRUE when an MH has made its first query after getting reconnected to the network but the data for it has not been received.

The following data structures are maintained at each MSS:

- $HLC[1..N]$: an array of lists; $HLC[i]$ is a list of records of the type $(x, T, invalid\_flag)$ one for each data item x cached by MH_i; N is the total number of MHs that are in the cell of the MSS. T is the time-stamp of the last invalidation of x. The $invalid\_flag$ is set to TRUE for data items for which an invalidation report has been sent but no implicit acknowledgment has been received.

3.2 Messages

- INVALIDATION_REPORT ($item\_list, T, first\_flag$): Sent to an MH by its home MSS to report invalidation of data items in $item\_list$. T is the time-stamp associated with this report. $first\_flag$ is set if this invalidation report is in response to the first query of the MH on waking up.
- DATA_REQUEST ($i, x, t, first\_request$) Sent by MH_i to its MSS to request data item x when x is not found in its local cache. t is set to the time-stamp t_s maintained by the MH. The flag $first\_request$ is set if this is the first data request after the MH regains connectivity to the network.
- DATA ($x, Data_x, T$): Broadcast by an MSS to send data to all MHs caching x. T is a time-stamp set to the current time at the MSS.

3.3 System Primitives

In designing our algorithm, we assume the availability of the simple system primitives:

- time(): returns the current time.
- add: insert an item in a set.
- delete: remove an item from a set.
- wait: delays the processing of an event until a given condition is satisfied.

3.4 Pseudo-code

An MSS continuously executes the procedure **MSS_Main**, shown in Figure 15. This procedure handles the following events.

1. **MSS receives a request for data from an MH (DATA_REQUEST)**: With each request, an MH sends the time-stamp of the last message it had received from the MSS. The MSS deletes all the entries in the HLC for the MH which had been invalidated before the time-stamp carried in the message. Since the messages are assumed to be received in order this ensures that the MH was awake at the time each of the invalidations was received and the MSS no longer needs to buffer the invalidation.

   If the data request is the first after a sleep, all the items cached by the MH, marked invalid since the last message received by the MH are repeated through an invalidation report. The invalidation report carries a time-stamp with it.

   Finally the requested data item is sent to the MH and added to its HLC.

2. **Data item(s) updated at MSS**: The MSS sends an invalidation report to all the MHs that are caching the changed data item and to whom a previous invalidation has not been sent for the same data. It also updates the data time-stamp in the HLC for these MHs and marks those items as invalid.

Each MH continuously executes the procedure **MH_Main**, shown in Figure 16. The following events are handled.

1. **MH generates a request for a data item**: If the MH has woken up after a sleep and this is the first request, it sends a data request for the item and sets a flag ($Flag\_Waiting$) to indicate that the buffered invalidations during the sleep period have not yet been received. On receiving those invalidations it answers successive queries from the cache.

   If the query is not the first after a wake up, the cache is checked for that data item. If the item is not in the cache a data request is sent to the MSS and the query is answered once the data arrives from the MSS.

2. **MH receives an INVALIDATION_REPORT from the MSS**: The MH sets its cache time-stamp to the to the time-stamp in the current message and invalidates in its cache all the data items mentioned in the report. All invalidations received between the time an MH awakens and receives the first query are ignored.

3. **MH receives DATA from the MSS**: It updates its cache with the current information and also updates its cache time-stamp.

4. **MH wakes up after a disconnection**: It sets the $First\_Request$ flag to TRUE.

4 Performance Analysis

We develop a simple model to analyze the performance of the proposed cache management scheme. Specifically, we want to estimate the miss probability and mean query delay for the proposed scheme. For the purpose of analysis, we consider the performance in a single cell, (as mobility is assumed to be transparently handled) with one MSS and N mobile hosts. We make the following assumptions:

- Total number of data items is M, each of size b_a bits.
- The queries generated by a sleeping MH (i.e., when the MH is disconnected from the MSS) are lost.
- A single wireless channel of bandwidth W is assumed for all transmissions taking place in the cell. All messages are queued to access the wireless channel and serviced according to the FCFS (First-Come First-Served) scheduling policy.
- Queries are of size b_q bits and invalidations are of size b_i bits.
- Software overheads are ignored.

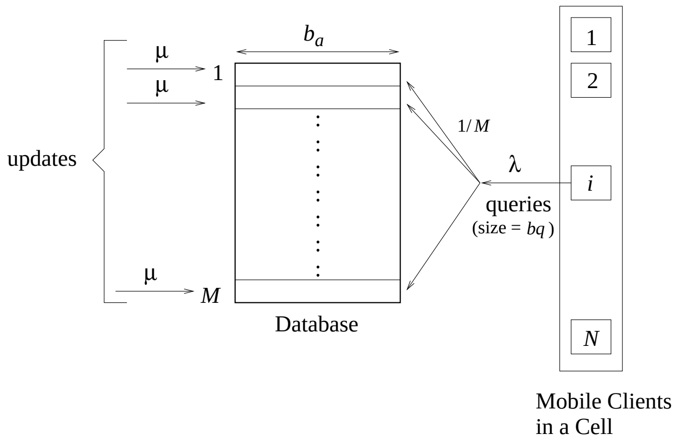


Figure 5: Modeling Query-Update Pattern.

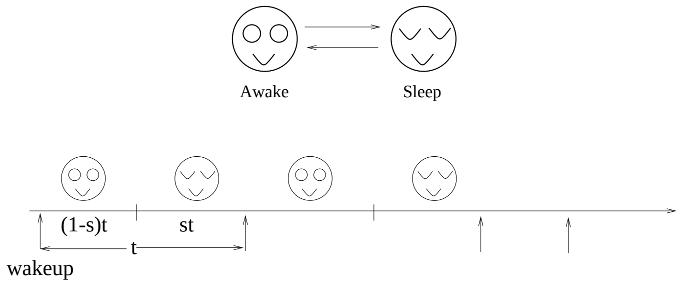


Figure 6: Modeling sleep behavior of an MH.

- **Modeling Query-Update Pattern**: The time between updates to any data item is assumed to follow an exponential distribution with mean $1/\mu$. Each MH generates a query according to a Poisson distribution with mean rate of λ. These queries are uniformly distributed over all data items in the database. The query-update model is shown in Figure 5.
- **Modeling Sleep Pattern for an MH**: An MH alternates between sleep and awake modes. The sleep/wakeup pattern of an MH is modeled by using two parameters (see Figure 6): (1) the fraction s, $0 \leq s, \leq 1$, of the total time spent by an MH in the sleep mode; (2) the frequency ω at which it changes state (sleeping or awake). We consider an exponentially distributed interval of time t with mean $1/\omega$. The MH is in the sleep mode for time st, and in the awake mode for time $(1-s)t$. By varying the value of ω, different frequencies of change of state can be obtained for the same total sleep time.

We want to estimate the average hit ratio at the local cache of an MH, i.e., the percentage of queries, generated at an MH, that would be satisfied by the local cache and the average delay experienced in answering a query.

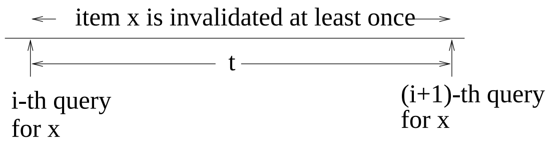
4.1 Estimation of Hit Ratio

Since the queries generated by a sleeping MH are lost, the *effective rate of query generation*, λ_e, by an MH can be approximated as

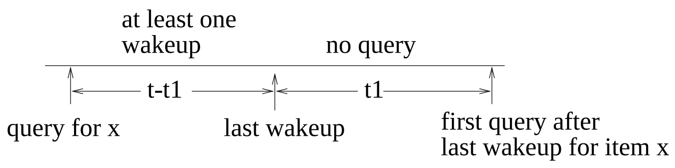
$$\lambda_e = (1-s)\lambda.$$

Since queries are uniformly distributed, the rate at which queries are generated for a *given* data item x, λ_x, is given by

$$\lambda_x = \frac{\lambda_e}{M} = \frac{(1-s)\lambda}{M}.$$



(a) Miss due to absence of valid data item in cache



(b) Miss due to diconnections.

Figure 7: Two mutually exclusive events when up-link request messages are needed.

A query made for a specific data item x by an MH would be a *miss* in the local cache (and would require an up-link request) in case of either of the following two events: (Consider the time interval t between the current query for x and the immediately preceding query for x by the MH)

1. Event 1: During this interval t, the data item x has been invalidated at least once (see Figure 7(a)).
2. Event 2: Data item x has not been invalidated during the interval t; the MH has gone to sleep at least once during the interval t, it woke up last time at time $t - t_1$ and the current query is the very first one after waking up from last sleep (Figure 7(b)). Note that the first query generated by a sleeping MH after waking up needs an up-link request to the MSS regardless of whether the required data item is in the local cache.

We compute the probabilities of Event 1 and Event 2 as follows.

- Probability of miss due to absence of valid data item in cache:

$$P[\text{Event 1}] = \int_0^\infty (\lambda_x e^{-\lambda_x t})(1 - e^{-\mu t})dt = \frac{\mu}{\lambda_x + \mu} = \frac{M\mu}{(1-s)\lambda + M\mu}.$$

- Probability of miss due to disconnection:

$$P[\text{Event 2}] = \int_0^\infty (P[\text{no invalidation and query for x during time t}] \times P[\text{the query (for x) is 1st after wakeup}])\, dt.$$

  The first factor in the expression is given by (note that there is a query at time t) $\lambda_x e^{-\lambda_x t} e^{-\mu t}$. The second factor can be evaluated as follows:

$$P[\text{the query (for x) is 1st after wakeup}] = \int_{t_1}^{t} P[\text{no query or sleep during interval } t - t_1 \text{ after last wakeup}] dt_1$$

$$= \int_{t_1}^{t} (P[\text{at least one sleep during interval } t - t_1] \times P[\text{no query or sleep during } t_1 \text{ given at least one during } t - t_1])\, dt_1$$

$$= \int_{t_1}^{t} \lambda_x e^{-\lambda_x t} e^{-\mu t} \left(1 - e^{-\omega(t-t_1)}\right) dt_1$$

$$= \frac{\omega}{\omega + \lambda_e}\left(1 - e^{-(\omega+\lambda_e)t}\right) - \frac{\omega}{\lambda_e}\left(e^{-\omega t} - e^{(-\omega+\lambda_e)t}\right) = \alpha, \text{ say.}$$

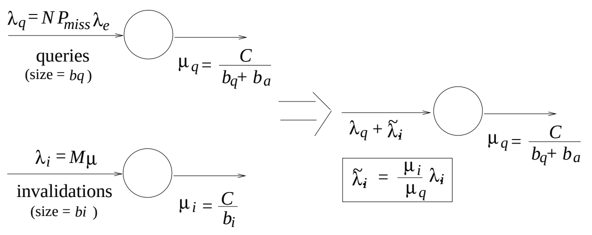

$\lambda_q = N P_{miss} \lambda_e$
queries
(size = bq)
$\mu_q = \frac{C}{b_q + b_a}$
$\lambda_i = M\mu$
invalidations
(size = bi)
$\mu_i = \frac{C}{b_i}$
$\lambda_q + \tilde{\lambda}_i$
$\mu_q = \frac{C}{b_q + b_a}$
$\tilde{\lambda}_i = \frac{\mu_i}{\mu_q} \lambda_i$

Figure 8: Combining the up-link and down-link M/D/1 queues.

Thus, the probability $P[\text{Event 2}]$ can now be evaluated as

$$P[\text{Event 2}] = \int_0^\infty \lambda_x e^{-\lambda_x t} e^{-\mu t} \alpha dt = \frac{\omega \lambda_x}{\lambda_e(\omega + \lambda_e)} \left(\frac{\lambda_e}{\mu + \lambda_x} - \frac{\omega + \lambda_e}{\mu + \lambda_x + \omega} + \frac{\omega}{\mu + \lambda_x + \lambda_x + \omega} \right).$$

The probability, P_{miss}, of a query requiring a up-link request is the sum of the probabilities for Event 1 and Event 2 and is given by

$$P_{miss} = P[\text{Event 1}] + P[\text{Event 2}],$$

and the probability of a hit is

$$P_{hit} = 1 - P_{miss}.$$

4.2 Estimation of Mean Query Delay

We now estimate the mean query delay T_{delay}. Note that a single wireless channel of bandwidth C is assumed for all transmissions taking place in the cell and all messages are queued to access the wireless channel and serviced according to the FCFS scheduling policy; also, queries are of size b_q bits and invalidations are of size b_i bits.

In order to determine T_{delay} we do the following:

- Model the servicing of up-link queries as a M/D/1 queue under the assumption that there is a dedicated up-link channel of bandwidth C. The query arrival rate λ_q is estimated to be $N P_{miss} \lambda_e$ since there are N MHs in a cell and for each MH in the cell, the up-link query generation rate is $P_{miss} \lambda_e$. The query service rate μ_q is then estimated to be $\frac{C}{(b_q + b_a)}$.
- Model the servicing of invalidation on down-link channel as a M/D/1 queue under the assumption that there is a dedicated down-link channel of bandwidth C. The average invalidation arrival rate λ_i is estimated to be $M\mu$ and the invalidation service rate μ_i is then estimated to be C/b_i.
- In order to model a single wireless channel of bandwidth C for both up-link and down-link traffic and estimate the mean query service rate T_q on this shared channel we combine both up-link and down-link M/D/1 model. Since we are interested in only the query service rate, the invalidations on the channel merely add to the delay in servicing the queries. Thus we assume that the service rate of the channel for both types of traffic to be μ_q, the service rate for queries, and adjust the arrival rate of invalidations in proportion to the service rate of queries. Thus the effective arrival rate of invalidations is taken as $\tilde{\lambda}_i = \frac{\mu_i}{\mu_q} \lambda_i$. The combined M/D/1 queue is shown in Figure 8. Using the standard queuing theory result for an M/D/1 queue, the average delay experienced by a query going up-link is given by [Jai91]:

$$T_q = \frac{2\mu_q - (\lambda_q + \tilde{\lambda}_i)}{2\mu_q(\mu_q - (\lambda_q + \tilde{\lambda}_i))}.$$

- All queries that are cache hits do not experience any delay. Thus the average delay experienced by any query in the system is given by $T_{delay} = P_{miss} T_q$.

Parameter	Value
N	25
M	100
λ	1/120 query/s
μ	10^{-4} updates/s (low), 1/1800 updates/s (high)
b_a	1200 bytes
b_q	64 bytes
b_i	64 bytes
C	10000 bits/sec
s	20%
ω	1800 sec
L	10 sec
w (TS)	$100L$

Table 3: Default parameters for AS and AT/TS.

5 Performance Comparison

5.1 Experimental Setup

We simulated our proposed scheme of cache invalidation in a single cell with a base station and varying number of mobile hosts; we experimented for different rates of invalidation of data items. The purpose of the experiments were twofold: (1) to investigate how closely the experimental results coincide with the values for performance metrics (delay, up-link requests) predicted by our simple model; (2) to investigate how efficiently our proposed scheme AS manages disconnection in a mobile environment; for this purpose we experimentally compared our scheme AS with the Ideal Scheme. The default parameters used for each scenario are as shown in Table 3. *Delay* is defined to be the time it takes to answer a query. The delay is assumed to be zero when there is a local cache hit.

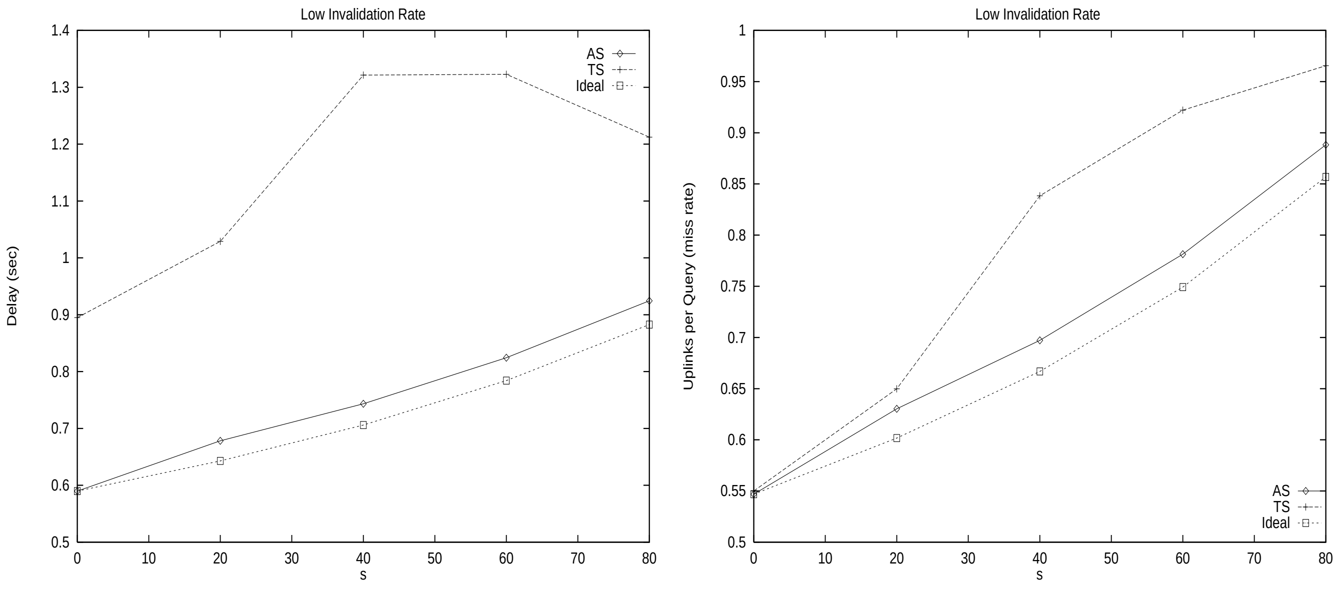
5.2 Experimental Results

5.2.1 Comparision at Low Data Invalidation Rate

The first scenario studies the performance of the TS and AS schemes when the data changes infrequently. The value of μ used is 10^{-4} updates / sec. Delay and number of up-links per query were studied by varying s. The TS scheme performs better than the AT scheme at low invalidation rates.

Delay: Figure 9(a) shows the variation in average delay with the sleep characteristics of the mobile hosts. For the TS scheme delay has two components: query's waiting time to be serviced by the MSS and the time it must wait for the next invalidation report to be broadcast (to ensure the data item has not been invalidated since the last update was received). Only the network delay component is shown. If the time to wait for the next invalidation report is added, an additional $L/2$ delay is seen for AT/TS. A significant improvement in total delay is seen when the AS strategy is used. The network delays are higher for TS as queries tend to go up-link in bursts causing congestion in the network. This is a consequence of waiting for the next invalidation report to arrive to answer the query. The delay for the TS scheme increases with s, as the cache has to be discarded more often reducing the hit rate and generating more up-link queries. However, the total query rate is reduced as the sleep rate increases. At very high sleep rates this effect dominates and the delay decreases.

Up-links: Figure 9(b) shows the variation in the number of up-links per query with the sleep characteristics of mobile hosts. AS requires an up-link for the first query after every sleep interval. If the item queried is already in the cache and not invalidated, then this up-link is additional to the Ideal scheme. Thus AS has a marginally higher number of up-links per query as opposed to the Ideal scheme which increase as the sleep rate increases. The shape of the TS curve can be explained based



(a) Delay (b) Up-links

Figure 9: Delay and up-links: low invalidation rate.

on the fact that it has to discard its cache after an extended sleep. This results in low hit rates and more up-link queries. The effect is not so dominant when the sleep rate is low but increases as the average sleep percentage increases.

5.2.2 Comparision at High Data Invalidation Rate

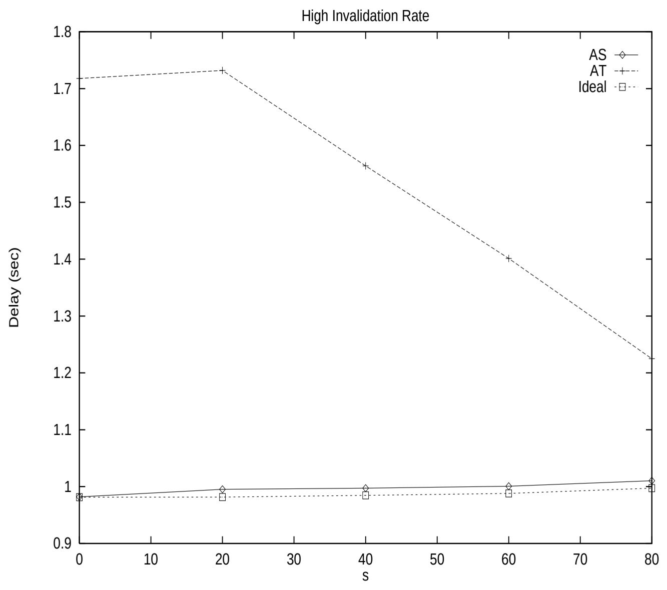
In this section we present results for a scenario where the data in the network changes frequently. The data change rate μ is assumed to be $1/1800$ updates/sec. All other parameters are as in Table 3. We compare our AS scheme to only the AT scheme since the AT scheme performs better than the TS scheme at high invalidation rates. This is because the AT scheme wastes less bandwidth than the TS scheme by not repeating the same invalidation report multiple times.

Delay: Figure 10(a) shows a plot of average delay against s. The network delays are slightly higher as compared to Figure 9. This is due to additional invalidation reports being sent in the AS scheme and the increase in size of each report for the AT scheme. The decrease at a very high sleep rate is due to the number of queries decreasing as most hosts are sleeping. There is an almost 7 to 8 times improvement in overall delay (network delay + wait for next report) when the AS strategy is compared to AT similar to figure 9.

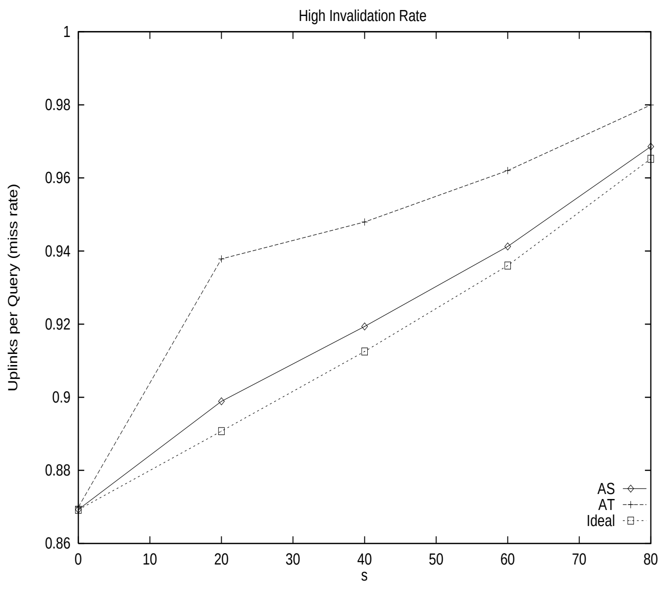
Up-links: Figure 10(b) shows the variation of the number of up-links per query with the sleep rate. As the invalidation rate is high the cache is less effective and more queries result in up-links as compared to the scenario of section 5.2.1. In the AT/TS strategy, the MH must wait for the next invalidation to come before it can answer a query. Since the time window for answering a query is greater, there is a greater probability that the item would be invalidated by that time. The hit rate of the cache is therefore poorer. This combined with the drop in hit rate due to discarding the cache contributes to the number of up-links for AT. AS performs marginally worse than the Ideal scheme as in the case of low invalidation rate.

5.2.3 Comparison with Ideal Scheme

Figures 11(a) and 11(b) show the delay and and up-links per query, respectively at different query rates for the ideal and AS schemes. As the number of queries per second increases, both the number of up-links per query and the average delay to answer a query increase. This is consequence of the decrease in hit rate. As there is a greater delay between queries the probability of an invalidation occurring between queries increases. This results in more up-links and higher delay. As the

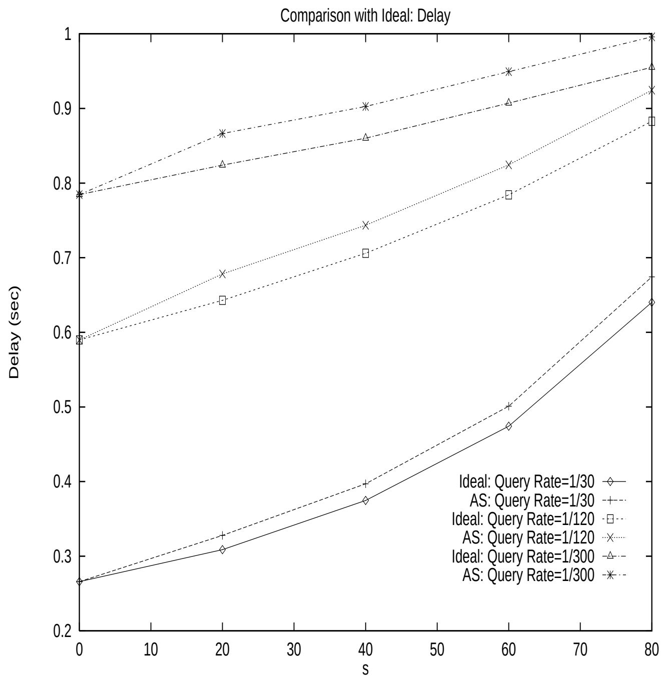


(a) Delay

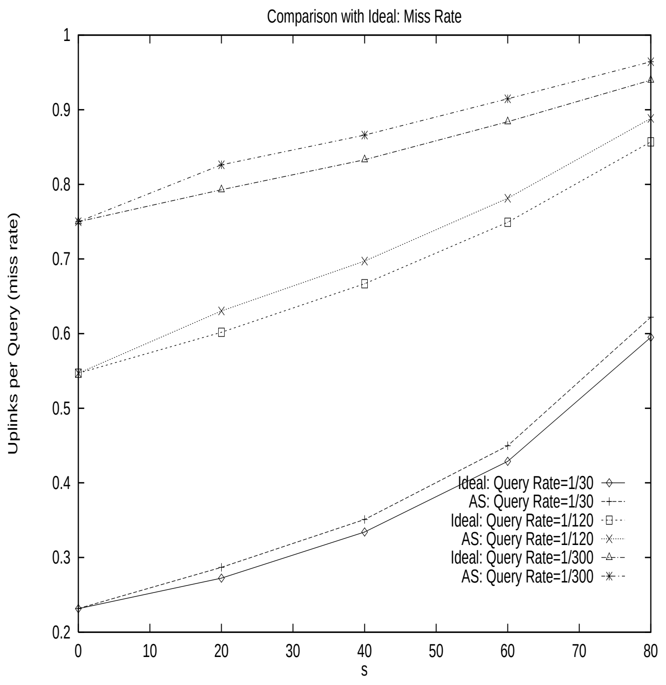


(b) Uplinks

Figure 10: Delay and up-links at high invalidation rate.

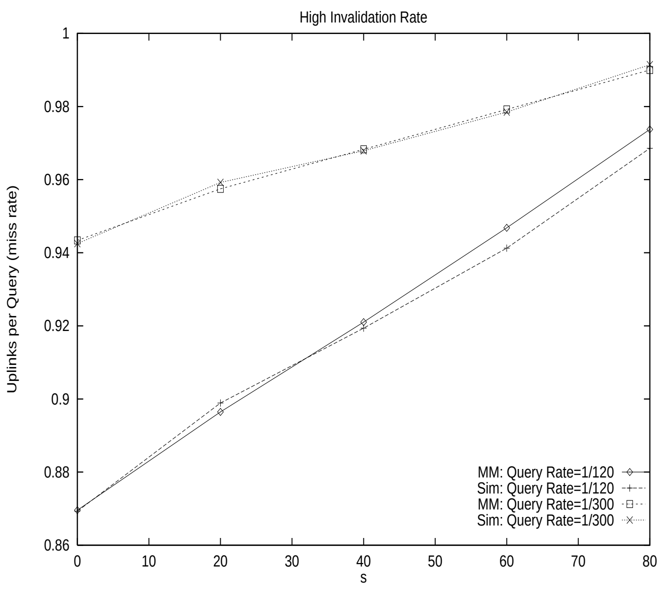


(a)Delay

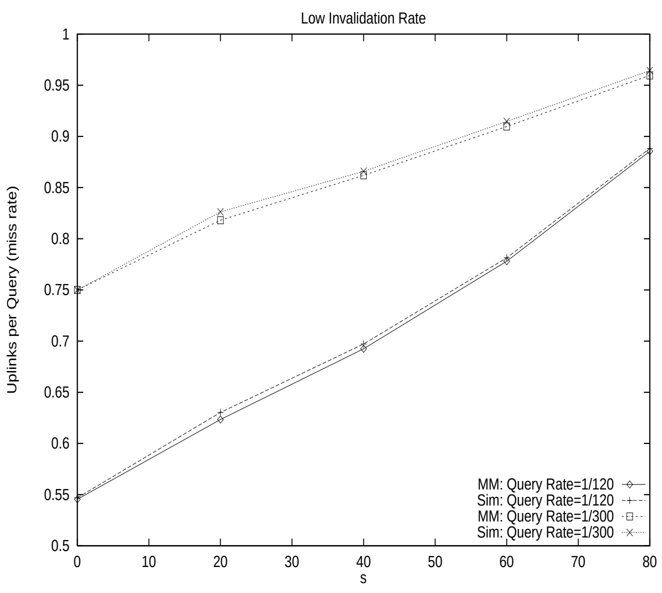


(b) Uplinks

Figure 11: Delay and up-links compared with Ideal scheme at low invalidation rate.



(a) High Invalidation Rate



(b) Low Invalidation Rate

Figure 12: Up-links: simulation and mathematical model.

sleep rate increases the probability of an invalidation between successive queries increases even further. In all cases the plots for AS closely follow that of the ideal scheme, with the gap increasing as the sleep rate increases.

5.2.4 Model Validation

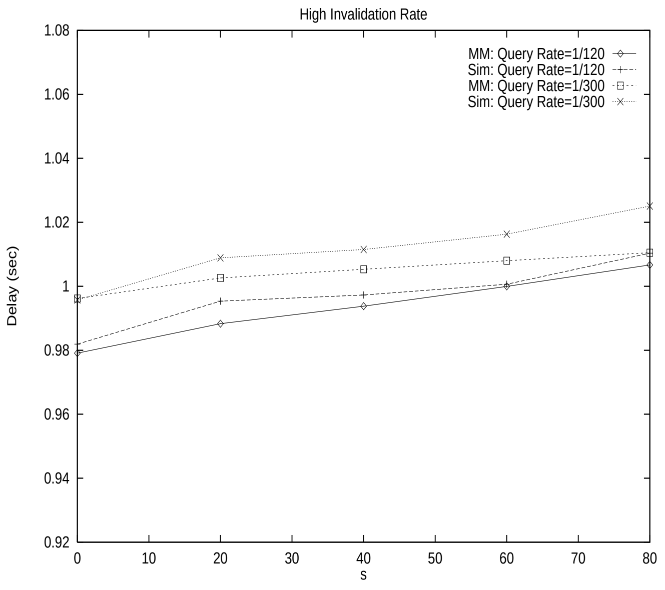
Figure 12 shows the comparison between the miss-rate for the proposed scheme AS obtained through simulation and that predicted by the mathematical model (MM). Figure 13 shows the comparison between the delay for the proposed scheme AS obtained through simulation (denoted as Sim in the plots) and that predicted by the mathematical model. The model captures the behavior very well and the results are closer at low invalidation rates. This is because of the heuristic used in the modeling for estimating the equivalent arrival rate of invalidation messages.

6 Conclusions

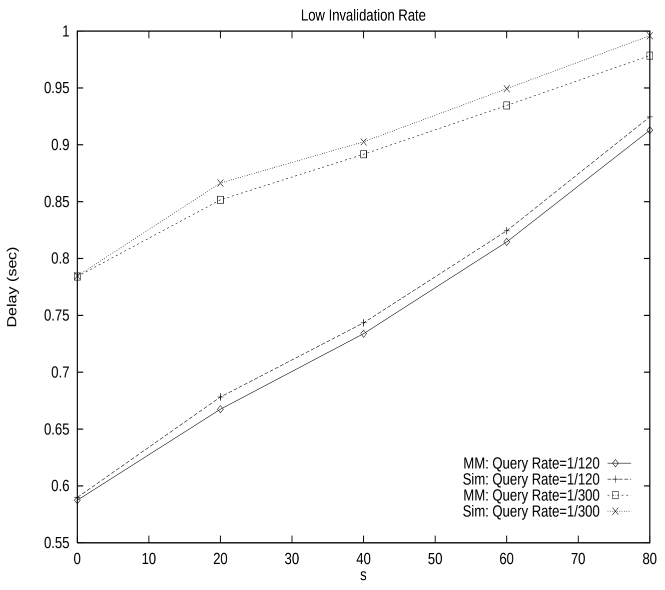
We have presented a cache maintenance (invalidation) strategy for a distributed system operating in a mobile wireless environment. The proposed algorithm minimizes the overhead preserving bandwidth, reducing the number of up-link requests and average latency. State information about the local cache at MH with respect to data items is maintained at the home MSS; by sending asynchronous call-backs and buffering them till implicit acknowledgments are received, the cache continues to be valid even after the MH is temporarily disconnected from the network. We have provided both theoretical and experimental comparison of the proposed scheme with the existing ones.

In Coda the server maintains state information for all the clients. As opposed to this, in our scheme, the state information is maintained at the home MSS of a mobile client. A server only needs to maintain information regarding which MSSs in the network need invalidation reports. It is expected that the number of mobile hosts will far exceed the number of static hosts. Hence, the amount of state information stored at each server is significantly reduced. Maintaining state information at the MSS can be considered as an overhead, but has the capability to provide various other benefits beyond this scheme. For example, profiling techniques can be used by the MSS to determine *what* to cache at the hosts when cache space is limited [TB97]. It is expected to provide a platform to enable prefetching of data[CB98] or hoarding of files [KP97] at the clients.

The proposed scheme ensures, in absence of any loss of invalidation reports in wired network, that the data returned to a



(a) High Invalidation Rate



(b) Low Invalidation Rate

Figure 13: Delay: simulation and mathematical model.

mobile client is at most τ seconds old, where τ is the maximum latency of forwarding an invalidation report from the server to the client via its home MSS. This is different from the Coda scheme where a diconnected client is allowed both read and write access to its cached data. When the client gets reconnected, its cached data is reintegrated with the server data. The user needs to resolve any conflict arising from independent modification of data by multiple disconnected clients. Coda's design is suitable for its goals i.e. providing Unix-like file system semantics in an environment where write-write conflicts are rare (e.g. college campus) and access to stale data does not lead to dire consequences or is detectable by the application. As opposed to this, our scheme is targeted towards applications which require strict data currency guarantees and where access to stale data is undesirable. Such applications include access to critical data such as bank account information and air traffic information.

The proposed scheme can be easily integrated with Mobile IP. In the mobility management scheme used in Mobile IP, each mobile host has a home address and a care-of-address. The home address is the IP address on the home network of the mobile host. The care-of-address is the address indicating the current location of the mobile host. Two architectural entities: home agent and foreign agent are used in Mobile IP to deliver datagrams to mobile clients. A home agent tunnels (encapsulates in another datagram packet) any datagrams sent to the mobile client at its home address to its current care-of-address(es). A foreign agent (in case mobile uses foreign care-of-address) on the current network of the mobile client decapsulates the packet and delivers it to the mobile client to which the datagram is addressed (see Fig. 14). We assume that the mobility agents (home or foreign agent) (MAs) are located at the MSSs. The HLC of a mobile host can be maintained by the home agent. Further, the HLC can be replicated with the foreign agent when the mobile host moves to a foreign network. A possible future work would be to incorporate our caching scheme in a Mobile IP implementation and measure its performance.

The performance analysis and simulation results show the benefits in terms of bandwidth savings (reduction in uplink queries) and data acess latency compared to Barbara and Imilienski's caching schemes which provides similar data currency guarantees as the proposed scheme. In this paper, we have studied the combined effect of a mobility pattern and disconnections on our mobility scheme. Future work includes studying the effect of different mobility patterns on latency, hit ratio, cost of invalidations.

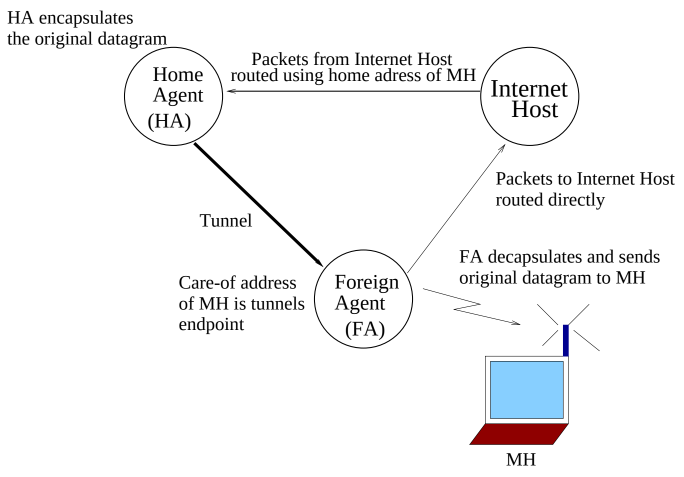


Figure 14: Deliverying message to a MH in Mobile IP.

7 Acknowledgements

The authors are grateful to Associate Editor Prof. Olariu and the four annonynous reviewers for many valuable comments that improved the final presentation of the paper.

References

[ABGM90] R. Alonso, D. Barbara, and H. Garcia-Molina. Data caching issues in an information retrieval system. *ACM Transactions on Database Systems*, 15:359–384, September 1990.

[AK92] R. Alonso and H. F. Korth. Database systems issues in nomadic computing. Technical Report MITL-TR-36-92, Matsushita Information Technology Laboratory, Princeton, NJ 08542-7072, December 1992.

[BI94] D. Barbara and T. Imielinski. Sleepers and workaholics: Caching strategies in mobile environments (extended version). *MOBIDATA: An Interactive Journal of Mobile Computing*, 1(1), November 1994.

[BI95] D. Barbara and T. Imielinski. Sleepers and Workaholics: Caching Strategies in Mobile Environments. *Very Large Databases Journal*, December 1995.

[CB98] M. Crovella and P. Barford. The Network Effects of Prefetching. In *Proceedings of INFOCOM '98*, 1998.

[HL98] Q. Hu and D. K. Lee. Cache algorithms based on adaptive invalidation reports for mobile environments. *Cluster Computing*, 1:39–50, 1998.

[HW93] Y. Huang and O. Wolfson. A competitive dynamic data replication algorithm. In *IEEE Proc of 9th Int'l conference on data engineering*, pages 310–317, 1993.

[HW94] Y. Huang and O. Wolfson. Dynamic allocation in distributed systems and mobile computers. In *IEEE Proc of 10th Int'l conference on data engineering*, pages 20–29, 1994.

[IB94] T. Imielinski and B. R. Badrinath. Wireless computing: Challenges in data management. *Communications of the ACM*, 37(10), October 1994.

[Jai91] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* John Wiley & Sons, Inc., 1991.

[JBEA95] J. Jing, O. Bukhres, A.K. Elmargarmid, and R. Alonso. Bit-sequences: A new cache invalidation method in mobile environments. Technical Report CSD-TR-94-074, Computer sciences department, Purdue university, May 1995.

[JEHA97] J. Jing, A. Elmagarmid, A. Helal, and R. Alonso. Bit-sequences: an adaptive cache invalidation method in mobile client/server environments. *Mobile Networks and Applications*, 2:115–127, 1997.

[KP97] G. H. Kuenning and G. J. Popek. Automated hoarding for mobile computers. In *16th ACM Symposium on Operating System Principles*, 1997.

[LJ96] G. Y. Liu and G. Q. McGuire Jr. A mobility-aware dynamic database caching scheme for wireless mobile computing and communications. *Distributed and Parallel Databases*, 4:271–288, 1996.

[Mic] Sun Microsystems. NFS: Network file system protocol specification. RFC 1094.

[MS94a] L. B. Mummert and M. Satyanarayanan. Large Granularity Cache Coherence for Intermittent Connectivity. In *Proceedings of the 1994 Summer USENIX Conference*, June 1994.

[MS94b] L. B. Mummert and M. Satyanarayanan. Variable granularity cache coherence. *Operating Systems Review*, 28(1):55–60, January 1994.

[Per96] C. Perkins. IP Mobility Support. RFC 2002, October 1996.

[S^+90] M. Satyanarayanan et al. Coda: A highly available file system for a distributed workstation environment. *IEEE Trans. Computers*, 39(4):447–459, April 1990.

[Sat90] M. Satyanarayanan. Scalable, secure, and highly available distributed file access. *IEEE Computer*, 23(5):9–21, May 1990.

[SWH94] A. P. Sistla, O. Wolfson, and Y. Huang. Minimization of communication cost through caching in mobile environments. In *Proc. of the ACM-SIGMOD*, May 1994.

[SWH98] A. P. Sistla, O. Wolfson, and Y. Huang. Minimization of communication cost through caching in mobile environments. *IEEE Transactions on Parallel and Distributed Systems*, 9(4):378–389, 1998.

[TB97] S. L. Tong and V. Bharghavan. Alleviating the latency and bandwidth problems in WWW browsing. *USENIX Symposium on Internet Technologies and Systems*, 1997.

[WH98] O. Wolfson and Y. Hang. Competitive analysis of caching in distributed databases. *IEEE Transactions on Parallel and Distributed Systems*, 9(4):391–409, 1998.

[WN90] K. Wilkinson and M.-A. Neimat. Maintaining consistency of client-cached data. In *Proc. 16th Int'l Conf. Very Large Data Bases (VLDB'90)*, pages 122–133, Brisbane, Australia, August 1990.

[WYC96] K. L. Wu, P. S. Yu, and M. S. Chen. Energy-efficient caching for wireless mobile computing. In *20th International conference on data engineering*, pages 336–345, March 1996.

```
Procedure MSS_Main()
begin
while (true)
    begin
        if(MSS receives DATA_REQUEST($i$,$x$,$t_s$,$first\_request$) message) then
            foreach item $(y, t, invalid\_flag) \in HLC[i]$ do
            begin
                if ($invalid\_flag$ == TRUE and $t \leq t_s$)
                    delete $(y, t, invalid\_flag)$ from $HLC[i]$
            end
            if ($first\_request$ == TRUE)
            begin
                $item\_list = \emptyset$
                foreach item $(y, t, invalid\_flag) \in HLC[i]$ do
                begin
                    if ($t > t_s$ and $invalid\_flag$ == TRUE)
                    /* item has changed while MH was sleeping */
                        add $(y)$ to $item\_list$
                end
                $T$ = time()
                send INVALIDATION_REPORT($item\_list, T$, TRUE) to $MH_i$
            end
            fetch ($Data_x$) from the Server
            delete $(x, t, invalid\_flag)$ from $HLC[i]$
            $T$ = time()
            add $(x, T$, FALSE) to $HLC[i]$
            send DATA($x, Data_x, T$) to $MH_i$

        if(MSS receives notification for change of data item $x$ from a server) then
            $T$ = time()
            foreach MH $i$ do
            begin
                $item\_list = \emptyset$
                if $(x, t, invalid\_flag) \in HLC[i]$ /* data item is being cached by $MH_i$
                    delete $(x, t, invalid\_flag)$ from $HLC[i]$
                    add $(x, T$, TRUE) to $HLC[i]$ /* update the timestamp in HLC*/
                    add $(x)$ to $item\_list$
                    send INVALIDATION_REPORT($item\_list, T$, FALSE) to $MH_i$
            end
    end
end
```

Figure 15: Event handler for a MSS.

```
Procedure MH_Main()
begin
    while (TRUE)
    begin
        if (MH receives a user request for data item ($x$)) then
        begin
            if ($First\_Request$ == TRUE) /* First request after disconnection*/
                send DATA_REQUEST($i, t_s, x$,TRUE)
                $First\_Request$=FALSE
                $First\_Waiting$=TRUE
                wait until DATA is received
            else
                if($First\_Waiting$ ==TRUE)
                    /* Buffer requests till the invalidation report is received for the first query. */
                    wait until $First\_Waiting$ = FALSE
                if( $x \notin cache$) /*item is not in the cache */
                /* make uplink query to MSS and wait for the data */
                    send DATA_REQUEST($i, t_s, x$, FALSE) to the $MSS$
                    wait until DATA is received
            /* use cached value to answer the query after data received or on a hit*/
            return $cache(x, Data_x)$
        end

        if (MH receives INVALIDATION_REPORT($item\_list, T, first\_flag$) then
        begin
            if ($First\_Request$ == FALSE or ($First\_Waiting$ == TRUE and $first\_flag$ == TRUE) )
                $First\_Waiting$ = FALSE
                $t_s = T$ /*set the timestamp of the local cache*/
                foreach ($x$) in $item\_list$ do
                    /*mark the mentioned items as invalid, remove from cache*/
                    delete ($x$) from $cache$
        end

        if (MH receives DATA($x, Data_x, T$)) then
        begin
            Update value of item $x$ in $cache$ to $Data_x$
            $t_s = T$
        end

        if (MH wakes up after disconnection) then
        begin
            $First\_Request$ = TRUE
        end
    end
end
```

Figure 16: Event handler for a MH.