Improving Data Accessibility and Query Delay in Cluster based Cooperative Caching (CBCC) in MANET using LFU-MIN

Madhavarao Boddu
Department of Computer Science
Pondicherry University

K. Suresh Joseph
Department of Computer Science
Pondicherry University

ABSTRACT
In order to improve data accessibility and reduce query delay in MANETs, cooperative caching approach is adapted in. So far, cache replacement algorithms like LRU, LFU, and LRU-MIN are used to reduce query delay and improve data accessibility in cluster based cooperative caching (CBCC) in MANETs. But LRU, LFU and LRU-MIN have its limitations: They have a high overhead cost of moving cache blocks into the most recently used position each time when a cache block is accessed and further they do not exploit the 'frequency' information of memory accesses. In this paper, we give an overview of caching policies designed specifically for Web objects and provide a new algorithm of our own to address these issues. This new algorithm can be regarded as a LFU-MIN algorithm. We examine the performance of this and other replacement algorithms via omnet++ simulation environment. Simulation results shows that the proposed LFU-MIN enhances the performance of cluster based cooperative caching in MANETs when compared with LRU and LFU.

Keywords
Adhoc Networks, cache replacement, clustering, cooperative caching, Prefetching, omnet++

1. INTRODUCTION
A mobile ad hoc network (MANETs) is a collection of wireless mobile nodes dynamically forming a network without the aid of any predefined network infrastructure. Despite the wide range of opportunities that MANETs provide, there are still research problems that need to be dealt with before it gets a vote of confidence from the public. Some of which are as follows. In MANETs, mobility of nodes, wireless transmission effect on attenuation, interference and multipath propagation due to the mobility nature of nodes in MANETs the topology changes dynamically. First, accessing remote information station via multi hop communication leads to longer query latency and causes high energy consumption. Second, when many clients frequently access the database server they cause a high load on the server and reduce the server response time. Third, multi hop communication causes the network capacity degrades when network partition occurs. To overcome the above limitations data caching is an efficient methodology to reduce query delay and bandwidth. To further enhance the performance of data caching cluster based cross layer and perfecting techniques are used. The focus of our research will be to improve the overall network performance by reducing the client query delay and response time. In this paper we propose a LFU-MIN cache replacement algorithm for cluster based cooperative caching (CBCC) in MANETs. The rest of the paper organized as follows: section 2 describes the related work. Section 3 describes the overview of CBCC approach. Section 4 describes the proposed cache replacement algorithm for CBCC approach. Section 5 describes the performance evaluation of cache replacement algorithms and section 6 concludes the paper and suggests possible future work.

2. BACKGROUNDAND RELATED WORK
Caching has been widely used in the wired area networks such as the internet, to increase the performance of web services. However the existing cooperative caching schemes cannot be implemented directly in MANETs due to the resource constraints that characterize the networks as a result new approaches have been proposed to tackle the challenges. Many cooperative caching proposals are available for wireless networks. The proposals are grouped based on the usage of underlying routing protocol, cache consistency management and cache replacement mechanism. In [1, 2] different approaches have been introduced to increase data accessibility and to reduce query delay. In cooperative cache based data access in ad hoc networks [1], a scheme is proposed. In this for caching they used cached data and cached path etc. more over the used cache replacement algorithm is only based on least recently used information. The used LRU as a cache replacement algorithm has certain limitations and the above proposed approach doesn't considered Prefetching technique. In [2], a similar approach is proposed for the network integrating ad hoc networks with the internet. Cache replacement algorithms have direct impact on the cache performance. In [3-7], a considerable number of proposals give much higher priority to data accessibility as opposed to accessed latency. So both factors are largely influenced by the caching scheme that the cache management adopted. In [3-7] new cache replacement algorithms are used to make the best use of cache space. But the used traditional replacement algorithms like LRU, LFU, and LRFU have problems. However caching alone is not sufficient to guarantee high data accessibility and low communication latency in dynamic system. To overcome these draw backs, a new approach is proposed in cluster based cross layer design for cooperative caching in MANETs [8].

In the above proposal [8] for cache replacement mechanism they used LRU-MIN as the cache replacement algorithm. But the used LRU-MIN has certain limitations. Among those first, it prefers only small objects to raise the hit ratio. Second, it doesn't exploit the frequency information of memory accesses. Third, the overhead cost of moving cache blocks into the most recently used position each time when a cache block is accessed. In this paper we propose a cache replacement algorithm called LFU-MIN which makes use of frequency of information with

minimal number of page replacements for evicting the objects from the cache. The proposed cache replacement algorithm further enhances the performance of cluster based cooperative caching (CBCC) in MANETs.

3. *SYSTEM ARCHITECTURE*

3.1 CBCC Architecture

CBCC is a cluster-based middleware which stays on top of the underlying network stack and provides caching and other data management services to the upper layer applications in MANET's environment. The instances of CBCC run in each mobile host. The network traffic information which is in the data link layer can be retrieved by the middleware layer for Prefetching purposes.

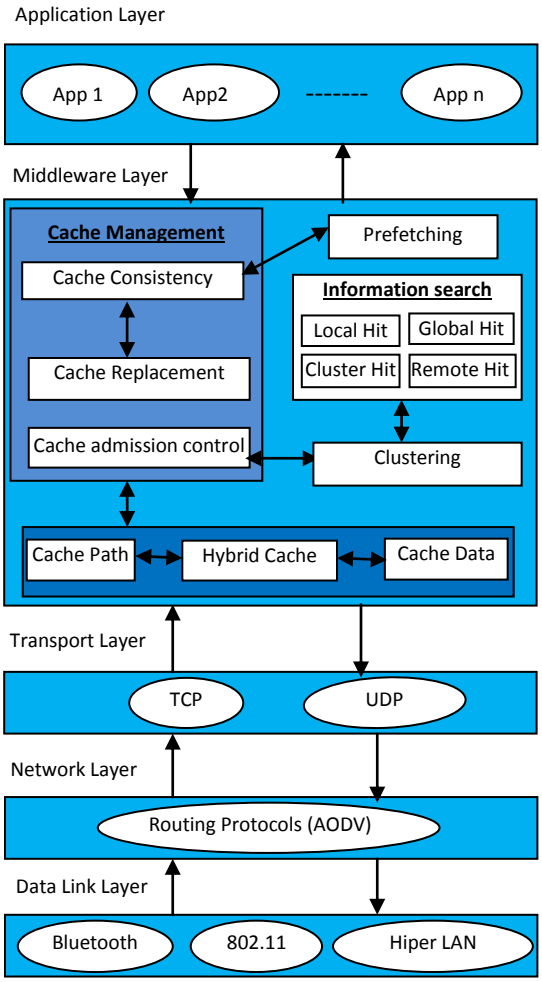

Fig1: System architecture for Cluster-Based Cooperative (CBCC) Caching(CBCC)

Application layer: It is responsible for providing an interface for users to interact with application services or networking services. Application layer uses HTTP, FTP, TFTP, TELNET etc.

Middleware layer: It is responsible for service location, group communications shared memory. Middleware layer consists of various blocks such as cache management, information search, Prefetching and clustering.

*Cache management***:** Cache management includes cache admission control, cache consistency maintenance, and cache replacement.

*Cache admission control***:** In this, a node will cache all received data items until its cache space full. After the cache space becomes full, the received data item will not be cached if the data item has a copy within the cluster.

Cache replacement: When fresh data item is arrived for caching and if cache space is full then the cache replacement algorithm is used to locate one or more cached data items to take out from the cache place. The cache replacement process involves two steps: First, if some of the cached data items become obsolete, these items will be detached to make space for the newly arrived data item. If there is still no enough cachespace after all obsolete items are removed, cache replacement will go to the second step, which is that one or more cached data items will be expelled from the cache space according to some criteria. The various cache replacement algorithms used for this mechanism are LRU, LRU-MIN etc.

Least Recently Used (LRU): It is one of the most widely used cache replacement algorithm, which evicts the objects based on the least recently used information. LRU maintains a hash table for the past accessing of the data. In the head of the table the most recently used information is placed and in the tail of the table the least recently used information is stored. When a new data item is added to the cache, it is added to the tail of the table. Whenever a cache hit occurs the access time of the requested data item is updated and it is moved into the head of the list. After the cache is full, it simply removes the tail element of the list.

LRU_MIN: It uses a technique called least recently used information with minimal number of page replacements. LRU-MIN is also just like LRU. Like LRU, LRU-MIN also maintains and sorted list of documents in the hash table based on the least recently used information i.e. based on the time the document was last used. The only difference between LRU and LRU-MIN is the method of selecting the document for the replacement. Whenever cache needs to replace the document, it searches from the tail of the hash table and evicts the data items only by which have equal or greater size than newly arrived data item size. If all cached documents are smaller than new document, the search is repeated looking for the first two documents greater than half the size of the new document. The process of halving the size and doubling the number of documents to be removed is repeated if large enough documents can still not be found for replacement.

Least frequently used (LFU): evicts the document that has been accessed the fewest times. It is realized by maintaining a reference count for each file in the cache. Each time a cache hit happens, the reference count of the file requested is increased by one. In case of a cache miss and these is no enough free space in the cache, the file(s) with the lowest reference count is (are) replaced.

*Cache consistency***:** The cache consistency strategy keeps the cached data items synchronized with the original data items in the data source.

Information search: Deals with locating and fetching the data item requested by the client.

Prefetching: Responsible for determining the data item to be prefetched from the Data Centre for future use.

Transport layer: It is responsible for providing data delivery transportation between the applications in the network by using the protocols like TCP, UDP. It includes the functionalities like Identifying the services, segmentation, sequencing and reassembling and error correction.

Network layer: It is responsible for providing logical addressing and path determination (routing).The routing protocols such as AODV, DSR, DYMO, etc. are responsible for performing path determination (routing). The current system architecture uses AODV protocol for path determination.

Data link Layer: Provides apparent network services so that network layer can be ignorant about the network topology and provides access to physical networking media. It includes error checking and flow control mechanism.

3.2 Cluster Formation and Maintenance

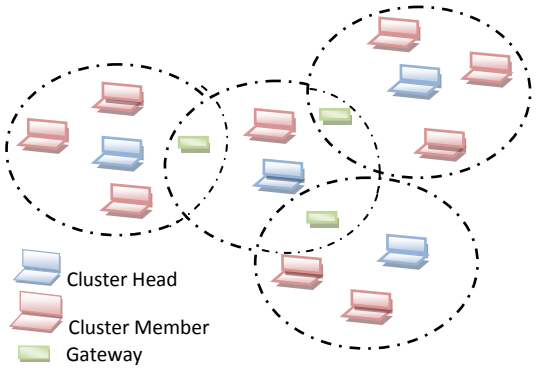

Fig. 2. Clustering Architecture

Clustering is a method used to partition the network into several virtual groups based on the some predefined method. For the cluster formation we use least cluster change algorithm [8] which is an improvement of lowest ID algorithm. Each mobile node has a unique id. The node which has least id in the group is elected as a cluster head. Cluster head maintains a list which maintains the information of all other nodes in the group. In a cluster, the number of hops between any two nodes is not more than two. In the whole network there is no direct connection between the cluster heads. Fig. 2 is an illustration of clustering architecture. In fig. 2, nodes which have pink color are cluster heads, nodes which have green color are gateways, and the rest are cluster members. Cluster member is just like a mobile node it does not have any extra functionality. The node which is common to two cluster heads is elected as a gateway. Gateway is used for providing the communication between two cluster heads. Whenever a node requests for the data, first it has to be checked in the cluster head list. If it is not available in the list of cluster head then the cluster head forwards the requested data item to the other cluster via gateway.

By using LCC we can reduce the frequent changes of cluster head formation. LCC adopts LID to create clusters. If a cluster member moves out of the cluster it won't affect the existing clustering architecture. If two cluster heads exist within the cluster, the lowest id mobile node is elected as a cluster head and if more number of nodes moves out of the cluster will form a new cluster.

3.3 Information Search Operation

Information search operation [8], mainly deals with locating and fetching the data item requested by the client from the cache. This Information search includes 4 cases.

Case 1: *Local hit:* When copy of the requested data item is ordered 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 in a client within the cluster of the requester, the requester sends a request to the Cluster head and the Cluster head returns the address of a client that has cached the data item.

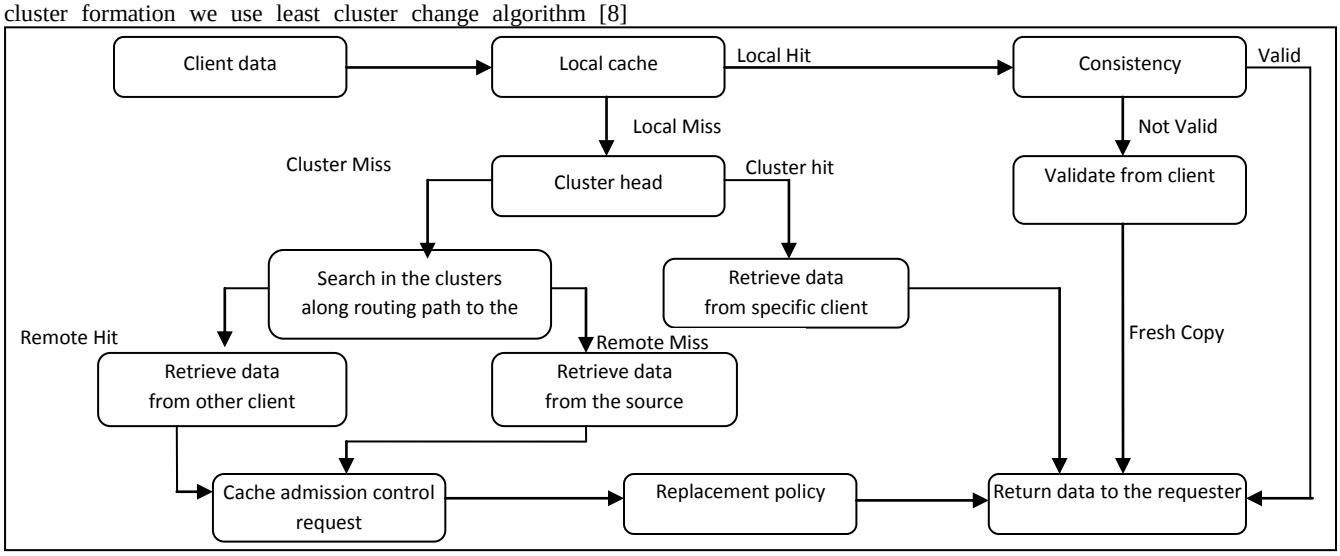

Fig. 3. Information Search Operation

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. When the client data request comes to the mobile node, first it checks in the local hard disk of mobile node i.e. local cache of mobile node. If it is available in the local cache it sends back the reply to the client. Otherwise the request is forwarded to the neighbors based on the cache current state information in the cluster head. If the cluster head has the requested cache state information cluster head gives back to the requester by giving the cluster member id. If it is not available within the cluster then the request is forwarded to the other cluster through gateway. The request is processed the same way and sends back the reply to the requester. Otherwise the request is reached to the data center, the datacenter processes the data request and sends backs the requested information to the client via multi hop communication then the client uses the cache admission control for the consistency check in the cluster. If the same data is available within the cluster, then it won't cache the objects information. If it is not available it will cache the data objects and sends back the cached information to the cluster head for updating in the cluster cache state.

4. PROPOSED CACHE REPLACEMENT ALGORITHM FOR CBCC APPROACH

Algorithm: LFU-MIN

1. Create data item list
2. While
3. Begin
4. for (cache. Length)
5. If (cache item size>=new-item size) then
6. Insert cached item into L and update the counter.
7. End if
8. End For
9. New-item size=new-item size/2
10. End while
11. // Remove items from L using LFU
12. While
13. Begin
14. for (L.length)
15. Delete least frequently used item from L
16. End for
17. End while

LFU-MIN uses a technique called least frequently used information with minimal number of page replacements. LFU-MIN is an advancement of LFU replacement algorithm.LFU evicts the document that has been accessed the fewest times. It is realized by maintaining a reference count for each file in the cache. Each time a cache hit happens, the reference count of the file requested is increased by one. In case of a cache miss and these is no enough free space in the cache, the file(s) with the lowest reference count is (are) replaced. In LFU-MIN we maintain a table which consists of the size of all the cached documents. Whenever a new document has to be fetched into the cache, it checks the availability of free space in the cache, if it is equal or greater than the new document then it fetches the document into the cache and updates the counter information and the size of the document is updated in the size table. In case of a cache miss and these is no enough free space in the cache, then the documents which has the document size equal or greater than the new item size with lowest reference rate are replaced. If the documents available in the cache do have the size which is equal or greater than the new item size and If all cached documents are smaller than new document, the search is repeated looking for the first two documents greater than half the size of the new document with lowest reference rate. The process of halving the size and doubling the number of documents to be removed is repeated if large enough documents can still not be found for replacement.

5. PERFORMANCE EVALUATION OF CACHE REPLACEMENT ALGORITHMS FOR CBCC APPROACH

5.1 Simulation Environment

In this section we have evaluated the performance of LRU, LFU and LFU-MIN cache replacement algorithms in CBCC approach using OMNET++ simulation environment. The simulation parameters used in the experiments are shown in table 1. The simulation was carried out in a grid of 4000×300m with 50 to 100 nodes. The time interval between two consecutive queries generated from each node/client follows an exponential distribution with mean node query delay T_q is taken as 6 sec. The node density can be selected by selecting the number of nodes; here we considered the number of nodes as 70 by default. The bandwidth selected for the transmission is 2mbps and the total transmission range of 250m is considered for the simulation. Each client generates a single stream of read only queries. After a query is sent out, the client does not generate new query until the pending query is served. Each client generates accesses to the data items following Zipf distribution [9] with a skewness parameter (θ) 0.8. If θ = 0, clients uniformly access the data items. As θ is increasing, the access to the data items becomes more skewed. Similar to other studies [10], [11] we choose θ to be 0.8. The AODV routing protocol was used in the simulation. The nodes/clients move according to the random waypoint model.

Initially, the clients are randomly distributed in the area. Each client selects a random destination and moves towards the destination with a speed selected randomly from $[v_{min}, v_{max}]$. After the client reaches its destination, it pauses for a period of time and repeats this movement pattern. The data are updated only by the server. The server serves the requests on FCFS (first-come-first-serve) basis. When the server sends a data item to a client, it sends the TTL value along with the data. The TTL value is set exponentially with a mean value. After the TTL expires, the client has to get the new version of the data item either from the server or from other client (having maintained the data item in its cache) before serving the query.

The Zipf-like parameter [9] can be expressed as

$$P_N(i) = \frac{\Omega}{i^{\theta}} \quad \text{- (3)}$$

Where

$$\Omega = \left(\sum_{i=1}^{N} \frac{1}{i^{\theta}} \right)^{-1} \quad 0 \le \theta \le 1$$

Here N is the total number of data items. And ϴ is the skewness parameter.

Table 1. Simulation Parameters

Parameter	Maximum Capacity	Default value
Simulation area	-	4000*300 m
Database size	-	750 items
Cache size (KB)	50 – 450	80
Size of the document(S_{min})	-	1kB
Size of the document(S_{max})	-	10kB
Transmission range	25-250M	250M
Number of clients	50-100	70
Zipf-like parameter	0.5-1.0	0.7
Time-To-Live (TTL)	200-1000 sec	500
Mean query delay(T_q)	2-100 sec	6 sec
Bandwidth	-	2mb/s
Node speed	2-20 m/s	2 m/s

5.2 Performance Metrics

Performance metrics are used to evaluate and to improve the efficiency of the process. The performance metrics Hit Ratio (HR), Delay Savings Ratio (DSR) are considered in the simulation experiment.

Hit ratio: It is defined as the ratio of number of successful requests to the total number of requests.

$$\text{Hit ratio} = \frac{successfulrequests}{totalnumberofrequests} \quad \text{- (4)}$$

Delay savings ratio: it is defined as the total time taken for the completion of successful requests..

$$DSR = \frac{\sum_i (nr_i * d_i - nv_i * c_i)}{\sum_i f_i * d_i} \quad \text{- (5)}$$

Where nr_i is the number of references to document i, f_i is the total number of references to document i. nv_i is the number of validations performed on document i.

5.3 Cache Performance Comparison

We compared the performance of LRU,LFU and LFU-MIN.As Fig. 4 indicates, LFU-MIN consistently provides better performance than LRU, LFU for all the cache sizes, it improves the Delay Savings Ratio(DSR)on average by 26.5 percent compared with LRU and 8.12percent compared with LFU.LFU-MIN improves the cache hit ratio performance when compared with LRU,LFU. it improves the cache hit ratio performance by 34.35 percent over LRU, and 6.7percent over LFU. LFU-MIN also improves the consistency of the cached documents in addition to improving performance of cache. The performance evaluations of various parameters are plotted by using the graphical representation. In Fig. 4, X-axis represents cache size and Y-axis represents DSR and for fig. 5, X-axis represents cache size and Y-axis represents Hit-Ratio.

Table 2. Obtained computational values

	LRU		LFU		LFU-MIN	
Cache Size (KB)	**DSR**	**HR**	**DSR**	**HR**	**DSR**	**HR**
80	0.14	0.08	0.22	0.11	0.25	0.14
100	0.19	0.12	0.25	0.18	0.28	0.19
150	0.22	0.15	0.28	0.23	0.32	0.24
280	0.26	0.19	0.32	0.28	0.35	0.28
340	0.32	0.25	0.38	0.34	0.39	0.34
400	0.36	0.3	0.41	0.39	0.42	0.43

DSR- Delay Savings Ratio, HR- Hit Ratio

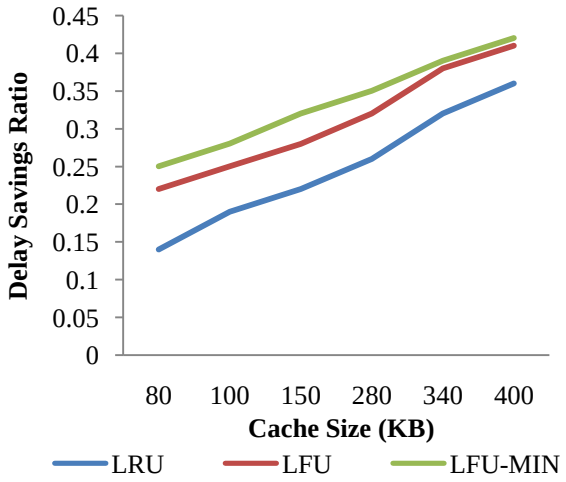

Fig 4: Performance comparison of DSR

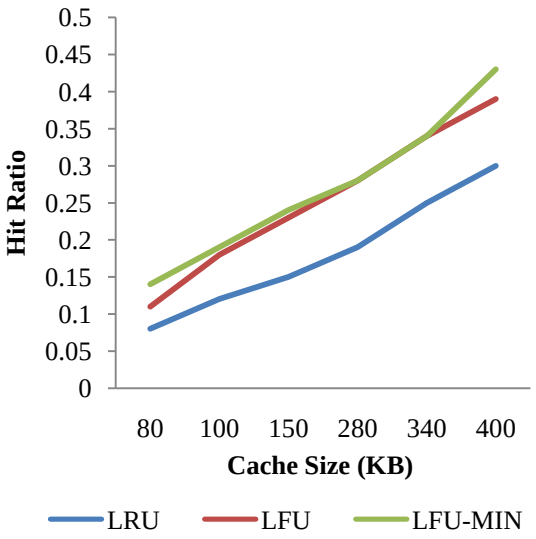

Fig 5: Performance comparison of HR

6. CONCLUSION

In this paper, we proposed a new cache replacement algorithm (LFU-MIN) which works on the principle of least frequently used information with minimal number of page replacements. The proposed new cache replacement algorithm is compared with other cache replacement algorithms using simulation modeling (omnet++). The Simulation results shows that the proposed LFU-MIN cache replacement algorithm enhances the performance of cluster-based cooperative caching (CBCC) in MANETs by improving the cache hit ratio and by reducing the client query response time while accessing the data items from the cache memory as compared with LRU and LFU cache replacement algorithms. For future work there is a need to find out an efficient Prefetching technique which further improves the data accessibility and reduce query delay to compliment the cooperative caching scheme.

7. REFERENCES

[1] G. Cao, L. Yin and C.R. Das. "Cooperative cache-based data access in adhoc networks," IEEE Computer Society, vol.37, 2004, pp.32-39.

[2] M. K. Denko and J. Tian, "Cross-layer design for cooperative caching in mobile adhocnetworks," in *Proc. 5th IEEE, Consumer Communications and Networking Conf. (CCNC)*, 2008, pp. 375–380.

[3] L. Yin and G. Cao, "Supporting cooperative caching in ad hoc networks," IEEE Trans. Mobile Comput., vol. 5, no. 1, pp. 77-89, Jan. 2006.

[4] J.Zhao, P.Zhang and G.Cao, "On cooperative caching in wirelessP2P networks", in Proc.28th Int.Conf.Distributed ComputingSystems(ICDCS2008),2008.

[5] H.Artail, H.Safa, K.Mershad, Z.Abou-Atme, andN.Sulieman, "COACS: A cooperative and adaptive caching system for MANETs,"IEEE Trans. Mobile Comput., vol. 7, no. 8, pp. 961-977, Aug. 2008.

[6] N.Chand,R.C.Joshi,andM.Misra,"Cooperativecachingstrategyin mobile ad hoc networks based on clusters,WirelessPerson.Commun" pp. 41-63, Dec. 2006.

[7] J. Tian and M. K. Denko, "Exploiting clustering and cross-layer design approaches for data caching in MANETs," in *Proc. 3rd IEEE Int. Conf. Wireless and Mobile Computing, Networking and Communications, (WiMob)*, 2007, p. 52.

[8] Mieso K. Denko, Jun Tian, Thabo K. R. Nkwe, and Mohammad S. Obaidat, "Cluster –Based Cross-Layer Design for Cooperative Caching in Mobile Ad Hoc Networks," IEEE Systems Journal, vol. 3, no. 4, Dec 2009.

[9] L. Breslau, P. Cao, L. Fan, G. Phillips and S. Sheker, "Web Caching and Zipf-Like Distributions: Evidence and Implications," *IEEE INFOCOM*, pp. 126-134, March 1999.

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

[11] HuapingShen, Sajal K. Das, Mohan Kumar and Zhijun Wang, "Cooperative Caching with Optimal Radius in Hybrid Wireless Networks," *NETWORKING*, pp. 841-853, 2004.