Enhanced Scalable Asynchronous Cache Consistency Scheme for Mobile Environments

Derar H. Alassi Reda Alhajj*

Abstract

An important technique to reduce the contention on the limited bandwidth of wireless channels between mobile units and base stations is caching frequently accessed data items. In the literature, two approaches were proposed for cache consistency: Stateful and Stateless. In the Stateful approach, the server has to keep information about all the mobile units in its cell. On the other hand, in the stateless approach, the server does not store any information about clients. In this paper, we propose a hybrid cache consistency approach which combines the advantages of both Stateless and Stateful approaches; our approach has several characteristics in common with the Scalable Algorithm for Cache Consistency Scheme "SACCS", which has been reported to have advantages compared to some major previous algorithms, including TimeStamps, Signatures, Amnesic Terminals and Asynchronous Stateful. The proposed approach reduces the side-effect of the sleep-wakeup patterns, and uses new communication messages intended to invalidate only the entries changed during the sleep time. Further, we propose a better replacement policy for the mobile unit cache, which considers the size of the removed entry to improve channel utilization. Experimental results show that the proposed approach increases the mobile cache hit, reduces the delay time of queries and reduces traffic in both uplink and downlink channels.

Keyword: mobile databases, cache consistency, invalidation report, cache invalidation scheme.

1 Introduction

Wireless networks are becoming more popular and widely spread because of the reduced cost of mobile terminals (laptop computers, personal digital assistants, hand-held computers, etc.) and communication channels, as well as the increased services provided by mobile networks [5]. These services include, but are not limited to, financial information, news and weather forecast, traffic schedules and conditions, online shopping catalogs, among others. So, caching frequently accessed data items is a very efficient way to save power consumption and bandwidth. In wireless networks we try to utilize the traffic of both Downlink and Uplink. However, mobile networks suffer from frequent disconnectedness of the mobile units, the limited bandwidth for communication and the power limitation of the mobile units. These main factors make mobile network communication and cache consistency challenging [27, 6, 5, 24, 19].

A naive approach to access data items which reside in the server database is querying the server every time a mobile unit needs the item; this happens in server-client architecture. But, it is clear that this approach generates a lot of network traffic, and it is inefficient because of the narrow bandwidth of wireless networks, especially when we have a large number of mobile units. So, caching frequently accessed data items is the best approach to solve this problem. Another challenge is cache consistency. Invalidation

*Authors are with the Computer Science Department, University of Calgary, Calgary, Alberta, Canada, alhajj@cpsc.ucalgary.ca

Reports (IRs) technique was proposed to keep the data in the mobile unit cache consistent (same as the data in the original server database) [2]. In this technique, the sever broadcasts IRs periodically to the mobile clients, which invalidate data items whose IDs appear in the received IR message. Apparently, using IRs will better utilize the downlink from the server to clients, but the average latency is high. We will discuss this in more details in section 3.

Described in the literature, there are two main approaches for cache invalidation, namely stateful and stateless; in addition, the Scalable Algorithm for Cache Consistency Scheme "SACCS" has been proposed as a hybrid approach that tried to combine advantages of both techniques. Our algorithm described in this paper has several issues in common with SACCS [24]. In particular, we tried to avoid reinventing the wheel and instead we build on top of and enhance the advantages brought in by the developers of SACCS. This way, we guarantee keeping at least the same functionality and attractiveness of SACCS. In addition, our algorithm has some other distinguishing features that further enhance the proposed hybrid approach into a method mainly in terms of scalability and bandwidth utilization.

Our main motivation for the work described in this paper is that SACCS invalidates all the data entries in the mobile unit cache when waking up, even if there could be many data entries which are still valid. We realized that the following three issues are worth further investigation:

1. Improving the way of dealing with the sleep-wakeup patterns.
2. Time Synchronization between the base station and clients.
3. Improving the replacement policy of the mobile unit cache.

Finally, the conducted experiments demonstrate the effectiveness of the proposed method as compared to SACCS. We decided to limit our comparison to SACCS because it has already been reported in the literature as better than the other approaches, and hence achieving performance better than SACCS directly indicates superiority over the other approaches.

The rest of the paper is organized as follows. Section 2 is an overview of the related work. Section 3 presents the proposed approach, named "ESACCS", which mainly enhances the original SACCS while keeping all its other characteristics. Section 4 reports the experimental results. Section 5 is conclusions and future research directions.

2 Related Work

Data dissemination is the delivery of data from a set of producers to a number of clients. This is characterized by an inherent asymmetry in the communication: The bandwidth in the downstream direction (server-to-clients) is much greater than in the upstream direction (clients-to-server). This model fits very well for a mobile system since mobile clients are usually unable to transmit data at a very high speed (sometimes this capacity is not even present). However, they can receive data at high rate. The model of sending information to a client population, without waiting for specific requests is also called push-based. By pushing data, the servers avoid interruptions caused by requests and are allowed to propagate data whose existence would otherwise be ignored by clients. On the other hand, push-based systems have the problem of deciding about the relevance of the data to be broadcasted to clients. The usefulness of the system depends on the ability to predict the clients' needs. A simple solution is to allow clients to subscribe to services while providing profiles of their interests. The clients subscribe to information providing queries that describe their interests. These queries can be viewed as continuous queries.

The server also needs to decide whether to send the data periodically or aperiodically. Periodic push has the advantage of allowing clients to disconnect for certain periods without missing out items (since

they will be broadcasted again after the client is reconnected). Aperiodic dissemination, on the other hand, is a more effective way of using the available bandwidth. Periodic push has been used in the past in many systems.

Wireless mobile communication has increasingly become an important means to access various kinds of dynamically changing data objects such as news, stock prices, and traffic information. However, wireless networks have very limited bandwidth and battery power [9]; they also have to deal with user mobility and disconnectedness. Thus, data communication in such networks is much more challenging than in wired networks.

Caching frequently accessed data objects at the local buffer of a mobile user (MU) is an efficient way to reduce query delay, save bandwidth, and improve overall system performance. But, frequent disconnections and roaming of MU turn cache consistency into a difficult task in mobile computing environments. A successful strategy must efficiently handle both disconnectedness and mobility. Broadcast has the advantage of being able to serve an arbitrary number of MUs with minimum bandwidth consumption. Thus, efficient mobile data transmission architectures must carefully design their broadcast and cache management schemes to maximize bandwidth utilization and also to minimize average query delay. Additionally, such architectures should be scalable to support large database systems as well as large number of MUs. Finally, there are two types of cache consistency maintenance approaches for wireless mobile environments: stateless, e.g., [2, 3, 11, 8, 16, 23, 26, 28] and stateful, e.g., [2, 4, 13]. On the other hand, the work of Wang *et al* SACCS as described in [24] is kind of a hybrid of stateful and stateless approach. It does not require periodic broadcast of IRs; rather it keeps one flag bit for each data object in MSS; this bit helps in determining when to broadcast IRs. In the following, we summarize existing algorithms for both stateful and stateless approaches.

To deal with the long disconnection problem, Elmagarmid *et al* [8] proposed a scheme called the Bit-Sequence (*BS*) with a compressed invalidation report consisting of a set of binary bit-sequences and their corresponding timestamps. This approach is mainly attractive for mobile clients with frequent and long disconnections. However, for large databases, the size of the set of binary bit-sequences becomes large. Consequently, the model requires broadcasting larger IR content than the approach described in [7].

Tan *et al* [21, 23] re-examined the Bit-Sequence scheme and studied different organizations of the invalidation report to facilitate clients so they can selectively tune to the portion of the invalidation report of interest to them. This allows mobile clients to minimize the power consumption when invalidating their cache content.

Hu and Lee [11] introduced several adaptive cache invalidation schemes to reduce the content of each IR and to handle the long disconnection problem. This leads to broadcasting different types of IRs based on update frequency, query rates/patterns, and client disconnection time and frequency. Thus, this scheme can effectively deal with the long disconnection problem. However, the average query latency is not improved.

Fong et al [10] proposed a distributed caching scheme that reduces the bandwidth consumption. The server broadcasts a subset of updated items from the previous IR interval to the mobile clients within the cell. Cao [5] proposed a modified version of the IR-based cache invalidation scheme to solve the long query delays problem in synchronous IR broadcasting schemes. This is possible by replicating and broadcasting several times within an IR interval a small fraction of the essential information of an IR, called Updated Invalidation Report (UIR). The client can process a query right after it receives the next IR or UIR. If the query result can all be found in the cache, the client can immediately send the result to the user. However, when the server receives a query, it defers the response until the next IR and broadcasts all the data identifiers and values queried during the previous IR interval to the clients. As a result, this is an active research area that has attracted the attention of several research groups to develop efficient and effective algorithms to cope with the mobile environment.

2.1 Stateless Approaches

Barabara and Imielinski [2] proposed three stateless algorithms. They are Timestamps (TS), Amnesic Terminals (AT), and Signature (SIG), in which the mobile MSS broadcasts IR messages every L seconds. An IR message includes all data object IDs updated during the past $k \times L$ seconds, where k is a positive integer. The advantage of these algorithms is that an MSS does not maintain any state information about its mobile units caches (MUCs), thus allowing simple management of the SC. However, they suffer from the following drawbacks:

1. They do not scale well to large databases and/or fast updating data systems due to the increased number of IR messages;

2. The average access latency is always longer than half of the broadcast period, simply because all requests can be answered only after the next IR;

3. When the sleep time (during which an MU is disconnected from its MSS) is longer than $k \times L$, all cache entries are deleted, thus leading to unnecessary bandwidth consumption, particularly if the data objects are still valid.

In order to handle the long sleep-wakeup patterns, several algorithms have been proposed. For example, in the bit-sequence (BS) algorithm [8], all cache entries are deleted only when at least half of the data entries in the cache have been invalidated. However, the model requires the broadcast of a larger number of IR messages than TS and AT schemes. Although the uplink validation check scheme [26] can deal with long sleep-wakeup patterns, it requires more uplink bandwidth and cannot handle very long sleep-wakeup patterns. In order to reduce the IR messages, adaptive methods are developed by Hu and Lee [11] to broadcast different IRs based on update frequency, MU access, and sleep-wakeup patterns. Yuen *et al* [28] developed an approach where an absolute validity interval (AVI) is employed for each data object; however, it fails to reduce the access delay introduced by periodic broadcast cycles. In the preceding approaches, all MUs can benefit from the broadcast only when they retrieve the same data objects from the MSS in the same broadcast cycle. If the MUs retrieve the same data objects in separate broadcast cycles, they cannot share the broadcasted data objects. This makes the broadcast inefficient and sensitive to the number of MUs in the cell. The TS strategy is modified in [3] by keeping the invalidated data objects in an MUC such that the MU can update a data object if it is received from the broadcast channel. This approach increases the broadcast channel utilization. However, keeping invalid data objects in an MUC wastes precious cache memory. Finally, a comprehensive performance evaluation of the existing stateless algorithms can be found in [23].

2.2 Stateful Approaches

Several stateful cache consistency maintenance algorithms have been proposed for wireless mobile environments. Described in [13] is an asynchronous stateful (AS) algorithm to maintain cache consistency in which an MSS records all retrieved data objects for each MU. An MU first retrieves a data object after it wakes up; it receives an IR from the MSS based on the MUC content record and sleep-wakeup time. Whenever an MSS receives an update from the original server for a recorded data object, it immediately broadcasts that object's IR to MUs. The advantage of the AS scheme is that MSS avoids unnecessary IR broadcasting to MUs. Moreover, MUs can deal with any sleep-wakeup pattern without losing valid data objects. However, in order to maintain each MUC, MSS must record all cached data objects for each MU. Hence, an MU can only download data objects which it requested through the uplink. This

makes the broadcast channel utility inefficient and sensitive to the number of MUs. Finally, a counter-based scheme is used by Cao as described in [4] to identify the hot data and save unnecessary IR traffic. Whenever MUC content is changed, the MU must piggyback the change to the server, thus consuming battery power and uplink bandwidth.

The above schemes assume reliable communication, but the associated costs are not evaluated. The cost of reliable communication for broadcasting (or multicasting) is significant since the uplink data transmission requires much more battery power than downlink data transmission, especially for the acknowledgment of IR broadcasting (or multicasting) due to the increased channel competition. On the other hand, the MSS cannot distinguish a disconnected MU from a connected one which has not received an IR. Therefore, it is necessary to allow a maximum number of retransmissions.

Barabara and Imielinski [2] present a novel way for sending invalidation messages over a limited bandwidth network. By using IRs, the server can notify clients about changes in the items that are being cached by them. The IRs are very succinct ways of transmitting this information. For instance, one could send a list of identifiers of the items that have changed since the last IR was sent. Depending on the technique used, IRs can cause "false negatives", making a client drop an item from its cache when in reality the item is still valid. In return for this anomalous behavior, one gets shorter report that consumes less bandwidth.

Barabara and Imielinski [2] study the tradeoffs involved in using different types of IRs. They also addressed the issue of relaxing consistency of the cache. For instance, if the mobile clients are caching stock prices, it may be perfectly acceptable to use values that are not completely up to date, as long as they are within 0.5% of the true prices. This can be accomplished by considering the cached values as *quasicopies* of the values in the server [1]. A quasicopy is a cached value that is allowed to deviate from the true copy in a controlled way. Using quasicopies, the IRs can be made even shorter, thereby saving extra bandwidth for their transmission.

Barabara and Imielinski [2] further extended the IRs strategies to dynamically adjust to changes in the query and update rates, and the disconnection patterns of the mobile clients. These new techniques, called adaptive IRs, take into account the on-demand queries that the clients make when they do not find the items in their local caches. Wu *et al* [26] address the problem of discarding cache content in MUs that have been disconnected for any period of time (sleepers). In [2], the strategy of MU after it disconnects is to purge its cache after it comes back in operation. Wu *et al* [26] propose a mechanism to decide whether some items in the cache can still be used by the mobile unit. To do so, the mobile units need to contact the server when they come back from a disconnection and ask if the items in the cache are still usable. To decrease uplink traffic, the database is partitioned into groups, and elements in the same group are cached together. This way, the MU only asks the server for the validity of specific groups. The server keeps track of the update history for elements in each group, and using an algorithm called Grouping with Cold Update-Set Retention (GCoRe), it determines whether or not all the items updated since the MU got disconnected have been included in the last IR. If the answer is yes, the cached group can be kept (since the mobile unit can hear about the updates in the IR). Otherwise, the cached group should be discarded (there is no enough information in the IR to determine which elements to keep). Using simulation, the authors determine that as the group size increases, the uplink traffic decreases, but the downlink traffic increases. The total bandwidth usage first decreases and then increases, suggesting an optimal operation point.

In the approach of Cao as described in [4, 5], all mobile clients in the cell keep the invalidated data items in their caches, and the server records the caching information for these clients and associates a counter to each data item in the database. Counters are used to identify hot (i.e., frequently requested) data items. The server broadcasts new values of hot data items right after each IR, and then connected clients prefetch the data items for their invalid cache entries. Once the server receives a data request, it processes

the request and broadcasts the answer immediately to all the connecting clients. Each time a client wants to check its cache validity after a long disconnection period, it simply sends a reconnection message to the server and the server replies with an invalidation report. Therefore, this approach can improve the broadcast channel utilization, reduce the average query delay, and solve the long disconnection problem. If considerable number of cached data items are replaced, then a large amount of uplink bandwidth is generated, which consumes the clients' limited battery power.

2.3 Cache Consistency Approaches

Caching frequently accessed data items is the best way to overcome the narrow-bandwidth of wireless networks. But, when dealing with caching, we have to ensure the consistency of the cached data items. The approaches described in the literature may be broadly classified under two categories, namely Stateful and Stateless.

In general, IRs broadcasting is an efficient way to ensure cache consistency. The algorithms using IRs for cache consistency follow either Synchronous or Asynchronous strategy, i.e., the base station sends IRs periodically (every L seconds) or immediately after a data entry has been changed, respectively. For the synchronous approach, the average delay is $\frac{L}{2}$, because a mobile unit cannot answer a query until it receives the next IR sent by the base station. On the other hand, it utilizes the downlink in a better way because less IRs are sent on the average [4, 27]. But for the Asynchronous approach, the average delay is usually very small and dependent on the rate of data change. Apparently, here the base station sends more broadcast messages (IRs); so more downlink traffic (could be sometimes unnecessary).

In the stateless approach, the base station does not have to store any information about the mobile units in its cell. Clearly, it is very simple in terms of database management. But, sleep of mobile units occurs frequently. So, if the base station did not store information about what each mobile unit cached and when it slept, it would not be able to deal with this situation efficiently. A naive way is to invalidate all the entries after waking up. An advantage of this approach over the stateful one is that the management of the base station database is very simple. However, it adds a lot of overhead on the uplink and downlink channel. Some of the algorithms following this approach, e.g., Amnesic Terminals (AS), Signatures and Timestamps, use periodic IR broadcast. So, the delay on average is $\frac{1}{2}$ the period, with worst case equal to the whole period [2, 24].

The other approach for the base station data management is the stateful approach, e.g., the Asynchronous Stateful (AS). Here, the base station has to store information (cached entries and sleep-wakeup time) for all the clients in its cell. An advantage for this approach is that the base station can deal with any sleep-wakeup patterns of the mobile units, and hence saves the bandwidth; but the management of the database of the base station is much more complicated than in the stateless approach. To over the high latency due to using the IR technique, the approach described in [3] reduces latency by sending *updated invalidation reports* which contains the data items that have been updated after the last IR has been broadcasted, i.e., the server sends smaller size packets more frequently.

2.4 SACCS

In this section, we provide a brief overview of SACCS on which we base our approach described in this paper. The additional distinguishing features of our approach are highlighted later in Section 3.

2.4.1 Description

SACCS stands for "Scalable Asynchronous Cache Consistency Scheme" [24]; it is a superior algorithm for cache consistency. Actually, it is a hybrid of both stateless and stateful approaches. It is considered to

be stateful because the base station keeps 1 bit flag to maintain whether this entry had been broadcasted before or not. On the other hand, it is considered to be stateless because the base station does not need to keep information about mobile units in its cell. For instance, comparing SACCS to AS, AS needs buffer space of $O(MN)$, but SACCS needs $O(N)$, where M is the number of mobile units and N is the number of the data entries in the base station database.

SACCS still has a drawback in dealing with the sleep-wakeup patterns of mobile units. It invalidates all the data entries of a mobile unit cache after it wakes up. But, in real cases, sleep-wakeup patterns happen more frequently for the sake of power saving or disconnectedness issues. The need for optimizing this is one of the main motivations for developing the approach described in this paper.

2.4.2 Database and Cache Management

The data entries to be stored in the server database have the following structure: (dx, tx, lx, fx): where tx is the last update time for the data object, lx is the estimated time to live (TTL), and fx is the flag bit [9]. On the other hand, the cache of a mobile unit keeps: (dx, tsx, llx, sx): where tsx is the time stamp denoting the last updated time for the cached data object dx, llx is an associated TTL, and sx is a two-bit flag identifying four data entry states: 0, 1, 2, and 3, indicating valid dx, uncertain dx, uncertain dx with a waiting query, and ID-only, respectively [24].

It can be easily seen that the flag bit in the server database reduces the size of the broadcasted IRs. On the other hand, the mobile unit does not delete any entry from its cache when it wakes-up, it just modifies the flags of all entries in its cache to be uncertain. Then, if the mobile unit wants to use any entry which is *uncertain*, it will send first *Uncertain* message to the server and the server will reply by a *Confirmation* message if it is valid; otherwise the base station will send the data itself. So, sending *Confirmation* and *Uncertain* messages, which are few bytes, is better than sending the data itself, which could be multiple of Kilo-Bytes.

An advantage of the TTL field in the mobile unit cache is that if the mobile unit misses some IRs because of network errors (not by the sleep state), the data entries will be invalidated automatically after the TTL expires. When a new data entry is received by a mobile unit and its cache is full, the LRU replacement policy is used. So, the data entry at the tail of the cache will be removed.

2.4.3 Shortcomings of SACCS: our motivation

Although SACCS is more effective when compared to the other algorithms described in the literature, we realized that it is possible to develop another approach that keeps the already demonstrated advantages of SACCS and at the same time overcomes the following issues. First, SACCS ignores the time synchronization between the mobile units and the base station. This is a very critical issue because if time is not synchronized, some data entries will be marked as invalid although they are valid. Second, SACCS uses LRU as the cache replacement policy in the mobile unit cache. In mobile cache, data entries mostly differ in size, and hence a better replacement policy is a policy which also considers the size of the removed entry. Finally, SACCS invalidates all the entries of a mobile unit cache after waking up; this is anticipated to add more overhead on the network traffic, especially the uplink channel. It is known that sleep-wakeup patterns are frequent for a mobile unit to save power, and hence a good algorithm has to deal with such patterns in an efficient way. Identifying these weaknesses motivated us to develop the novel approach ESACCS described in this paper.

3 The Proposed Approach: ESACCS

3.1 Overview

The main theme of the approach proposed in this paper is to build on top of SACCS in order to produce a more effective method to deal with the cache in the mobile environment. In particular, ESACCS brings in techniques to deal with the three main issues enumerated in Section 2.4.3.

ESACCS proposes two new communication messages along with all the previous messages for SACCS. Also, a better replacement policy for the mobile unit cache is used, which considers the size of the removed record. Usually LRU is good when we have all the records in a cache having the same size, but if they vary in size, it is better to put into consideration the size of the removed record as well. As SACCS just marks records as invalid without deleting them physically, they could be used later by sending a *uncertain* message to the base station. To benefit from this point, it is better to consider the size of the record to be removed before it is removed.

ESACCS deal with roaming in the same way as SACCS does. ESACCS invalidates all the cache entries and sets them to *uncertain state*. In the following sections we will discuss these issues in more details.

3.1.1 Time Synchronization

ESACCS ensures the time synchronization between the base station and the mobile unit; there are many protocols which can be used here, e.g., the standard RFC 1305 Network Time Protocol (NTP), Simple Network Time Protocol (SNTP), Time Synchronized Mesh Protocol (TSMP) and a proposed protocol for wireless network time synchronization "The Flooding Time Synchronization Protocol" [10]. We will not go in details in this issue; it is beyond the scope of this paper. Here is an example which demonstrates the need to synchronize time between the base station and mobile units.

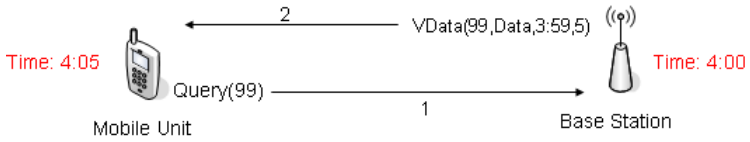

Figure 1. Time Synchronization

As demonstrated in Figure 1, the time on the mobile unit leads that of the base station by 5 minutes. When the mobile unit gets the record which was last updated on the base station at 3:59, this means it will expire at 4:03 (because the TTL for the record is 4 minutes). But the time on the mobile unit when it receives this record is 4:04 or more (because of network delay). So even if the record is valid, it will be marked as invalid and can not be used until another query is submitted to the base station, and so on. So, it is very clear that time synchronization should be done.

3.1.2 Replacement Policy

It might be good to use the LRU policy in CPU cache because the entries inside the cache are all of the same size, and there would not be any overhead when removing entries. However, in mobile cache, the entries vary in size. So, there could be a difference in the size in multiples of KB between two consecutive records. Hence, ESACCS extends the LRU policy to consider the size in the replacement policy. If the mobile cache is full and there is no free space to insert a new record, it searches the last 5% of the records

in the Mobile Cache, and removes among them the record (or group of records) that fits most the size of the new record; a group of records are considered when the incoming record is larger than any single existing record. We repeat this step towards the head of the cache until we have enough free space to insert the new record, but each time we decrease the number of records we are searching. This would decrease the downlink traffic.

3.1.3 Sleep-Wakeup Patterns

Saving power is one of the most important challenges that wireless networks. Batteries have a limited working time. So, mobile units sleep to reduce power consumption. During this time, there might be many broadcasted IRs missed by a sleeping mobile unit. As a result, here we lose consistency, which is the main issue. Stateful algorithms like AS, can easily deal with this issue. The base station checks the locally stored information about each mobile unit (entries cached, sleep time, etc) and according to this information it will update that mobile unit. But, this is not efficient for large number of mobile units in a cell. Stateless algorithms delete all the cache entries after sleeping, which is not efficient at all. SACCS has a hybrid technique in between, using only a bit in the base station database, but still can be improved.

ESACCS enhances the way we deal with the sleep-wakeup patterns using a very simple technique. ESACCS requires any mobile unit that decides to sleep, to keep the time stamp before going into the sleep mode. After this mobile unit wakes up, it will send to the base station a *Wakeup(TimeStamp)* message. When the base station receives this message, it checks the database to find all the previously broadcasted entries (IDs) having the TimeStamp of the last update greater than the TimeStamp sent in the *WakeupInvalid(TimeStamp,IDs)* message. Using this technique, we reduce the traffic in both the uplink and downlink bandwidths. The experimental resultsreported in this paper demonstrate that the cache hit (ratio between the total number of hits and requests) rate of ESACCS is higher than that of SACCS. As well, the average delay of queries in ESACCS is less than that in SACCS.

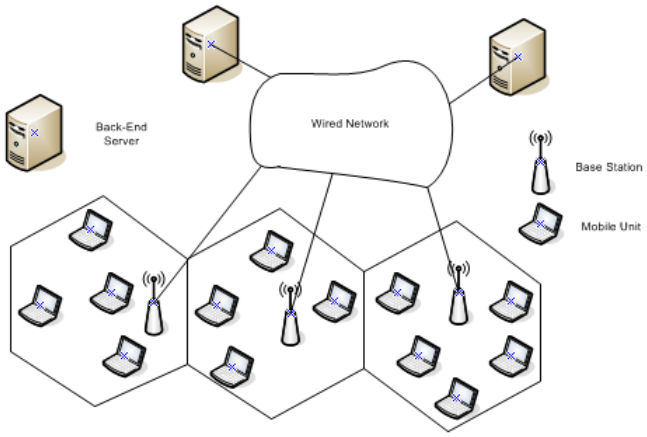

Figure 2. Wireless Network Architecture

3.2 System Architecture

The wireless network we assume is shown in Figure 2. We consider that this network consists of many cells; each cell has a base station, and includes many mobile units [24, 25]. We assume that all the

databases are replicated on all the base stations, i.e., we have the same content in all the databases.

3.3 Communication Messages and the Algorithms

3.3.1 Communication Messages

ESACCS has communication messages issued by the mobile units and other communication messages sent by the base station. All the communication messages used in SACCS are used in ESACCS with further two communication messages to be used in the sleep-wakeup states (assuming that the data is replicated between all the base stations and the original servers).

Messages from Mobile Unit to Base Station

A. *QUERY(x)*: Query for data object entitled x.

B. *Uncertain(x, tsx)*: Querying whether the data object entitled x with the timestamp txs is valid or not.

C. *Wakeup(ts)*: Querying the IDs (titles) of the data objects which have been changed since that sleep time.

Messages from Base Station to Mobile Unit

A. *Vdata(x, dx, lx, tx)*: Broadcast valid data object dx with update time at tx and TTL=lx

B. *IR(xs)*: Invalidation Report containing IDs (titles) of data objects which are not valid anymore.

C. *Confirmation(x, lx, tx)*: Ensure that the data object dx is valid if $tsx = tx$ and TTL = lx.

D. *WakeupInvalid(tx, x)*: data objects with time stamp tx are invalid.

3.3.2 Algorithm

ESACCS has two versions of algorithms. One runs on mobile units and the other runs on the base stations.

We will consider the algorithms proposed in SACCS as basis for both the client-side and the server-side algorithms in ESACCS. In particular, we adapted for ESACCS the algorithms developed foe SACCS by incorporating and planting in them the statements necessary and sufficient to handle the novel capabilities of ESACCS, which were missing in the original SACCS.

Client-Side Algorithm As we mentioned, each entry in the mobile cache has a 2-bit flag (Sx) to show its state.

If the mobile unit was queried, it first searches in its local cache, if the data object queried does not exist or $Sx = 3$ (ID-only) it will send a message Query(x) to the base station; otherwise, it will check if it is valid ($Sx = 0$) it will answer the query using the locally cached data. But if $Sx = 1$, it will send a *Uncertain(tsx, dx)* to the base station to check whether this entry is valid or not.

If the mobile unit receives an $IR(x)$, it will change the state of the object x to be ID-only ($Sx = 3$) if it has this data object in the local cache.

Table 1. Data Objects States

Sx	***Data object state***
0	*Valid*
1	*Uncertain*
2	*Uncertain with a waiting query*
3	*ID-only (deleted)*

If the mobile receives *Confirmation(x, lx, tx)* and it has this data object in the cache, it will compare the time stamps and then refresh the state of this entry.

If the mobile unit wants to sleep, it first stores the current timestamp locally. When it wakes up, it will send a *Wakeup(ts)* message to the base station.

So, if a mobile unit receives a *WakeupInvalid(ts, x)*, it will update all the entries it has to be ID-only, and leaves the others the same state.

But if the mobile unit receives a *VData(x, dx, lx, tx)*, it will download it if the received object was previously in the cache and its state is ID-only, otherwise if it is uncertain it will be refreshed, otherwise if it has a waiting query, it will answer the query and then download it to the cache.

Finally, when TTL expires, it will change the state of that entry to be uncertain. The pseudo code for the client algorithm is given next:

```
MobileUnitAlgorithm()
   {
   while(true)
   {
   if ( going to sleep)
   {
       store the current timestamp locally;
   }
   if ( waking up)
   {
       send Wakeup(ts) to the BS;
   }
   if ( TTL expires for entry x)
   {
       set the valid entry x into uncertain state ( sx =1 );
   }
   if ( mesage was received)
   {
       if(it is a Request message for dx)
       {
           if( dx is valid in the cache)
           {
               answer the request with cached data object dx;
               move the entry into the head of the cache head;
           }
           else if ( dx is in uncertain state)
           {
               send Uncertain(x,tsx) message to the BS;
               add the ID, i.e., x, to query waiting list;
               set sx = 2 and move the entry into the head of cache list;
           }
           else if ( x is ID-only entry in the cache)
           {
```

```
        send Query(x) message to the BS;
        remove the entry x in the cache;
        add x to query waiting list;
    }
    else
    {
        send Query(x) message to the BS;
        add x to query waiting list;
    }
}
if( it is Vdata(x, dx, lx, tx) message)
{
    if( x is in query waiting list)
    {
        answer the request with dx;
        remove the uncertain entry x if it exists in the cache;
        add the valid entry x at the cache list head;
    }
    else
    {
        if (x is ID-only entry in the cache)
        {
            download dx to the original entry location in the cache;
        }
        else if ( x is an uncertain entry in the cache)
        {
            if(tsx<ts)
            {
                download dx to the original entry in the cache;
                set tsx = tx; llx = lx; sx = 0;
            }
            else
            {
                set sx = 0;
            }
        }
    }
}
if ( it is an IR(x) message)
{
    if( entry x is valid or uncertain in the cache)
    {
        delete dx and set sx = 3;
    }
}
if(it is Confirmation(x, lx, tx) message)
{
    if(x is an uncertain entry in the cache)
    {
        if(tsx == tx)
        {
            set sx = 0 and llx = lx;
        }
        else
        {
            delete dx and set sx = 3;
        }
    }
```

```
    }
}
}
}
```

Server-Side Algorithm For the server side, the algorithm is simpler. If the base station (server) receives a *Query(x)* message, it will reply with a *Vdata(x, dx, lx, tx).*

If it receives a *Uncertain(tsx,dx)*, it will check whether the entry is valid or not by comparing the timestamps. If the entry is valid, *Confirmation(x, lx, tx)* is broadcasted; otherwise, *VData(x, dx, lx, tx)* is broadcasted.

Finally, whenever a data entry in the local database of the base station changes, the base station immediately broadcasts an *IR(x).* The pseudo code for the server-side algorithm is given next:

```
BSAlgorithm(){
       for BS receiving a message
       {
           if (it is Query(x) message)
           {
               fetch data entry x from the database;
               broadcast Vdata(x, dx, lx, tx) to all Mobile Units;
                   if (fx == 0)
                       set fx = 1;
           }
           if (it is Uncertain(x, tx) message)
           {
               fetch data entry x from the database;
               if (tx == tsx)
               {
                   broadcast Cofirmation(x, lx, tx) to all Mobile Units;
               }
               else
               {
                   broadcast Vdata(x, dx, lx, tx) to all Mobile Units;
               }
           if (fx == 0)
           {
               set fx = 1;
           }
           }
           if ( entry x has been changed)
           {
               broadcast IR(x) to all Mobile Units;
           }
           if( it is Wakeup(ts))
           {
             fetch all IDs of entries that has been changed after this time stamp ts;
             broadcast WakeupInvalid(ts, IDs);
           }
       }
       }
```

4 Experimental Results

In our experiments, for fair comparison, we use the same parameters used in [24]. *Uplink Bandwidth* = 1Kbps, *Downlink Bandwidth* = 200Kbps. Uplink message = 20B and the *Downlink confirmation* or

invalidation messages = 20B. Some other parameters may be changed for different cases, *number of data objects in the system*, *mobile unit cache*, *zipf coefficient* and *number of mobile units in the system*. As well, we modified the implementation code used in [24] to meet the new extensions we added on top of the SACCS, and we carried out these experiments. We get the implementation of SACCS from its original authors. We extended and modified the received implementation as necessary to cover the different novel ideas proposed for ESACCS.

In these experiments, we did not compare our proposed algorithm ESACCS against other algorithms like: AS, TS, Signatures, however, we compared it only against SACCS, because SACCS has already been demonstrated more efficient and scalable than all these algorithms.

We assumed in these experiments different scenarios. For example, we considered high and slower update rates of the data objects in the database of the base station. We also considered different sizes of the data objects to be cached in the mobile units' cache. Further, we executed these experiments on different number of mobile units.

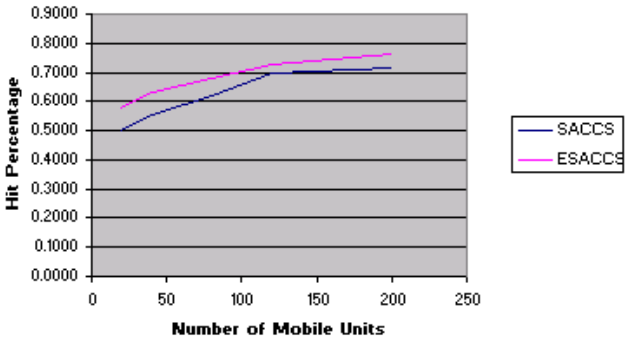

Figure 3. Hit Ratio

4.1 Cache Hit

Cache hit is defined as the number of hits divided by the number of requests. As shown in Figure 3, ESACCS has a better cache hit copared to SACCS; the improvement in some cases is 10%, and it is 7% on average. Apparently, because ESACCS does not invalidate or change the state of the records in the mobile unit cache into uncertain state when waking up, ESACCS achieves a better cache hit percentage. This improvement will be crucial in systems having high frequency of queries.

4.2 Average Delay

Average delay is defined as the total delay divided by the number of requests. ESACCS achieves better average delay compared to SACCS. For investigating the performance of ESACCS vs SACCS in terms of average delay, we carried out 3 types of experiments. First, we investigated the effect of the number of mobile units on the system. As we see in Figure 4, when the number of mobile units increases, the difference between ESACCS and SACCS increases, which means that ESACCS is more scalable than SACCS in terms of delay time. However, the improvement is not significant in terms of average delay; this is because both ESACCS and SACCS have the same scheduling policies for queries. A small improvement in average delay is gained because a better cache hit is achieved.

Then, we investigated the effect of the Zipf coefficient. Figure 5 shows that ESACCS still gives a better performance compared to SACCS.

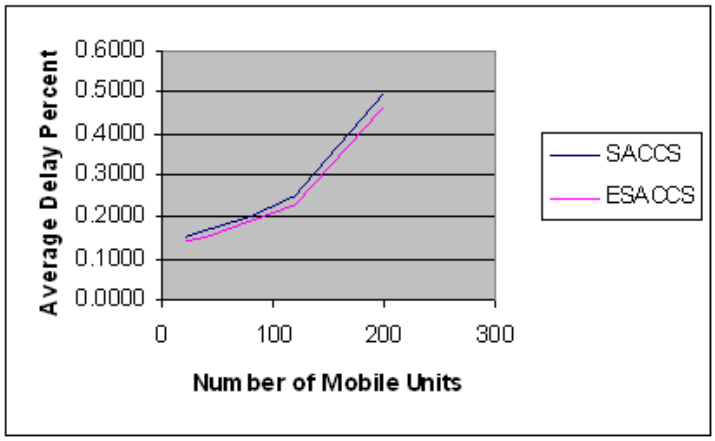

Figure 4. Average Delay Time vs. No. of Mobile Units

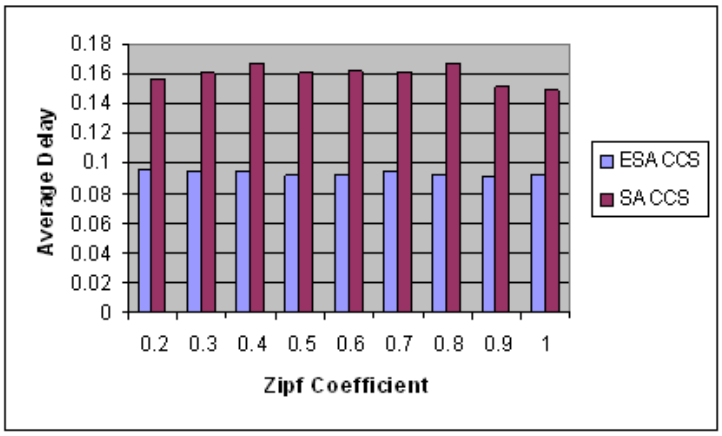

Figure 5. Average Delay Time vs. Zipf Coefficient

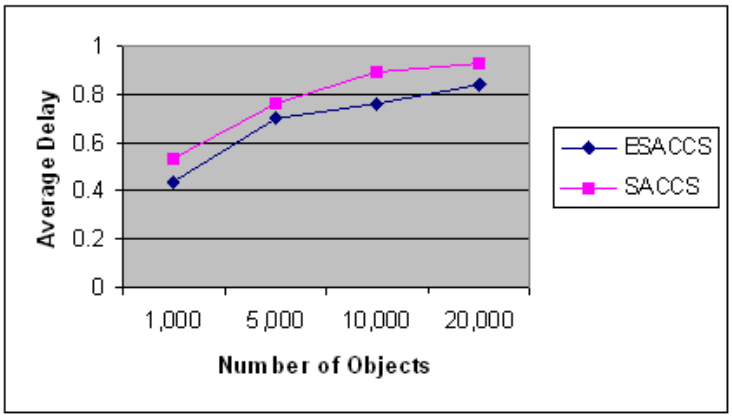

Figure 6. Average Delay vs. No. of Data Objects

We also checked out the performance of ESACCS vs. SACCS for different numbers of data objects in the system. ESACCS shows a better performance for different number of data objects in the system as demonstrated in Figure 6, which shows that when the number of objects in the system increases a better average delay is gained.

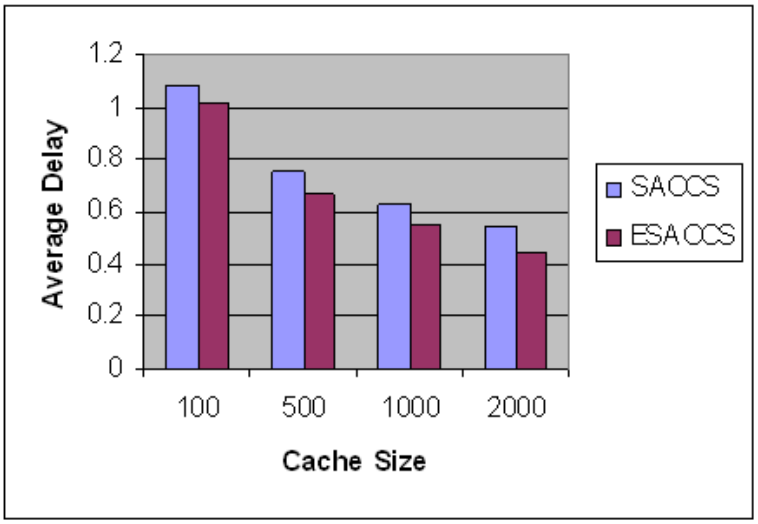

Figure 7. Average Delay vs. Mobile Unit Cache Size

Finally, we checked the performance of ESACCS in terms of average delay for different mobile unit cache sizes. ESACCS still performs better than SACCS as demonstrated in Figure 7.

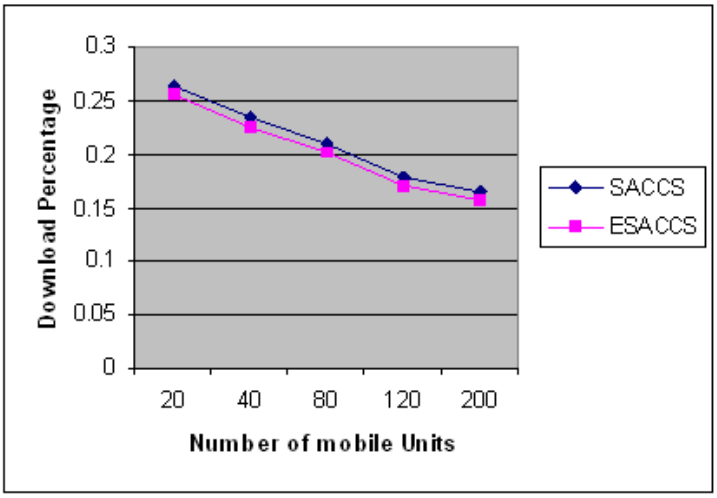

Figure 8. Download Percentage

4.3 Download Ratio

The download ratio is the ratio between the cache missed and the total requests. Each miss causes the mobile unit to request the data object which causes the base station to reply to that message by

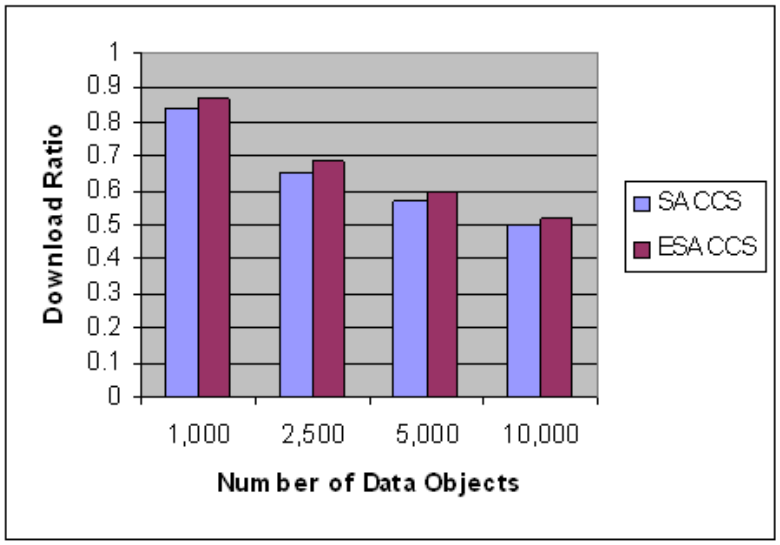

Figure 9. Downlink vs. Data Objects

a *Vdata* message. ESACCS reduces the downlink message because it increases the cache hits. This is demonstrated in Figure 8, which shows the download ratio for each of SACCS and ESACCS. From Figure 8, it can be easily seen that as the number of mobile units increases the average download ratio decreases because data objects will be more likely to be broadcasted previously. Also, ESACCS performs better than SACCS in terms of downlink channel traffic, especially because of the better way ESACCS deals with the frequent sleep-wakeup patterns. Further, ESACCS shows a better performance in terms of downlink traffic for different numbers of data objects in a system. Figure 9 shows the performance of ESACCS vs. SACCS.

5 Conclusions

In this paper, we proposed ESACCS as an efficient and effective algorithm for cache consistency; ESACCS inherits its superiority by maintaining all the characteristics of SACCS and adding new novel features that have positive effect on the overall performance. By reducing the number of requests from mobile units (uplink messages) to a base station, therefore, increasing the cache hit in the mobile units we improved the efficiency of the algorithm. Also, by trying to keep big-sized entries inside the mobile cache and deleting the smaller-sized entries, we reduced also the downlink traffic because these entries could be used later. Finally, ESACCS and SACCS have to ensure time synchronization between mobile units and the base station.

6 Acknowledgement

We thank Prof. Zhijun Wang for his assistance in providing the implementation code for the original SACCS scheme which we extended to implement our ESACCS scheme; this helped in having a more fair comparison between ESACCS and SACCS.

References

[1] R. Alonso, D. Barbara, and H. Garcia-Molina, "Data Caching Issues in an Information Retrieval System," *ACM Transaction on Database Systems,* Vol.15, No.3, Sept. 1990.

[2] D. Barabara and T. Imielinski, "Sleepers and Workaholics: Caching Strategies in Mobile Environments," *VLDB Journal,* Vol.4, No.4, pp.567-602, 1995.

[3] G. Cao, "A Scalable Low Latency Cache Invalidation Strategy for Mobile Environments," *Proceedings of ACM MOBICOM,* Boston, MA, pp.200-209, 2000.

[4] G. Cao, "On Improving the Performance of Cache Invalidation in Mobile Environments," *Mobile Networks and Applications,* Vol.7, No.4, pp.291-303, 2002.

[5] G. Cao, "A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments," *IEEE Transactions on Knowledge and Data Engineering,* Vol.15, No.5, pp.1251-1265, 2003.

[6] M.-S. Chen, K.-L. Wu, P. S. Yu, "Optimizing Index Allocation for Sequential Data Broadcasting in Wireless Mobile Computing," *IEEE Transactions on Knowledge and Data Engineering,* vol.15, no.1, pp.161-173, 2003.

[7] A. Elmagarmid, A. Helal, and C. Lee, "Scalable Cache Invalidation Algorithms for Mobile Data Access," *IEEE Transactions on Knowledge and Data Engineering,* Vol.15, No.6, pp.1498-1511, 2003.

[8] A. Elmagarmid, A. Heal and R. Alonso, "Bit-Sequences: An Adaptive Cache Invalidation Method in Mobile Client/Server Environments," *Mobile Networks and Applications,* Vol.2, No.2, pp.115-127, 1997.

[9] L. Feeney and M. Nilsson, "Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment," *Proceedings of IEEE INFOCOM,* 2001.

[10] C. C. F. Fong, J. C. S. Lui, and M. H. Wong, "Distributed Caching and Broadcast in a Wireless Mobile Computing Environment," *The Computer Journal,* Vol.42, No.6, pp.455-472, 1999.

[11] Q. Hu and D.K. Lee, "Cache Algorithms Based on Adaptive Invalidation Reports for Mobile Environments," *Cluster Computing,* Vol.1, No.1, pp.39-50, 1998.

[12] H. Kang and S. Lim, "Bandwidth-Conserving Cache Validation Schemes in a Mobile Database System," *Proceedings of the International Conference on Mobile Data Management,* pp.121-132, 2001.

[13] A. Kahol, S. Khurana, S.K.S. Gupta and P.K. Srimani, "A Strategy to Manage Cache Consistency in a Distributed Mobile Wireless Environment," *IEEE Transaction on Parallel and Distributed Systems,* Vol.12, No.7, pp.686-700, 2001.

[14] S. K. Lee, C. S. Hwang and H. C. Yu, "Supporting Transactional Cache Consistency in Mobile Database Systems," *Proceedings of ACM MobiDE,* Seattle, WA, 1999.

[15] D. Li and R. Cheriton, "Scalable Web Caching of Frequently Updated Objects Using Reliable Multicast," *Proceedings of USENIX Symposium on Internet Technologies and Systems,* pp.1-12, Oct. 1999.

[16] 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,* vol. 4, no. 5, pp. 271-288, 1996.

[17] A. Madhukar and R. Alhajj, "An Adaptive Energy Efficient Cache Invalidation Scheme for Mobile Databases," *Proceedings of ACM Annual Symposium on Applied Computing,* France, Apr. 2006.

[18] M. Maròti , B. Kusy , G. Simon and À. Lèdeczi, "The flooding time synchronization protocol," *Proceedings of the 2nd international conference on Embedded networked sensor systems,* Baltimore, MD, pp.39-49, 2004.

[19] W.-C. Peng and M.-S. Chen, "Query Processing in a Mobile Computing Environment: Exploiting the Features of Asymmetry," *IEEE Transactions on Knowledge and Data Engineering,* vol.17, no.7, pp.982-996, 2005.

[20] T.S. Rappaport, *Wireless Comm.: Principles and Practice,* Prentice Hall, 1996.

[21] K.L. Tan, "Organization of invalidation reports for energy-efficient cache invalidation in mobile environments," *Mobile Networks and Applications,* Vol.6, pp.279-290, 2001.

[22] H. Shen, M. Kumar, S. K. Das and Z. Wang, "Energy-Efficient Data Caching and Prefetching for Mobile Devices Based on Utility," *Proceedings of the International Parallel and Distributed Processing Symposium,* Santa Fe, New Mexico, April 2004.

[23] K. Tan, J. Cai and B. Ooi, "An Evaluation of Cache Invalidation Strategies in Wireless Environments," *IEEE Transactions on Parallel and Distributed Systems,* Vol.12, No.8, pp.789-807, 2001.

[24] Z. Wang, S.K. Das, H. Che and M. Kumar, "A Scalable Asynchronous Cache Consistency Scheme (SACCS) for mobile environments," *IEEE Transactions on Parallel and Distributed Systems,* Vol.15, no.11, pp.983-995, Nov. 2004.

[25] Z. Wang, M. Kumar, S.K. Das and H. Shen, "Dynamic cache consistency schemes for wireless cellular networks," *IEEE Transactions on Wireless Communications,* Vol.5, no.2, pp.366-376, 2006.

[26] K.L. Wu, P.S. Yu and M.S. Chen, "Energy-Efficient Caching for Wireless Mobile Computing," *Proceedings of IEEE International Conference on Data Engineering,* pp.336-345, 1996.

[27] M.K.H. Yeung, Y.-K. Kwok, "Wireless Cache Invalidation Schemes with Link Adaptation and Downlink Traffic," *IEEE Transactions on Mobile Computing,* vol.4, no.1, pp.68-83, 2005.

[28] J.C. Yuen, E. Chan, K. Lam and H.W. Leung, "Cache Invalidation Scheme for Mobile Computing Systems with Real-Time Data," *SIGMOD Record,* pp.34-39, Dec. 2000.

[29] L. Yin, G. Cao and Y. Cai, "A Generalized Target-Driven Cache Replacement Policy for Mobile Environments," *Proceedings of the International Symposium on Applications and the Internet (SAINT),* pp.14-21, Jan. 2003.

[30] H. Yu, L. Breslau and S. Shenker, "A Scalable Web Cache Consistency Architecture," *Proc. of ACM SIGCOMM,* pp.163-174, Aug. 1999.

[31] J. Zhang, R. Izmailov, D. Reininger and M. Ott, "Web Cache Framework: Analytical Models and Beyond," *Proceedings of IEEE Workshop on Internet Applications,* pp.132-141, 1999.