RMCC: An Efficient Cooperative Caching Scheme for Mobile Ad-hoc Networks

Po-Jen Chuang, Yu-Yiu Chen, and Hang-Li Chen

Department of Electrical Engineering, Tamkang University
Tamsui, New Taipei City, Taiwan 25137, R.O.C.
Email: pjchuang@ee.tku.edu.tw

Abstract

*For mobile ad-hoc networks (MANETs), caching data is important and useful. By exercising cache cooperation between mobile nodes, cooperative caching schemes can improve data accessibility and system performance. Seeing that the cache hit ratios tend to degrade due to high node mobility, restricted battery energy and limited wireless bandwidth in a MANET, we introduce a new cooperative caching scheme, the Regionally Maintained Cooperative Caching (**RMCC**) scheme, to solve the problem. **RMCC** allows one node in a region to cache a data item while the other nodes in the same region to cache the path to the node when to acquire the same item. Such a design helps generate more cache space and wider cached data variety for nodes in one region, and as a result attains higher cache hit ratios as well as lower data access latency. Experimental evaluation shows that the proposed **RMCC** outperforms other schemes in terms of cache hit ratios, in-cache ratios and amounts of required file packets.*

Keywords: *Mobile ad-hoc networks (MANETs), cooperative caching schemes, cache hit ratios, access latency, experimental evaluation*

1. Introduction

A cache is a component that transparently stores data items to facilitate future data requests. When a node needs a data item which is in the cache, it can directly read the cache, instead of going to the server, to attain the item in a faster and more efficient way. If a cache can serve more data requests, the overall system performance can be largely intensified. Wireless caching is similar, in principle, to content caching which has been long used by Internet Service Providers (ISPs) to accelerate web content acquirement. A wireless cache will temporarily store popular content that is flowing into an ISP's network. If the temporary storage can satisfy a subscriber's data request, we can avoid data transfer by expensive transit links and meanwhile reduce network congestion. For mobile ad-hoc networks (MANETs), caching data is equally important and useful. To give an example, in a battlefield, a MANET may contain several commanding officers each of whom has a powerful data center, and a group of soldiers who need to access from the centers for such data as geographic information, enemy information or new commands. As neighboring soldiers tend to have similar missions and share common interests, it is very likely that when soldier A has accessed a data item from the data center, nearby soldiers will access the same item some time later. In such a case, if later accesses to the same data item can be served by soldier A (who has the item) rather than by the distant center, we can save significant energy, bandwidth and time.

By exercising cache cooperation between mobile nodes, cooperative caching schemes can improve the system performance of MANETs, mesh networks or sensor networks.

Cooperative caching can improve data accessibility in MANETs in which a node may communicate with others anytime anywhere. For instance, ***CacheData*** [1] lets each node cache a relayed data item (d_i) locally when it finds d_i is popular (frequently accessed) or when cache space is available. ***CachePath*** [1, 2] lets each node cache the shortest path to the available cache that has the requested item. ***Zhao's*** [3] introduces a symmetric cooperative caching approach to improve the performance of ***CacheData***. ***GroupCache*** [4] allows a node and its 1-hop neighbors to form a group, then periodically exchanging and maintaining the caching status in the group.

In MANETs, cooperative caching faces a major problem: cache hit ratios may go down while access latency may go up due to high node mobility, restricted battery energy and limited wireless bandwidth. Another major problem is: Cooperative caching schemes cannot effectively distribute varied data to the caches in a region. This will keep nodes in a region from caching more hot data, degrading cache hit ratios while increasing access latency. To improve the situation, this paper presents an advanced new cooperative caching scheme, the Regionally Maintained Cooperative Caching (***RMCC***) scheme. Preserving the advantages of existing caching schemes (such as ***Zhao's*** and ***GroupCache***), ***RMCC*** lets only one node (say A) in a region cache a data item while the other nodes in the same region cache the path to A when pursuing the same item. Such a design can produce more cache space for nodes in a region to store more data and practically raise the cache hit ratios. To maintain cache validity of adjacent nodes, ***RMCC*** takes advantage of the hello message broadcasting in on-demand routing for MANETs. It exchanges and maintains the cache status when any node receives a data reply message. For a data miss in the CacheData space, each node will search the item in its CachePath table before forwarding the request to the next node in the routing path towards the server.

Experimental evaluation using NS2 [5] is carried out to check and compare the performance of the proposed ***RMCC*** and other caching mechanisms, including ***SimpleCache***, ***GroupCache*** and ***Zhao's***. The results indicate more favorable performance in cache hit ratios, in-cache ratios, and the amount of file packets for our ***RMCC*** than for other schemes.

2. Related Works

Most of previous researches in MANETs focus on the development of dynamic routing protocols [6-8], to improve the one-hop/multi-hop connectivity among mobile hosts (MHs) – such as notebooks, PDA or cell phones in which every node can move arbitrarily and communicate with one another by multi-hop wireless links.

2.1. Ad-hoc On-demand Distance Vector Routing (AODV) [9-12]

AODV is composed of route request, route reply and route maintenance. In ***AODV***, nodes will build and repair a path only when necessary – to reduce the extra cost of building routes in a dynamic network topology.

2.1.1. Route request: Route discovery starts upon request. When a node needs to send a packet to a destination, it first searches its routing table for usable routing information. If there is valid information, the node will send out the packet along with the next hop indicated in the table; otherwise, it will start the route request by flooding a RREQ (route request) packet. A RREQ carries such information as source IP, destination IP and broadcast ID. Each node maintains a broadcast ID which will increase whenever a RREQ is sent (to mark the event). After sending out a RREQ, the source will set a timer and wait for the RREP (request reply). Sending a RREQ

will create a route reaching out to the destination and also a reverse route for the destination to return the RREP to the source.

2.1.2. Route reply: In the process of sending a RREQ, if there is a valid route to the destination in the routing table of a middle node, the middle node will return the RREP to the source; otherwise, the destination node needs to return the RREP by the end. Different from a RREQ, RREP will be transmitted in unicast, i.e., the middle nodes will build a forward route to the destination. After the source receives the RREP, it then transmits the data packets by the built forward route to the destination.

2.2. CacheData and CachePath [1, 2]

CacheData and ***CachePath*** are two cooperative caching schemes. In ***CacheData***, a node will cache a passing-by data item d_i locally when it finds that d_i is popular (with many requests) or when it has free cache space. For example, in Figure 1, both nodes 6 and 7 request d_i through node 5, node 5 knows that d_i is popular and caches it locally. Future requests by node 3, 4, or 5 can be served by node 5 directly. Figure 1 can also illustrate the idea of ***CachePath***. Suppose node 1 has requested a data item d_i from node 11. When node 3 forwards d_i back to node 1, it knows that node 1 has a copy of d_i. Later, if node 2 requests d_i, node 3 finds out node 11 is three hops away while node 1 is only one hop away. Node 3 hence forwards the request to node 1 instead of node 4. Note that many routing algorithms (such as ***AODV*** and ***DSR*** [11-13]) provide hop count information between the source and destination. Besides ***CacheData*** and ***CachePath***, there are hybrid cooperative caching schemes of the two, *e.g.*, [1, 14].

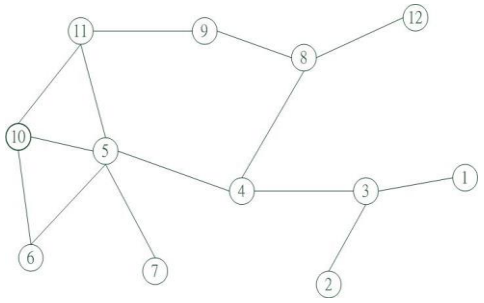

Figure 1. An Example of Performing *CacheData* and *CachePath*

2.3. Zhao's Method [3]

Zhao *et al.*, propose a symmetric cooperative cache approach, where data requests are sent to the cache layer at every node but data replies are sent only to the cache layer at the intermediate nodes which need to cache the data.

2.3.1. The placement and replacement policy: Designed based on ***CacheData***, ***Zhao's*** also lets a node cache a passing-by data item locally when it finds the item is popular or has enough cache space.

2.3.2. The data discovery process: After a request is generated by the application, it is passed down to the cache layer. To send the request to the next hop, the cache layer wraps the original message with a new destination address – the next hop to reach the data server (the real destination). We assume that the cache layer can access the routing table and locate the next hop

to reach the data center. It can be easily accomplished if the routing protocol is based on ***DSR*** or ***AODV***. That is, the packet can be received and processed hop by hop by all nodes along the path from the requester to the data server.

When an intermediate node receives the request and delivers it to the cache layer, the cache manager will check if it has the requested data in its local cache. If yes, add its local information, including ID and TS, to the request packet. (Also add its node id to the Path List, which is a linked list encapsulated in the cache layer header.) If not, forward the request to the next node until the request arrives at the data center.

2.4 GroupCache [4]

GroupCache lets each mobile host and its 1-hop neighbors form a group, to exchange and maintain the caching status periodically. By using the proposed group caching, the caching space in mobile hosts can be efficiently utilized, to reduce the redundancy of cached data and the average access latency.

2.4.1. The placement and replacement policy: When a node receives a data item, it will cache the item if there is free cache space. If there is no free space, it will look for available cache space in group members. If none of the group members has enough cache space to cache the received item, the receiving node will look up the group_table to select an "appropriate" group member (which has the oldest timestamp of the cached item) and send the item to the selected member.

2.4.2. The data discovery process: A requester will build a routing path to the destination and send the request to the next hop to reach the source (destination). When an intermediate node in the routing path receives the request, it will check its self_table (cachedata space) or group_table (cachepath space) for the requested data. If unable to find the data, it will forward the request to the next node in the routing path. The same process will go on until the server attains the data.

3. The Proposed Regionally Maintained Cooperative Cache (RMCC) Scheme

To enhance the cooperation of node caches based on the ***Zhao's*** mechanism, we let nodes broadcast hello messages with node cache information. Nodes will cooperate to catch current cache distribution. When a node receives a requested file forwarding packet with the file's popularity degree over a predefined threshold (*i.e.*, a *popular* file), it will add the file name, file version and popularity degree of the cache information into the next hello message and broadcast it to neighbor nodes. Receiving the hello message, a neighbor node will check its own cache space to see if it has the same file: if yes, delete it and cache the path only. By allowing only one node to cache the file while the other nodes in the same region to cache the path, we can attain more cache space for storing other files and increase the cache hit ratios substantially.

3.1. The Flow of Our Cache Scheme

3.1.1. The task of each node: Our scheme adopts similar routing as the ***AODV*** scheme [9-12]. While ***AODV*** uses neighbor tables to store the information of neighbor nodes, our ***RMCC*** combines the cache space and neighbor tables. Besides caching data, each of our nodes also stores in its neighbor table the cache paths to neighbor nodes, to facilitate future

requests for popular data. When a node receives a hello message with the cache information of a data item, it will first search its own cache for such an item. If the item is in the cache: check its validity; if not, check the neighbor table for the corresponding cache path.

3.1.2. The cooperation of nodes: When updating cached data, two nodes in a region, say A and B, may cache the same data item. At this point, compare the popularity degree of this item in the two nodes: If the cached item is more popular in A than in B, A will keep the item in its cache whereas B will move to add the cache path in its neighbor table and delete the cached item. A then includes the updated information in the next hello message and broadcast it to the neighbors. Receiving the message, each neighbor will delete this specific cached item from its cache and store the cache path in the neighbor table. It is through such cooperation between nodes that we are able to increase cache efficiency.

3.1.3. An example: Figure 2 helps illustrate the basic design of our scheme.

In Figure 2, we assume node 0 is a wireless node adjacent to a wired network and via node 0 we can fetch a requested data item stored in the wired network. Sometime later, when a few neighbor nodes of node 0 repeatedly cache the same data item, we find it is quite a waste of the limited resources in the wireless environment. Previous cache schemes cannot avoid this kind of resource waste, but our scheme can -- because we employ the regionally maintained cooperative caching by way of hello messages. Now suppose nodes 4 and 5 in the figure will cache the same data item and node 4 is about to broadcast a hello message. Node 4 has the information of this item, including the time stamp and popularity degree, adds the information to the hello message and broadcast it out. (Note that by combining cache information with hello messages, we can maintain caches and meanwhile confirm the presence of neighbor nodes, avoiding the drawback of ***CachePath*** due to absence of neighbor nodes.) Receiving the hello message, node 5 will search its own cache for this specific data item: if having the item, delete it and cache the path only (saving the cache space for other data).

Later, when node 9 queries the same data (by the red line) by way of node 5, node 5 will attach the queried information (including the time stamp) for the server to check its validity in node 4. The server then sends a control message to node 4 (by the blue line) asking it to return the requested item (by the green line) to node 9. Thus completes the query.

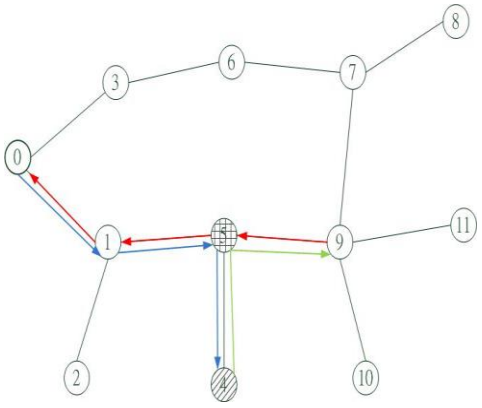

Figure 2. An Example to Illustrate our Scheme

3.2. Data Placement and Replacement

In our scheme, when a packet containing a requested file reaches a node, the node will cache the file when there is cache space. If the cache space is full and the file is already there, the node will add the file's popularity degree by one. If the cache space is full without the file, the node then follows the LRU cache replacement mechanism in [1] to cache the file. We can also store cache paths via periodical hello message broadcasting in MANET routing, like the ***AODV*** routing. That is, by allowing only one node to cache a file and the other nodes in the same region to cache the path leading to the file, we are able to gain more cache space for caching more files and eventually enhance the cache hit ratios.

3.3. Data Discovery

In our scheme, when a node (client) requests for a file, each node on the way of exploration to the server needs to check if such a file is stored in its cache. If yes, append the file name, version and node name to the exploration packet; otherwise, look for cache path space and append related information to the exploration packet. If a node caches neither the file nor the path, it will pass the exploration attempt to the next node which will repeat the same process until reaching the server. In the data discovery process, if an intermediate node caches the valid information of the requested file and directly returns it to the client, we can save significant bandwidth resources due to reduced control packet transmissions. In case the requested file cached in intermediate nodes is invalid, the server will send the file to the client.

4. Experimental Evaluation

Experimental evaluation using NS2 [5] has been conducted to check and compare the performance of the proposed ***RMCC*** and related schemes, including ***Zhao's*** [3], ***GroupCache*** [4] and ***SimpleCache*** [3]. (***SimpleCache*** is the traditional caching scheme which caches the received data at the query node only).

4.1. Simulation Parameters

In the simulation, we set client and server models based on [1], exponentially distribute the query interval of each node and slowly increase the query frequency over time. A new query will take place only after a current query is served. Considering the actual trace of webpage packets, we let 10% of the network's popular pages provide 80% of users' requests for web pages (the Zipf-like distribution [15]). To simulate data updates in the server, we assume the server with an opportunity to update data every five seconds. As our scheme employs ***AODV*** routing, we adopt its routing parameters as well. For instance, the hello message is broadcast per second, the transmission distance is assumed to be 250 meters, and the simulation area is set to be rectangular to reflect network cache performance under long-distance transmission. The adopted simulation parameters are listed in Table I.

Table I. Simulation Parameters

Parameter	value
Simulation area	1500m*500m
Number of nodes	50
Communication range	250
Client cache size	800kb
Server database size	1000 items
Data item size	20Kb

4.2. Simulation Results

Figure 3 depicts the cache hit ratios vs. the server update ratios for the schemes. The hit ratio (calculated by the following formula) indicates the ratio of requested data items being in the cache and valid to all requested items.

$$hit\_ratio = \frac{N_{hit}}{N_{total\_query}} = \frac{N_{hit}}{N_{hit} + N_{miss} + N_{nc}}$$

where

N_{hit} = the number of requested items found in the cache of an intermediate node and valid.

N_{miss} = the number of requested items found in an intermediate node cache but invalid

N_{nc} = the number of requested items not found in the cache of any intermediate node

$N_{total\_query} = N_{hit} + N_{miss} + N_{nc}$ = the total number of requests

The result shows that ***SimpleCache*** which lets each node check its own cache space for the requested item yields undesirable performance, and so does ***GroupCache*** which does not check validity from the server. Our ***RMCC*** generates the best hit ratios among all mainly because it uses hello messages to maintain caches regionally and to store cache paths.

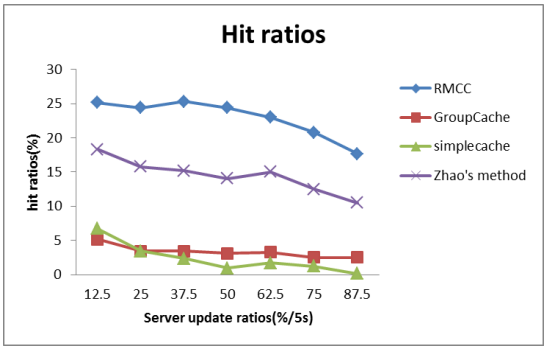

Figure 3. Cache Hit Ratios vs. Server Update Ratios

Figure 4 depicts the cache miss ratios vs. the server update ratios for the schemes. The miss ratio (attained by the following formula) indicates the ratio of requested data items being in the cache but invalid to all requested items.

$$miss\_ratio = \frac{N_{miss}}{N_{total\_query}} = \frac{N_{miss}}{N_{hit} + N_{miss} + N_{nc}}$$

Compared with ***Zhao's***, our ***RMCC*** yields slightly higher miss ratios at some server update ratios because of its additional cached paths. ***RMCC*** nevertheless produces more desirable overall performance than ***Zhao's*** due to its constantly higher hit ratios as depicted above.

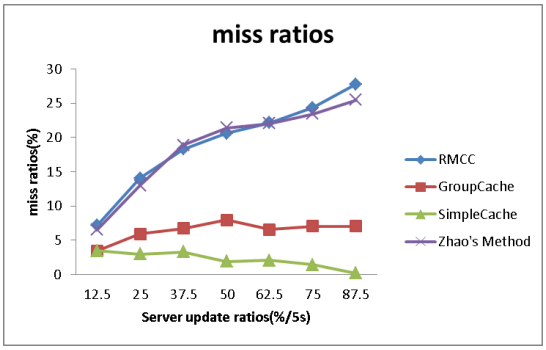

Figure 4. Cache Miss Ratios vs. Server Update Ratios

Figure 5 gives the in-cache ratios vs. the server update ratios for the schemes. The in-cache ratio (obtained by the following formula) is the ratio of requested data items being in cache, *valid* or *invalid*, to all requested items.

$$in\_cache\_ratio = \frac{N_{hit} + N_{miss}}{N_{total\_query}} = \frac{N_{hit} + N_{miss}}{N_{hit} + N_{miss} + N_{nc}}$$

RMCC is shown to yield the highest in-cache ratios because its regional inter-node cooperation policy helps gain more cache space for storing more items and thus increases the probability of locating a requested data item in the cache. Note that the in-cache ratios of both ***RMCC*** and ***Zhao's*** go up with the server update ratios. This is because it gets easier to locate a requested item in the cache when cached data are distributed more evenly. Between the two schemes, ***RMCC*** is able to attain higher in-cache ratios than ***Zhao's*** because it caches not only data but also the paths leading to cached data, as mentioned before.

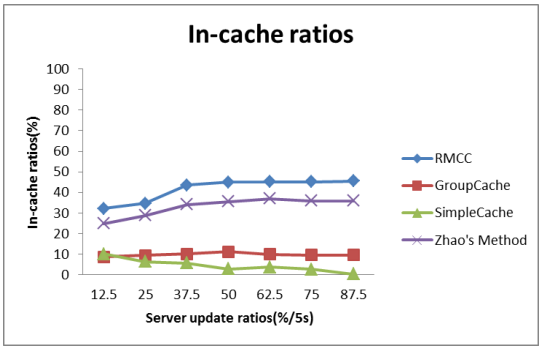

Figure 5. In-cache Ratios vs. Server Update Ratios

Figure 6 gives the in-cache valid ratios vs. the server update ratios. The in-cache valid ratio (attained by the following formula) is the ratio of requested data items being in cache and valid to all requested items that are in cache.

$$in\_cache\_valid\_ratio = \frac{N_{hit}}{N_{hit} + N_{miss}}$$

RMCC has much higher in-cache valid ratios than ***Zhao's*** because it adopts the following approaches: (1) using extra storage to cache the paths (to help locate a requested data – in cache and valid), (2) additionally broadcasting the cached paths (to increase the hit ratios and

maintain caches regionally) and (3) broadcasting the most stable data when broadcasting the cached paths (to minimize the probability of additional cache misses).

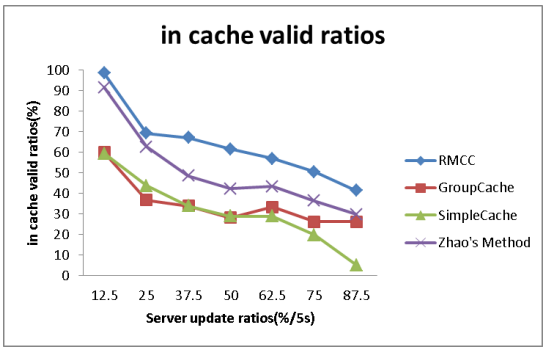

Figure 6. In-cache Valid Ratios vs. Server Update Ratios

Figure 7 illustrates the amount of file packets vs. the server update ratios for the schemes. Here, the total of file packet transmissions (not including control packet transmissions) indicates the percentage of file packet transmissions for these caching schemes over file packet transmissions for original ***AODV*** (with no caching mechanisms). The result will help reveal how much caching mechanisms can reduce file packet transmissions. Here, ***SimpleCache*** and ***GroupCache*** both require considerable file packets, indicating they save only small amounts of file packets. ***GroupCache***, which acquires requested data from the server upon cache miss, requires even more file packets than original ***AODV***, especially when the server update ratios are high. ***RMCC*** achieves better caching efficiency than ***Zhao's*** (by its regional inter-node cooperation) and hence reduces the most file packet transmissions.

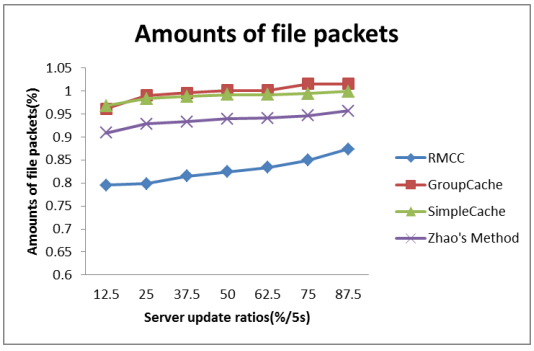

Figure 7. The Amounts of File Packets vs. Server Update Ratios

Figure 8 gives the amounts of control packets in bytes. ***SimpleCache*** is shown to have the least amount of control packets because it lets each node check the cache for a requested data. Our ***RMCC*** needs slightly more control packets than ***SimpleCache*** and ***Zhao's*** – which is acceptable and worthwhile when compared with the significant performance gain illustrated above. Of all schemes, ***GroupCache*** requires obviously the most control packets because its nodes need to broadcast periodically.

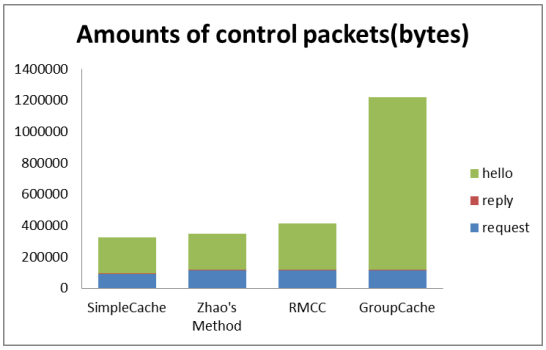

Figure 8. The Amounts of Required Control Packets in Bytes

5. Conclusions

Cooperative caching can improve the data accessibility and system performance of a MANET. But high node mobility, restricted battery energy, limited wireless bandwidth and ineffective caches may degrade the cache hit ratios while increase the access latency. This paper introduces a new cooperative caching scheme, the Regionally Maintained Cooperative Caching (***RMCC***) scheme, to solve the problem. ***RMCC*** is built on a regional inter-node cooperation concept: A data item will be cached in only one node of a region; when the other nodes in the same region need that specific item, they simply cache the path to the node whose cache has the item. The unique design is desirable as it (1) increases cache space for nodes in a region, (2) widens the variety of cached data, (3) elevates cache hit ratios in data discovery, (4) shortens data access latency and (5) upgrades the overall performance for a MANET. To maintain cache validity of adjacent nodes, ***RMCC*** exchanges/maintains the cache status using hello message broadcasting. Simulation results show that when compared with existing caching schemes, ***RMCC*** performs better in several performance parameters, including *cache hit ratios*, *in-cache ratios* and *file packet transmissions*. It eventually reduces data access latency and bandwidth consumption.

References

[1] L. Yin and G. Cao, "Supporting Cooperative Caching in Ad Hoc Networks", 23rd Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 4, **(2004)**, pp. 2537-2547.
[2] H. Artail, H. Safa and S. Pierre, "Database Caching in MANETs Based on Separation of Queries and Responses", 2005 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, vol. 3, **(2005)**, pp. 237-244.
[3] J. Zhao, P. Zhang, G. Cao and C. R. Das, "Cooperative Caching in Wireless P2P Networks: Design, Implementation and Evaluation", IEEE Transactions on Parallel and Distributed Systems, vol. 21, no. 2, **(2010)**, pp. 229-241.
[4] Y.-W. Ting and Y.-K. Chang, "A Novel Cooperative Caching Scheme for Wireless Ad Hoc Networks: GroupCaching", 2007 International Conference on Networking, Architecture, and Storage, **(2007)**, pp. 62-68.
[5] NS2, http://www.isi.edu/nsnam/ns/.
[6] D. Wang, X. Wang, X. Yu, K. Qi and Z. Xia, "A Truthful and Low-Overhead Routing Protocol for Ad Hoc Networks", International Journal of Future Generation Communication and Networking, vol. 6, no. 2, **(2013)**, pp. 127-138.
[7] S. Gupta and C. Kumar, "An Intelligent Efficient Secure Routing Protocol for MANET", International Journal of Future Generation Communication and Networking, vol. 6, no. 1, **(2013)**, pp. 111-132.
[8] N. Karthikeyan, V. Palanisamy and K. Duraiswamy, "Performance Comparison of Broadcasting methods in Mobile Ad Hoc Network", International Journal of Future Generation Communication and Networking, vol. 2, no. 2, **(2009)**, pp. 47-58.
[9] C. E. Perkins and E. M. Royer, "Ad hoc On-Demand Distance Vector Routing", 2nd IEEE Workshop on Mobile Computing Systems and Applications, **(1999)**, pp. 90-100.

[10] C. Perkins, E. Royer and S. Das, "Ad Hoc On-Demand Distance Vector (AODV) Routing", Internet Draft, Internet Engineering Task Force, http://www.itef.org/internet-drafts/draft-ietf-manet-aodv-08.txt, **(2001)**
[11] M. Abolhasan, T. Wysocki, E. Dutkiewicz, "A Review of Routing Protocols for Mobile Ad Hoc Networks", J. Elsevier Ad Hoc Networks, vol. 2, no. 1, **(2004)**, pp. 1-22.
[12] P. Nandl and S. C. Sharma, "Performance study of Broadcast based Mobile Ad Hoc Routing Protocols AODV, DSR and DYMO", International Journal of Security and Its Applications, vol. 5, no. 1, **(2011)**, pp. 53-64.
[13] D. Johnson, D. Maltz, Y.-C. Hu and J. Jetcheva, "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks. Internet Draft", Internet Engineering Task Force, http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-05.txt, **(2001)**.
[14] H. Artail, H. Safa, K. Mershad, Z. Abou-Atme and N. Sulieman, "COACS: A Cooperative and Adaptive Caching System for MANETs", IEEE Transactions on Mobile Computing, vol. 7, no. 8, **(2008)**, pp. 961-977.
[15] L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and implications", 18th Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 1, **(1999)**, pp. 126-134.