# SDCI: SCALABLE DISTRIBUTED CACHE INDEXING FOR CACHE CONSISTENCY FOR MOBILE ENVIRONMENTS

**G. Lingamaiah[1], S. G. Nawaz[2], B. Dhanunjaya[3], G. Peddi Raju[4], K. Somasena Reddy[5]**

***Gujjala.lingamaiah@gmail.com, sngnawaz@gmail.com, Bandidhanunjaya700@gmail.com, peddi.raju15@gmail.com, Somasena12@gmail.com***

## Abstract

*In this paper, we suggest a new cache consistency maintenance scheme, namely Scalable distributed cache indexing for Cache Consistency (SDCI), for mobile environments. It mainly depends on 3 key features. They are: (1) Utilization of standard bits at server and Mobile Node's cache for maintaining cache consistency; (2) Utilization of Local Cache Standard (LC-standard) for all entries in Mobile Node's cache after its invalidation for maximizing the broadcast bandwidth efficiency; (3) Making every valid entry of Mobile Node's cache to whenever it wakes up. These features make the SDCI a good scalable algorithm with minimum database management overhead. By observing Comprehensive simulation results we can see that the performance of SDCI is superior to those of existing algorithms.*

***Keywords:*** *indexing, cache updating, mobile network, SDCI, SSUM*

----------***----------

## 1. INTRODUCTION

The use of wireless communication has been increasing day-by-day and has become a significant means for people to access different kinds dynamically changing data objects, such as news, stock price, and traffic information. But, wireless mobile computing environments are restricted by communication bandwidth and battery power [1], and have to survive with Mobile Node's disconnectedness and mobility. So, data communication in wireless mobile networks is difficult compared to that of wired networks.

*Caching* frequently used data objects at the local buffer of a Mobile Node is a better way to decrease query delay, save bandwidth and ameliorate system performance. However, in wireless mobile computing environments, difficulty in cache consistency arises with the frequent disconnection and roaming of an Mobile Node. A strategy that is successful must be able to efficiently handle both disconnectedness and mobility. The advantage with the broadcast is that it is able to serve an arbitrary number of Mobile Nodes with minimum bandwidth consumption. So, efficient mobile data transmission architecture should prudently design its broadcast and cache management schemes to maximize and minimize *delay.* Efficient mobile data transmission architecture should also be *scalable,* such that it works efficiently for large database systems and also upholds a large number of Mobile Nodes.

In the literature, there exist 2 types of cache consistency maintenance algorithms for wireless mobile computing environments. They are stateless and state-full. In the case of stateless approach, the server doesn't know the client's cache content. The client requires for verifying the validity of cached entries from the server before each and every query. Even though stateless approaches use simple database management, their scalability and capacity to uphold disconnectedness are poor. On the other hand, approaches are scalable. However, they cause significant overhead because of server database management. So, there is a necessity to develop scalable and efficient algorithms for maintaining cache consistency in mobile environments.

Inspired by the necessity for a scalable and efficient cache consistency maintenance mechanism, we put forward a new algorithm, namely scalable asynchronous cache consistency schema (SDCI) that maintains cache consistency between the Central Data Provider (CDP) and Mobile Node's caches. SDCI is a highly scalable, efficient, and low complexity algorithm because of the 3 key features: (1) Utilization of standard bits at server and Mobile Node's cache for maintaining cache consistency; (2) Utilization of an iD for each and every entry in Mobile Node's cache after its invalidation for maximizing the broadcast bandwidth efficiency; (3) Making all valid entries of Mobile Node's cache to *uncertain state* when it wakes up.

Comprehensive simulation results reveal that SDCI gives superior performance than existing algorithms. Taking an example, in a system having different types of Mobile Node access patterns and data object update frequencies, SDCI will be able to support about 44 percent and 270 percent more Mobile Nodes than Timestamp (SSUM) schemes, respectively. It also makes sure that the average access delay is no larger than seconds.

Remaining part of the paper is organized as given below. Section ii is a brief overview of the related work. In Section III, complete description of the SDCI algorithm is given. Section IV gives comprehensive simulation results of the algorithm and after that compares it with already existing procedures. Section V is about the conclusions drawn.

## 2. RELATED WORK

We summarize existing stateless and stateful processes in this section. In the *stateless* approach [2]-[7], a CDP presumes no knowledge of Mobile Node's cache contents. CDP sends CUAs to its Mobile Nodes sporadically. At an Mobile Node, a data object request cannot be serviced until the next IR from the CDP is received. In the *stateful* approach [8], an CDP preserves object state for each Mobile Node's cache and only broadcasts CUAs for those objects.

Barbara and Imielinksi [2] put forwarded 3 stateless algorithms: Timestamps (SSUM), Amnesic Terminals *(AT)* and Signature *(SIG).* In these algorithms, the CDP broadcasts CUAs, that consist all data object IDs that are updated during the past seconds (where is a positive integer), every seconds. The benefit of these algorithms is that an CDP does not maintain any state information about its Mobile Node's caches, and this makes the management of CDP database very easy. But, there are many disadvantages with these algorithms. The first problem is that they do not scale well to large databases and/or fast updating data systems, because of increased number of IR messages. The second one is that the average access latency is usually longer than half of the broadcast period. This is because all requests can be answered only after the next IR. Lastly all cached data objects are dropped in the case of the sleep time is longer than.

For handling the long sleep-wake up patterns, various algorithms have been suggested. Going by example, in the bit-sequence *(BS)* algorithm because of Jing, et al. [3], every cache entry is deleted only when half or more of the data entries in the cache have been nullified. But, the model needs the broadcast of a very large number of IR messages compared to that of SSUM and AT schemes. Wu et al. [5] suggested an uplink validation check scheme which will be able to deal with long sleep-wake up themes better than SSUM and AT. However the problem with this approach is that this approach needs more uplink bandwidth and cannot deal with long sleep-wake up patterns.

A very few stateful cache consistency maintenance algorithms have been suggested for wireless mobile computing environments. Khaleel Mershad et al[12] proposed a smart server update mechanism for Maintaining Cache Consistency in Mobile Environments. Though the solution is impressive when compared to most frequently cited solutions in earlier literature, but still it is probabilistic to make available data in cache for nodes,.

In AS, an CDP records all retrieved data object for each and every Mobile Node. After an Mobile Node first gets back a data object after it wakes up, the CDP sends an IR, depending on the Mobile Node's cache content record and sleep-wake up time, to that respective Mobile Node. When an CDP gets an update from the original server for all the recorded data objects, it broadcasts that data object's IR to Mobile Nodes. The benefit of the AS scheme is that the CDP averts needless broadcast of CUAs to Mobile Nodes and also it can deal with any sleep-wake-up pattern without losing any valid data object. But, for maintaining each Mobile Node's cache state in the CDP, the CDP must and should record all cached data objects for each and every Mobile Node. So an Mobile Node can be able to download data objects which it requested using the uplink. This renders the channel utility inefficient and sensitive to the number of Mobile Nodes.

## 3. SCALABLE DISTRIBUTED CACHE INDEXING FOR CACHE CONSISTENCY (SDCI)

In this particular paper, we suggest a new Scalable distributed cache indexing for Cache Consistency (SDCI) for maintaining Mobile Node's cache consistency that is used in a read-only system. SDCI is a hybrid of stateless and stateful procedures which means that it maintains minimum state information. But, unlike the stateful algorithm [12] that makes the CDP to remember all data objects for all Mobile Node's cache, SDCI needs only the CDP to recognize which data objects in its database might be valid in Mobile Node's caches, making the management of the CDP database much simpler than usual. However, unlike prevailing synchronous stateless processes, SDCI does not need periodic broadcast of CUAs, decreasing IR messages that are required to be sent through the downlink broadcast channel. Also, by appending uncertain and Local-Cache states in Mobile Node's caches, SDCI makes easy handling of arbitrary sleep-wake-up patterns and mobility, and good cooperation among all Mobile Nodes that greatly enhances broadcast channel efficiency. The subsections that follow explain the suggested algorithm in detail.

### A. Data Structures and Message Formats

For each and every data object having distinct identifier x , the data structure for CDP and Mobile Node's cache are as given below:

- $(d_x, t_x, f_x)$ Data entry format for each data object in CDP. Here $d_x$ , is the data object, $t_x$ is the last update time for the data object and $f_x$ is a flag bit.
- $(d_x, ts_x, s_x)$ :data entry format in an Mobile Node's cache. Here $d_x$ is defined above, $ts_x$ is the time stamp showing the last updated time for the cached data object $d_x$ $d_x$, and $s_x$ is a two-bit flag recognizing 4 data entry

states: *stable*, *uncertain*, *waiting* And $LC-\texttt{standard}$ respectively.

### B. Mobile Node Cache Management

As this paper is mainly about the cache consistency maintenance design, we utilize the LRU (Least Recently Used) based replacement algorithm for managing Mobile Node's caches. The effect of the cache replacement algorithms on SDCI is studied later.

In LRU based replacement scheme, a data object that is recently cached or a cached data object that gets a hit is relocated to the head of the cache list. Whenever a data object requires to be cached and the cache is full, data entries with *sx* $\neq$ *2* from the tail are removed for creating enough space in order to accommodate this new data object (the data object with *sx = 2* must be kept so that some requests are waiting for its confirmation).

Any refreshed data objects from uncertain state or Local-Cache state are moved to their original location and again, if required, sufficient data entries from the tail are deleted.

### C. Algorithm Description

We have given 2 procedures, i.e., CDP-trans () and Mobile Node transactions, for the SDCI. The CDP continuously performs the CDP-trans () and each Mobile Node continuously performs the Mobile Node Transactions method during its awake period.

The pseudo codes for CDP-trans () and Mobile Node Transactions are as shown in Figures 1 and 2.

**CDP-trans (** $request(x)$ **) {**

$sts = findStatus(x)$

$if\ (sts \cong available)$ **begin**

**Broadcast** $d_x, t_x, x$

$rValue \leftarrow available$

$if\ (f_x \cong 0)$

$f_x \leftarrow 1$

*End;*

$if\ (sts \cong uncertain)$

$if\ (t_x \cong ts_x)$

Broadcast $(x, t_x)$

$rValue \leftarrow confirm$

*else*

**Broadcast** $d_x, t_x, x$

$rValue \leftarrow available$

$if\ (f_x \cong 0)$

$f_x \leftarrow 1$

$if\ (sts \cong update)$

Update the database entry ( $d_x \leftarrow d_x$ '&&$t_x \leftarrow t_x$ ')

$if\ (f_x \cong 1)$ **begin**

Broadcast $cua(x)$

$f_x \leftarrow 0$

End;

**Fig1:**CDP-trans

### D. Cache Consistency Maintenance

We explain in detail how SDCI sustains consistency between an CDP database and Mobile Node caches. We presume that the consistency between the CDP database and original servers is sustained using wired network consistency algorithms [9], [10].

For each and every cached data object, SDCI utilizes a single flag bit $f_x$, for maintaining the consistency between the CDP and Mobile Node caches. Whenever $d_x$ is recovered by an Mobile Node, *fx* is set, showing that a valid copy of $d_x$ may be ready to use in an Mobile Node's cache. Whenever the CDP gets an updated $d_x$, it broadcasts and resets. This action shows that there are no valid copies of $d_x$ in any Mobile Node's caches. Also, further updates do not necessitate broadcast of IR(x).

The flag is set again whenever the CDP services restoration (comprising request and confirmation) for $d_x$ by an Mobile Node.
In mobile environments, an Mobile Node's cache is in either of two states: (i) awake or (ii) sleep. In the case of an Mobile Node is *awake* at the time of IR(x) broadcast, the $d_x$ copy is invalidated and an ID-only

***mnRequestHandler***(
$req(d_x)$ ) {
$sts = reqStatus(d_x)$
$if\ (sts \cong stable)$
Update cache list
Return
$cached(d_x)$
$if\ (sts \cong uncertain)$
Add $x$ to waiting list
Set $s_x \leftarrow 2$
Update cache list head with this entry
Return
$uncertain(x, ts_x)$
$if\ (sts \cong unavailble)$
send *request to* CDP
add $x$ to query waiting list
}

***mnResponseHandler***(
$resp(d_x)$ ){
$sts = status(x)$
$if\ (sts \cong available)$
Remove $x$ from cache
Add $x$ to cache head
*return* $d_x$
$if\ (sts \cong waiting)$
//ELSE **IF** (entry $x$ is ID-only entry in cache )
mnRequestHandler(
$req(d_x)$
$else if\ (sts \cong uncertain)$
$if\ (ts_x < t_x)$
$ts_x \leftarrow t_x$
$s_x \leftarrow 0$
mnRequestHandler(
$req(d_x)$
*else*
$s_x \leftarrow 0$
**}**

***mnCacheUpdateAlertHandler***( $cua(x)$ ){
$msg = reqStatus(cua(x))$
$sts = reqStatus(d_x)$
$if\ ((msg \cong update)\ \&\&(sts \cong available$
**begin**
$s_x \leftarrow 3$
$remove(d_x)$
end
}
$elseif\ (msg \cong confirm)\ \&\&(sts \cong unce$
$s_x \leftarrow 0$
$elseif\ (msg \cong confirm)\ \&\&(sts \cong waiti$
Return $d_x$
*else*
$delete(d_x)$
$s_x \leftarrow 3$
**}**

**mnWakeupHandler(){**
$foreach(s \cong 0)$
$s \leftarrow 1$
**}**

**Fig2**: Mobile Node Transactions

Entry is sustained by the Mobile Node. The data objects of a Mobile Node in the *sleep* state are not affected until it wakes up. Whenever an Mobile Node wakes up, it sets all cached valid data objects (including) into the uncertain state. As a result sleeping Mobile Nodes and the cached objects are not affected by missing broadcast.

### E. Efficiency and Cooperation

As stated earlier, a good cache consistency maintenance algorithm must be able to get scaled and also must be efficient in terms of the database size and the number of Mobile Nodes. SDCI are able to handle large and fast updating data systems as the CDP has some knowledge of Mobile Node's cache. Only data entries that have standard bits set result in the broadcast of IBs whenever data objects get updated. As a result, the *IR* broadcast frequency is the minimum of the uplink query/confirmation frequency and the data object update frequency. Similarly, the broadcast channel bandwidth consumption for CUAs is minimized.

In addition to IR traffic, all other traffic in SDCI is also minimized because of the strong cooperation among the Mobile Node's caches. Specifically, this is because of the introduction of the uncertain state and the Local-Cache state for the Mobile Node's caches. The recovery of a data object, $d_x$ from the CDP allotted by any given Mobile Node makes the x entries in the uncertain or Local-Cache state in all the awake Mobile Node's caches to a valid state. Also, a single uplink confirmation for in the uncertain state makes the entries in uncertain state for all the awake Mobile Node's caches to either valid or Local-Cache state. The addition of the uncertain state also helps an Mobile Node's cache in order to keep all the valid data objects after it wakes up from duration of sleep time. In contrast, for SSUM algorithms, every invalidated data object is completely removed from the Mobile Node's cache. This helps in bringing little cooperation among the Mobile Nodes, creating a dramatic increase of traffic volume between the CDP and the Mobile Nodes as the number of Mobile Nodes increases (see in Section V). Even though it improves the scalability of SSUM by holding the invalid data objects, it decreases the cache efficiency by keeping the invalid data objects, instead of IDs as is the case in our SDCI approach, in the Mobile Node's cache.

In variance with the AS scheme that needs O(MN) buffer space in the CDP for keeping all the Mobile Node's cache state, our procedure only requires *one bit* per data object in the CDP showing that if the broadcast is necessary whenever the data object is updated. Also, the added management overhead is minimal as it needs only a single bit check and set/reset.

### F. Mobility

Generally, synchronous stateless procedures handle Mobile Node's mobility by presuming that all CDPs broadcast the same CUAs .But, in the case of the number of CDPs being large, those systems cannot be scaled. There is no other alternate way for handling the Mobile Node's mobility in the literature. In our approach, an Mobile Node roaming into a new cell is considered as if it just woke up from the sleep state, i.e., every valid data object is set to the uncertain state. The consistency is guaranteed by using this approach and every valid data object is retained. Moreover, SDCI is simple because it is transparent to the CDPs engaged. In this approach, the roaming effect is nothing but the addition of a new sleep-wake-up pattern and should not have any substantial impact on the overall performance. This paper emulates the roaming effect by a sleep-wake-up pattern.

## 4. PERFORMANCE EVALUATION BY SIMULATION

### A. simulation setup

We take into consideration a single cell system with one CDP database and multiple Mobile Nodes having identical cache size. The request process for each Mobile Node is presumed to be Poisson distributed and the update processes for data objects are also Poisson distributed. Moreover, the sleep-wake-up process is made into a model having a two-state Markov chain.

In our simulation, we utilized a single channel with bandwidth *CB* for both downlink as well as uplink data transmission. Every message is queued and serviced based on a first-come-first- serve basis. Every request is neglected whenever an Mobile Node is in the sleep state. Error recovering cost and software overhead are also neglected. Whenever a requested data object is available at an Mobile Node's local cache, the delay *AQD* is counted as 0. A *Zip f—like* distribution Mobile Node access pattern [11] is utilized in the simulation.

The following subsections give a comprehensive comparison of the suggested SDCI with SSUM algorithms by means of metrics and for 3 distinct cases. The average access delay is a significant measurement of system performance, a shorter *AQD*, a better performance. The *uplink per query* is associated to cache hit ratio. Whenever an Mobile Node receives a query, in the case of the queried data object being valid in the cache, a cache hit is counted, and no uplink is required for the query. So, higher hit ratio, fewer uplink per query.

### B. Case Studies

Three cases are given here. In each case, $b_u$ =64 and *bd = 64* for both SDCI, and *bu = 10* and *bd = 10* for SSUM[12].

The bandwidth is set as CB=10000bps. Cache size is in units of the number of data objects and the maximum number of ID-only entries is also set to in the first two cases. Data object sizes and data object update frequencies. They will be provided later in detail and @ is used to denote them in Table II. Each and every case study correlates to a parameter effect that is shown by * in the column.

**Case 1:** Effect of CDP Database Size

Table 1 shows the simulation results of AQD and UPQS for two algorithms with distinct database sizes.

From Table III, we see that SDCI outperforms the SSUM [12] on both performance metrics for distinct database sizes (N). Whenever the database size reaches *approx 13000, AQD* for SSUM is over *160 sec*. But in case of SDCI, we can observe a slow increase of the performance metrics as increases all the way up to approx N value 13000.

In brief, the performances of SDCI are insensitive to the database size N and so it can be used to scale large database sizes. But, the performance of SSUM is responsive to the database size, specifically in terms of the delay performance, and so SSUM cannot be used to scale large database sizes.

**Case 2:** Effect of Data Update Rate

Table IV shows the simulation results of the effect of update interval $t_u$. Moreover, it is observed SDCI outperforms SSUM [12] for all these performance metrics. As assumed, as $t_u$ declines, both performance metrics increase for all these algorithms. But, SDCI show slower rates of increase compared to SSUM for all the metrics. Particularly, whenever $t_u$ is declined to 10 sec, the delay performance of SSUM is very large (about 50 sec) beyond acceptance. At this update interval, SDCI gets considerable delay performance gain over SSUM. For *uplink per query*, SDCI outperforms SSUM in the range of 5 to 30 percent.

Case 3: Effect of the Number of Mobile Nodes. In this case study, we take into consideration a system identical to a real situation. We presume that a cell has 5 distinct Mobile Node query patterns as $\lambda = \frac{1}{(10+50*i)}$ and s=0.9-0.2*I and *Ts* = 500 * (*i* + 1) *sec* with i = 0,1,2,3 and 4.

Each and every query pattern group has *M/5* Mobile Nodes in it. We presume the query patterns for all the groups follow Zip f-like (z=1 ) distribution. The access popularity ranking for the neighboring groups is moved *by 10*, i.e., group I has declining popularity from data object 1 to 1000 and group 2from 11to 1000 and then from 1to 10, and so on.

We also presume that there are 5 types of data objects in the CDP as: $b_p = 500*(i+1)$ and $T_u = 10^{i+1}\,\text{sec}$ with i= 0,1,2,3 and 4

**Table1:** Average Query Delay per no of records

| | 100 | 200 | 400 | 800 | 1600 | 3200 | 6400 | 12800 |
|---|---|---|---|---|---|---|---|---|
| SDCI | 0.165 | 0.298 | 0.429 | 0.564 | 0.679 | 0.784 | 0.896 | 1.015 |
| SSUM | 12.314 | 12.974 | 13.862 | 14.4399 | 14.994 | 15.494 | 17.465 | 161.977 |

per no of transactions

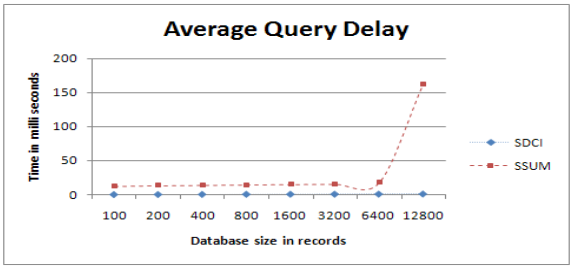
| | 10 | 40 | 160 | 640 | 2560 | 10240 |
|---|---|---|---|---|---|---|
| SDCI | 3.004 | 1.6 | 0.934 | 0.667 | 0.596 | 0.573 |
| SSUM | 50.006 | 36.9 | 17.139 | 15.488 | 15.096 | 14.879 |

**Table 2:** Uplink per query per no of records

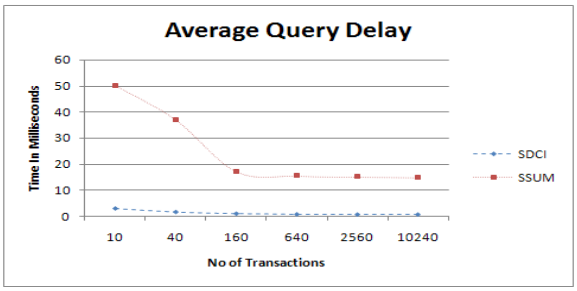
| | 100 | 200 | 400 | 800 | 1600 | 3200 | 6400 | 12800 |
|---|---|---|---|---|---|---|---|---|
| SDCI | 0.234 | 0.334 | 0.428 | 0.495 | 0.558 | 0.599 | 0.648 | 0.673 |
| SSUM | 0.7 | 0.7 | 0.8 | 0.861 | 0.89 | 0.89 | 0.92 | 0.924 |
| UM | 56 | 93 | 32 | | 3 | 5 | 2 | |

per no of transactions

| | 10 | 40 | 160 | 640 | 2560 | 10240 |
|---|---|---|---|---|---|---|
| SDCI | 0.838 | 0.717 | 0.584 | 0.509 | 0.513 | 0.520 |
| SSUM | 0.999 | 0.986 | 0.956 | 0.893 | 0.839 | 0.825 |

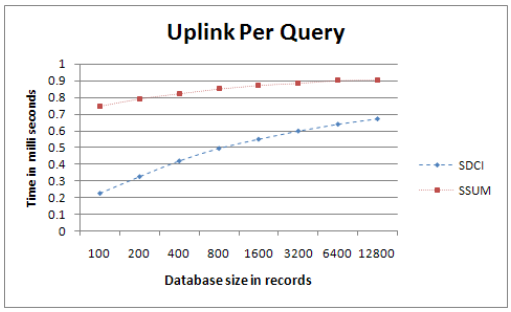



(a) Per no of records

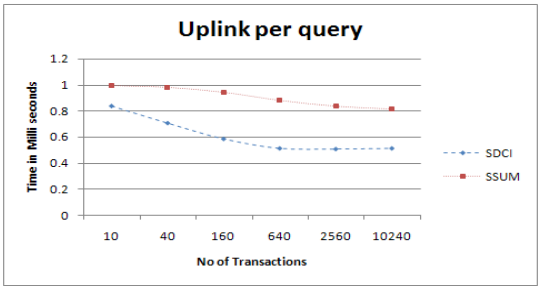



(b) Per no of transactions

**Fig 3:** Line chart representation of Average query delay comparison between SDCI and SSUM




(a)Per no of records




(b)Per no of transactions

**Fig 4:** Line chart representation of uplink per query comparison between SDCI and SSUM

Each and every data type group has N/5 data objects.

The parameter values that are selected above depend on the understanding that a highly frequent query of Mobile Node usually implies a shorter awake time and in the case of a faster updated data object generally has a smaller size. The cache size *is 150 Kbytes.* The maximum number of data ID-only entries which can be put in each Mobile Node cache is set at 100.

Figures 3 and 4 describe AQD and uplink per query comparison between SDCI and SSUM, respectively. As we can observe, SDCI scales much better compared to SSUM in terms of both performance metrics.

## CONCLUSIONS AND FUTURE WORK

In this paper, we suggested a Scalable distributed cache indexing for Cache Consistency (SDCI) for mobile environments. 3 key features of SDCI are: (i) use of standard bits at CDP and Mobile Node's caches to sustain cache consistency; (ii) utilization of a Local-Cache state for each and every entry in Mobile Node's cache after a data object becomes nullified; (iii) all acceptable data entries are set to the uncertain state after an Mobile Node wakes up. These main features make the suggested algorithm highly scalable and efficient. To be straight forward, SDCI is a hybrid of stateful and stateless algorithms. But, unlike stateful algorithms, SDCI sustains one flag bit for each and every data object in CDP to ascertain whenever to broadcast CUAs. On the flipside, unlike prevailing synchronous stateless procedures, SDCI does not need periodic broadcast of CUAs. So SDCI largely declines IR messages that are required to be sent through the downlink broadcast channel. SDCI gets the positive features from both stateful as well as stateless algorithms. Comprehensive simulation results reveal that the suggested algorithm higher performance compared to SSUM scheme. Future work will comprise studying the effect of distinct replacement algorithms on the performance of SDCI. Future study will also delve into the CDP cache management algorithm and the effective transfer cached data objects among CDPs when Mobile Nodes roam among distinct CDPs.

## REFERENCES

[1]G. Forman and J. Zahorjan, "The challenge of mobile computing", IEEE Computer, 27(6), pp38-47, April 1994.
[2]D. Barbara and T. Imielinksi," Sleeper and Workaholics: caching strat¬egy in mobile environments ", In Proceedings of the ACM SIGMOD Conference on Management of Data, pp1-12, 1994.
[3]J. Jing, A. Elmagarmid, A. Heal, and R. Alonso. "Bit-sequences: an adaptive cache invalidation method in mobile client/server environ¬ments", Mobile Networks and Applications, pp 115-127, 1997.
[4]Q. Hu and D.K. Lee, "Cache algorithms based on adaptive invalidation reports for mobile environments", Cluster Computing, pp 39-50, 1998.
[5]K.L. Wu, P.S. Yu and M.S. Chen, "Energy-efficient caching for wire¬less mobile computing", In 20th International Conference on Data En¬gineering, pp 336-345, 1996
[6]G. Cao, "A scalable low-latency cache invalidation strategy for mobile environments", ACM Intl. Conf. on Computing and Networking (Mo- bicom), pp200-209, August, 2001
[7]K. Tan, J. Cai and B. ooi, "An evaluation of cache invalidation strate¬gies in wireless environments", IEEE Trans. on Parallel and Dis¬tributed System, 12(8), pp789-897, 2001
[8]A. Kahol, S. Khurana, S.K.S. Gupta and P.K. Srimani," A strategy to manage cache consistency in a distributed mobile wireless environ¬ment", IEEE Trans. on Parallel and Distributed System, 12(7), pp 686¬700, 2001.
[9]H. Yu, L. Breslau and S. Shenker, "A scalable web cache consistency architecture", In Proceedings ofthe ACM SIGCOMM, August, 1999.
[10]J. Gwertzman and M. Seltzer, "World-Wide Web cache consistency", In Proceedings of The USENIX Symposium on Internet Technologies and Systems, December, 1997.
[11]L. Breslau, P. Cao, J. Fan, G. Phillips and S. Shenker, "Web caching and Zipf-like distributions: evidence and implications, ", Proceedings of IEEE INFOCOM'99,pp126-134, 1999
[12]Khaleel Mershad and Hassan Artail, Senior Member, IEEE; "SSUM: Smart Server Update Mechanism for Maintaining Cache Consistency in Mobile Environments"; IEEE TRANSACTIONS ON MOBILE COMPUTING, VOL. 9, NO. 6, JUNE 2010
[13] A. Elmagarmid, J. Jing, A. Helal, and C. Lee, "Scalable Cache Invalidation Algorithms for Mobile Data Access," IEEE Trans. Knowledge and Data Eng., vol. 15, no. 6, pp. 1498-1511, Nov. 2003.
[14] H. Jin, J. Cao, and S. Feng, "A Selective Push Algorithm for Cooperative Cache Consistency Maintenance over MANETs," Proc. Third IFIP Int'l Conf. Embedded and Ubiquitous Computing, Dec. 2007.

[15] IEEE Standard 802.11, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification, IEEE, 1999.
[16] J. Jing, A. Elmagarmid, A. Helal, and R. Alonso, "Bit-Sequences: An Adaptive Cache Invalidation Method in Mobile Client/Server Environments," Mobile Networks and Applications, vol. 15, no. 2, pp. 115-127, 1997.
[17] J. Jung, A.W. Berger, and H. Balakrishnan, "Modeling TTL-Based Internet Caches," Proc. IEEE INFOCOM, Mar. 2003.
[18] X. Kai and Y. Lu, "Maintain Cache Consistency in Mobile Database Using Dynamical Periodical Broadcasting Strategy," Proc. Second Int'l Conf. Machine Learning and Cybernetics, pp. 2389- 2393, 2003.
[19] B. Krishnamurthy and C.E. Wills, "Piggyback Server Invalidation for Proxy Cache Coherency," Proc. Seventh World Wide Web (WWW) Conf., Apr. 1998.
[20] B. Krishnamurthy and C.E. Wills, "Study of Piggyback Cache Validation for Proxy Caches in the World Wide Web," Proc. USENIX Symp. Internet Technologies and Systems, Dec. 1997.
[21] D. Li, P. Cao, and M. Dahlin, "WCIP: Web Cache Invalidation Protocol," IETF Internet Draft, http://tools.ietf.org/html/draftdanli- wrec-wcip-01, Mar. 2001.
[22] W. Li, E. Chan, Y. Wang, and D. Chen, "Cache Invalidation Strategies for Mobile Ad Hoc Networks," Proc. Int'l Conf. Parallel Processing, Sept. 2007.
[23] S. Lim, W.-C. Lee, G. Cao, and C.R. Das, "Performance Comparison of Cache Invalidation Strategies for Internet-Based Mobile-Ad Hoc Networks," Proc. IEEE Int'l Conf. Mobile Ad-Hoc and Sensor Systems, pp. 104-113, Oct. 2004.
[24] M.N. Lima, A.L. dos Santos, and G. Pujolle, "A Survey of Survivability in Mobile Ad Hoc Networks," IEEE Comm. Surveys and Tutorials, vol. 11, no. 1, pp. 66-77, First Quarter 2009.
[25] H. Maalouf and M. Gurcan, "Minimisation of the Update Response Time in a Distributed Database System," Performance Evaluation, vol. 50, no. 4, pp. 245-266, 2002.
[26] P. Papadimitratos and Z.J. Haas, "Secure Data Transmission in Mobile Ad Hoc Networks," Proc. ACM Workshop Wireless Security (WiSe '03), pp. 41-50, 2003.
[27] J.P. Sheu, C.M. Chao, and C.W. Sun, "A Clock Synchronization Algorithm for Multi-Hop Wireless Ad Hoc Networks," Proc. 24th Int'l Conf. Distributed Computing Systems, pp. 574-581, 2004.
[28] W. Stallings, Cryptography and Network Security, fourth ed. Prentice Hall, 2006.
[29] D. Wessels, "Squid Internet Object Cache," http://www. squid-cache.org, Aug. 1998.
[30] J. Xu, X. Tang, and D. Lee, "Performance Analysis of Location- Dependent Cache Invalidation Schemes for Mobile Environments," IEEE Trans. Knowledge and Data Eng., vol. 15, no. 2, pp. 474-488, Feb. 2003.
[31] L. Yin, G. Cao, and Y. Cai, "A Generalized Target Driven Cache Replacement Policy for Mobile Environments," Proc. Int'l Symp. Applications and the Internet (SAINT '03), Jan. 2003.