Global Cluster Cooperation Strategy in Mobile Ad Hoc Networks

*Naveen Chauhan[1], Lalit K. Awasthi[1], Narottam Chand[1], R.C. Joshi[2] and Manoj Misra[2]

[1]Department of Computer Science and Engineering
National Institute of Technology, Hamirpur – 177005, India
[2]Department of Electronics and Computer Engineering
Indian Institute of Technology, Roorkee – 247667, India

Abstract: ***MANETs or mobile ad hoc networks are a form of wireless networks which do not require a base station for providing network connectivity. Mobile ad hoc networks have many characteristics which distinguish them from other wireless networks. Frequent network disconnection is one among various characteristics, due to which data availability is lower than traditional wired networks. Cooperative caching helps MANETs in alleviating from the situation of nonavailability of data. In this paper we have proposed a cache cooperation strategy named global cluster cooperation (GCC) which is based on clusters. This approach fully exploits the pull mechanism to facilitate cache sharing in a MANET. We have evaluated the performance of our strategy using simulation and compared with existing cooperative caching schemes****.***

Keywords: MANETs, cluster, cooperative caching, cache state node, global cache state.

I. INTRODUCTION

Due to information overflow, people can no longer be disconnected from their information systems. Caching plays a vital role in providing access of data to the information systems in case of disconnection. This is a well establish way of providing faster data in the area web caching, proxy servers and browsers [8]. With the advent of mobile ad hoc networks (MANET), which is demand based infrastructure less network, being resource poor, caching plays a pivotal role in making MANETs a success in many applications like rescue operations, military operation, etc. A mobile node (MN) is envisioned to be equipped with more powerful capabilities, like sufficient storage space, more processing power etc. 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 multi-hop 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.

To date there are some works in literature on cooperative caching in ad hoc networks, such as consistency [1, 3] and placement [4]. To the best of our knowledge, only few of previous works [2, 3, 6] have exploited clustering as caching mechanism in MANETs. Cooperative caching has been studied in web environment [8], but efficient cache management is still a hot research area in MANETs. CoCa, a cooperative caching protocol [9] have been proposed, which reduces the number of server requests as well as number of cache miss by sharing the cache contents. Further built on the CoCa framework a grou[p based cooperative caching scheme called GroCoCa has been proposed in [10] in which a centralized incremental clustering algorithm is adopted by taking into consideration node mobility and data access pattern. GroCoCa improves system performance at the cost of extra power consumption. Chiu et al. [3] proposed two protocols IXP and DPIP. In IXP each node share its cache contents with the nodes in its zone. The disadvantage of the IXP protocol is that when a node enters into a new zone, the nodes of the new zone are not aware about the cache contents of the new entrant.

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 (see Figure 1). To enhance the system performance, within a cluster, individual caches interact with each other and within a network, the designated CSN of clusters interact with each other such that 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 ad hoc networks.

The rest of the paper is organized as follows. Clustering strategy employed in GCC is presented in Section II. Section III describes the proposed GCC caching scheme for data retrieval.

Section IV is devoted to performance evaluation. Section V concludes the paper.

II. CLUSTER HANDLING

Our 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 and 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 left lower 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 its network domain. 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.

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. If g is set to $r/\sqrt{8}$, all clients in a cluster can connect to one another in one-hop communication. Where r is transmission range of a client.

In GCC, a typical cluster consists of a CSN and a number of clients, and a client only belongs to one cluster. Since a CSN is expected to handle additional load in the system, 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 be CSN, which is function of node staying period in the cluster and available battery power. A node with the highest candidacy factor is elected as CSN.

III. GLOBAL CLUSTER COOPERATIVE (GCC) CACHING

The design rationale of GCC is that, there is no dearth of storage space in present scenario, so the information regarding the cached contents of various clients in a cluster would be kept with each node in the cluster. 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 nodes' 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 found 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. Assume that MHi denotes mobile node/client i. In Figure 1, let us assume MHi sends a request for a data item dx and MHk is located along the path through which the request travels to the data source MHs, 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/MHs 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.

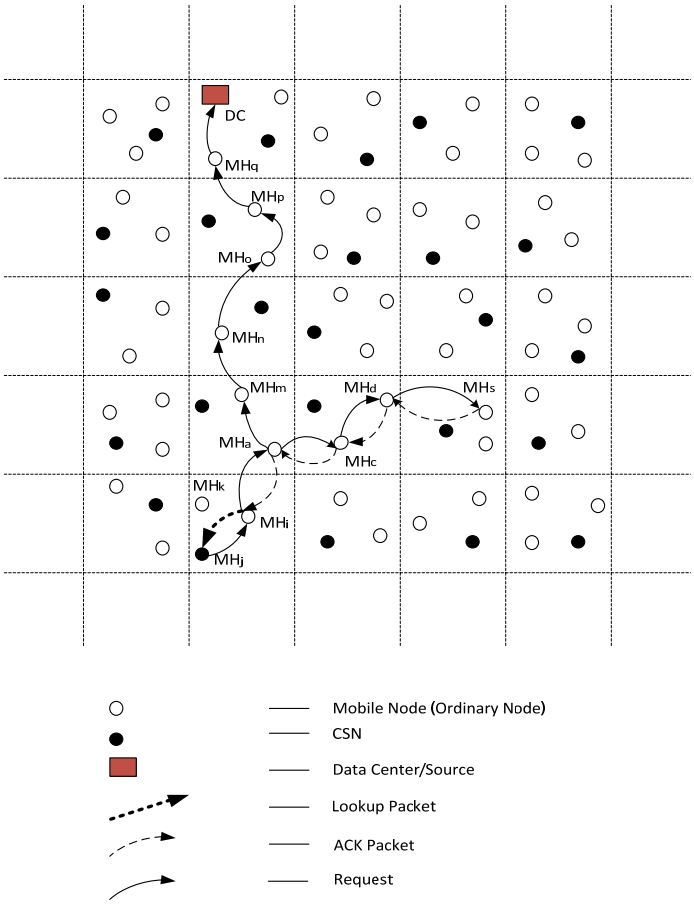

Figure 1. Request packet from client MHi to data source

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. Figure 2 shows the behavior of GCC caching strategy for a client request.

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.

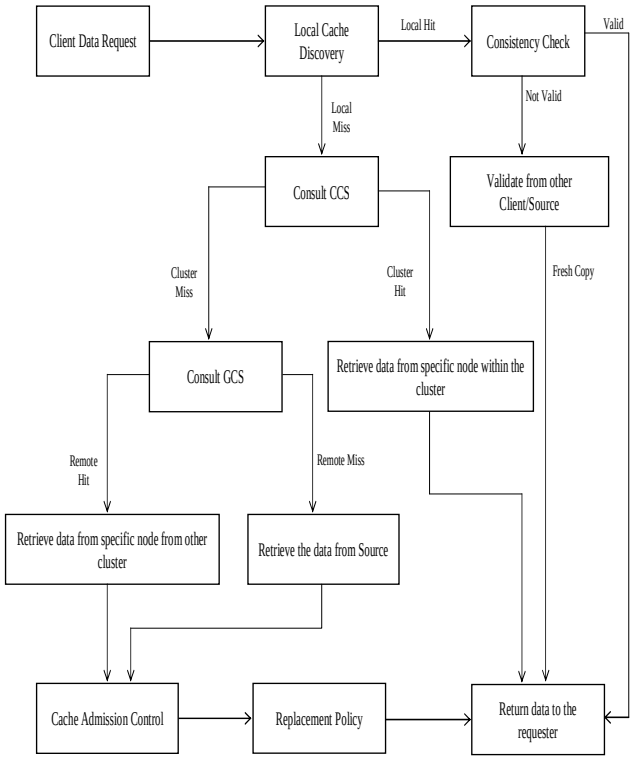

Figure 2. Service of a client by GCC caching strategy

IV. SIMULATION RESULTS

The simulation area is assumed of size 1500 m x 1500 m The clients move according to the random waypoint model [7]. The time interval between two consecutive queries generated from each client follows an exponential distribution with mean Tq. 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 si 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 I. For performance comparison with GCC, two other schemes non-cooperative (NC) caching and CacheData [1, 3] are also implemented. 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.

TABLE I

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

a. Effects of cache size

Figure 3 and Figure 4 show the effects of cache size on average query latency and message overhead by varying the cache size from 200 KB to 1400 KB. From Figure 3, 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 much better than CacheData. When the cache size is small, more required data could be found in local+cluster cache for CC 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.

Figure 4 shows that GCC performs much better than NC and CacheData in terms of message overhead. The reason is that due to cache cooperation among clusters GCC gets data from nearby clusters instead of far away data source. Therefore, the data requests and replies need to travel smaller number of hops and mobile nodes need to process lower number of messages. As the cache size grows, the byte hit ratio of GCC increases and its message overhead decreases.

b. Effects of mean query generate time

Figure 5 shows the average query latency as a function of the mean generate time Tq. The GCC scheme performs better than NC and CacheData schemes at all values of Tq. At small value of Tq, the query generate rate is high and system workload is more. This results in high value of average query latency. When Tq increases, fewer queries are generated and average query latency drops. If Tq keeps increasing, the average query latency drops slowly or even increases slightly due to decrease in cache byte hit ratio. Under extreme high Tq, most of the queries are served by the remote data server and the difference between different schemes is not very large. Figure 6 shows that NC has worst message overhead among all the schemes.

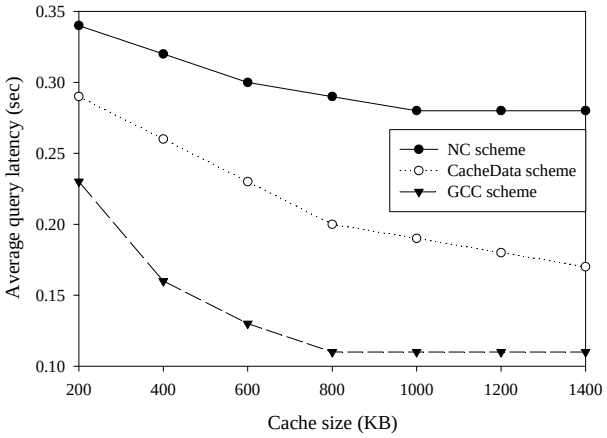

Figure 3. Effects of cache size on average query latency

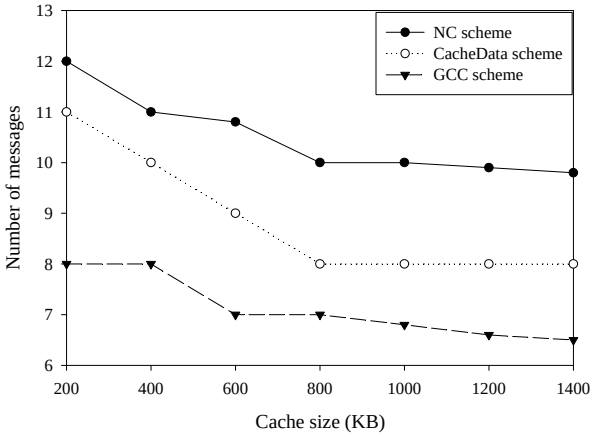

Figure 4. Effects of cache size on message overhead

c. Effects of mobility

Figure 7 and Figure 8 show 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 Figure 7, 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.

Figure 8 shows that the message overhead increases with increasing mobility. In GCC, the number of messages due to CSN role change/election and new registration of cache states with CSN increases with the node mobility. Experiments show that the overall performance degrades with higher mobility.

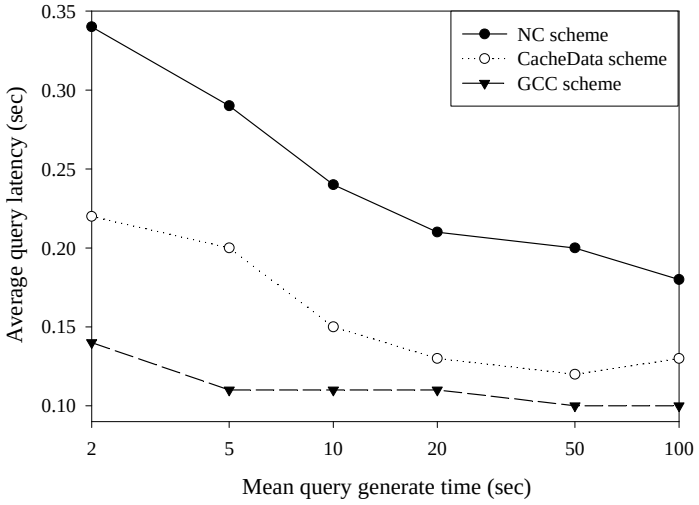

Figure 5. Effects of query generate time on average query latency

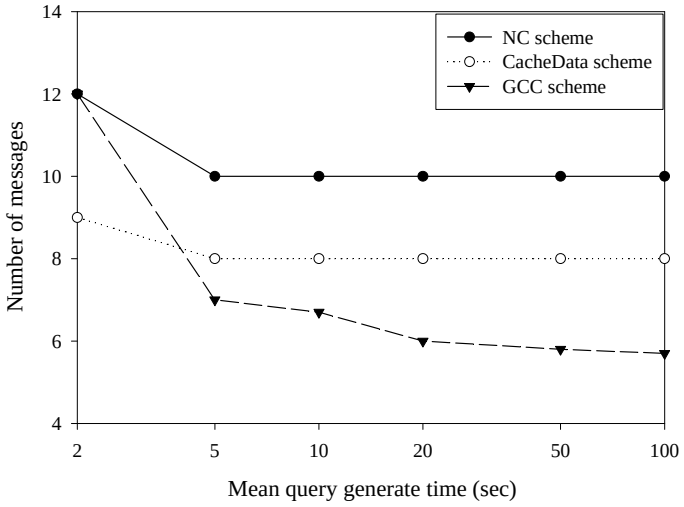

Figure 6. Effects of query generate time on message overhead

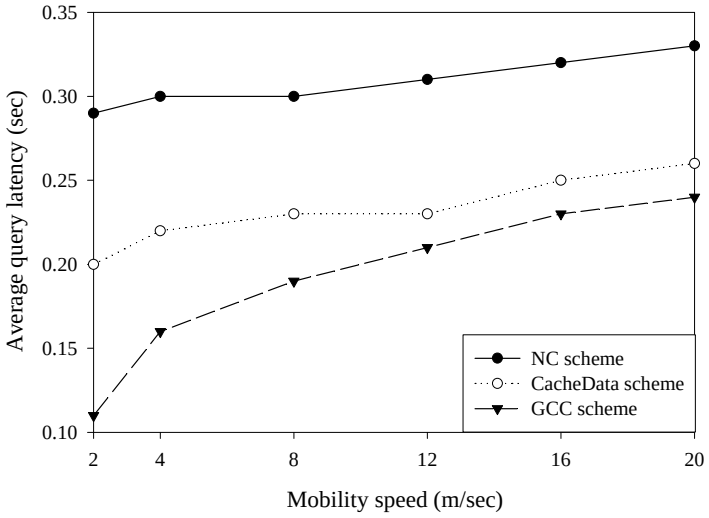

Figure 7. Effects of node mobility on average query latency

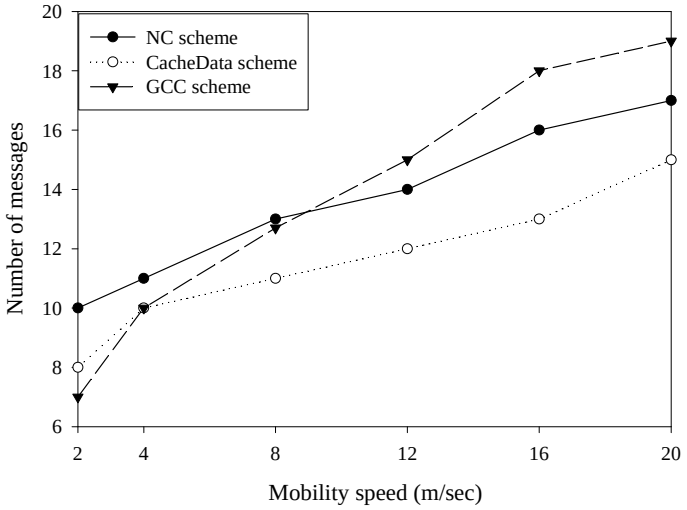

Figure 8. Effects of node mobility on message overhead

d. Effects of transmission range

Figure 9 shows that increase in transmission range increases the expected progress of the packet towards its final destination but at the expense of a higher energy consumption per transmission. On the other hand, a shorter transmission range consumes less per transmission energy, but it requires a large number of hops for the packet to reach its destination.

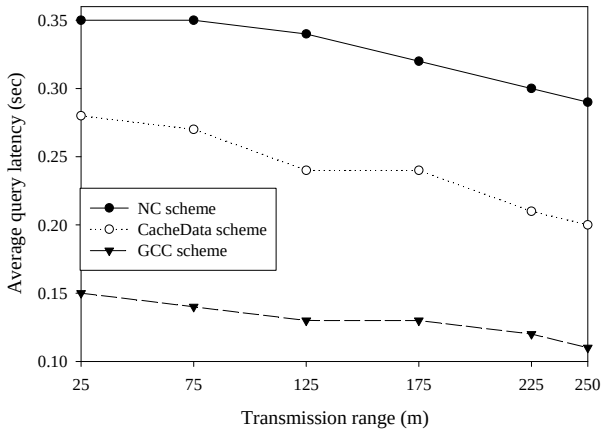

Figure 9. Effects of transmission range on average query latency

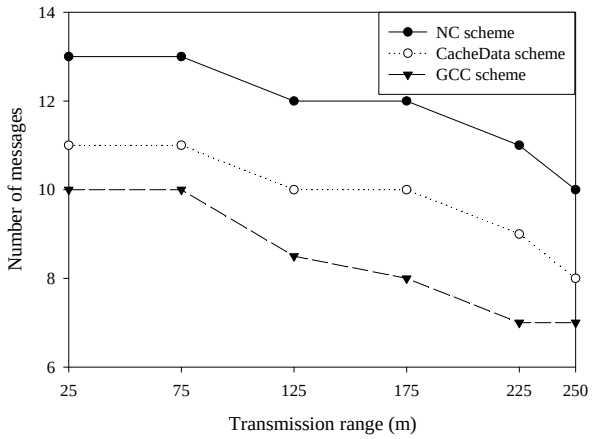

Figure 10. Effects of transmission range on message overhead

Figure 10 shows that for all the strategies the message overhead decreases with increasing transmission range because smaller numbers of hops are needed for packet to reach their destination.

V. CONCLUSION

In this paper, we have addressed cache cooperation issue in mobile ad hoc networks. We have presented a caching strategy named GCC. This strategy is unique such that in a cluster, the information about what all other clusters are retaining with themselves is available. 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. GCC is capable of supporting efficient data retrieval in ad hoc networks. This scheme exploits clustering for efficient data caching. Simulation results demonstrate that the proposed scheme reduces the message overheads and enhances the data accessibility as compared to other strategies.

REFERENCES

[1] L. Yin and G. Cao, "Supporting Cooperative Caching in Ad Hoc Networks," IEEE INFOCOM, pp. 77-89, March 2004.

[2] Narottam Chand, R.C. Joshi and Manoj Misra, "A Zone Co-operation Approach for Efficient Caching in Mobile Ad Hoc Networks," International Journal of Communication Systems, Vol. 19, Issue 9, pp. 1009-1028, Nov 2006.

[3] Ge-Ming Chiu and Cheng-Ru Young, "Exploiting In-Zone Broadcast for Cache Sharing in Mobile Ad Hoc Networks," IEEE Transactions on Mobile Computing, Vol. 8, No. 3, pp. 384-396, March 2009.

[4] Bin Tang, Himanshu Gupta and Samir Das,"Benefit-Based Data Caching in Ad Hoc Networks," IEEE Transaction on Mobile Computing, Vol. 7, No. 3, pp. 289-303, March 2008.

[5] Takahiro Hara and Sanjay K. Madria,"Data Replication for Improving Data Accessibility in Ad Hoc Networks," IEEE Transaction on Mobile Computing, Vol. 5, No. 11, pp. 1515-1532, Nov 2006.

[6] Narottam Chand, R.C. Joshi and Manoj Misra, "Cooperative Caching Strategy in Mobile Ad Hoc Networks Based on Clusters," International Journal of Wireless Personal Communication, Vol. 43, Issue 1, pp. 41-63, Oct 2007.

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

[8] R. Malpani, J. Lorch and D. Berger, "Making World Wide Web Caching Servers Cooperate," World Wide Web Journal, Vol. 1, No. 1, 1996.

[9] C.Y. Chow, H.V. Leong and A Chan, "Peer-to-Peer Cooperative Caching in Mobile Environments," Proceedings of 24th International Conference on Distributed Computing Systems Workshop (ICDCSW), pp. 528-533, 2004.

[10] C.Y. Chow, H.V. Leong and A Chan, "Cache signature for Peer-to-Peer Cooperative Caching in Mobile Environments," Proceedings of 18th International Conference on Advanced Information Networking and Applications (AINA), pp. 96-101, 2004.