A Scalable Asynchronous Cache Consistency Scheme (SACCS) for Mobile Environments

Zhijun Wang, Sajal K. Das, Hao Che, and Mohan Kumar, *Senior Member*, *IEEE*

Abstract—In the literature, there exit two types of cache consistency maintenance algorithms for mobile computing environments: stateless and stateful. In a *stateless* approach, the server is unaware of the cache contents at a mobile user (MU). Even though stateless approaches employ simple database management schemes, they lack scalability and ability to support user disconnectedness and mobility. On the other hand, a *stateful* approach is scalable for large database systems at the cost of nontrivial overhead due to server database management. In this paper, we propose a novel algorithm, called *Scalable Asynchronous Cache Consistency Scheme* (SACCS), which inherits the positive features of both stateless and stateful approaches. SACCS provides a weak cache consistency for unreliable communication (e.g., wireless mobile) environments with small stale cache hit probability. It is also a highly scalable algorithm with minimum database management overhead. The properties are accomplished through the use of flag bits at the server cache (SC) and MU cache (MUC), an identifier (ID) in MUC for each entry after its invalidation, and estimated time-to-live (TTL) for each cached entry, as well as rendering of all valid entries of MUC to uncertain state when an MU wakes up. The stale cache hit probability is analyzed and also simulated under the Rayleigh fading model of error-prone wireless channels. Comprehensive simulation results show that the performance of SACCS is superior to those of other existing stateful and stateless algorithms in both single and multicell mobile environments.

Index Terms—Mobile environments, cache consistency, disconnection, bandwidth utilization, stale cache hit.

1 Introduction

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 [8] and 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 an MU make *cache consistency* 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, an efficient mobile data transmission architecture must carefully design its broadcast and cache management schemes to maximize *bandwidth utilization* and also to minimize *average query delay*. Additionally, such an architecture should be *scalable* to support large database systems as well as a large number of MUs.

Two types of cache consistency maintenance algorithms have been proposed for wireless mobile environments: stateless and stateful. In the *stateless* approach [1], [4], [9], [10], [13], [15], [17], [19], the server is unaware of the client's cache content. The server periodically broadcasts the data object's *invalidation report* (IR) to all the MUs. Even though stateless approaches employ simple database management schemes, their scalability and ability to support disconnectedness are poor. On the other hand, *stateful* approaches [11] are scalable, but incur significant overhead due to server database management. Existing caching schemes assume reliable communication between the mobile support station (MSS) and MUs for IR broadcast. However, any reliable communication mechanism requires acknowledgment from the MUs. After an IR is broadcast (or multicast for stateful schemes), the increased competition for uplink channel between the MSS and MUs will have an impact on the uplink queries and, hence, on the average access delay and MU's battery consumption. If an MU is disconnected during the IR broadcast, the server cannot get its acknowledgment back and must retransmit the IR because it does not know if the IR is lost or the MU is disconnected. Moreover, the existing schemes proposed in the literature do not study the possible inconsistency and performance loss due to wireless channel errors. Thus, there is a need for scalable and efficient algorithms for maintaining cache consistency in the error-prone wireless channels.

In this paper, we propose a novel algorithm, called the *Scalable Asynchronous Cache Consistency Scheme* (SACCS), which maintains cache consistency between the MSS cache, called the *server cache* (SC), and the MU caches (MUCs). SACCS is a highly scalable, efficient, and low complexity algorithm, and provides only *weak* cache consistency[1] with a small probability of stale cache hit under unreliable IR broadcast environments. The properties are accomplished through the use of flag bits at SC and MUC, an identifier (ID) in MUC for each entry after its invalidation, and estimated time-to-live (TTL) for each cached entry, as well

• *The authors are with the Center for Research in Wireless Mobility and Networking (CReWMaN), Department of Computer Science and Engineering, The University of Texas at Arlington, Arlington, TX 76019. E-mail: {zwang, das, hche, kumar}@cse.uta.edu.*

Manuscript received 2 Apr. 2003; revised 10 Nov. 2003; accepted 9 Apr. 2004. For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0048-0403.

1. As is well-known, IR-based schemes cannot provide *strong* cache consistency in the sense that an MU must confirm with the server on the cached data object upon each cache access.

as rendering of all valid entries of MUC to uncertain state when an MU wakes up. A preliminary version of this paper appeared in [16].

Each data object in an SC is associated with a flag bit, which is set when an MU retrieves the object from the MSS. The MSS broadcasts a data object's IR only if the corresponding flag bit is set. When an object is updated, its corresponding flag bit is reset. Hence, unlike synchronous periodic IR broadcast schemes (e.g., [1]), most of the unnecessary IRs can be avoided and, consequently, substantial bandwidth is saved. The SACCS algorithm maintains *only one extra flag bit* for each data object in the SC. In contrast, the Asynchronous Stateful (AS) algorithm [11] requires $O(MN)$ buffer space in the MSS to maintain states of MUCs, where M is the number of the MUs and N is the number of data objects in SC. Once a data object is invalidated, its entry in an MUC is set to *ID-only state*, i.e., the object is deleted but its ID is kept. All the valid MUC data entries are set to *uncertain* state (i.e., the validity of a cache entry is not clear) after an MU wakes up. This mechanism makes the handling of disconnection and mobility very simple. The MSS sets an estimated TTL for each data object based on its update history. This TTL is also cached together with the data object in MUC when acquired from the MSS. The cached data entry in the MUC is automatically set to an uncertain state when its TTL expires. This will protect a stale data object from being used for an arbitrarily long time due to *IR loss*, which means that a connected MU cannot correctly receive a broadcast IR. All entries in the uncertain or ID-only state can be used to identify useful broadcast messages for validation and triggering of data object downloading. Hence, all the MUs strongly cooperate in sharing the broadcast resource.

An analytical model is proposed to estimate the upper bound on the stale cache hit probability. A two-state Markov chain model is introduced to characterize the Rayleigh fading error-prone wireless channel. Simulation experiments are conducted to measure of SACCS in terms of stale cache hit probability and average access delay with various MU speeds.

Our comprehensive simulation results demonstrate that SACCS offers superior performance over the existing stateful and stateless algorithms in both single and multicell environments. For example, in a single-cell system with five types of MU access and sleep-wakeup patterns and 10 types of data object update frequencies, the average query delay for SACCS is less than 50 percent of AS and less than 88 percent of the Timestamp (TS) [1] scheme. In a seven-cell system, SACCS achieves more than 30 percent gain in terms of average access delay than AS in a wide range of MU roaming frequencies.

The rest of the paper is organized as follows: Section 2 gives a brief overview of the related work. A detailed description of the SACCS algorithm is presented in Section 3. In Section 4, the stale cache hit probability is analyzed under unreliable IR broadcast environments and simulated for Rayleigh fading channels. Section 5 presents simulation results and compares our algorithm with existing approaches. Section 6 concludes the paper.

2 Related Work

There exist two types of cache consistency maintenance approaches for wireless mobile environments: stateless [1], [4], [9], [10], [13], [15], [17], [19] and stateful [11]. In the following, we summarize existing algorithms for both approaches.

2.1 Stateless Approaches

In [1], three stateless algorithms have been proposed. They are Timestamps (TS), Amnesic Terminals (AT), and Signature (SIG), in which the MSS broadcasts IR messages every L seconds. An IR message includes all data object IDs updated during the past kL seconds, where k is a positive integer. The advantage of these algorithms is that an MSS does not maintain any state information about its 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 kL, all cache entries are deleted, thus leading to unnecessary bandwidth consumption, particularly if the data object is still valid.

In order to handle the long sleep-wakeup patterns, several algorithms have been proposed. For example, in the bit-sequence (BS) algorithm [10], all cache entries are deleted only when half or more 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 [17] 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 in [9] to broadcast different IRs based on update frequency, MU access, and sleep-wakeup patterns. In [19], 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 broadcast data objects. This makes the broadcast inefficient and sensitive to the number of MUs in the cell. The TS strategy is modified in [4] 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. A comprehensive performance evaluation of the existing stateless algorithms is studied in [15].

2.2 Stateful Approaches

Very few stateful cache consistency maintenance algorithms have been proposed for wireless mobile environments. In [11], an asynchronous stateful (AS) algorithm is proposed to maintain cache consistency in which an MSS records all retrieved data objects for each MU. When an MU first retrieves a data object after it wakes up, based on the MUC content record and sleep-wakeup time, the MSS sends an IR to that particular MU. Whenever an MSS receives an update

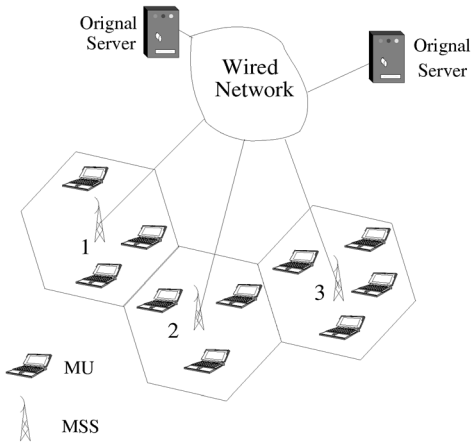


Fig. 1. Wireless data communication system architecture.

from the original server for each recorded data object, it immediately broadcasts that object's IR to MUs. The advantage of the AS scheme is that the MSS avoids unnecessary IR broadcast to MUs. Moreover, MUs can deal with any sleep-wakeup pattern without losing valid data objects. However, in order to maintain each MUC, the 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. More recently, a counter-based scheme is used in [5] to identify the hot data and save unnecessary IR traffic. Whenever an 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 broadcast (or multicast) is significant since the uplink data transmission requires much more battery power than downlink data transmission, especially for the acknowledgment of IR broadcast (or multicast) due to increased channel competition. On the other hand, the MSS cannot distinguish a disconnected MU from a connected one which did not receive an IR. Therefore, it is necessary to allow a maximum number of retransmissions.

3 Proposed Scheme SACCS

In this section, we propose a novel *Scalable Asynchronous Cache Consistency Scheme* (SACCS) for *read-only* MUCs.

3.1 System Architecture

The wireless mobile data communication system architecture is as illustrated in Fig. 1. We consider a system with a wired network, multiple original servers, mobile support stations (MSSs), and MUs. Each (hexagonal) cell has a base station or an MSS which is connected to the original servers through the wired network. Each MSS serves multiple MUs through wireless channels and has a server cache (SC) to store data objects. Each MU has a local cache (MUC) which stores some retrieved data objects. An MSS is also responsible for cache consistency between the original servers and MUCs. In this paper, we mainly focus on cache consistency between the SC and MUCs, and assume that the consistency between the SC and original servers is maintained in the wired network [6], [12] [18].

SACCS provides a *weak* cache consistency for MUC with a small stale cache hit probability (the MUs requiring *strong* cache consistency can check the cached data entry through MSS for each cache access). In the proposed scheme, an MU does not need to send an acknowledgment for each IR broadcast. Hence, an MU gets a stale cache hit for data entry x when the following two events occur: 1) the MU misses the IR of x and 2) the update time for x is smaller than TTL expiration time. The details of stale cache hit probability analysis are presented in Section 4.

SACCS provides cache consistency for dynamic public data objects such as news, traffic information, live sport scores, etc. All such objects are shared by MUs without security considerations. The IRs of these objects are broadcast to all MUs without acknowledgment. If reliable communication is used between the MSS and an MU retrieving the data object, the MU needs to send acknowledgment to the MSS. All other awake MUs can download the data object without acknowledgment if they can correctly receive it.

3.2 Data Structures and Message Formats

For each data object d_x with x as the unique identifier (ID), the data structures for SC and MUC are defined as follows:

In SC:

- (d_x, t_x, l_x, f_x): where t_x is the last update time for the data object, l_x is the estimated TTL, and f_x is the flag bit such that $f_x = 1$ indicates that the next IR will be broadcast.

In MUC:

- (d_x, ts_x, ll_x, s_x): where ts_x is the time stamp denoting the last updated time for the cached data object d_x, ll_x is an associated TTL, and s_x is a two-bit flag identifying four data entry states: 0, 1, 2, and 3, indicating valid d_x, uncertain d_x, uncertain d_x with a waiting query, and ID-only, respectively.

The communication messages are as defined in Table 1. Each cache entry has three states: *valid*, *uncertain*, and *ID-only*. Fig. 2 shows how the entry x changes from one state to another.

3.3 MUC Management

Since we mainly focus on cache consistency maintenance in this paper, we use the Least Recently Used (LRU) based replacement algorithm for the management of MUC. The impact of the cache replacement algorithms on SACCS is a subject of future study. In the adopted LRU-based replacement scheme, a newly cached data object or one that receives a hit is moved to the head of the cache list. When an object needs to be cached while the cache is full, data entries with $s_x \neq 2$ from the tail are deleted to make enough space for accommodating this new data object (the object with $s_x = 2$ must be kept because some requests are waiting for its confirmation). Any refreshed data objects from the uncertain or ID-only state are placed in their original location and again, if necessary, enough data entries from the tail are removed.

We limit the number of ID-only entries that can be used at any given instant to a certain value. This is to minimize frequent refreshment of old ID-only entries, which are likely to be replaced before they are requested. One may set this number close to the average number of data objects that can

TABLE 1
Communication Messages in SACCS

Name	Sender	Receiver	Comments
$Update(x, d_x', t_x')$	original servers	MSS	d_x has been updated to d_x' at time t_x'
$Vdata(x, d_x, l_x, t_x)$	MSS	MUs	broadcast valid data object d_x with update time at t_x and TTL = l_x
$IR(x)$	MSS	MUs	cached d_x is invalid
$Confirmation(x, l_x, t_x)$	MSS	MUs	d_x is valid if $ts_x = t_x$ and TTL = l_x
$Query(x)$	MUs	MSS	query for data object d_x
$Uncertain(x, ts_x)$	MUs	MSS	querying if d_x, which is in uncertain state with update time ts_x, is valid or not

be cached. For example, if a cache can hold C objects, then the number of ID-only entries is limited to C. Because the MU has limited cache size, during a broadcast, it caches only those objects whose ID-entries are already in the cache. The memory overhead of ID-only entries is insignificant since the size of a data object's ID is usually much smaller than the object itself.

3.4 Algorithm Description

We present two procedures for SACCS, namely, MSSMain() and MUMain(), as shown in Figs. 3 and 4, respectively. The MSS continuously executes the MSSMain() procedure to handle MUs' query and data object update. Each MU continuously executes the MUMain() procedure to handle its query and broadcast messages.

MSSMain(): When an MSS receives a *Query* message, it broadcasts the queried data object to all MUs. When it receives an *Uncertain* message, it checks if the uncertain data entry is valid or not by comparing the cached entry time stamps. If the uncertain data entry is valid, a confirmation message is broadcast; otherwise, a valid data entry is broadcast. Whenever an MSS receives an *Update* message from the original server, it updates its cache entry in the database and also estimates a new TTL for the entry based on the last TTL and this update interval. If the corresponding flag bit is set for the entry, an IR is immediately broadcast and the flag bit is reset.

MUMain(): When an MU has a request for a data object, it first searches its local cache. If the requested object is valid, the MU immediately answers the request using the cached object. If the cached data entry is in uncertain state, an *Uncertain* message is sent to the MSS. If the data entry is in an ID-only state or not in the local cache, a *Query* message is sent to MSS to retrieve the data object. When the MU receives a $Vdata$ message, if it has a query waiting for the data object, the MU answers the query and caches the data entry. If the MU has an uncertain entry in cache, the entry is refreshed. If the MU has an ID-only entry, the data entry is

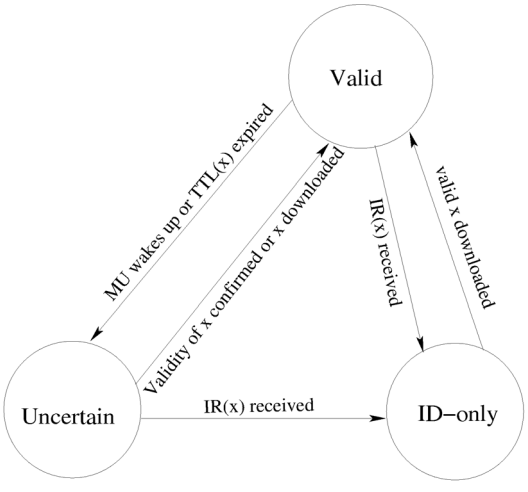


Fig. 2. State diagram of cache entry x.

MSSMain() {
For MSS receiving a message
IF (It is $Query(x)$ message)
fetch data entry x from the database
broadcast $Vdata(x,\ d_x,\ l_x,\ t_x)$ to all MUs
IF ($f_x == 0$)
set $f_x = 1$
IF (It is $Uncertain(x,\ ts_x)$ message)
fetch data entry x from the database
IF ($t_x\ ==\ ts_x$)
broadcast $Confirmation(x,\ l_x,\ t_x)$ to all MUs
ELSE
broadcast $Vdata(x,\ d_x,\ l_x,\ t_x)$ to all MUs
IF ($f_x == 0$)
set $f_x = 1$
IF (It is $Update(x,\ d_x',\ t_x')$ message from the original server)
update the database entry x, as $d_x = d_x'$ and $t_x = t_x'$
update the TTL l_x
IF ($f_x\ ==\ 1$)
broadcast $IR(x)$ to all MUs and reset $f_x = 0$
}

Fig. 3. MSSMain procedure.

```
MUMain() {
  For MU receiving a message
    IF (It is a Request message for d_x)
        IF ( d_x is valid in the cache)
          answer the request with cached data object d_x
          move the entry into the head of the cache list
        ELSE IF (d_x is in uncertain state )
          send Uncertain(x, ts_x) message to the MSS
          add ID, i.e., x, to query waiting list
          set s_x = 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 MSS
          remove the entry x in the cache
          add x to query waiting list
        ELSE
          send Query(x) message to the MSS
          add x to query waiting list
    IF (It is a Vdata(x, d_x, l_x, t_x) message)
        IF (x is in query waiting list)
          answer the request with d_x
          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 d_x to the original entry location in the cache
          ELSE IF (x is an uncertain entry in the cache)
            IF (ts_x < t_x)
              download d_x to the original entry location in the cache
              set ts_x = t_x, ll_x = l_x and s_x = 0
            ELSE
              set s_x = 0
    IF (It is an IR(x) message)
        IF (entry x is valid or uncertain in the cache)
          delete d_x and set s_x = 3
    IF (It is a Confirmation(x, l_x, t_x) message)
        IF (x is an uncertain entry in the cache)
          IF (ts_x == t_x)
              set s_x = 0 and ll_x = l_x
              IF (x is in query waiting list)
                answer the request with d_x
          ELSE
              delete d_x and set s_x = 3
    IF (MU wakes up from the sleep state)
        set all valid (s_x = 0) entries into uncertain state (s_x = 1)
    IF (TTL expires for entry x)
        set the valid entry x into uncertain state (s_x = 1)
  }
```

Fig. 4. MUMain procedure.

downloaded. Upon receiving an IR message, the MU sets the entry into ID-only state. When a *Confirmation* message is received, if the MU has a corresponding uncertain entry in the cache, it refreshes the entry by comparing the corresponding time stamps. When a TTL is expired for an entry, it is set to uncertain state.

3.5 Illustration of MUC Management

Let us illustrate the MUC management with the help of a simple example as shown in Fig. 5. Assume there are n entries $(1, 2, \ldots, n)$ in the cache list and m queries $(w_1, w_2, \ldots, w_m)$ in the query waiting list. In particular, we illustrate various actions of cache management at an MU such as the ones upon receipt of a request, a valid data object, a confirmation message, and an invalidation message.

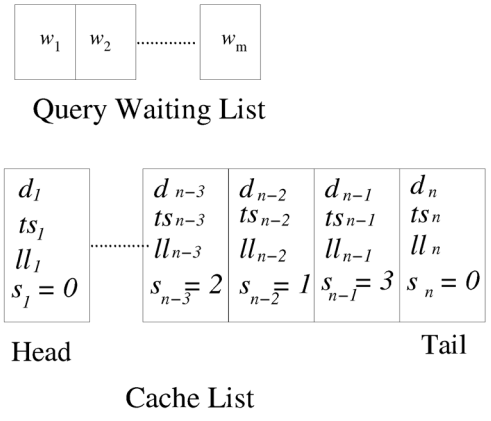


Fig. 5. MUC management scheme.

1. **Request** for d_x:
   a. If $s_x = 0$, such as for $x = n$, the MU answers the request immediately using d_x and moves the entry x to the head of cache list; in other words, (d_x, ts_x, ll_x, s_x) is inserted to the head of the cache list.
   b. If $s_x = 1$, such as for $x = n - 2$, the MU sends an $Uncertain(x, ts_x)$ message to the MSS, sets $s_x = 2$, moves the entry x to the cache list head, and adds x to the query waiting list.
   c. If $s_x = 3$, such as for $x = n - 1$, the MU deletes the entry x from the cache list, sends $Query(x)$ message to the MSS, and adds x to the query waiting list.
   d. If x is not in the cache list, the MU sends $Query(x)$ message to the MSS and adds x to the waiting list.
2. $\mathbf{Vdata}(x, d_x, l_x, t_x)$:
   a. If x is on the waiting list, the MU answers the request by downloading the data object d_x and adds the entry x at the head of the cache list. If the MUC has an uncertain entry x (e.g., $x = n - 3$), the entry is deleted.
   b. If x is not in the query waiting list but in the cache list, for example, $x = n - 1$, the MU downloads d_x at the location of entry $n - 1$ and sets $ts_{n-1} = t_x$, $ll_{n-1} = l_x$, and $s_{n-1} = 0$. Assuming $x = n - 2$, the MU sets $s_{n-2} = 0$ when $ts_{n-2} = t_x$ or downloads d_x instead of d_{n-2} and sets $s_{n-2} = 0$, $ts_{n-2} = t_x$, and $ll_{n-2} = l_x$ when $ts_{n-2} < t_x$.
   c. If x is not in the query waiting list and cache list, the MU does nothing on x.

   Each time before the MU adds or downloads a data object, if the free buffer space in the cache is not enough, the entries n, $n - 1$, $n - 2$ from the tail such that $s_x \neq 2$ are deleted.
3. $\mathbf{IR}(x)$:
   a. If $s_x < 3$, the MU deletes d_x and sets $s_x = 3$. If the total number of ID-only entries is over a maximum limit, the last ID-only entry is deleted from the cache.
   b. If x is the ID-only entry or not in the cache list, the MU does nothing on it.

4. **Confirmation**(x, l_x, t_x):
   a. If $x = n-3$ and $ts_{n-3} = t_x$, answer the request with the cached data object d_{n-3}, set $s_x = 0$, and move the entry $n-3$ to the head of cache list. Else if $ts_{n-3} < t_x$, the request is still in query waiting list; so, delete d_{n-3} from the cache.
   b. If $x = n-2$, the MU checks the timestamp ts_{n-2}; if $ts_{n-2} = t_x$, set $s_{n-2} = 0$ and $ll_{n-2} = l_x$, else delete d_{n-2} from the entry and set $s_{n-2} = 3$.
   c. Otherwise, the MU does nothing on it.

3.6 MUC Consistency Maintenance

Let us explain how the SACCS maintains consistency between an SC and MUCs. We first assume an error-free channel in which no IR is lost by an awake MU (i.e., the MU is connected to the MSS). Then, we discuss how to deal with the IR loss situation.

For each cached data object, SACCS uses a single flag bit, f_x, in SC in order to maintain the consistency between the SC and MUC. When d_x is retrieved by an MU, f_x is set indicating that a valid copy of d_x may be available in an MUC. If and when the MSS receives an updated d_x, it broadcasts an $IR(x)$ and resets f_x. This action implies there are no valid copies of d_x in any MUC. Furthermore, while $f_x = 0$, subsequent updates do not entail broadcast of $IR(x)$. The flag f_x is set again when the MSS services a retrieval (including request and confirmation) for d_x by an MU.

In mobile environments, an MUC belongs to one of two states: awake or sleep. If an MU is *awake* at the time of $IR(x)$ broadcast, the copy of d_x is invalidated and an ID-only entry is maintained by the MU. The data objects of an MU in the *sleep* state are unaffected until it wakes up. When an MU wakes up, it sets all cached valid data objects (including d_x) into the uncertain state. Consequently, MUs and their cached objects are unaffected if $IR(x)$ broadcast occurs during their sleep times.

A TTL is associated with each cache entry. When the TTL of a cache entry expires, an MU automatically sets it into uncertain state. There are some probabilities for an MU to get a stale cache hit in the case of IR loss. We give a detailed analysis on this in Section 4.

3.7 Efficiency and Cooperation

As mentioned earlier, a good cache consistency maintenance algorithm must be scalable and efficient in terms of the database size and the number of MUs. We claim that the proposed scheme SACCS can handle large and fast updating data systems because the MSS has some knowledge of MUC. Only data entries which have flag bits set result in the broadcast of IRs when data objects are updated. Consequently, the IR broadcast frequency is the *minimum* of the uplink query/confirmation frequency and the data object update frequency. In this way, the broadcast channel bandwidth consumption for IRs is minimized. Besides IR traffic, all other traffic in SACCS is also minimized due to the strong cooperation among the MUCs. This is specifically due to the introduction of the uncertain state and the ID-only state for the MUCs. The retrieval of a data object, d_x, from the MSS issued by any given MU brings the entries of x in the uncertain or ID-only state in all the awake MUCs to a valid state. Moreover, a single uplink confirmation for entry x causes all entries of x in the uncertain state for all the awake MUCs to be in either valid or ID-only state. The addition of the uncertain state also allows an MUC to keep all the valid data objects when it wakes up after an arbitrary sleep time. In contrast, for the AS and TS algorithms, all the invalidated data objects are completely deleted from the MUC. This allows little cooperation among the MUs, resulting in a dramatic increase of traffic volume between the MSS and the MUs as the number of MUs increases (see Section 5). Although the scalability of the TS scheme can be improved by retaining the invalid data objects [4], the cache efficiency is reduced by having to keep in the MUC the invalid data objects, rather than IDs as is the case in our SACCS approach.

In contrast to the AS scheme which requires $O(MN)$ buffer space in the MSS to keep all the MUCs, the SACCS requires only *one bit* per data object in the SC, thus indicating if the IR broadcast is required when the data object is updated. Moreover, the database management overhead is minimal requiring only a single bit check and set/reset.

The TTL expiration of a valid cache entry is checked only when its data object is accessed or available on the channel; otherwise, we do not care about the entry status. When the data object of a valid entry is accessed or available on the channel, its TTL is first checked. If the TTL has not expired, the entry is treated as valid. Otherwise, it is handled as an uncertain entry. The cost of each TTL expiration check is the execution of an addition (namely, $ts_x + ll_x$) and a comparison (e.g., comparing $ts_x + ll_x$ with the current time). The cost of these two operations is not much as compared with the cost of a data object query and/or data object download. Thus, the cost of execution on the TTL expiration is not significant.

3.8 Mobility

One advantage of the AS scheme is the efficient treatment of mobility. SACCS can also handle MUs' mobility effectively. When an MU roams, it is either in awake or in sleep state. If it is in the sleep state, there is no extra action in SACCS as well as AS. In SACCS, an MU sets all its valid cache entries to uncertain state after it wakes up. Hence, for caching purposes, upon wake up, the mobility and location of an MU is irrelevant. In AS, when a roamed MU wakes up, its first query will be sent to the new MSS, which can retrieve its cache state from the previous MSS, which maintains the cache consistency.

If a roaming MU is awake, SACCS treats it as if it just woke up from the sleep state, i.e., all valid data entries are set to an uncertain state. The consistency is guaranteed with this approach and all valid data objects are retained. Also, SACCS is a simple scheme in the sense that it is transparent to the MSSs involved. But, for AS, there exist two situations:

1. An MU is sending a message to the MSS when it roams. In this situation, the MSS can handle its handoff and transfer its cache state to the new MSS. So, there is no extra action for cache consistency maintenance.
2. An MU is not sending a message to the MSS when it roams. A wakeup event is forced in this case. When its next query comes, it sends a message to the MSS, which includes the queried data object and cache state request. The cache state request contains the previous MSS and roaming time. After the MSS gets the cache state request, the MSS retrieves its state from the previous MSS and sends the data object IRs (which are updated after the MU roamed) to that particular MU. Thus, the cache consistency is maintained.

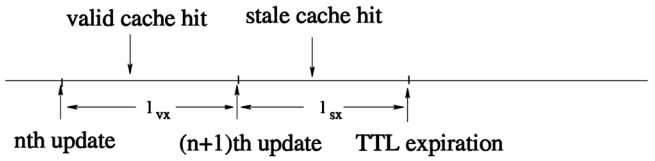


Fig. 6. Data object update process.

3.9 Failure Handling

Handling of MU failures is the same as handling MU disconnections. If an MU recovers from a failure, it sets all cached valid data entries into an uncertain state. SACCS treats this situation as a wakeup from the sleep state. The handling of server failures is also simple. When an MSS server is back after a failure, it simply broadcasts a server-down message to all MUs which, in turn, set all valid data entries into the uncertain state. The MUs in sleep state miss the server-down message, but after they wake up, all valid entries are automatically set to uncertain state. Thus, the cache consistency is maintained even if some cached data objects are updated during the MSS server failure. This is because the validation of any cached data object must be refreshed or checked before its usage. Finally, all valid data objects are retained after a server failure due to the fact that they are set to the uncertain state, thus avoiding extra data object download.

4 Stale Cache Hit Probability

In this section, the *stale cache hit probability* as a function of IR loss probability is analyzed and simulated in Rayleigh fading wireless channels.

4.1 Analytical Modeling

Our analytical model derives an upper bound of the *stale cache hit probability* in SACCS under different channel conditions. The following assumptions are made in our model:

1. The update process for a data object d_x follows a Poisson distribution with average update rate μ_x.
2. A data object d_x is associated with a TTL, l_x. The TTL equals the average update interval time T_{u_x}. That is, $l_x = T_{u_x} = 1/\mu_x$.
3. An awake MU has a probability, P_{mIR}, to miss an IR broadcast.

Let P_{s_x} be the stale cache hit probability for data object d_x. Fig. 6 shows an object update process. After the nth update of d_x, the $(n+1)$th update may occur either before or after the TTL expiration. When an MU misses the $(n+1)$th IR, there is no stale hit if the update is after the expiration of the TTL because the data entry is automatically set to an uncertain state. If the update is before the TTL expiration and the MU accesses the cached data object d_x between the $(n+1)$th update time and the TTL expiration time, it gets stale cache hits. Let P_{short_x} be the probability of the next update interval which is before the TTL expiration, l_{v_x} be the average of all update intervals which are earlier than the TTL expiration, and l_{s_x} be the interval between the $(n+1)$th update and estimated TTL expiration. These terms can be calculated as:

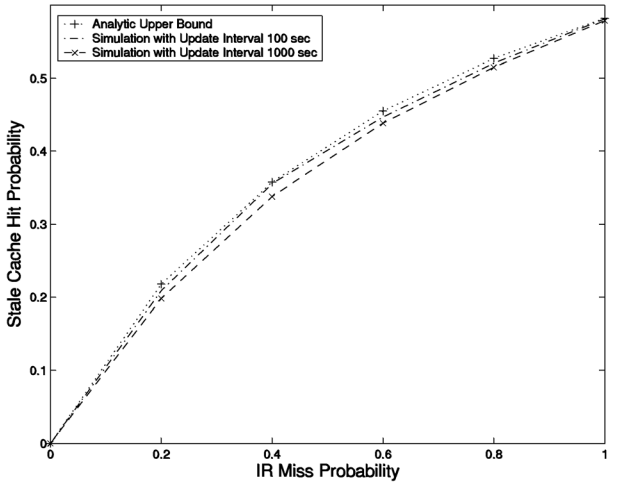


Fig. 7. Upper bound on stale cache hit probability versus IR miss probability.

$$P_{short_x} = \int_0^{l_x} \mu_x e^{-\mu_x t} dt = 1 - \frac{1}{e} \tag{1}$$

$$l_{v_x} = \frac{\int_0^{l_x} \mu_x t e^{-\mu_x t} dt}{P_{short_x}} = \left(\frac{e-2}{e-1}\right) l_x \tag{2}$$

$$l_{s_x} = l_{x-l_{v_x}} = \left(\frac{1}{e-1}\right) l_x. \tag{3}$$

Assume an MU is always awake and misses all IRs. Then, the stale cache hit for data object d_x is given by:

$$P_{s_x} = \frac{l_{s_x}}{l_x} = \frac{1}{e-1}. \tag{4}$$

If an MU has IR broadcast miss probability P_{mIR}, then the stale cache hit probability is:

$$P_{s_x} = \frac{l_{s_x} P_{mIR}}{l_x P_{mIR} + l_{v_x}(1 - P_{mIR})} = \frac{P_{mIR}}{P_{mIR} + e - 2}. \tag{5}$$

Equation (5) implies that the stale cache hit probability is only dependent on P_{mIR}. Thus, we conclude that the stale cache hit probability for any data object d_x of an MU is the same, i.e., $P_s = P_{sx}$. Fig. 7 shows the stale cache hit probability for various IR miss probabilities. The results show that the stale cache hit probability is about 10 percent (21 percent) for an MU with 10 percent (20 percent) IR miss probability, independent of the data object update frequency.

The above analytical model assumes that an MU is always awake. If we consider the sleep-wakeup event for an MU, then the stale hit probability is reduced because, when the MU wakes up from the sleep state, all valid entries will be checked prior to their usage. The stale cache hit probability for a frequently disconnected MU is much smaller than the upper bound in (5). A system with 100 MUs is simulated. In the simulation, each MU has a sleep-wakeup period randomly picked from the set of values (600, 1,200, 1,800, 2,400, 3,000) sec, the sleep ratio picked from (0.2, 0.35, 0.5, 0.65, 0.8), and the request arrival rate from (1/20, 1/40, 1/60, 1/80, 1/100). The detailed simulation setup is described in the Section 5. Fig. 8 shows the simulation results for the stale cache hit probability of the system, which is reduced significantly as compared to that of MUs which are always awake. For example, when the IR miss probability is 10 percent, the stale hit probability is only about 4 percent; and for IR miss probability of 20 percent,

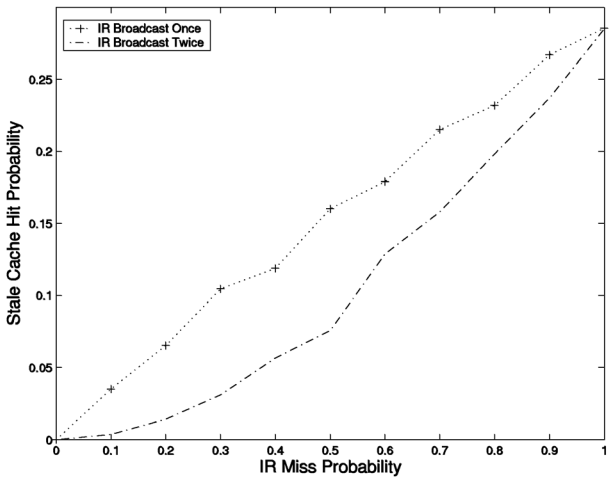


Fig. 8. Simulation of stale cache hit probability versus IR miss probability.

the stale hit probability is about 7 percent. In order to reduce the stale hit probability, we can broadcast the IR multiple times when an object is updated earlier than its TTL expiration. Fig. 8 shows the stale cache hit probability for broadcasting each IR twice if its update is earlier than the TTL expiration. The results indicate that, if the IR miss probability is less than 40 percent, the stale cache hit probability is less than 5 percent. These results demonstrate that SACCS can provide very small stale cache hit probability by broadcasting IR multiple times when the update is earlier than TTL expiration.

4.2 Stale Cache Hit Probability for Rayleigh Fading Channels

The stale cache hit probability of SACCS with Rayleigh-fading model for wireless channel is simulated in this section. We consider a Rayleigh-fading channel between the MSS and an MU be modeled by a two-state (*good* and *bad*) Markov chain, as shown in Fig. 9. An MU can successfully receive packets from the MSS if the channel between them is in *good* state and lose packets if the channel is in *bad* state. The channel states between the MSS and different MUs are independent. According to [14], a channel can be assumed to be effectively constant during a period of $T_c \approx 9c/(16\pi f_c V)$, where c is the speed of light, V is the MU speed (along the wave path), and f_c is the carrier frequency. The channel in the next T_c period has a probability of transiting to the other state. The state transition probabilities such as P_{gb} (from *good* to *bad* state) and P_{bg} (from *bad* to *good* state) are determined by V and the fading margin F, that is, the maximum fading noise of received signal without system performance falling below a specified value [7]. The probability of a channel in *bad* state

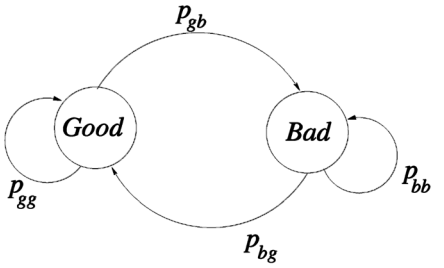


Fig. 9. Two-state Markov chain describing the *good-bad* channel model.

(P_B) is dependent on F [7]. Table 2 shows P_{gb}, P_{bg}, and P_B values for a carrier frequency $f_c = 900\ MHz$ with various values of V and F. Note that $P_{gg} = 1 - P_{gb}$ is the transition probability from *good* to *good* state and $P_{bb} = 1 - P_{bg}$ is the transition probability from *bad* to *bad* state. The packet duration is set to $4\ ms$ (millisecond) because $T_c \geq 4\ ms$ for $f_c = 900\ MHz$ and $V \leq 50\ kmph$ (kilometer per hour).

From Table 2, we note that the slower the MU speed, the smaller the state transition probability. This means that the channel condition is more stable for a slow moving MU than a fast moving one.

We study the performance by simulation of SACCS under Rayleigh fading channel model. In this simulation, an MU requesting a data object uses reliable communication and all other awake MUs can passively download the data objects if they can successfully receive all the packets of a data object. Each packet duration is set to 4 ms. In this period of time, 100 *bytes* (i.e., packet size) can be transmitted for a 200 $Kbps$ (kilobits per second) channel. A Go-Back-N ARQ (Automatically Repeat reQuest) scheme is used between the MSS and the MU for retrieving data objects. The number of MUs is set to 50. Each MU's query access and sleep-wakeup pattern are set the same as in the previous section.

Table 3 shows the performance of SACCS with the MU speed. We conclude that the stale cache hit probability depends only on the fading margin. A smaller fading margin leads to a larger probability of a channel in *bad* state, resulting in a larger IR loss probability. For $F = 10\ dB$, $P_B = 0.095$ (note that $P_{mIR} = P_B$), the stale cache hit probability is about 4 percent; while for $F = 15\ dB$ (i.e., $P_B = 0.065$), the stale cache hit probability is about 2 percent. These results, similar to that of Fig. 8, demonstrate that the proposed SACCS scheme can provide small stale cache hit probability in the error-prone wireless environments. The average access delay and the data download ratio (defined as the number of broadcast data objects divided by the number of queries) increase as the MU speed increases. This is because a slow-moving MU has a more stable channel and stands a better chance of sharing the broadcast data objects. A larger F results in a smaller P_B, thus a smaller number of retransmis-

TABLE 2
State Transition Probabilities and *Bad* State Probability versus Different V and F

$V(kmph)$	5		10		20		40		For all V
	P_{gb}	P_{bg}	P_{gb}	P_{bg}	P_{gb}	P_{bg}	P_{gb}	P_{bg}	P_B
$F = 10\ dB$	0.0132	0.1252	0.0261	0.2480	0.0498	0.4735	0.0786	0.7470	0.0951
$F = 15\ dB$	0.0107	0.1557	0.0211	0.3066	0.0381	0.5664	0.0564	0.8176	0.0645

TABLE 3
Performance of SACCS versus Different MU Speed

Fading Margin	Performance Metrics	Speed $V(kmph)$			
		5	10	20	40
$F = 10\ dB$	Stale Cache Hit Probability	0.0358	0.0374	0.0367	0.0371
	Average Access Delay (sec)	1.387	1.595	1.650	1.762
	Data Download Ratio	0.844	0.857	0.867	0.873
$F = 15\ dB$	Stale Cache Hit Probability	0.0236	0.0230	0.0223	0.0228
	Average Access Delay (sec)	1.368	1.475	1.514	1.586
	Data Download Ratio	0.843	0.855	0.866	0.871

sions and, ultimately, a smaller average access delay.

5 Performance Evaluation by Simulation

The performance of SACCS is evaluated and compared with the timestamp (TS) and asynchronous stateful (AS) schemes. Recall that TS is a popular stateless scheme and has been widely compared with other algorithms. For meaningful comparison, we extend TS with some advanced features of SACCS, such as: 1) introduction of uncertain state for an MU keeping its valid data entry after long disconnection, 2) use of ID-only state in MUC to trigger data object download, and 3) use of flag bits in SC to reduce the IR broadcast traffic. We call a TS with these additional features an *extended TS* (ETS). The AS scheme is also used for performance comparison because it is one of the few stateful schemes that handles the MU disconnection and mobility better. For SACCS, an IR of a data object is broadcast twice if its update time is earlier than the TTL expiration to reduce the stale hit probability. The other error recovery costs, such as data retransmission, are ignored in all three algorithms. The TTL, l_x, of a data object is dynamically calculated as: $l_x = l_x * 0.5 + l_{interval} * 0.5$, where $l_{interval}$ is the current update interval for the data object.

We present the simulation results with single cell as well as multicell environments.

5.1 Single Cell Environment

We consider a single cell system with one SC and multiple MUs with identical cache size. The parameters are defined as in Table 4.

Each MU's request process and the data object update process are assumed to follow Poisson distributions. The sleep-wakeup process is modeled as a two-state Markov chain (i.e., *sleep* and *awake*). The state transition probability from awake to sleep state is $\alpha = 1/(1-s)T_p$ and that from sleep to awake state is $\beta = 1/sT_p$.

In the simulations, we use two channels with bandwidth W_d and W_u for downlink and uplink data transmission, respectively. In the uplink channel, all messages are buffered as FIFO (first in first out) queue. In the downlink, there are two FIFO queues, one having higher priority than the other. The IR messages are buffered in the higher priority queue. All other messages are buffered in the lower priority queue; this queue can be scheduled only if the higher priority queue is empty. All requests are ignored when an MU is in the sleep state. When a requested data object is available at an MUC, the average query delay (D) is counted as 0. We consider *Zipf-like* distribution for MU access pattern [3], [20] such that the access probability (p_x) for data object d_x is proportional to its popularity rank, $rank(x)$. More specifically, $p_x = const/rank(x)^z$, where $const$ is the normalization constant and z is the Zipf coefficient.

In the following, we present the performance comparison of the proposed SACCS with AS, TS, and ETS in terms of such metrics as D and UPQ for three different cases. The average waiting time (i.e., half of the IR broadcast period, $L/2$) is removed from D for ETS to make a better comparison with SACCS and AS in all figures. As shown in Section 4, for the same sleep-wakeup pattern, the stale cache hit probability is less than 5 percent if the IR miss rate is smaller than 40 percent, hence the stale hit probability is not presented as a metric in the result. In each case, $b_u = b_d = 20$ *bytes* for both SACCS and AS; and $b_u = b_d = 10$ *bytes* for TS and ETS. The bandwidth is set as $W_d = 200\ Kbps$ and $W_u = 1\ Kbps$. The other parameters may be changed in some cases. Some default values are set as: $N = 10,000$, $M = 100$, $C = 5$ MBytes, $z = 0.9$, $L = 20$ *sec*, and $wsz = 5$.

In all cases, we consider a system with 10 types of data objects. The data object update rate (T_u), size, and percentage of each type of objects over the total objects are shown in Table 5. The chosen parameter values are based on the understanding that a faster updated object

TABLE 4
Parameter Definition

M	number of MUs in the system.
N	number of data objects in the system.
C	cache size for MU $(bytes)$.
λ	average arrival rate of request for an MU.
T_u	average update time interval for a data object (sec).
T_p	period for a sleep-wakeup cycle of an MU (sec).
s	ratio of the sleep time to the sleep-wakeup period for an MU.
b_o	data object size $(bytes)$.
b_u	uplink message size $(bytes)$.
b_d	downlink invalidation or confirmation message size $(bytes)$.
D	average query delay, i.e., the interval between the time request is issued and the time the result is received by the application (sec).
UPQ	uplink per query, defined as the total number of queries through uplink channel divided by the total number of queries.
L	invalidation broadcast period for TS scheme (sec).
wsz	broadcast window size for TS scheme.

TABLE 5
Ten Types of Data Objects in Database

$Data\ Type$	1	2	3	4	5	6	7	8	9	10
$Size(Bytes)$	$1K$	$5K$	$10K$	$15K$	$20K$	$25K$	$30K$	$35K$	$40K$	$45K$
$T_u(sec)$	50	100	200	400	800	1600	3200	64000	12800	25600
$Percentage(\%)$	5	5	10	10	20	20	10	10	5	5

usually has smaller size. The average data object size is about 25 Kbytes, which is based on the Internet measurements [2].

The MUs may be different from one another in terms of λ, s, T_p, and T_r. These parameters for each MU can take values from the corresponding given sets. Each value has equal probability of being chosen for each MU. The set of values for arrival rate λ is (1/20, 1/40, 1/60, 1/80, 1/100), for sleep ratio s is (0.2, 0.35, 0.5, 0.65, 0.8), and for sleep-wakeup period time T_p is (600, 1,200, 1,800, 2,400, 3,000) sec.

The query patterns for each MU are assumed to follow Zipf-like distribution. The access popularity ranking for each MU is shifted by a random number between 0 and 99. For example, an MU picks up a shift number 50, which means the MU has the highest access popularity for data object number 51. The popularity decreases from 51 to N, then from 1 to 50. The data object 50 has the lowest access popularity.

Case 1: Effect of the number of MUs. In this case, we study the impact of three features of SACCS on the system performance as compared with TS, ETS, and AS. Let SACCS-nfg stand for SACCS without flag bit set in SC; let SACCS-nid be SACCS without ID in MUC; and let SACCS-nuc be SACCS without uncertain state in MUC. Recall that ETS is an extension of TS with all SACCS features. We use tables to present the results due to their wide range of values or small changes (as in Table 7).

Tables 6 and 7 present the D and UPQ values for a varying number (M) of MUs. For all algorithms, the average delay (D) increases as the number of MUs increases. The SACCS-based algorithms have much shorter D compared with AS, TS, and ETS, especially when $M > 100$. Moreover, the turning off flag bit in SC has the least impact on D. This is due to the fact that the IR message is very small compared to the data object size. SACCS has about 10 percent less delay than SACCS-nfg when $M = 120$. Turning off the ID or uncertain state makes SACCS less scalable and leads to a larger D as M increases. This is because the ID-only and uncertain states allow MUs to share the broadcast data objects, thus saving the downlink bandwidth and, consequently, reducing the access delay. AS has smaller D than TS, but it does not scale as much as ETS, which allows strong cooperation among MUs because ETS incorporates all three features of SACCS.

For SACCS-based algorithms and ETS, the UPQ metric decreases as M increases. But, for AS and TS, it is almost constant. This is due to the cooperation among MUs in SACCS-based algorithms and ETS. Note that SACCS has the least UPQ, while turning off ID has the largest increase on the UPQ.

The simulation studies validate our initial claims, namely, ID-only entry and uncertain state in MUC are critical features of SACCS; and use of flag bit in SC reduces IR traffic. Thus, ETS performs better than TS and, hence, we will use ETS instead of TS in the following cases.

Case 2: Effect of Database Size. Figs. 10 and 11 present the simulation results showing the effects of database size. For ETS, the average query waiting time ($L/2 = 10\ sec$) is not counted. In other words, only the queue delay and transmission time are counted for ETS in all the following cases. SACCS outperforms AS and ETS in both D and UPQ because SACCS avoids all the unnecessary IR traffic while retaining all the valid data objects in MUCs. As expected, with an increased number of data objects, the performance metrics also increase for all three algorithms, but SACCS has much smaller D than AS and ETS. Additionally, the average gain (in terms of D) of SACCS over AS and ETS is more than 50 percent throughout the range of database sizes. The UPQ of SACCS is a little bit lower than that of ETS and about 6 percent less than that of AS.

TABLE 6
Average Access Delay D (Sec) versus the Number of MUs

M	20	40	60	80	100	120
$SACCS$	0.907	1.006	1.129	1.329	1.836	2.999
$SACCS\text{-}nfg$	0.912	1.021	1.153	1.372	1.932	3.293
$SACCS\text{-}nid$	0.968	1.129	1.346	1.736	3.128	13.400
$SACCS\text{-}nuc$	1.044	1.149	1.391	1.693	2.674	10.033
AS	0.969	1.139	1.376	1.824	3.619	18.429
TS	13.242	14.444	15.818	17.585	27.244	125.164
ETS	12.774	13.779	14.674	15.488	16.587	18.767

TABLE 7
The Uplink per Query (UPQ) versus the Number of MUs

M	20	40	60	80	100	120
$SACCS$	0.902	0.894	0.874	0.866	0.854	0.838
$SACCS\text{-}nfg$	0.904	0.895	0.876	0.868	0.856	0.839
$SACCS\text{-}nid$	0.927	0.926	0.927	0.925	0.920	0.917
$SACCS\text{-}nuc$	0.909	0.896	0.884	0.871	0.862	0.850
AS	0.9000	0.904	0.906	0.910	0.909	0.901
TS	0.926	0.925	0.929	0.930	0.930	0.922
ETS	0.914	0.892	0.889	0.872	0.861	0.847

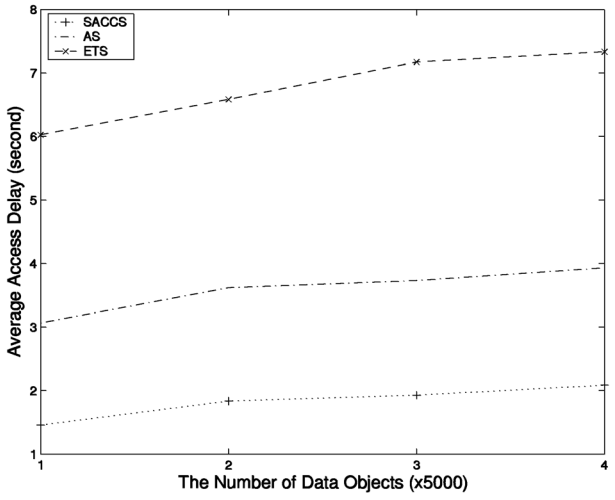


Fig. 10. Average access delay versus number of data objects.

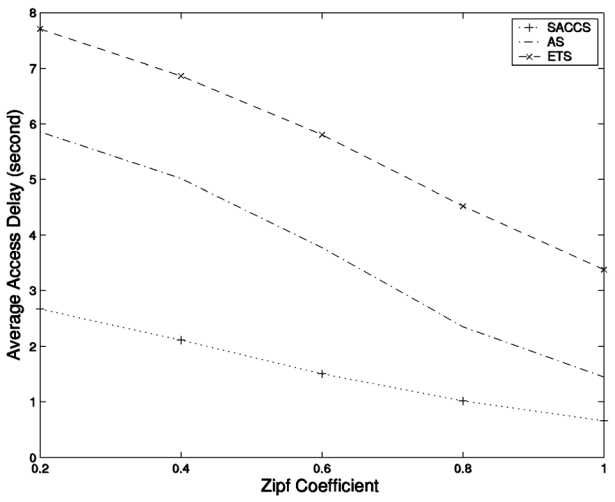


Fig. 12. Average access delay versus Zipf coefficient.

Case 3: Effect of Zipf Coefficient. In this case, we study the effect of the Zipf coefficient, z, on the system performance. Here, we choose a small database with $N = 1,000$ objects. This is because, for small Zipf coefficient, the access frequencies for different data objects are very close to each other. Hence, using a large database size results in very few cache hits, which makes the comparison meaningless.

From Figs. 12 and 13, we conclude that SACCS has much smaller D than both AS and ETS. The average gain is more than 50 percent over the other two algorithms. AS has the largest UPQ, while SACCS has the lowest UPQ when $z > 0.6$ and a little bit more than ETS when $z < 0.6$. All three schemes perform better as z increases because the data accesses are more concentrated for larger z, thus increasing the cache hit ratio and then reducing access delay.

5.2 Multicell Environment

In this section, we consider a hexagonal cell and its six neighboring cells, as shown in Fig. 14. Initially, we assume each cell has an equal number of MUs (i.e., $M/7$). An MU roaming process is assumed to be Poisson with an average sojourn time T_r (sec) in a cell. An MU from cell 1 has equal probability (1/6) of roaming into any of its six neighbors. Similarly, an MU from any other cell has 1/6 probability of roaming into cell 1 and 5/12 probability roaming to another neighboring cell. This balances the number of MUs in each cell which has one uplink channel with bandwidth W_u (bps) and one downlink channel with bandwidth W_d (bps).

The simulation results in the previous section showed that the performance of TS (or ETS) is much worse than that of SACCS and AS. In addition, there is no good mobility handling scheme for TS. So, we only study here the impact of mobility on SACCS and AS. As stated in Section 3, a *roaming* is treated as a forced wakeup event in both SACCS and AS. In SACCS, all valid cache entries are set to uncertain state after the roaming. In AS, the MU's first query and roaming time are sent to the MSS after the roaming.

We focus on the cache consistency scheme in multicell environments in our simulations, rather than the in-session data transaction. Hence, when an MU roams from one cell to another, all requests that are sent to the former MSS are dropped. We study the effect of the MU's mobility in this section.

Case 4: Effect of Average Sojourn Time. The values of parameters for this case are the same as shown in Table 5. Fig. 15 demonstrates that SACSS has much better perfor-

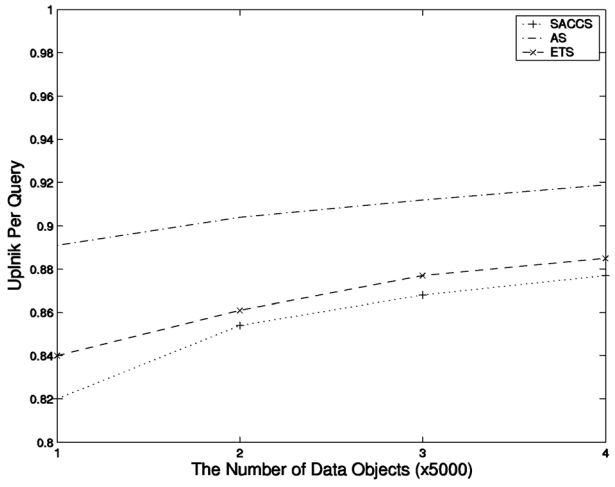


Fig. 11. Uplink per query versus number of data objects.

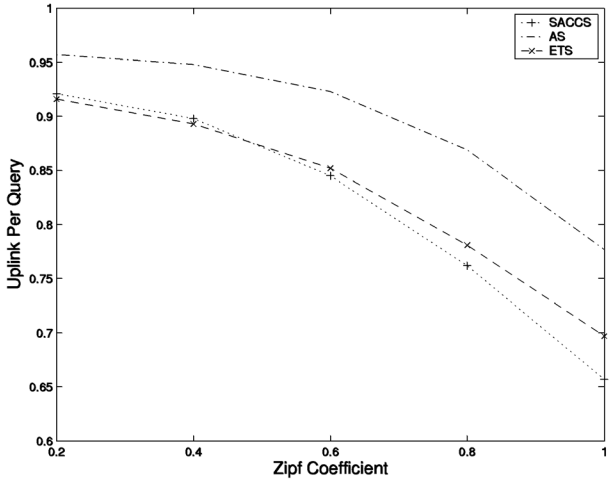


Fig. 13. Uplink per query versus Zipf coefficient.

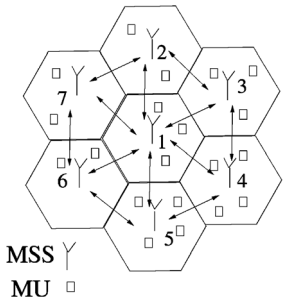


Fig. 14. Seven cell configuration; each cell has one MSS.

mance in terms of D (average query delay) for all ranges of average sojourn time. Moreover, SACCS has lower UPQ for long average sojourn time ($\geq 800\ sec$), but higher UPQ for very short average sojourn time than that of AS. In SACCS, all cache entries are set to uncertain state after each roaming. In AS, the first query after each roaming needs to be forwarded to the MSS to retrieve the cache state. For very short average sojourn time, the uncertain entries in SACCS have very few chances to be refreshed before its usage, thus resulting in a larger UPQ of SACCS than AS. For both SACCS and AS, the performance gets better as the average sojourn time increases (roaming frequency decreases). The average delay (D) of SACCS decreases from 1.7 sec to 1.55 sec. In AS, it decreases from 2.5 sec to 2.1 sec. This is due to fewer forced wakeup events and, hence, fewer extra uplinks. These results show that the impact of mobility is not significant enough in both schemes due to the fact that all valid data objects are retained after roaming. In other words, the uncertain state is a powerful method for SACCS to treat MU's disconnectedness and mobility.

6 Conclusions

In this paper, we proposed the Scalable Asynchronous Cache Consistency Scheme (SACCS) for mobile environments and evaluated its performance analytically as well as experimentally. Unlike the previous methods, SACCS provides a weak cache consistency under realistic environments for an MU with IR broadcast miss. It is a highly scalable and efficient scheme. The basic idea involves the use of flag bits at server cache (SC) and mobile user cache (MUC), an identifier (ID) in MUC for each entry after its invalidation, and estimated time-to-live (TTL) for each cached entry, as well as rendering of all valid entries of MUC to uncertain state when an MU wakes up. Strictly speaking, SACCS is a hybrid of stateful and stateless algorithms. However, unlike stateful algorithms, SACCS maintains only one flag bit for each data object in mobile support station (MSS) to determine when to broadcast the IRs. On the other hand, unlike the existing synchronous stateless approaches, SACCS does not require periodic broadcast of IRs, thus significantly reducing IR messages that need to be sent through the downlink broadcast channel. SACCS inherits the positive features of both stateful and stateless algorithms. Our comprehensive simulation results show that the proposed algorithm offers significantly better performance than the TS and AS schemes in both single and multicell environments.

An LRU-based cache replacement algorithm is used in this paper. Future work will investigate the impact of other replacement algorithms on the performance of SACCS. Further study is also needed for the MSS cache management algorithm and effective transfer of cached data objects among MSSs in response to MUs' roaming among different MSSs.

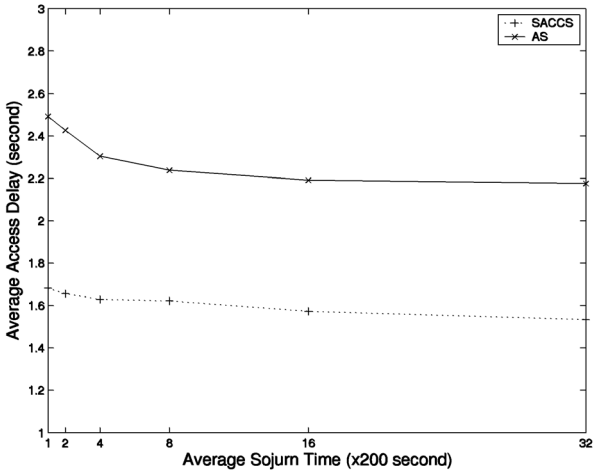


Fig. 15. Average access delay versus MU average sojourn time.

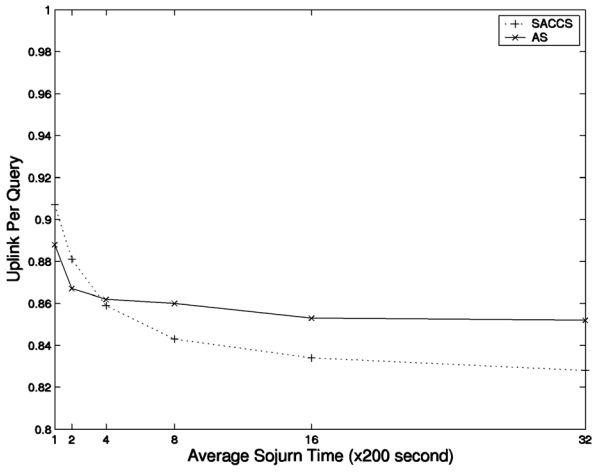


Fig. 16. Uplink per query versus MU average sojourn time.

Acknowledgments

The authors are grateful to the anonymous reviewers for their constructive comments that helped them improve the quality of the paper. This work is supported by a grant from the Texas Advanced Research Program under Grant Number 14-771032.

References

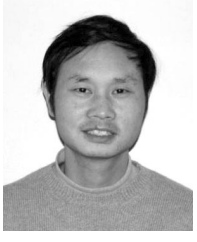
[1] D. Barbara and T. Imielinksi, " Sleeper and Workaholics: Caching Strategy in Mobile Environments," *Proc. ACM SIGMOD Conf. Management of Data,* pp. 1-12, 1994.

[2] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," *Proc. ACM SIGMETRICS Conf.,* May 1998.

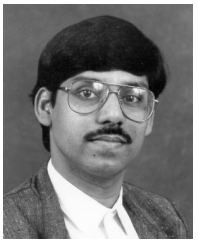
[3] L. Breslau, P. Cao, J. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-Like Distributions: Evidence and Implications," *Proc. IEEE INFOCOM,* pp. 126-134, 1999.

[4] G. Cao, "A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments," *Proc. ACM Int'l Conf. Computing and Networking (Mobicom),* pp. 200-209, Aug. 2001.

[5] G. Cao, "On Improving the Performance of Cache Invalidation in Mobile Environments," *ACM/Kluwer Mobile Network and Applications,* vol. 7, no. 4, pp. 291-303, 2002.
[6] P. Cao and C. Liu, "Maintaining Strong Cache Consistency in the World-Wide Web," *Proc. Int'l Conf. Distributed Computing Systems,* pp. 12-21, 1997.
[7] A. Chockalingam, M. Zorzi, L.B. Milstein, and P. Venkataram, "Performance of a Wireless Access Protocol on Correlated Rayleigh-Fading Channels with Capture," *IEEE Trans. Comm.,* vol. 46, pp. 644-655, 1998.
[8] L. Feeney and M. Nilsson, "Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment," *Proc. IEEE INFOCOM,* 2001.
[9] 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.
[10] J. Jing, 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.
[11] 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 Trans. Parallel and Distributed Systems,* vol. 12, no. 7, pp. 686-700, July 2001.
[12] D. Li and R. Cheriton, "Scalable Web Caching of Frequently Updated Objects Using Reliable Multicast," *Proc. USENIX Symp. Internet Technologies and Systems,* pp. 1-12, Oct. 1999.
[13] 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.
[14] T.S. Rappaport, *Wireless Comm.: Principles and Practice.* Printice Hall, 1996.
[15] K. Tan, J. Cai, and B. Ooi, "An Evaluation of Cache Invalidation Strategies in Wireless Environments" *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 8, pp. 789-807, Aug. 2001.
[16] Z. Wang, S.K. Das, H. Che, and M. Kumar, "SACCS: Scalable Asynchronous Cache Consistency Scheme for Mobile Environments," *Proc. Int'l Workshop Mobile and Wireless Networks,* pp. 797-802, 2003.
[17] K.L. Wu, P.S. Yu, and M.S. Chen, "Energy-Efficient Caching for Wireless Mobile Computing," *Proc. 20th Int'l Conf. Data Eng.,* pp. 336-345, 1996.
[18] H. Yu, L. Breslau, and S. Shenker, "A Scalable Web Cache Consistency Architecture," *Proc. ACM SIGCOMM,* pp. 163-174, Aug. 1999.
[19] J.C. Yuen, E. Chan, K. Lam, and H.W. Leung, "Cache Invalidation Scheme for Mobile Computing Systems with Real-Time Data," *SIGMOD Record,* Dec. 2000.
[20] J. Zhang, R. Izmailov, D. Reininger, and M. Ott, "Web Cache Framework: Analytical Models and Beyond," *IEEE Workshop Internet Applications,* pp. 132-141, 1999.

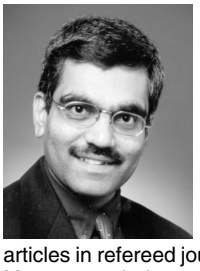


Zhijun Wang received the MS degree in electrical engineering from Pennsylvania State University, University Park, 2001. He is working toward the PhD degree in the Computer Science and Engineering Department at the University of Texas at Arlington. His current research interests include data management in mobile networks and peer-to-peer networks, mobile computing, and networking processors.



Sajal K. Das received the BS degree in 1983 from Calcutta University, the MS degree in 1984 from the Indian Institute of Science, Bangalore, and the PhD degree in 1988 from the University of Central Florida, Orlando, all in computer science. He is currently a professor of computer science and engineering and also the founding director of the Center for Research in Wireless Mobility and Networking (CReWMaN) at the University of Texas at Arlington (UTA). Prior to 1999, he was a professor of computer science at the University of North Texas (UNT), Denton, where he founded the Center for Research in Wireless Computing (CReW) in 1997 and also served as the director of the Center for Research in Parallel and Distributed Computing (CRPDC) from 1995-1997. Dr. Das was a recipient of the UNT Student Association's Honor Professor Award in 1991 and 1997 for best teaching and scholarly research; UNT's Developing Scholars Award in 1996 for outstanding research; UTA's Outstanding Faculty Research Award in Computer Science in 2001 and 2003; and the UTA College of Engineering Research Excellence Award in 2003. His current research interests include resource and mobility management in wireless networks, mobile and pervasive computing, wireless multimedia and QoS provisioning, sensor networks, mobile Internet architectures and protocols, parallel processing, grid computing, performance modeling, and simulation. He has published more than 250 research papers in these areas, directed numerous industry and government funded projects, and holds four US patents in wireless mobile networks. He received the Best Paper Awards from the Fifth Annual ACM International Conference on Mobile Computing and Networking (MobiCom '99), 16th International Conference on Information Networking (ICOIN-16), Third ACM International Workshop on Modeling, Analysis, and Simulation of Wireless and Mobile Systems (MSWiM 2000), and the 11th ACM/IEEE International Workshop on Parallel and Distributed Simulation (PADS '97). He serves on the editorial boards of *IEEE Transactions on Mobile Computing*, *ACM/Kluwer Wireless Networks*, *Parallel Processing Letters*, and the *Journal of Parallel Algorithms and Applications*.

Hao Che received the BS degree from Nanjing University, Nanjing, China, in 1984, the MS degree in physics from the University of Texas at Arlington, Texas, in 1994, and the PhD degree in electrical engineering from the University of Texas at Austin, Texas, in 1998. He was an assistant professor of electrical engineering at Pennsylvania State University, University Park, from 1998 to 2000, and a system architect with Santera Systems, Inc., Plano, Texas, from 2000 to 2002. Since September 2002, he has been an assistant professor of computer science and engineering at the University of Texas at Arlington. His current research interests include network architecture and design, network resource management, multiservice switching architecture, and network processor design.



Mohan Kumar received the PhD (1992) and MTech (1985) degrees from the Indian Institute of Science and the BE degree (1982) from Bangalore University in India. He is an associate professor of computer science and engineering at the University of Texas at Arlington. His current research interests are in pervasive computing, wireless networks and mobility, active networks, mobile agents, and distributed computing. He has published more than 95 articles in refereed journals and conference proceedings and supervised Masters and doctoral theses in the areas of pervasive computing, caching/prefetching, active networks, wireless networks and mobility, and scheduling in distributed systems. Dr. Kumar is on the editorial board of *The Computer Journal* and he has guest edited special issues of several leading international journals including MONET and WINET issues and the *IEEE Transactions on Computers*. He is a senior member of the IEEE.