An Efficient Caching Strategy in Mobile Ad Hoc Networks Based on Clusters

Narottam Chand*, R. C. Joshi, and Manoj Misra
Department of Electronics and Computer Engineering
Indian Institute of Technology, Roorkee 247 667, India
*Email: nchand@ieee.org

***Abstract*—We propose a novel scheme, called *cluster cooperative (CC)* for caching in mobile ad hoc networks where network topology is partitioned into non-overlapping clusters. In each cluster a "super" node called cache state node (CSN) is chosen to maintain the cluster cache state (CCS) information for nodes within its cluster domain. For a local cache miss, each client checks the data with its home CSN before forwarding the request towards server. Simulation experiments show that the CC caching mechanism achieves significant improvements in average query latency and reduces the cache management overheads in comparison with other caching strategies.**

***Index Terms*—Ad hoc networks, cooperative caching, admission control, replacement, consistency, cluster.**

I. INTRODUCTION

The ultimate goal of mobile ad hoc network (MANET) is to provide efficient information/data access to mobile nodes. Caching is one of the most attractive techniques that improves data retrieval performance in wireless mobile environment [1], [5]. 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 [2], [3] and placement [4]. To the best of our knowledge, none of previous works has exploited clustering as caching mechanism in MANETs.

In this paper, we investigate the data retrieval challenge of mobile ad hoc networks and propose a novel scheme, called cluster cooperative (CC) for caching. The goal of CC is to reduce the cache discovery overhead and provide better cooperative caching performance. CC partitions the whole MANET into equal size clusters based on the geographical network proximity (see Fig. 1). To enhance the system performance, within a cluster, individual caches interact with each other such that combined result is a larger cumulative cache. In each cluster, CC dynamically chooses a "super" node as cache state node (CSN), to maintain the cluster cache state (CCS) information of different nodes within its cluster domain. The CCS 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 CC 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 CC is presented in Section II. Section III describes the proposed CC 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[1]. Grid size captures the maximum distance between two nodes inside a cluster. Size is selected such that, each node is at one-hop distance from every other node inside a cluster. 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 zigzag diagonal-wise fashion. In each cluster area a "super" node is selected to act as CSN, which is responsible for maintaining the cluster cache state (CCS) information of different nodes within its cluster domain. CCS for a node is the list of data items along with their TTL stored in its cache. When a node caches/replaces a data item, its CCS 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

[1] The problem of finding an optimal clustering is out of the scope of this paper. For the sake of simplicity, in this paper we assume that clusterization phase gives a partition of the network into grids. However, any clustering algorithm can be used as our CC caching scheme is compatible with any non-overlapping clustering strategy.

bottleneck and/or a point of failure when the system has high network density. Unlike this, CSN works only as CCS holder to save the information about the cached items by different clients belonging to the cluster, 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 $r/\sqrt{2}$, all clients in a cluster can connect to one another in one-hop communication. Where r is transmission range of a client.

In CC, 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. CLUSTER COOPERATIVE (CC) CACHING

The design rationale of CC 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 CC, 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 by sending a request to the CSN. Only when the client cannot find the data item in the cluster members' caches (called cluster cache miss), it will request the item from the client that lies on the routing path towards server. If a cluster along the path to the server 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 copy of the requested data item is stored inside the hard disk of the requester. The data item is retrieved to serve the query and no cooperation is necessary.

Case 2: Cluster hit. When the requested data item is stored by a client within the cluster of the requester. The requester sends a request to the CSN and the CSN returns the address of the client that has cached the data item.

Case 3: Remote hit. When the data is found with a client belonging to a cluster (other than home cluster of the requester) along the routing path to the data source.

Case 4: Global hit. Data item is retrieved from the server.

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 MH_i denotes mobile node/client i. In Fig. 1, 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 sends a *lookup* packet to the CSN MH_j in its cluster. Upon receiving the *lookup* message, the CSN searches in the CCS 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. If no client is caching the item, the *ack* packet returned to the requester MH_i contains *Null* address and the MH_i sends a *request* packet to next hop node MH_a.

When MH_k receives a *request* packet, it sends a *lookup* packet to its CSN if it does not have d_x in its local cache. When MH_k receives an *ack* packet, it sends *request* packet to the client within cluster or to next hop node based on the address returned in *ack* packet as described in step 1 above.

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.

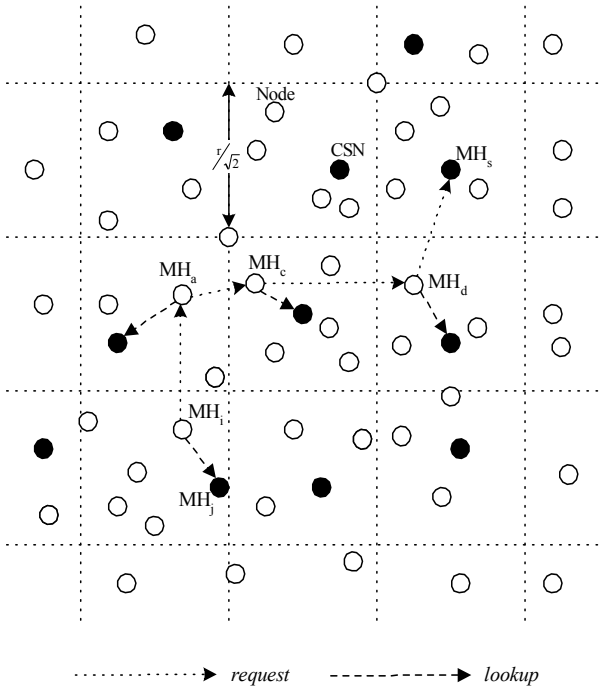

Fig. 1. Request packet from client MH_i is forwarded to the data source MH_s.

In CC, 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. Fig. 2 shows the behavior of CC

caching strategy for a client request.

The CC 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.

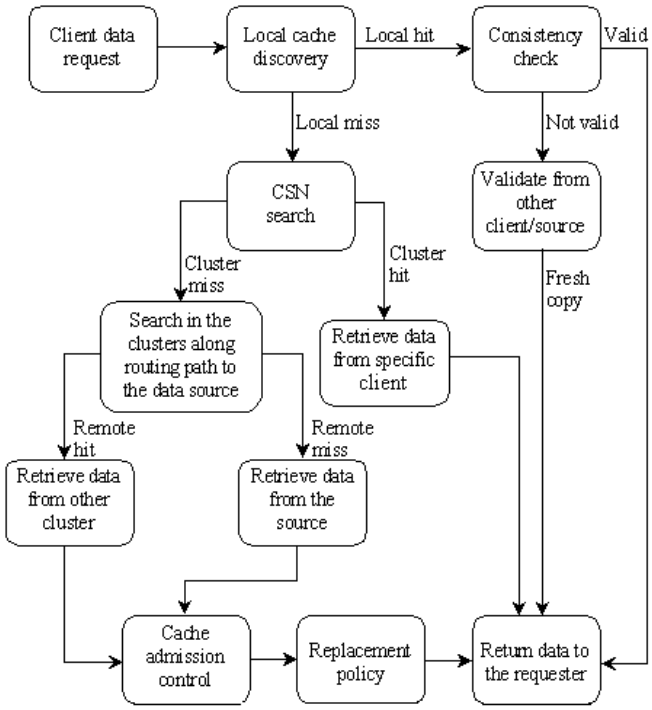

Fig. 2. Service of a client request by CC caching strategy.

TABLE I
SIMULATION PARAMETERS

Parameter	Default Value	Range
Database size (N)	1000 items	
s_{min}	1 KB	
s_{max}	10 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)	250 m	25~250 m
Skewness parameter (θ)	0.8	0~1

IV. SIMULATION RESULTS

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 I.

For performance comparison with CC, two other schemes non-cooperative (NC) caching and CacheData [2], [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.

A. Effects of cache size

Fig. 3 and Fig. 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 Fig. 3, we can see that the CC 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, thus alleviating the need for remote and global cache access. Because the hop count of cluster data hit is one and is less than the average hop count of remote data hit, CC 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 CC reaches its best performance when the cache size is 800 KB. This demonstrates its low cache space requirement.

Fig. 4 shows that CC performs much better than NC and CacheData in terms of message overhead. The reason is that due to cache cooperation within a cluster CC gets data from nearby node/cluster 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 CC increases and its message overhead decreases.

B. Effects of mean query generate time

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

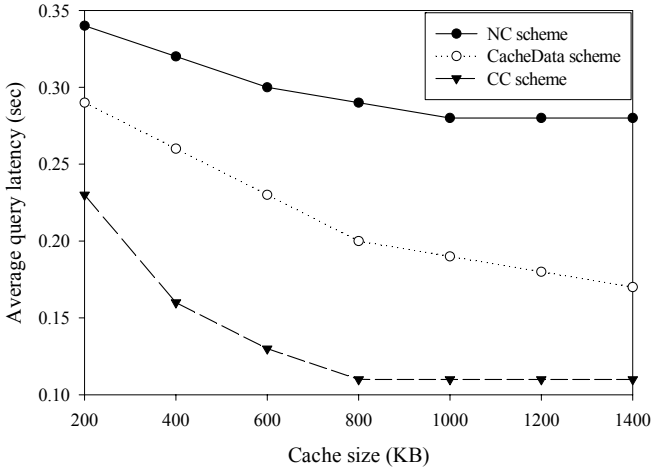

Fig. 3. Effects of cache size on average query latency.

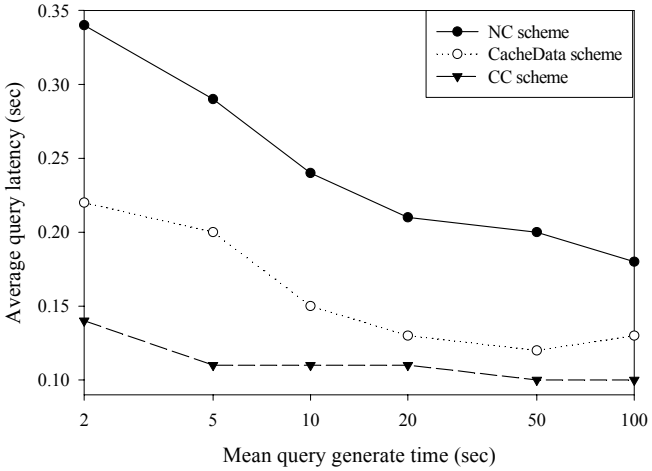

Fig. 5. Effects of query generate time on average query latency.

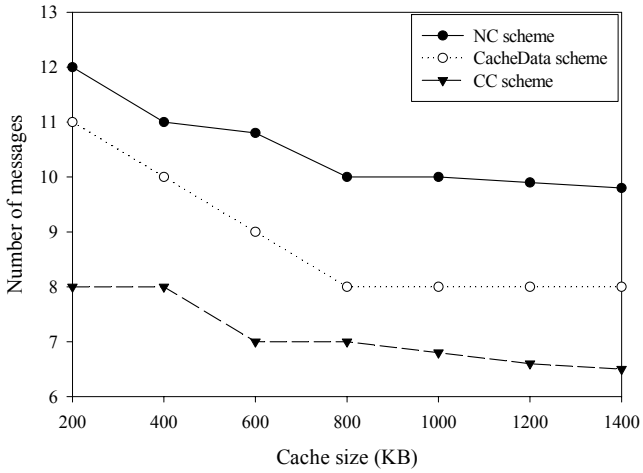

Fig. 4. Effects of cache size on message overhead.

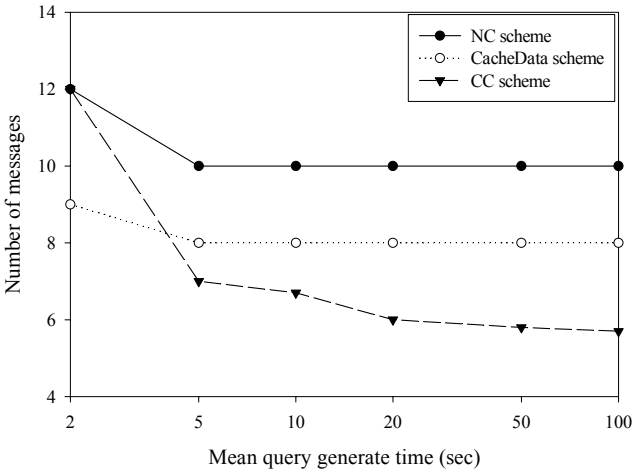

Fig. 6. Effects of query generate time on message overhead.

C. Effects of mobility

Fig. 7 and Fig. 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 Fig. 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 CC caching performance in terms of average query latency.

Fig. 8 shows that the message overhead increases with increasing mobility. In CC, 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.

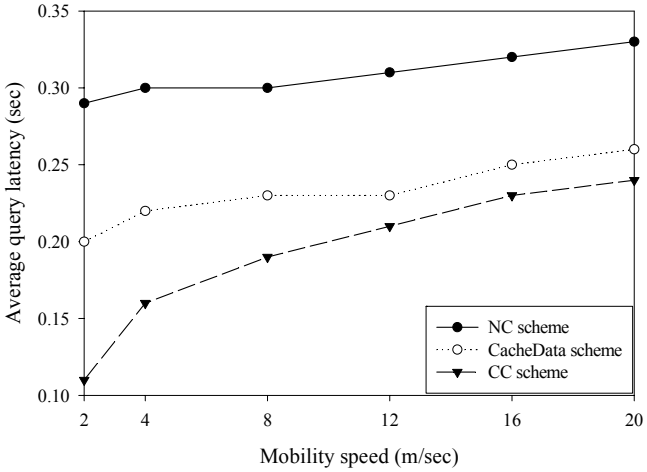

Fig. 7. Effects of node mobility on average query latency.

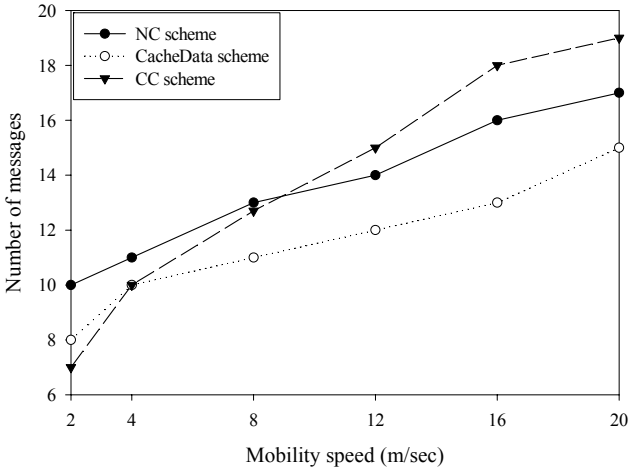

Fig. 6. Effects of node mobility on message overhead.

V. CONCLUSIONS

This paper exploits clustering for efficient data caching in ad hoc networks. The proposed CC caching scheme reduces the message overheads and enhances the data accessibility as compared to other strategies. We anticipate 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] G. Cao, "A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 15, No. 5, pp. 1251-1265, 2003.

[2] L. Yin and G. Cao, "Supporting Cooperative Caching in Ad Hoc Networks," *IEEE INFOCOM*, pp. 2537-2547, March 2004.

[3] G. Cao, L. Yin and C. Das, "Cooperative Cache Based Data Access Framework for Ad Hoc Networks," *IEEE Computer*, pp. 32-39, February 2004.

[4] Pavan Nuggehalli, Vikram Srinivasan and C.-F. Chiasserini, "Energy-Efficient Caching Strategies in Ad Hoc Wireless Networks," *MobiHoc*, pp. 25-34, 2003.

[5] N. Chand, R.C. Joshi and Manoj Misra, "Energy Efficient Cache Invalidation in a Mobile Environment," *International Journal of Digital Information Management (JDIM) special issue on Distributed Data Management*, Vol. 3, No. 2, pp. 119-125, June 2005.