Modified Energy Efficient Cache Invalidation Algorithm in Mobile Environment

S. Sankara Gomathi, S. Krishnamurthi

***Abstract* – Maintenance of the cache consistency is a complicated issue in the wireless mobile environment. Caching of frequently accessed data items on the node can reduce the bandwidth requirement of the mobile node environment. In this paper, we present a new technique called Modified Energy Efficient Cache Invalidation Algorithm (MEECIA) especially to reduce the uplink bandwidth consumption in the wireless ad-hoc network, by choosing following modes: Slow, Fast, Super-fast. The mode is selected based on threshold specified for time and the number of node requesting the updated data to other node. Simulation results demonstrate that our algorithm is efficient in improving mobile caching, reducing the communication bandwidth and also efficiently utilizing the energy in ad-hoc network. Compared to the previously reported cache based Invalidation Report (IR), our scheme can significantly improve the performance in terms of energy consumption and reduce the query latency in ad-hoc network.**

***Index Terms* - Mobile caching, Cache consistency, Query latency, Broadcast bandwidth, IR, Timestamp.**

I. Introduction

A wireless ad-hoc network is a computer network, in which the communication links are wireless. The network is ad-hoc because each node is willing to broadcast and requests the data for other nodes and so the determination of which node broadcast the data, is made dynamically based on the number of request from other nodes. Nodes participating in ad-hoc network have limited bandwidth and battery power [1]. So the data communications in these networks are more challenging than in wired network.

Introducing caching technique for frequently accessed data item at each node leads to better utilization of bandwidth and improve the overall performance of the system [2],[3]. This cache consistency between each node is maintained by generating IR for updating the cache. Caching of frequently accessed data at the mobile node has been considered to be very useful and efficient mechanism for conserving the bandwidth [11],[3],[4]. Because average data access latency is reduced as several data access requests can be satisfied from the cache. The IR consists of the information about objects that have been updated recently [13]. Node needs to invalidate an object that are found in the report and salvage their content that is still valid.

S.Sankara Gomathi is pursuing research work at Anna University in the field of mobile communication. She is an Assistant professor of Sri Venkateswara College of Engineering in India (email: sgomathisubhra@yahoo.com)

S. Krishnamurthi is a Professor of Electronics and Communication department, Anna University, India (email : skmurthi@gmail.com)

In wireless ad-hoc network every node will act as a server and each node maintains certain data items. A node can request for a particular object which will go to the corresponding server. Transition of the node to corresponding mode will depend on the number of node request to the particular object and each node request is quenched as per the mode.

An algorithm proposed by this paper provides both packet and power efficiency in ad-hoc network. That is, the ability of the algorithm is to minimize the total number of packets sent on a wireless link, and the ability to minimize the battery power consumed by the node. Also we address the problem of cache invalidation scheme in ad-hoc network environment and proposes new cache invalidation scheme for improving the energy consumption with low query latency and also it improves the packet and power efficiency by introducing different modes of operation by each node. Thus an efficient implementation of the MEECIA algorithm is presented and simulations have been carried out using java simulator to evaluate its cache effectiveness.

II. Related Work

In the literature there are two types of cache consistency maintenance algorithms proposed [3], [6], [10]. They are stateful and stateless. Stateful approaches are scalable, but they incur significant overhead due to server database management. In stateless approach, the server is not aware of the state of the client's cache. The clients need to query the server to verify the validity of their cache before each use, which not only wastes the scarce wireless bandwidth, but also consumes a lot of battery energy. Therefore there is a need for developing scalable and efficient algorithm for maintaining cache consistency in mobile environments.

Authors in [5],[8] explained that the data dissemination is the delivery of data from memory of one node to the number of other nodes. Data dissemination strategy follows periodic and aperiodic dissemination. Periodical broadcasting of the data allows the node to disconnect for certain period, without missing out items. However aperiodic dissemination is a more effective technique of using the available bandwidth.

Authors in [15] proposed that the mobile station broadcasts the IR periodically to other node. Once the report has been generated, the node invalidates its cache accordingly. But some times the node cannot receive the IR (if it is in the disconnected state), then the node may be forced to discard the entire cache content, when the disconnection time exceeds certain threshold.

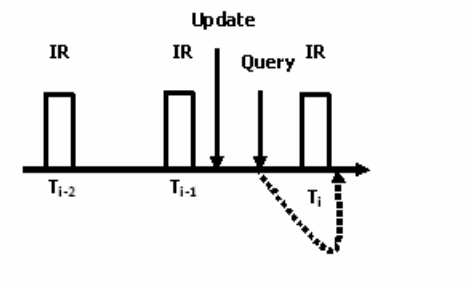

Fig 1. Periodic Broadcasting

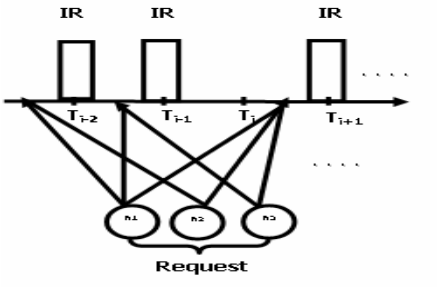

Fig 2. Aperiodic Broadcasting

Each node needs to decide whether to send the data periodically or aperiodically. Periodic transmission has the advantages over aperiodic transmission. Periodic transmission of data can reduce the uplink request, and also it consumes large amount of energy as shown in fig 1. Whereas in aperiodic transmission, each time the node is requesting the data as in fig 2 more bandwidth is required. The bandwidth in the downstream direction is much greater than the upstream direction [6].

Authors in [15] also explained that the cache invalidation is based on the periodic broadcasting of invalidation report. The node listens to the report to identify the cache validity. If the cache is not valid the node sends the query and refreshes its cache, and the node keeps a list of items queried during an interval and answers them after receiving the next IR. If two or more queries for the same item are requested, then all will be answered at the next IR as shown in fig 1.

Authors in [9] explained that, each node is maintaining certain data item depending upon the application. The node issues only simple requests to read the most recent copy of a data item from other nodes. Invalidation report has been generated, either when any updates take place at a particular node or the data items needed by other node. Additionally, such architectures should be scalable to support large database system as well as large number of nodes. This algorithm maintains the cache consistency between nodes and reduces the query latency.

III. Cache Invalidation Scheme

A. Broadcasting Timestamp Algorithm:

In Timestamp Algorithm (TS), the server identifies the data that has changed in the last '*w*' second. Thus, the invalidation report is composed by the timestamps of the latest change for items. The Mobile User (MU) listens to the report and updates the status of its cache. For each item cached, the MU either purges it from the cache (when the items is reported to have changed at a time larger than the timestamp stored in the cache), or updates the cache's timestamp to the timestamp of the report (if the item was not mentioned). Notice that this is done for every item in the cache.

The server begins to broadcast the invalidation report periodically at times Ti = i×L. The server keeps a list LS_i, Qi as follows.

LS_i - {[d, ts_d] | (d ε D) Ti-w < ts_d < Ti}

ts_d - The timestamp of the last update of 'd'

Qi - {j | j has been queried in the interval [Ti-1, Ti]}

The MU keeps a variable 'T1' that indicates, the last time it received a report. If the difference between the current report timestamp and this variable is larger than w, the entire cache is dropped. More formally, the MU runs the algorithm as:

```
if (Ti – Tl > w) { drop the entire cache; }
else {
  for every item d in the MU cache {
    if ([d, ts_d] is in LS_i) {
      if (ts^c < ts_d)   { throw d out of the cache;}
      else { ts^c = Ti; }}}}
  for every item d ε Qi  {
   if d is in the cache
       {use the cache's value to answer the query; }
    else { go uplink with the query; }}Tl = Ti;
```

Algorithm 1. Checking Cache Validity using Broadcasting Timestamp Algorithm

B. Simple Checking Algorithm:

After a long period of disconnection, most of the cached data still have not been updated. But they are retained in the local cache after the node is disconnected, results in significant reduction of bandwidth because the MU can use its local cached data. But in order to retain the cached data, it is necessary to identify the cached data that are still valid and should be purged.

A straightforward way of checking cache validity, either at the object level or the group level, is to simply send the object or group ids and their timestamp to the server. But it requires uplink communication cost to send. Instead, it is sufficient to just send T_{lb} with object ids or group ids to the server. This is because, once a MU processes a new IR, all the valid cache entries can be viewed as being time stamped at that moment. The server can check the validity of an object or a group of objects based on T_{lb}, the timestamp of the latest broadcast report received by a MU. The server then compares T_{lb} with the object update timestamp stored in the server, and sends a validity report back to the MU. The algorithm for query processing using simple checking scheme as:

```
for every ob_i ε IR{
 if (ob_i is in cache and ts_d > ts^c) { Invalidate ob_i ; } }
 if (T_lb < (T-w×L)){
   send T_lb and all object ids that are not yet invalidated
   to the server and wait for the validity report;
   Invalidate the cache according to the report;}
 for every item q ε Q_List {
  if all object referenced by q are in the cache
     { process q;}
  else { send missed object ids to the server;}}T_lb = T;
```

Algorithm 2. Query Processing using Simple Checking Algorithm

Once MU receives an IR, it first invalidates its cache according to the IR. Then, when it wakes up from a disconnection and $T_{lb} < (T - w \times L)$, it sends the cached objects that are not yet invalidated to the server for validity checking. Note that since IR contains the object ids that were updated during the past w broadcast intervals, no validity checking is necessary if $T_{lb} \geq (T - w \times L)$. After receiving the validity report from the server, the MU then processes the queries.

These algorithms generate the periodical broadcasting of IR for updating the cache. This may require large amount of bandwidth and energy that leads to the node to be suspended from its network soon. Hence cache consistency problem occurs. But our algorithm is efficient in improving mobile caching, reducing the communication bandwidth and also efficiently utilizing the energy in ad-hoc network.

IV. Proposed Work

A. Cache Invalidation in Ad-hoc Network:

Our work introduces the stateful cache invalidation algorithm. This mainly focuses to optimize the bandwidth utilization and energy consumption in ad-hoc network. The major focus is that every node is having the capability of broadcasting the updated data items. Each node is having the same amount of energy to maintain certain data item. Frequently accessed data items are caches for reducing the bandwidth requirement. Nodes in ad-hoc network contain the following information (gid, id, object, TS). Each node maintains a counter for counting the number of nodes requesting the data item and the request registry for identifying which node is requesting the updated data item. The bandwidth utilization is in the order of (gid, TS) > (id, TS) > (object, TS). TS specify the time value of the latest changes for the item in the group.

Our algorithm specifies three modes of operation for broadcasting the data to invalidate the cache. These modes are named as: Slow mode, Fast mode and Super-fast mode.

a. Slow Mode:

The system is said to be in slow mode when the broadcasting node will be initially broadcasting (gid, TS) and also assumes not too many nodes are requesting the updated objects. The broadcasting node counts the number of request sent by the nodes for the updated object within a threshold period ($Time_{th}$) and keeps track of how many nodes are participated in requesting the object.

b. Fast Mode:

In fast mode the node broadcast (id, TS) while the number of request from the node reaches the specified threshold level ($node_{th\text{-}high}$) and the number of request increases than the lower threshold value ($node_{th\text{-}low}$). At the particular time interval, the node counts the number of request sent by the nodes for broadcasting the IR in fast mode. If the number of request sent by the nodes decreases than higher threshold value ($node_{th\text{-}high}$), the system may enter either into the slow mode or enter into the super-fast mode.

c. Super-fast Mode:

In the super-fast mode the node acts like asynchronous manner for fixed time ($Time_{fixed}$) interval, and send the point to point message to the node whose data has been changed. If the number of request sends by the node increases than the higher threshold value ($node_{th\text{-}high}$), then the node is answered by super-fast mode. After the time ($Time_{fixed}$) is elapsed, the system again enters into the fast mode. Compared to other two modes, the super-fast mode saves the uplink bandwidth efficiently.

This algorithm uses varying time intervals instead of fixed time intervals for broadcasting IRs or object. So we start this algorithm with the long time interval and make the time window half if the number of updates exceeds a threshold i.e., the time interval for slow mode is 20 sec, the time interval for fast mode is 10 sec and the fixed time interval for super-fast mode is 2 sec. And also each node will be able to broadcast IR and receives IR to update its cache.

B. Broadcasting Node:

In an ad-hoc network, each node is responsible for constructing IR at a predefined time period. The broadcasting node follows three modes of operation for generating the invalidation report, to update the cache. Depends on the number of request sent by the node, the respective mode has to be selected to update the cache. The formal description of the algorithm at the broadcasting node is as follows. The notations related to data items and modes of operations are specified in Table1 and Table 2.

Table 1. Data items description

Symbols	Description
Id	id of the data
ID	The set of database id
gid	Group id of the data
D	The set of data item
nid	id of all node
IR	Invalidation Report
TS	Time Stamp
w	Window size

Table 2. Modes of operation

Symbols	Description
L_{th}	The time threshold
L_{low}	Time interval for slow mode
L_{high}	Time interval for fast mode
L_{fixed}	Fixed time interval for super-fast mode
Req_{data}	An id list of data items a node requested from other node
$Cache_{data}$	An id list of data items a node contains in its cache
Reg_{reg}	Request registry for identifying which node is Requesting the data item
$node_{cnt}$	Number of node count
$node_{th\text{-}low}$	the lower threshold number of nodes
$node_{th\text{-}high}$	the higher threshold number of nodes

Table 3. Description of the data at the receiving node

Symbols	Description
Req_i	{d | d has been queried before T_i}
T_l	The timestamp of the last received IR
TS_c	Timestamp of cached data item d
Req_{data}	Request registry for identifying which node is Requesting the data item

(A)Slow Mode:
At interval time T_i construct IR_i as follows:
IR_i = {[gid, TS] | [gid ε {ID}]Λ[T_i- L_{low} ×w<TS< T_i]}
Broadcast IRi, L_{low}; Receive Req_{data};
Compare Req_{data} with its $Cache_{data}$ and Start Timer;
for every node (id ε nid) {
  for every (id($Cache_{data}$) ε Req_{data}) {
    Update ($node_{cnt}$) and get id of node to (Reg_{reg}); }
 if {($node_{cnt}$<=$node_{th\text{-}low}$)Λ(L_{th} =L_{low})} {
    Broadcast d ε D and Clear ($node_{cnt}$) and (Reg_{reg}); }
else if ($node_{cnt}$>$node_{th\text{-}low}$) { Execute Step (B); }
else if ($node_{cnt}$>$node_{th\text{-}high}$) { Execute Step (C); }
else {Refer the next id(Req_{data}) ; } }

(B)Fast Mode:
At interval time T_i construct IR_i as follows:
IR_i = {[id, TS]| [id ε {ID}]Λ[T_i- L_{high} ×w<TS< T_i]}
Broadcast IR_i, L_{high}; Receive Req_{data};
Compare Req_{data} with its $Cache_{data}$ and Start Timer;
for every node (id ε nid) {
  for every (id($Cache_{data}$) ε Req_{data}) {
    Update ($node_{cnt}$) and get id of node to (Reg_{reg}); }
  if {($node_{cnt}$<=$node_{th\text{-}high}$)Λ(L_{th} =L_{high})}{
    Broadcast d ε D and Clear ($node_{cnt}$) and (Reg_{reg}); }
  else if ($node_{cnt}$<$node_{th\text{-}low}$) {Execute Step (A); }
  else if ($node_{cnt}$>$node_{th\text{-}high}$) {Execute Step (C); }
  else {Refer the next id(Req_{data}); } }

C)Super-fast Mode:
At interval time T_i construct IR_i as follows:
IR_i = {[d, TS]| [d ε D]Λ[T_i- L_{high} ×w<TS< T_i]}
Broadcast IR_i, L_{fixed}(Point-to-Point);
Execute Step (B) after L_{fixed} elapsed;

C. Receiving Node:

The receiving node validates its cache based on the received IR. If the node missed the IR, it will wait for the next IR. Then it prepares the list of invalidated data items. Those are sending back to the broadcasting node. Using any one of three modes the updated data items are reported to the receiving node. Upon reception, the node updates its cache. The node serves this data item to answer the query. The formal description of the algorithm at the receiving node is as follows. The notations are described in Table-3.

(A)When the node receives IR_i and L_{th}
for every receiving node [id($Cache_{data}$) ε id(IR_i)]{
  if(L_{th} = L_{fixed}) {
    Download d to its local cache and update;
    Use d to answer the previous query;
    Wait until L_{fixed} elapsed; }
else if(L_{th} = L_{high}) {
  if(T_l<(T_i- L_{th} ×w)) {
    Drop the entire cache or send request to verify the cache;
  for each data item [d, TS_c]in the [$Cache_{data}$] {
      id_{node} = id(IR_i);
      if(([d, TS] ε data(id_{node})) Λ(TS_c < TS) {Invalidate d;}
        for each id received {
          if (id ε Req_{data}(id)) {
            Download d to its local cache and update;
            Update its cache with newly arrived data;
            Use d to answer the previous query; }}
          if d is an invalid cache item {
            Download d into local cache update;}T_l=T_i;
        if($Req_i \neq 0$) then request(Req_i); }}}
else if(L_{th} = L_{low}) {
  if(T_l<(T_i- L_{th} ×w)) {
  Drop the entire cache or send request to verify the cache;
    for each data item [d, TS_c]in the [$Cache_{data}$] {
      id_{node} = id(IR_i);
      if(([d, TS] ε data(id_{node})) Λ(TS_c < TS) {Invalidate d;}
        for each id received {
          if (id ε Req_{data}(id)) {
            Download d into local cache update;
            Use d to answer the previous query; } }
          if d is an invalid cache item {
            Download d into local cache;}
            T_l=T_i;
        if($Req_i \neq 0$) then request(Req_i); } }

(B) request(Req)
Req_{data} = 0; For each d ε Req {
  If d is valid in the [$Cache_{data}$]
  { Use cache's value to answer the request ;}

else { Req_{data} = Req_{data} U d;}
Send request (Req_{data}) to the broadcasting node;

Each node is broadcasting the data, which resides in its cache. The receiving node either updates its cache, or sending the request back for some particular data item, thereby the requesting node has been answered by proper mode of operation for the application. Thereby we can efficiently increase network communication and throughput. In this algorithm, we can also minimize the number of request of the system, compared to other algorithm.

V. Performance Analysis

We developed a simple model to analyze the performance of the proposed cache invalidation scheme in ad-hoc network. Most existing IR-based cache invalidation algorithms [3],[5],[7],[15] are proposed to deal with the long disconnection problem. Since they are based on the TS algorithm [15], but our major concern is not to deal with the long disconnection problem, we only compare the performance of the TS algorithm and the proposed algorithm.

The performance of MEECIA algorithm has been improved by implementing three modes of operation. These modes are evaluated based on the metrics such as energy consumption and query delay. The energy consumption contains three components: the energy consumed on cache invalidation, uplink request and to download the desired data and updating the cache. Based on this the bandwidth consumed when the number of request by the node increases at the appropriate level. Query delay occurs because of the waiting time increases for updating the cache. In order to evaluate the efficiency of various invalidation algorithms, a simulation model has been developed.

A. Simulation Model and System Parameters:

The simulation model consists of the wireless nodes without any base station. Each one is maintaining certain data items in its cache. The data base of the node can only be updated when the queries are made by other nodes. The simulation model has been implemented in Java based simulation package.

The nodes are broadcasting the IRs periodically to other nodes. The broadcasting nodes assign the highest priority to IR broadcasting, and equal priorities to the rest of the messages. All other messages are served on FCFS basis. The node defers IR broadcasting, until it finishes the current packet transmission. However, the next IR should be broadcasted at its originally scheduled time interval.

Each node is generating the read-only queries for updating its cache. If the data item is needed by the application, the node checks whether the data item is presented in its local cache memory. If the referenced data item is not in the cache, the item ids are sent to that corresponding node for fetching the data items. The node cache management follows the LRU (Least Recently Used) replacement policy [14]. In our algorithm, if there are invalid data items, the node replaces the oldest invalid item. If there is no invalid cache item, the node replaces the oldest valid cache item. The difference is due to the fact that the node in our algorithm can download data from the broadcast channel.

B. Simulation results:

Experiments were run using different workloads and system settings. The performance analysis presented here is designed to compare the effects of different workload parameters such as energy consumption level and query delay on the relative performance of the TS algorithm and MEECIA algorithm. Our goal is to minimize the number of node request and the reduction of query delay in mobile environment for the limited battery power.

a. Energy Consumption:

In the mobile environment, each node has certain energy level for transmitting and receiving the data. The energy of the node reduces, when the node requests the data item and the waiting time for updating its cache increases [12]. A detailed explanation is given below by comparing our algorithm with TS algorithm.

According to our algorithm the node has to perform three modes of operations. Based on this, we can efficiently utilize the energy when the number of request is increased. The fig 3 shows the energy consumption of the node while the node request varies from 2 to 8. At first all node has the same energy level of 1000. Energy is consumed when the node request the data. From the result we can say that our algorithm consumes less energy than TS algorithm.

The fig 4a shows the energy consumption range of both algorithms. At every second the node energy is consumed depends on its communication pattern. Say for example at 5^{th} sec. the energy level of our algorithm is making the communication effectively. Otherwise, the node may miss its IR and cache consistency problem will occur. The fig 4b shows the energy consumption of each node. Once the node updates its cache either by MEECIA algorithm or TS algorithm the energy level of the node is measured. Energy utilization of each node in TS algorithm is greater than the MEECIA algorithm (see fig 4b). From the simulation results based on the energy consumption parameter the MEECIA algorithm is considered as a best energy efficient algorithm for the nodes in ad-hoc network.

b. The Query Latency:

We measure the query delay as a function of mean query arrival time, and the mean update arrival time. Comparing with TS algorithm, our algorithm significantly improves the query delay. Each node generates the query according to the mean query generate time. If the queried data item is present in its local cache, the node serves the data from its local cache or the query has to be served by other nodes. So the node is waiting for processing the query by observing the IR. The IR contains the limited amount of data items. It cannot answer for all queried items at IR interval. So some queries are answered with delay.

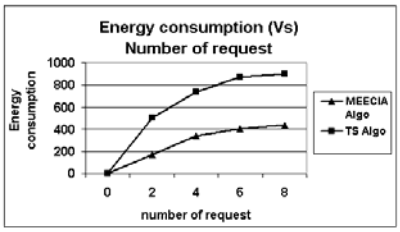

Fig 3. Energy consumption parameter on request

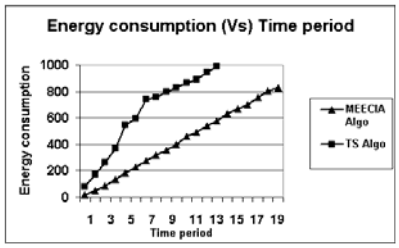

Fig 4a. Energy consumption verses Time Period

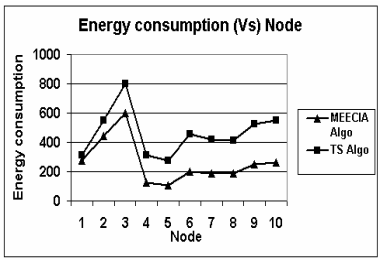

Fig 4b. Energy consumption of each node

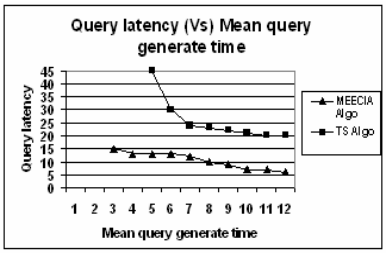

Fig 5a. Query latency as a function of mean query generate time

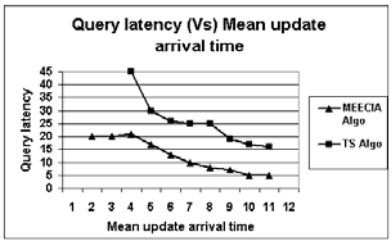

Fig 5b. Query latency as a function of mean update arrival time

The fig 5a shows the query delay as a function of the mean query generated time. When the query generate time is lower than 5sec, the query delay of the TS algorithm becomes infinitely long. However, in MEECIA algorithm even when the query generates time reaches 3sec, the query delay becomes very less compared to TS algorithm.

In our algorithm, without considering the priority request, a node cannot answer the query until it receives the IR. As shown in fig 5b, as the mean update arrival time increases, the cache hit ratio increases and the query delay decreases. Since our algorithm has high cache hit ratio than the TS algorithm, the query delay of our algorithm is shorter than TS algorithm. Comparing our algorithm with TS algorithm, our algorithm significantly improves the energy and reduces the query delay.

VI. Conclusion

Thus the IR-based cache invalidation has received considerable attention due to its scalability. Most of the existing IR-based cache invalidation algorithm concentrates on dealing with the long disconnection problem, and not much work has been reported to address the demerits associated with the IR-based algorithm. In this paper, we propose a cache invalidation algorithm for improving the energy consumption and query latency by reducing the number of request sends by the node in an ad-hoc network. Our algorithm implements three modes of operation for reducing the request of the nodes and responding quickly for reducing the query delay. As a result, most unnecessary request and broadcast bandwidth can be avoided and the energy can be saved. The various simulation results show that the proposed algorithm can reduce the number of request by the node, improve the energy consumption and also reduce the query latency compared to the TS algorithm.

An LRU based cache replacement is used in this paper. In future, we can study the impact of various replacement algorithms on the performance of MEECIA algorithm.

References

[1] T. Imielinski and B. R. Badrinath. " wireless computing: challenges in data management" *Communications of the ACM*, 37(10), Oct. 1994.

[2] K. Wu, P.S. Yu, and M. Chen, "Energy- Efficient Caching for Wireless MobileComp," *Proc. of IEEE ICDE*,New Orleans, pp.336-343, Feb.'96.

[3] G. Cao, "A Scalable Low Latency Cache Invalidation Strategy for Mobile Environments," *Proc. of ACMMOBICOM*, Boston, MA, 2000

[4] G. Cao, "On Improving the Performance of Cache Invalidation in Mobile Environments," *Mobile Networks and Applications,* vol.7, Pp.291-303, 2002

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

[6] Q. Hu and D.K. Lee, "Cache Algorithms Based on Adaptive Invalidation Reports for Mobile Environments," *Cluster Computing*, vol.1,pp. 39-50, 1998.

[7] J. Jing, A. Elmagarmid, A. Heal, 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.

[8] 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 TPDS*, vol.12, 2001.

[9] K. Tan, J. Cai and B. Ooi, "An Evaluation of Cache Invalidation Strategies in Wireless Environments" *IEEE TPDS*, vol.12, no.8, pp.789-807, Aug. 2001.

[10] Z. Wang, et al, "SACCS: Scalable Asynchronous Cache Consistency Scheme for Mobile Environments," *Proc. Of Workshop on Mobile and Wireless Networks*, 2003.

[11] Jun Cai and Kian-Lee Tan, "Energy-efficient selective cache invalidation" *Proc of Wireless Network*, vol.5, 489- 502, 1999

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

[13] Mark Kai Ho Yeung and Yu-Kwong Kwok,"Wireless Cache Invalidation Schemes with Link Adaptation and Downlink Traffic" *IEEE Transactions on Mobile Computing,* vol.4, No. 1, February 2005

[14] A. Kahol, S. Khurana, S. Gupta, and P. Srimani,"An Efficient Cache Management Scheme for Mobile Environment," *Proc. 20thInt'l Conf.Distributed Computing Systems*, pp. 530-537, Apr. 2000.

[15] D. Barabara and T. Imielinski, "Sleepers and Workaholics: Caching Strategies in Mobile Environments," *VLDB Journal*, 4, 567-602,1995.