An Efficient Cache Maintenance Scheme for Mobile Environment

A. Kahol
Cisco Systems Inc.
170 W. Tasman Dr.
San Jose, CA 95134
akahol@cisco.com

S. Khurana
Telcordia Technologies
445 South St.
Morristown, NJ 07960
sumit@research.telcordia.com

S. K. S. Gupta and P. K. Srimani
Dept. of Computer Science
Colorado State University
Ft. Collins, CO 80523
{gupta,srimani}@cs.colostate.edu

Abstract

In this paper we present a new cache maintenance scheme, called AS, suitable for wireless mobile environment. Our scheme integrates mobility management scheme of Mobile IP with cache maintenance scheme used in Coda file system. As opposed to broadcasting invalidation report schemes [1], AS supports arbitrary disconnection patterns and uses less wireless bandwidth. We present analytical and simulation results to show the superiority of our caching scheme.

1 Introduction

Data caching is an important technique for improving data availability and access latencies. It is especially important for mobile computing environments which are characterized by narrow bandwidth wireless links and frequent voluntary/involuntary disconnections from the server. These features of a mobile environment coupled with the need to support seamless mobility of the clients distinguish its cache maintenance algorithms unique and different from those used for wired networks since the protocols must try to optimize conflicting requirements under the constraints of limited bandwidth and frequent disconnection from the server. Further, these protocols should be energy-efficient, and adaptable to the varying QoS provided by the physical layer of the wireless network.

Existing cache coherency protocols proposed for mobile environment are all based on call-back mechanism; since the mobile clients get disconnected voluntarily or involuntarily, the invalidation messages (call-back break) used in this scheme often gets lost. To address this problem, Barbara and Imilienski [1] have developed a *invalidation report broadcasting* scheme. Several other schemes have also been proposed to extend this basic scheme with respect to optimizing the size of invalidation report [2], adjusting the periodicity of invalidation reports in accordance to the query rate and client disconnection time [3] and restricting broadcast to vicinity of the mobile client [4]. These schemes based on invalidation report broadcasting have the following common characteristics: (1) they assume a stateless server and do not address the issue of host mobility (except the work by Liu and Maguire [4]); and (2) the entire cache has to be discarded if the client is disconnected for a period larger than the periodicity of the broadcast (or some multiple of it), even when many of the data items stored in the local cache are still consistent or valid.

Coda file system [5, 6] provides support for disconnected operations on shared files in UNIX-like environment. Coda uses two mechanisms for cache coherency. As long as the client is reachable from at least one server, call-back mechanism is used. When disconnection occurs, access to possibly stale data is permitted at a client to improve availability; these data updates are checked for consistency upon reconnection and only those modifications that do not have any conflict are committed. Balance between speed of validating a 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 each reconnection puts an unnecessary burden on the client. Further, since Coda is a distributed file system it assumes a state-full server which may not be appropriate for other applications such as web caching.

In this paper we present a new cache maintenance scheme, called AS (Asynchronous Stateful) scheme, suitable for wireless mobile environment. Our scheme integrates mobility management scheme of Mobile IP with cache maintenance scheme used in Coda file system. In our proposed scheme each mobile host or client (MH) has one designated *home* Mobile Support Station (MSS) which maintains certain specific information about the client even when the client moves out of its immediate area of supervision. The home MSS maintains for each MH a data structure called *Home Location Cache (HLC)* which stores the latest time-stamp for each data item cached by the MH (i.e., when last time the data item was invalidated). This concept of HLC fits perfectly into existing architectures to support mobility in wireless networks (e.g. mobile IP[7]) which also uses the concept of a *home location* for each MH. In implementing our scheme, the same home location could be used to maintain the HLC. The proposed AS scheme 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; an MH can continue to use its cache even after prolonged periods of disconnection from the network, without the need of discarding the entire cache. AS supports arbitrary disconnection patterns and uses less wireless bandwidth. We present analytical and simulation results to show the improvements in bandwidth usage of our caching scheme.

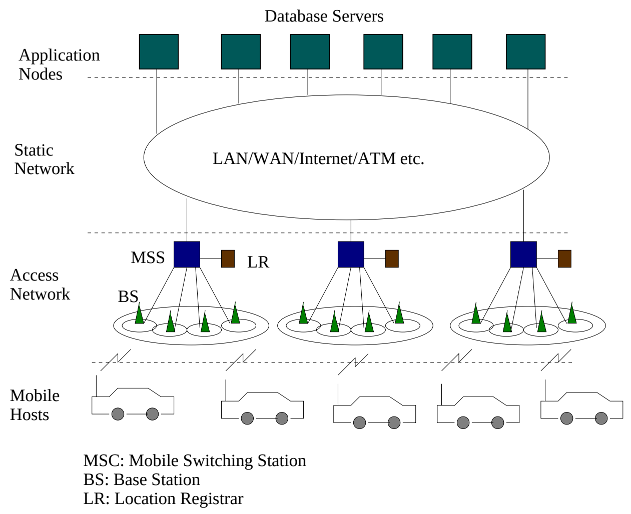


Figure 1. Mobile Computing Environment.

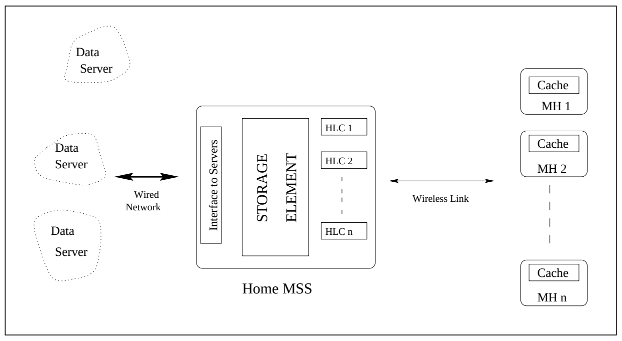


Figure 2. System Architecture

2 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 (BS). In accordance with 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 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. We assume that the mobility agents (home or foreign agent) (MAs) are located at the MSSs. In this paper we design our caching architecture based on Mobile IP.

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. For our purpose, a disconnected client is in sleep mode; we use the term *wakeup* to indicate reconnection. 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 mobility agents of a mobile host.

3 Overview of Proposed Scheme

The caching architecture is shown in Figure 2. The home agent of the mobile host maintains its *home location cache* (HLC). If a mobile host (MH) is roaming, its HLC is duplicated at the MSS of its foreign agent. Thus, a MSS always maintains a HLC for each MH in its coverage area. HLC_i for MH_i, as maintained in the MSS, keeps track of what data has been cached at MH_i. In general, HLC_i is a list $(x, TS, invalid\_flag)$ of each data item x being cached by MH_i, where x is the identifier of a data item and TS is the time-stamp of the last invalidation of x. The $invalid\_flag$ is set to TRUE for data items for which an invalidation has been sent to the MH but no (implicit) acknowledgment has been received. Note that this time-stamp is the same as that provided by the server in its invalidation message. Our scheme makes the following assumptions:

- Whenever a data item is updated at the server, it sends out invalidation messages to all the MSS over the wired network.
- An MH informs its mobility agent before caching any data item in its local cache.
- The mobility agent, which is nearest to the MH and maintains the HLC of the MH, forwards the MH any invalidation it receives from the server.

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

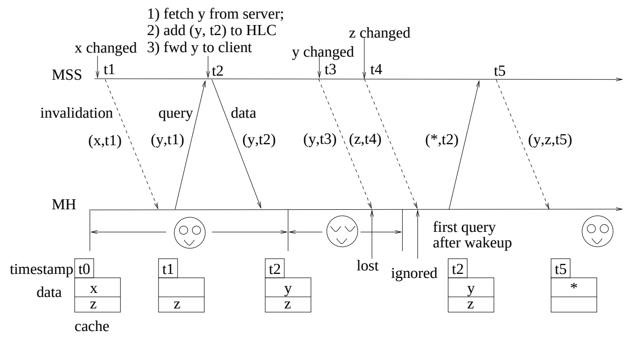


Figure 3. An Example Scenario

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 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 home agent with its cache time-stamp. In response to this probe message the home agent sends it a 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 home agent. 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 failures and voluntary disconnections. Even if the MH wakes up and then immediately goes 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 wakeup call again and 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 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 it 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 since later upon first query after wakeup it sends a invalidation check message to the MSS. The MSS uses the time-stamp in the cache check message to determine send a invalidation report with all the missed invalidations by the MH. In this case, the MSS determines from MH' cache time-stamp t2 that MH has missed invalidation for y and z and so it resends them to the MH.

4 Formal Description

4.1 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 structure is 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.

4.2 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.

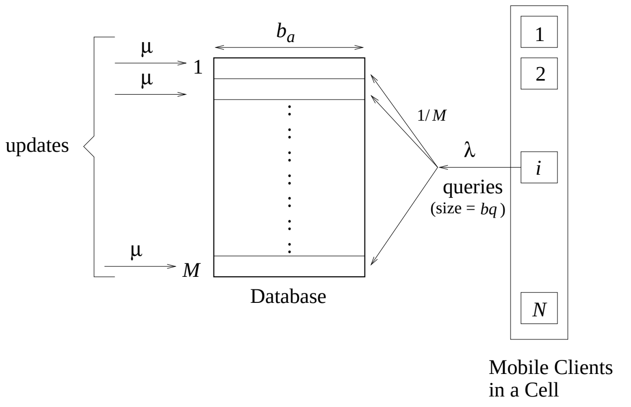


Figure 4. The Query Update Model

4.3 Protocol

An MSS continuously executes the procedure **MSS_Main**, whose response to various events is as follows:

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**, which responds to the various events in the following manner:

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.

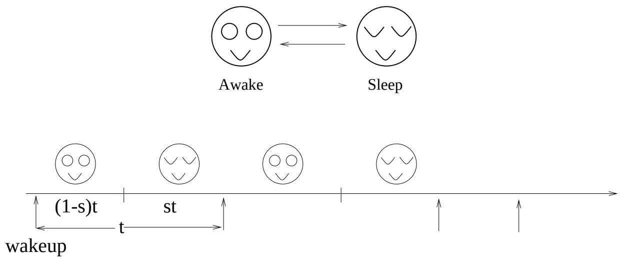


Figure 5. The Sleep Model

5 Performance Analysis

We provide a simple theoretical analysis 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.

5.1 Modeling Query-Update Pattern

The query-update model is shown in Figure 4. We assume that there are M data items in the entire database. Each data item is of size b_a bits. The time between updates to a data item is assumed to follow an exponential distribution with mean $1/\mu$. Each MH queries data items according to a Poisson distribution with mean rate of λ. These queries are uniformly distributed over all data items in the database.

5.2 Modeling Sleep Pattern

An MH can get connected and disconnected while it is in the cell. The sleep/wakeup pattern of an MH is modeled by using two parameters (see Figure 5). One is the fraction s of the total time spent by an MH in the sleep mode and the other is the frequency at which it changes it's state (sleeping or awake). To model these 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.

5.3 Estimation of Miss Ratio

We assume that all queries generated by an MH when it is sleeping, i.e., disconnected from the server, are lost. Thus the effective rate of query generation by an MH can be approximated as

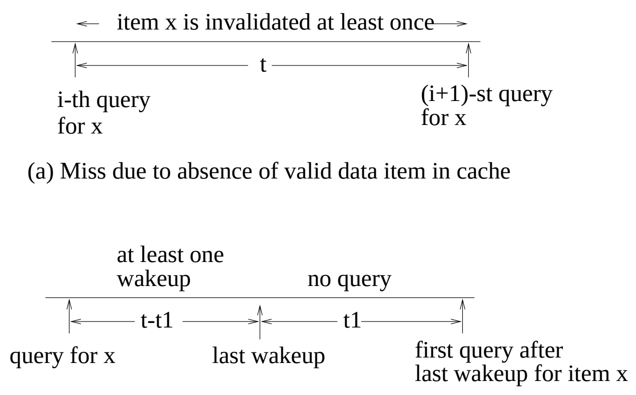

item x is invalidated at least once
t
i-th query for x
(i+1)-st query for x
(a) Miss due to absence of valid data item in cache
at least one wakeup
no query
t-t1
t1
query for x
last wakeup
first query after last wakeup for item x
(b) Miss due to diconnections.

Figure 6. Two Mutually Exclusive Events When Up-links Are Needed

$\lambda_e = (1-s)\lambda$. Since queries are uniformally distributed, the rate at which queries are generated for a *given* data item is given by $\lambda_x = \lambda_e/M$. A query made for a specific data item x by an MH would be a *miss* in the local cache (and would require an uplink 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. During this interval t, the data item x has been invalidated at least once (see Figure 6(a)).
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 6(b)).

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:

$$P[\text{Event 2}] = \int_0^\infty (P[\text{no invalidation and query for x during t}]$$

$$\times P[\text{the query (for x) is 1st after wakeup})dt$$

$$= \int_0^\infty \lambda_x e^{-\lambda_x t} e^{-\mu t} (\int_0^t e^{-\lambda_e t_1} \omega e^{-\omega t_1} (1-e^{-\omega(t-t_1)}) dt_1) dt$$

$$= \frac{\omega\lambda_x}{\lambda_e(\omega+\lambda_e)} \left(\frac{\lambda_e}{\mu+\lambda_x} - \frac{\omega+\lambda_e}{\mu+\lambda_x+\omega} + \frac{\omega}{\mu+\lambda_x+\lambda_x+\omega} \right).$$

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

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

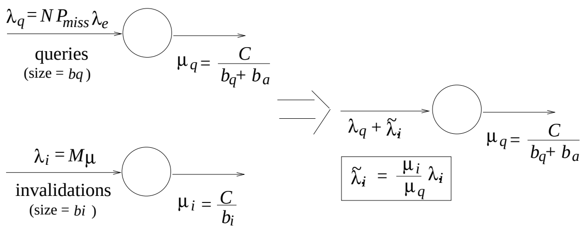

$\lambda_q = N P_{miss} \lambda_e$
queries
(size = bq)
$\mu_q = \frac{C}{b_q + b_a}$
$\lambda_i = M\mu$
invalidations
(size = bi)
$\mu_i = \frac{C}{b_i}$
$\lambda_q + \tilde{\lambda}_i$
$\tilde{\lambda}_i = \frac{\mu_i}{\mu_q}\lambda_i$
$\mu_q = \frac{C}{b_q + b_a}$

Figure 7. Combining the Up-link and Down-link M/D/1 Queues

5.4 Estimation of Mean Query Delay

We now estimate the mean query delay T_{delay}. A single wireless channel of bandwidth C 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 scheduling policy. Further, we assume that 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 $NP_{miss}\lambda_e$ since there are N many 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 7. 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

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

- 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 $T_{delay} = P_{miss}T_q$.

5.5 A Simple Comparison

Various strategies have been suggested in literature which use synchronous invalidation reports. They build on the basic ideas in

TS/AT [1]	**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 [1]	Ideal	AS
Max 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

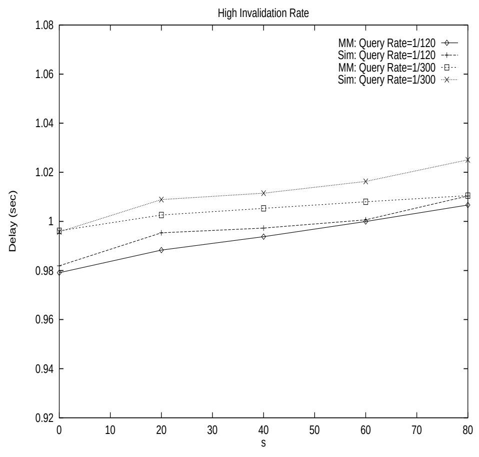
[8]. Wu in [9] has proposed an enhancement in which the mobile host sends back to the server the ids of all cached data items, along with their time-stamps after a long disconnection. The server identifies the changed data items and returns a validity report. This results in wastage of bandwidth and 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 [10] 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 [3] 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 basic drawbacks of a stateless scheme still hold.

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.

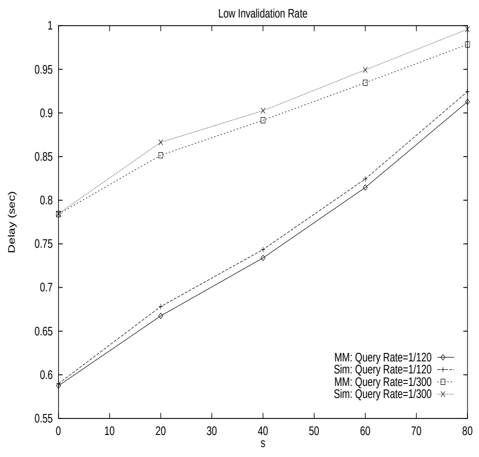
Unlike these strategies, all of which use synchronous invalidation reports and are based on the basic scheme of [8], our strategy is (A)synchronous and (S)tateful; we use the name **AS**. In order to compare our scheme with the sleepers and workaholics scheme of [8], we note the salient features of that 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 $k = 1$.
- 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.



(a) High Invalidation Rate



(b) Low Invalidation Rate

Figure 8. Delay: Simulation and Mathematical Model

6 Simulation Results & Comparison with Analysis

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, uplink 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 an *idealized scheme* where it is assumed that the invalidation information is available to the clients instantaneously at no cost. 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. We summarize our observations as follows:

- **Comparison with Analytical Results:** Figure 8 shows the comparison between the delay obtained through simulation 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.
- **Comparison with Ideal Scheme**: Figure 9 compares AS scheme with the *ideal scheme* in which invalidation information is instantly available to the clients at no cost. Figures 9(a) and (b) show the results for query delay and up-link messages (in case of a cache miss), respectively. As the number of queries per second decreases, both the number of up-links per query and the average delay to answer a query increase. This is a direct consequence of decrease in hit rate. As delay between queries increases, the probability of an invalidation occurring between queries increases; this results in more up-links and higher delay. As the 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 difference increasing as the sleep rate increases.

7 Conclusions

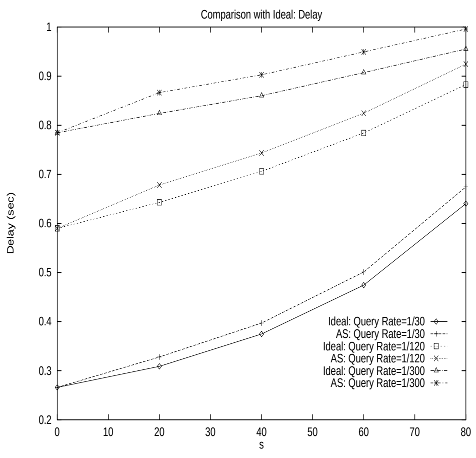
We have proposed a new cache maintenance scheme for wireless mobile environment that integrates mobility management scheme of Mobile IP with cache maintenance scheme used in Coda file system. Our initial theoretical and experimental studies suggest the following advantages: i) the server can be stateless and the overhead of maintaining state of the cache at the server is offloaded to the home agent, ii) upon reconnection the client has to send only a single message to the home agent to get any invalidations that occurred while the client was disconnected, and iii) being asynchronous and state-full it avoids repeated broadcast of invalidations for data items which may not be relevant to specific mobile clients.

Several enhancements can be made to this basic scheme. This includes handling lost invalidation message due to network failure in the wired network and handling the case when co-located care-of-address is used by the MH. The lost message can be handled by using sequence numbers and periodic pooling by the foreign agent on behalf of the mobile host. In the case when mobile host uses co-located care-of address, the home MSS can take the full responsibility of maintaining MH's HLC. Further, as in Coda the client can itself do periodic polling to detect any lost invalidation reports.

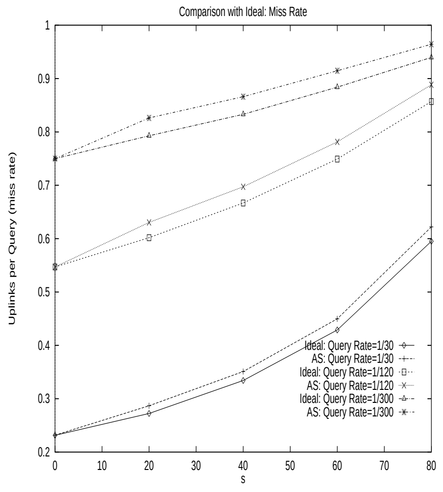
References

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

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



(a) Delay



(b) Uplinks

Figure 9. Comparison of Delay and Uplinks between AS and Ideal Scheme (s is the percentage of time a client is disconnected from the network)

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

[4] 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.

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

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

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

[8] 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.

[9] 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.

[10] 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.

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

Table 3. Default Parameters