Cooperative Caching in Mobile Ad Hoc Networks Through Clustering

NAROTTAM CHAND AND NAVEEN CHAUHAN
Department of Computer Science & Engineering
National Institute of Technology
Hamirpur (HP) – 177 005
INDIA
{nar.chand, naveenchauhan.nith}@gmail.com

Abstract: - Constrained communication environment in mobile ad hoc networks (MANETs) due to insufficient wireless bandwidth and limited battery energy makes data retrieval a challenging problem. Due to frequent network disconnections in MANETs data availability is lower than traditional wired networks. Caching of frequently accessed data in such environment is a potential technique that can improve the data access performance and availability. Cooperative caching, which allows the sharing and coordination of cached data among clients, can further explore the potential of caching techniques. In this paper, we present a scheme, called global cluster cooperation (GCC) for caching in mobile ad hoc networks. In this scheme network topology is partitioned into non-overlapping clusters based on the physical network proximity. This approach fully exploits the pull mechanism to facilitate cache sharing in a MANET. Simulation experiments show that GCC mechanism achieves significant improvements in cache hit ratio and average query latency in comparison with other caching strategies.

Key-Words: - MANETs, cluster, cooperative caching, cache state node, global cache state, hit ratio

1 Introduction

Recent explosive growth in computer and wireless communication technologies has led to an increasing interest in mobile ad hoc networks (MANETs) which are constructed only from mobile nodes. The interest in developing mobile wireless ad hoc networking solutions has been due to their flexibility, ease of deployment and potential applications such as battlefield, disaster recovery, outdoor assemblies, etc. In MANETs, every mobile node (MN) plays the role of a router for communication with other mobile nodes. Even though there is no dearth of storage space in present scenario, it is always better to utilize the resources optimally. With caching, the data access delay is reduced since data access requests can be served from the local cache, thereby obviating the need for data transmission over the scarce wireless links. However, caching techniques used in one hop mobile environment may not be applicable to multihop ad hoc environment since the data or request may need to go through multiple hops. Variable data size, frequent data updates, limited client resources, insufficient wireless bandwidth and clients' mobility make cache management a challenging task in mobile ad hoc networks. As mobile nodes in ad hoc networks may have similar tasks and share common interest, cooperative caching, which allows the sharing and coordination of cached data among multiple nodes, can be used to reduce the bandwidth and power consumption.

Most of the previous researches in ad hoc networks focus on the development of dynamic routing protocols that can improve the connectivity among mobile nodes. Although routing is an important issue in ad hoc networks, other issues such as data access are also very important since the ultimate goal of using such networks is to provide information access to mobile nodes [1, 5, 8, 9]. One of the most attractive techniques that improves data availability is caching.To date there are some works in literature on cooperative caching in ad hoc networks, such as consistency [1, 2, 3] and placement [4]. In this paper, we investigate the data retrieval challenge of mobile ad hoc networks and propose a novel scheme, called global cluster cooperation (GCC) for caching. The goal of GCC is to reduce the cache discovery overhead and provide better cooperative caching performance. GCC partitions the whole MANET into equal size clusters based on the geographical network proximity as shown in Fig. 1. To enhance the system performance, within a cluster, individual caches interact with each other and combined result is a larger cumulative cache. In each cluster, GCC dynamically chooses a "super" node as cache state node (CSN), to maintain the global cache state (GCS) information of different nodes within the network. The GCS for a client is the list of cached items along with their time to live (TTL) field. Simulation experiments are performed to evaluate the proposed GCC caching scheme and compare it with existing strategies in the MANETs.

The rest of the paper is organized as follows. Section 2 describes the system model. Clustering strategy employed in GCC is presented in Section 3. Section 4 describes the proposed GCC caching scheme for data retrieval. Section 5 is devoted to performance evaluation. Section 6 concludes the paper.

2 System Model

We consider a MANET environment where mobile nodes access the data items stored by other nodes. A mobile node that holds the original copy of a data item is called data center. A data request initiated by a node is forwarded hop-by-hop along the routing path until it reaches the data center and then it sends back the requested data. Mobile nodes frequently access the data, and cache some data locally to reduce network traffic and data access delay. We assume that weak cache consistency is required and time to live (TTL) based consistency model is used [7]. As mobile nodes may not have sufficient cache storage e.g. in case of multimedia data, caching strategy is to be devised efficiently.We make the following assumptions and introduce some common definitions.

Assumptions

- The network is divided into several non-overlapping clusters where in each cluster; a node could be in one of two roles: cache state node (CSN) or ordinary node. CSN is a node that maintains global cache state (GCS) information of different clusters, whereas ordinary node maintains cluster cache state (CCS) information corresponding to all the nodes in its cluster. The ordinary node also maintains local cache as per its requirement.
- Unique host identifier is assigned to each mobile node in the system. The system has total of M hosts and MH_i ($1\leq i\leq M$) is a host identifier. The set of one hop neighbors of a host MH_i is denoted by MH_i^1.
- Nodes can physically move, so there is no guarantee that a neighbor at time t will remain in the cluster at time $t+\tau$. The devices might be turned off or on at any time, so the set of active nodes varies with time and has no fixed size.
- The set of data items is denoted by $D = \{d_1, d_2, \ldots\ldots d_n\}$, where N is the total number of data items and d_j ($1\leq j\leq N$) is a data identifier. D_i denotes the actual data for id d_i. Size of data item d_i is s_i. The original of each data item is the data source/center.
- Each mobile node has a cache space of C bytes.
- Each data item is periodically updated at source. After a data item is updated, its cached copy (maintained on one or more hosts) may become invalid.

Definitions

Grids. The mobile ad hoc network topology is divided into equal size parts called grids. Grids are formed to cluster the ad hoc network. Grid size is such that two nodes located in adjacent grids (horizontally, vertically & diagonally) are at a distance of one hop. A grid consists of a cache state node (CSN) and a number of ordinary nodes, and each node belongs to only one grid.

CSN. In each cluster area, a super node is selected to act as the cache state node (CSN), which is responsible for recording the global cache state (GCS) of all clusters in the network area. When a node in any cluster stores/deletes some data item into/from its cache, it sends the information to its CSN so that the corresponding GCS can be updated. Since a CSN handles additional load of GCS maintenance, it must be relatively stable and capable to support this responsibility. In order to ascertain such qualification of a node, we assign to each node a candidacy factor to qualify as CSN, which is a function of node staying period in the cluster, available battery power and memory space. A node with the highest candidacy factor is elected as CSN.

CCS. Cluster cache state (CCS) for a node is the list of cached items along with time to live (TTL) field. When cache content at a node changes, it broadcasts the information to all the nodes within its cluster. CCS is maintained at each node of the cluster.

GCS. Global cache state (GCS) for a network is the list of cached items maintained at different clusters along with their time to live (TTL) field. When cache content changes, it sends the information to the CSN and the CSN updates the corresponding GCS of the cluster. The information at other CSN is updated periodically after certain time period.

3 Cluster Handling

The clustering algorithm divides the network topology into predefined equal sized geographical grids called clusters. The problem of finding an optimal clustering is out of the scope of this paper. For the sake of simplicity, we assume that clustering phase gives a partition of the network into grids. However, any clustering algorithm can be used as our GCC caching scheme is compatible with any non-overlapping clustering strategy. Grid size captures the maximum distance between two nodes in adjacent clusters (horizontally, vertically & diagonally). It is ensured that the coordinators in adjacent grids are within the transmission range of each other. Network area is assumed to be virtually extended such that boundary clusters also have same size as other clusters. Beginning with the lower left cluster, the clusters are named as 1, 2, ..., in a column-wise fashion. In each cluster area a "super" node is selected to act as CSN, which is responsible for maintaining the global cache state (GCS) information of different clusters within the network. GCS for a network is the list of data items along with their TTL stored in its cache. When a node caches/replaces a data item, its GCS is updated at the CSN.

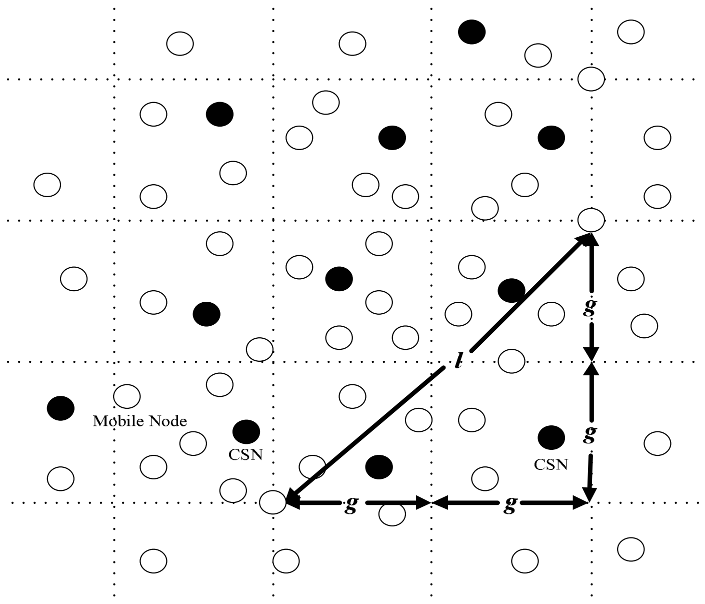


Fig. 1. Partitioning of MANET into clusters

It may be noted that CSN is quite different from conventional "clusterhead" that is used to forward requests for a group of nodes. In each cluster of such a clusterhead networked system, all the requests from/to a client are forwarded by the clusterhead, which tends to make it a bottleneck and/or a point of failure when the system has high network density. Unlike this, CSN works only as GCS holder to save the information about the cached items by different clients belonging to the entire network partitioned into clusters, and provides additional service during cache discovery, admission control and replacement. Compared to clusterhead, CSN deals with much less workload and does not have to as powerful as a clusterhead. In the proposed clustering method, grid side g is a key factor to the clustering. A node in one cluster can communicate with a node in adjacent cluster (horizontally, vertically & diagonally), if the required side g is derived as follows. The maximum distance l between two hosts MH_i and MH_j is given as: $\sqrt{2g^2+(2g)^2}=\sqrt{8}g$. To ensure one hop communication among the hosts within a cluster, l should be less than or equal to transmission range (r), i.e. $\sqrt{8}g \le r$. Hence for $g \le r/\sqrt{8}$, all MNs in a cluster can connect to one another in one hop communication.

4 GCC Caching

This section describes our global cluster cooperation (GCC) caching scheme that exploits the above described clustering mechanism for data retrieval in MANETs. The design rationale of GCC is that, for a mobile client, all other mobile clients within its cluster domain form a cooperative cache system for the client since local caches of the clients virtually form a cumulative cache. In GCC, when a client suffers from a cache miss (called local cache miss), the client will look up the required data item from the cluster members. Only when the client cannot find the data item in the cluster members' caches (called cluster cache miss), it will request the CSN which keeps the global cache state (GCS) and maintains the information about the node in the network which has copy of desired data item. If a cluster other than requesting node's cluster has the requested data (called remote cache hit), then it can serve the request without forwarding it further towards the server. Otherwise, the request will be satisfied by the server. For each request, one of the following four cases holds:

Case 1: Local hit. When a node requires a data and finds it in the local cache.

Case 2: Cluster hit. When a node requires the data, it checks its local cache, in case of local miss, node consults its CCS which is maintained by this node only, to check whether data is available in one of the neighboring nodes within the cluster.

Case 3: Remote hit. When the requested data item is not stored by a client within the cluster of the requester. The requester checks with CSN which is maintaining GCS and then returns the address of the client that has cached the data item.

Case 4: Global hit. When the data is not found even remotely data is retrieved from data center.

Based on the above idea, we propose a cache discovery algorithm to determine the data access path to a node having the requested cached data or to the data source. In Fig. 2, let us assume MH_i sends a request for a data item d_x and MH_k is located along the path through which the request travels to the data source MH_s, where $k \in \{a, c, d\}$. The discovery algorithm is described as follows:
When MH_i needs d_x, it first checks its own cache. If the

data item is not available in its local cache, it checks with CCS which is maintained by MH_i to see whether any of neighboring node in the cluster has a copy of desired data. If it is not available at cluster level, it sends a lookup packet to the CSN MH_j in its cluster. Upon receiving the lookup message, the CSN searches in the GCS for the requested data item. If the item is found, the CSN replies with an ack packet containing id of the client who has cached the item. MH_i sends a request packet to the client whose id is returned by MH_j and the client responds with reply packet that contains the requested data item. When a node/MH_s receives a request packet, it sends the reply packet to the requester. The reply packet containing item id d_x, actual data D_x and TTL_x, is forwarded hop-by-hop along the routing path until it reaches the original requester. Once a node receives the requested data, it triggers the cache admission control procedure to determine whether it should cache the data item.

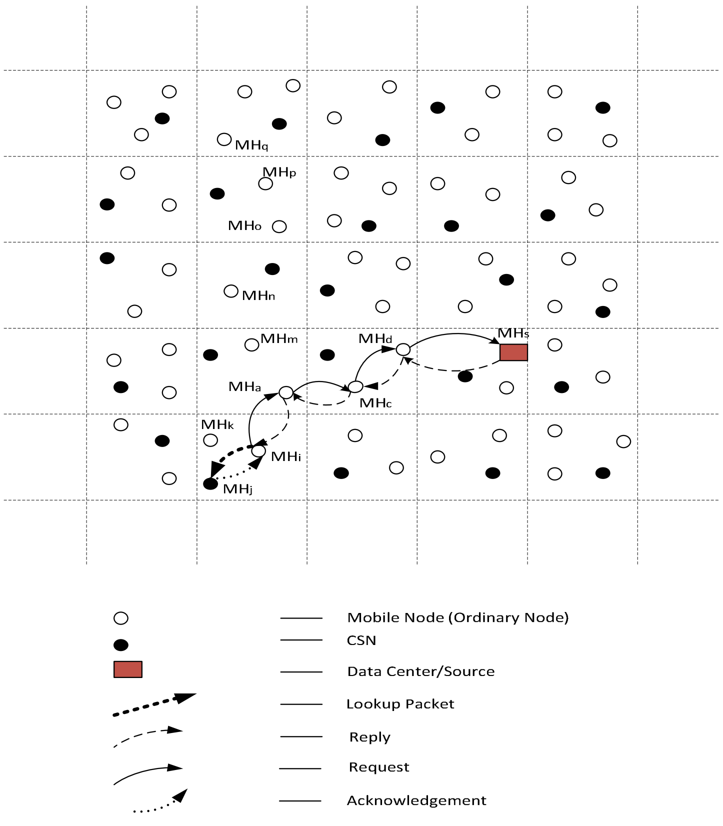


Fig. 2. Request packet from client MH_i to server

Cache admission control decides whether a data item should be brought into cache. Inserting a data item might not always be favorable because incorrect decision can lower the probability of cache hits. For example, replacing a data item that will be accessed soon with an item that will be accessed in near future degrades performance. In GCC, the cache admission control allows a client to cache a data item based on the location of data source or other client that has the requested data. If the origin of the data resides in the same cluster of the requesting client, then the item is not cached, because it is unnecessary to replicate data item in the same cluster since cached data can be used by closely located hosts. In general, same data items are cached in different clusters without replication.

The GCC caching uses a simple weak consistency model based on time to live (TTL), in which a client considers a cached copy up-to-date if its TTL has not expired. The client removes the cached data when the TTL expires. A client refreshes a cached data item and its TTL if a fresh copy of the same data passes by.

5 Performance Evaluation

In this section, we evaluate the performance of GCC through simulation. The simulation area is assumed of size 1500m x 1500m. The clients move according to the random waypoint model [6]. The time interval between two consecutive queries generated from each client follows an exponential distribution with mean T_q. Each client generates accesses to the data items following Zipf distribution with a skewness parameter θ. There are N data items at the server. Data item sizes vary from s_{min} to s_{max} such that size s_i of item d_i is, $s_i = s_{\min} + \lfloor random().(s_{\max} - s_{\min} + 1)\rfloor$, i = 1, 2,... N, where random() is a random function uniformly distributed between 0 and 1. The simulation parameters are listed in Table 1. For performance comparison with GCC, two other schemes non-cooperative (NC) caching and CacheData [1, 3] are also simulated. In NC, received data are cached only at query node and locally missed data items are always fetched from the origin server. In our experiments, the same data access pattern and mobility model are applied to all the three schemes. All the schemes use LRU algorithm for cache replacement. We evaluate two performance metrics:

Average query latency (T_{avg}). The query latency is the time elapsed between the query is sent and the data is transmitted back to the requester, and average query latency (T_{avg}) is the query latency averaged over all the queries.

Byte hit ratio (B). Byte hit ratio is defined as the ratio of the number of data bytes retrieved from the cache to the total number of requested data bytes. Here byte hit ratio (B) includes local byte hit (B_{local}), cluster byte hit ($B_{cluster}$) and remote byte hit (B_{remote}). If n_{local}, $n_{cluster}$ and n_{remote} denote the number of local byte hits, cluster byte hits and remote byte hits respectively out of total n_{total} requested bytes, then

$B_{local} = n_{local}/n_{total} \times 100\%$, $B_{clutser} = n_{cluster}/n_{total} \times 100\%$, and $B_{remote} = n_{remote}/n_{total} \times 100\%$

Table 1. Simulation parameters

Parameter	Default Value	Range
Database size (N)	1000 items	
s_{min}	10 KB	
s_{max}	100 KB	
Number of clients (M)	70	50~100
Client cache size (C)	800 KB	200~1400 KB
Client speed (v_{min}~v_{max})	2 m/s	2~20 m/s
Bandwidth (b)	2 Mbps	
TTL	5000 sec	200~10000 sec
Pause time	300 sec	
Mean query generate time (T_q)	5 sec	2~100 sec
Transmission range (r)	25 m	25~250 m
Skewness parameter (θ)	0.8	0~1

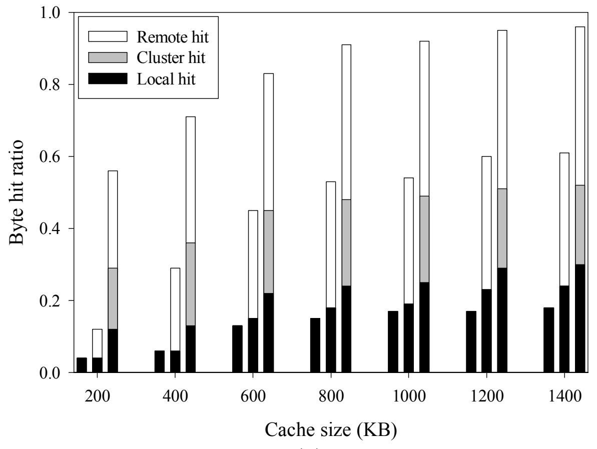
Effects of cache size

Fig. 3 shows the effects of cache size on byte hit ratio and average query latency by varying the cache size from 200 KB to 1400 KB. Fig. 3(a) and 3(b) show that all schemes exhibit better byte hit and average query latency with increasing cache size. This is because more required data items can be found in local cache as the cache gets larger. From Fig. 3(b), we can see that the GCC scheme performs much better than NC scheme. Because of the high byte hit ratio due to cluster cooperation, the proposed scheme also performs better than CacheData. When the cache size is small, more required data could be found in local+cluster+global cache for GCC as compared to CacheData which utilizes only the local cache. Because the hop count of cluster data hit is one and is less than the average hop count of remote data hit, GCC scheme achieves lower average query latency. As the cache size is large enough, the nodes can access most of the required data items from local and cluster cache, so reducing the query latency. It is worth noting that GCC reaches its best performance when the cache size is 800 KB. This demonstrates its low cache space requirement.

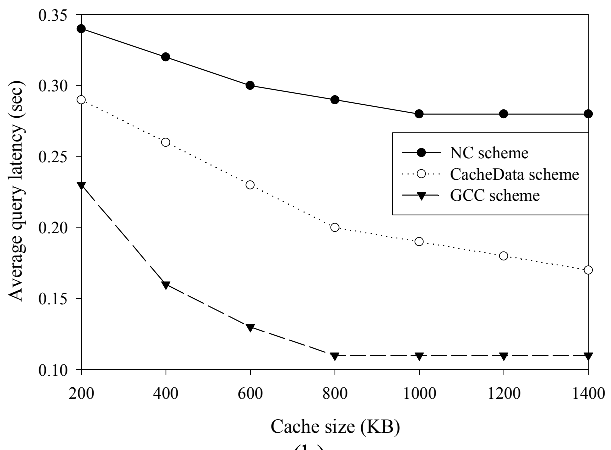
Effects of mobility

Fig. 4 shows the comparison of caching strategies, where each node is moving with a speed uniformly distributed between 0 and a given value along x-axis. We vary the maximum speed of nodes from 2, 4, 8, 12, 16, to 20 m/sec. From Fig. 4(a), we can see that local byte hit ratio in NC is not affected due to client mobility. The local byte hit ratio of CacheData and GCC decreases slightly as the mobility speed increases because some of the invalid data items may not be refreshed before their TTL expires. The remote byte hit ratio of CacheData decreases with an increase in mobility speed. The cluster and remote byte hit ratio decreases in GCC with increasing speed. This is because, when every node is moving with very high speed, the GCC recorded for each node with its CSN does not help much in the near future, since the nodes are moving arbitrarily.

From Fig. 4(b), we see that performance of all the caching strategies degrades with increasing mobility. This is due to overheads caused by mobility induced route failures and route re-computations. If mobility increases, the frequency of nodes with different data affinity leaving/joining a cluster increases thus degrading the GCC caching performance in terms of average query latency.

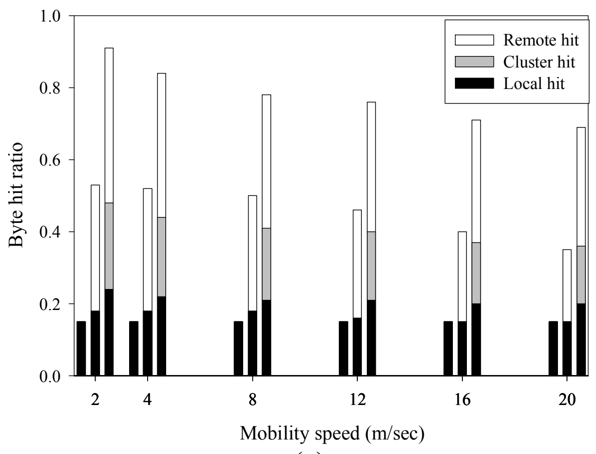


(a)

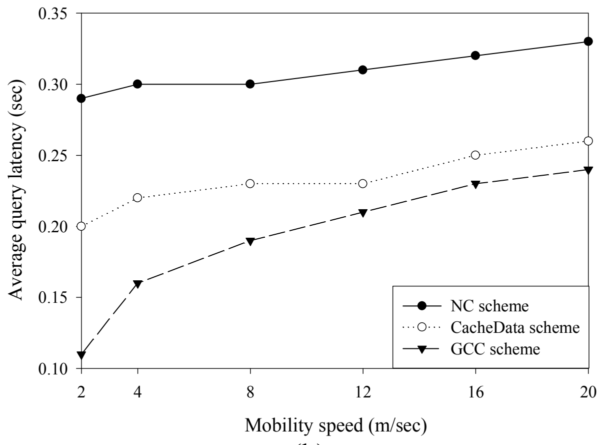


(b)

Fig. 3. Effects of cache size on (a) byte hit ratio, and (b) average query latency



(a)



(b)

Fig. 4. Effects of mobility on (a) byte hit ratio, and (b) average query latency

6 Conclusions

This paper presents a cooperative caching strategy named GCC for MANETs. This strategy is unique that in a cluster, the information about what data all other nodes in different clusters are storing is maintained. All this is possible due to the emergence of powerful mobile nodes along with advances in wireless communication technology. As there is no dearth of storage and computing capabilities in mobile nodes, GCC fits best in present scenario. Simulation results show that the proposed scheme reduces the message overheads and enhances the data accessibility as compared to other strategies. We hope that our work will stimulate further research on cooperative cache based data access by considering various issues such as cooperative cache replacement, strong cache consistency, prefetching, etc.

References:

[1] Yin L. and Cao G.. Supporting Cooperative Caching in Ad Hoc Networks. *IEEE Transactions on Mobile Computing*, Vol. 5, No. 1, 2006, pp. 77-89.

[2] Chand N., Joshi R.C. and Misra, M. A Zone Co-operation Approach for Efficient Caching in Mobile Ad Hoc Networks. *International Journal of Communication Systems*, Vol. 19, No. 9, 2006, pp. 1009-1028.

[3] Chiu G.M. and Young, C.R. Exploiting In-Zone Broadcast for Cache Sharing in Mobile Ad Hoc Networks. *IEEE Transactions on Mobile Computing*, Vol. 8, No. 3, 2009, pp. 384-397.

[4] Tang B., Gupta H. and Das S.R. Benefit Based Data Caching in Ad hoc Networks, *IEEE Transactions on Mobile Computing*, 2009, pp. 289-303.

[5] Hara T. and Madria S. K. Data Replication for Improving Data Accessibility in Ad Hoc Networks. *IEEE Transactions on Mobile Computing*, Vol. 7, No. 3, 2008, pp. 289-304.

[6] Bettstetter C., Resta G. and Santi P. The Node Distribution of the Random Waypoint Mobility Model for Wireless Ad Hoc Networks. *IEEE Transaction on Mobile Computing*, Vol. 2, No. 3, 2003, 257-269.

[7] Chow C.Y., Leong H.V. and Chan A. Peer-to-Peer Cooperative Caching in Mobile Environments. *Proceedings of the 24th International Conference on Distributed Computing Systems*, 2004, pp. 528-533.

[8] Lim S., Lee W.C., Cao G. and Das C.R. A Novel Caching Scheme for Improving Internet Based Mobile Ad Hoc Networks Performance. *Elsevier Journal of Ad Hoc Networks*, Vol. 4, No. 2, 2006, pp. 225-239.

[9] Lim S., Lee W.C., Cao G. and Das, C.R. (2004). Performance Comparison of Cache Invalidation Strategies for Internet Based Mobile Ad Hoc Networks. *Proceedings of IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, 2004, pp. 104-113.