A Novel Cooperative Caching Scheme for Wireless Ad Hoc Networks: *GroupCaching*

Yi-Wei Ting and Yeim-Kuan Chang
Department of Computer Science and Information Engineering
National Cheng Kung University
701 Tainan, Taiwan R.O.C.
{p7893113, ykchang}@mail.ncku.edu.tw

Abstract

In the mobile ad hoc network, a mobile host can communicate with others anywhere and anytime. Cooperative caching scheme can improve the accessibility of data objects. However, the cache hit ratio is reduced and access latency becomes longer significantly due to the mobility of MHs, energy consumption in battery, and limited wireless bandwidth. In this paper, we propose a novel cooperative caching scheme called *GroupCaching (GC)* which allows each MH and its 1-hop neighbors form a group. The caching status is exchanged and maintained periodically in a group. By using the proposed GroupCaching, the caching space in MHs can be efficiently utilized and thus the redundancy of cached data is decreased and the average access latency is reduced. We evaluate the performance of the *GroupCaching* by using NS2 and compare it with the existing schemes such as *CacheData* and *ZoneCooperative*. The experimental results show that the cache hit ratio is increased by about 3%~30% and the average latency is reduced by about 5%~25% compared with other schemes.

Keywords: Ad hoc, Cache placement, Cooperative caching, Group

1. Introduction

Mobile ad hoc networks (MANETs) are formed with mobile hosts (MHs) such as notebook, PDA, or cell phone and so on. These mobile devices can create a wireless network dynamically among themselves without the aid of any network infrastructure. Every MH can move arbitrarily and communicate with one another by using multi-hop wireless links. Each MH acts as a router and forwards data packets to other neighbors in the coverage of transmission range. MANET is very useful under certain environments, such as battles, disaster rescue, earthquake recovery, and exploration of an area, etc.

In the last years, research studies in MANETs mostly focus on designing efficient multi-hop ad hoc routing protocols for packet forwarding between two nodes. However, the ultimate goal of a routing protocol is to create a routing path for data communication. Therefore, how to improve the accessibility of data object by using caching techniques is another important issue.

Caching techniques are an efficient solution for increasing the performance in message or data communication. It has been widely used in different fields such as CPU design, multi-processor, memory architecture or router design. Furthermore, Interent uses cache placement and replacement in proxy servers [4] and cooperative caching architecture [5] to reduce the network traffic and average latency of data query significantly. The original idea of caching is that the data accessed by MHs has the properties of temporal and spatial locality. Higher temporal and spatial locality ensures that most accesses will go to the data that were accessed recently in the past and that reside in the cache. Therefore, caching frequently requested data can improve the performance of data communication.

In a MANET, if the MHs cache some frequently requested data, the cached data can be served for others later. The requester needs not to retrieve the data from the remote server (data source) and then requests the data from the caching node. As a result, the data accessibility is enhanced. Furthermore, cooperative caching techniques allow several caching nodes to coordinate the caching tasks such as the redirection of data requests, cache placement, or cache replacement. Existing cooperative caching schemes [1] [3] [6] [7] [8] [11] [12] allow a data object can be cached by multiple hosts for enhancing the accessibility of data objects.

There are some challenges and issues such as mobility of MHs, power consumption in battery, and limited wireless bandwidth when we employ caching techniques in MANETs for data communication. Due to the movement of MHs, MANETs may be partitioned into many independent networks. Hence, the requester can not retrieve the desired data from the remote server (data source) in another network. The entire data accessibility will be reduced. Also, the caching node may be disconnected from the network for saving power. Thus, the cached data in an MH may not be retrieved by other MHs and then usefulness of the cache is reduced.

The MHs also decide the caching policy according to the caching status of other MHs. However, the existing cooperative caching schemes in a MANET lack an efficient protocol among the MHs to exchange their localized caching status for caching tasks. Therefore, in this paper, we propose a novel cooperative caching scheme called *GroupCaching (GC)* which maintains localized caching status of 1-hop neighbors for performing the tasks of data discovery, caching placement, and caching replacement when a data request is received in an MH. Each MH and its 1-hop neighbors form a group by using the *"Hello"* message mechanism. In order to utilize the cache space of each MH in a group, the MHs periodically send their caching status in a group. Thus, when caching placement and replacement need to be performed, the MH selects the appropriate group member to execute the caching task in the group. In our proposed cooperative caching protocol, the MHs know the caching status of their neighbors. Based on the proposed *GroupCaching*, the redundancy of cached data objects can be reduced because the MHs can check the caching status of other group members for deciding the placement and replacement. Because more cache space can be utilized, each MH can store more different data objects in a group and then increases the data accessibility.

The rest of the paper is organized as follows: Section 2 reviews the related works for cooperative caching schemes in mobile ad hoc networks. The proposed *GroupCaching* scheme is presented in Section 3. Section 4 shows the experimental results of GroupCaching compared with existing schemes. The conclusion and future work are presented in section 5.

2. Related works

2.1 CacheData and CachePath

The CacheData [1][12] scheme considers the cache placement policy at intermediate nodes in the routing path between the source and the destination. The node caches a passing-by data item locally when it finds that the data item is popular, i.e., there were many requests for the data item, or it has enough free cache space. Since CacheData needs extra space to save the data, it should be used prudently. A conservative rule is proposed as follow: *A node does not cache the data if all requests for the data are from the same node*. However, there is no cooperative caching protocol among MHs. Each MH independently performs the caching tasks such as placement and replacement. CachePath [1][12] is also proposed for redirecting the requests to the caching node. In MANETs, the network topology changes fast and thus, the cached path may become invalid due to the movement of MHs.

2.2 ZoneCooperative

The ZoneCooperative [11] scheme considers the progress of data discovery. Each client has a cache to store the frequently accessed data items. The data items in the cache satisfy not only the client's own requests but also the data requests passing through it from other clients. For a data miss in the local cache, the client first searches the data in its zone before forwarding the request to the next client that lies on a path towards server. However, the latency may become longer if the neighbors of intermediate nodes do not have a copy of the requested data object for the request.

2.3 NeighborCaching

The concept of NeighborCaching (NC) [13] is to utilize the cache space of inactive neighbors for caching tasks. The basic operations of NC are as follows. When a node fetches a data from a remote node, it puts the data in its own caching space for reuse. This operation needs to evict the least valuable data from the cache based on a replacement algorithm. With this scheme, the data that is to be evicted is stored in the idle neighbor node's storage. In the near future, if the node needs the data again, it requests the data not from the far remote source node but from the near neighbor that keeps the copy of data. The NC scheme utilizes the available cache space of neighbor to improve the caching performance. Howerver, it lacks of the efficiently cooperative caching protocol among the MHs.

3. Proposed GroupCaching scheme

3.1 Motivation

In mobile ad hoc networks, MHs can move arbitrarily anytime and communicate with another one for data transmission. Although cooperative caching can provide the high accessability of data objects, the caching performance (cache hit ratio and average latency) can be reduced significantly due to the property of dynamic topology in MANETs. Possible reasons are listed as follows:

1. The caching nodes maybe shut down (leave) from the MANET because the energy of battery is exhausted. In other words, the cached data of caching nodes can not be serviced for others and will be removed after it re-connects the network. On the contrary, when an MH connects (joins) the network, its content of cache space is empty. If we can utilize its cache space as soon as possible, the caching performance will be enhanced.
2. The cache size of MHs always smaller than general personal computer. If there is no cooperative caching protocol among MHs, the cache space in all MHs can not be utilized effectively. If MHs can know the caching status of their neighbors, MHs can integrate their available cache space and efficiently cache data.
3. In MANETs, on-demand routing protocol is prefered to be used for saving eneregy and bandwidth. In this way, the caching node is selected based on the routing path between the source and the destination. Thus, if the multiple routing paths are passed through the same nodes, their cache space easily become full and the energy will be consumed fast.

In order to deal with above conditions, we design a cooperative caching protocol among MHs. Our goal is to provide a caching and power efficient protocol in MANETs. First, an MH and its neighbors form a group. The group definition is presented in Section 3.2.2.

Second, each MH sends their caching status to its group periodically. Third, when a data object needs to be placed or replaced in an MH, its group members cooperate to perform the tasks of placemant and replacement. We assume MHs have the ability to place the data in its group member [13]. The group caching has several benefits. First, it can reduce the redundancy of cached data object because MHs can check the caching status of other group members when receiving a data object. Second, it can store more different data objects and then increase the data accessibility. Third, the group can store more data objects from the destinations than an MH because the group members are cooperative to cache the data objects.

3.2 GroupCaching (GC)

3.2.1 Network model

We model the MANET environment as a graph *G = (V, E)*, where *V* represents the set of mobile hosts (MHs) in the network, and *E* is the set of links. An edge $e = (u, v) \in E$, where $u, v \in V$, exists if and only if *u* is in the transmission range of v and vice versa.

All links in *G* are bi-directional, i.e., if *u* is in the transmission range of *v*, *v* is also in the transmission range of *u*. The network is assumed to be in a connected state. If it is partitioned, each component is treated as an independent network.

3.2.2 Group definition

Each MH and its one-hop neighbors form a group. A one-hop neighbor can be covered in the area of transmission range from an MH. Each MH has a group member ID. The group member ID may be the IP address or unique host ID. In order to maintain the connectivity of a group, we employ the mechanism of *"Hello"* messages. *"Hello"* messages are sent locally in each MH. These messages are sent periodically as a *"Keep-Alive-Signal"*. In this way, each MH knows who its one-hop neighbors are. Generally, we can obtain the *k*-hop neighbor information by piggybacking the *(k–1)*-hop neighbor information in *"Hello"* messages. However, the biggest concern is the energy consumption in MHs and constrain of wireless bandwidth. Therefore, in our proposed GC scheme, each MH only maintains one-hop neighbors in a group. Figure 1 illustrates the group in the view of MH *D*. MH *D* and its one-hop neighbors {*C*, *E*, *G* and *H*} form a group. Each MH sends their caching status to its group members. Thus, when a data item is received in MH *D*, MH *D* can check the caching status of each group member and select the appropriate group member to place the cached data.

3.2.3 Cooperative caching tables

In order to record the caching status of group members, each MH maintains two tables (*self_table* and *group_table*) to record caching status of itself and its group members respectively. The *self_table* contains the following fields: {*Cached data id*, *Cached data item, Data source id*, *Timestamp*}. The *self_table* is updated when the placement is performed in an MH. The

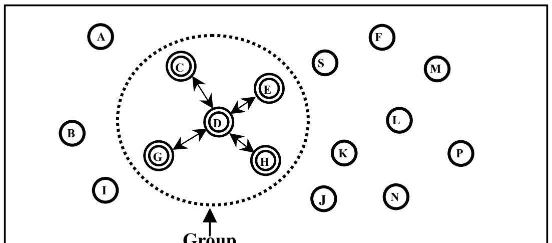


Figure 1: The group in the view of MH *D*.

group_table contains the following fieldss: {*Cached data id*, *Data source id*, *Group member id*, *Timestamp*}. The *Timestamp* is the caching time of the data object. When an MH receives the notification of caching status from the group member, it updates the *group_table*. If an MH does not receive the *"Hello"* message during a pre-defined number of *"Hello"* cycles from the neighbor, it means the neighbor is leaving or shutdown. The *group_table* also needs to be updated and remove the related records of the leaving neighbor when an MH leaves. From the *group_table*, an MH knows which data object is cached in which group member. Therefore, when a request is received in an MH, it can search the *self_table* and *group_table* to find the record of the request for data object exist.

3.2.4 Caching Control message

We design a caching control message to exchange the caching status in a group periodically. In our experimental, we set exchange interval at every second. The caching control message contains the following fields: {*Group member id*, *Cached data id*, *Timestamp, Remaining available cache space*}. The caching control message is periodically sent by MHs. When an MH receives the caching control message, it updates its *group_table*. In other words, each MH can maintain localized caching statues of one-hop neighbors for performing cache placement and replacement.

3.3 Placement and replacement policy

In this section, we present how and where to place the data object in a group member when an MH receives a data object from the destination. The detailed algorithm is shown in Figure 2. Based on the usage of caching control message, each MH knows the remaining available cache space of other MHs in a group and the IDs and timestamps of their cached data objects.

First of all, when an MH receives a data object (called receiving MH), it caches the data object if the cache space is enough. Otherwise, the receiving MH checks the available cache spaces of its group members. If the available cache space of any group member is sufficient to store the data object, the receiving MH puts the data object to the group member randomly.

Second, if the available cache space of every group member is not sufficient to cache the received object, the reciving MH lookups the *group_table* to see if there exists a group member that already caches the data object. If yes, the data object is not cached. If no, the

```
When a data object $d_i$ arrives
01  IF (there is a valid copy in self_table or group_table) THEN
02      no cache
03  ELSE
04      Placement_Algorithm($d_i$)

Placement_Algorithm ($d_i$)
01  IF (available cache space of receiving MH > size of $d_i$) THEN
02      Cache $d_i$; Return
03  ELSE IF (available cache space of group member > size of $d_i$)
        THEN Push $d_i$ to the group member randomly; Return
04  ELSE
05      Lookup the $d_i$ in the group_table
06      IF (Find) THEN
07          No cache
08      ELSE
09          Select the "appropriate group member" in its group
10          Push $d_i$ to selected group member
```

Figure 2: The process of receiving a data item.

receiving MH selects the "appropriate group member" which has the oldest timestamp of the cached data object in the *group_table* and sends the data object to that group member to do placement. When a group member receives a data object from the receiving MH, it repeatedly performs the LRU replacement operations to increase available cache space until the received data object can be cached.

3.4 Data discovery process

The process of data discovery performs the searches in the caching nodes for the requested object. In GroupCaching, when a requester (source) wants to retrieve a data object from the data source, it first checks its *self_table* to see if the data object exists locally. If yes, it returns the data object (cache hit) to the application. If no, it lookups its *group_table* for the data object, if yes, the requester redirects the data request to that group member, and waits the replied data object (remote cache hit).

If the requester can not find any cached record for the desired data object in the *self_table* and *group_table, it starts to execute the data discovery process.* Initially, the requester constructs a routing path to the destination and sends the data request to the next MH in order to reach the data source (destination). Figure 3 shows the algorithm of receiving a data request in an MH. When the intermediate nodes receive a data request in the routing path, they lookup their *self_table* and *group_table* for the data request. The process of lookup first searches *self_table* and then searches the *group_table*. If the receiving MH can not find the record for the request in its *selft_table* and *group_table*, it forwards the request to the next MH on the routing path.

If the destination (data source) receives the data request, it replies the data object via the routing path. When the intermediate node receives the pass-by data object, it performs the cache placement and replacement described in section 3.3 according to their *self_table* and *group_table*.

3.5 Cache consistency problem

There are two schemes that can deal with the cache consistency problem: *weak consistency* and *strong consistency*. Under the weak consistency, a cached data object is associated with an attribute, TimeToLive (TTL). If the TTL time expires, the cached data object is removed. Under the strong consistency, if a cached data object is requested, the caching node first asks the data source to see if the cached data object is valid or not. Because of the energy concern and the constrain of wireless bandwidth, we prefer using the weak consistency in mobile ad hoc networks.

```
When a request for data item $d_i$ arrives
01  IF (there is a valid copy in self_table) THEN
02      send $d_i$ to the requester
03  ELSE IF (there is a valid entry in group_table) THEN
04      re-direct the request to the group member
05  ELSE
06      forward the request to the next MH by routing path
```

Figure 3: The process of receiving a data request in an MH.

Table 1: The simulated paremeters.

Simulator	Network Simulator (NS2) [9]
Simulation time	6000 seconds
Network size	1500m x 500m
Mobile host	100nodes
Transmission range of MH	100m
Mobility model	Random way point
Speed of mobile host	1~10 m/s randomly
Total of data item set	1000 data item
Average query rate	0.2 / second
Hot data	20% of total data item set
Probability of query in hot data	80%
Data size	10kBytes
Cache size	200kBytes, 400kBytes, 600kBytes, 800kBytes, 1000kBytes, 1200kBytes, 1400kBytes
Compared schemes	CacheData [1], ZoneCooperative [2], Proposed GroupCaching
Replacement policy	LRU

4. Performance Evaluation

The performance evaluation is shown in this section. The simulation model is given in Section 4.1. In Section 4.2, we verify the results of SimpleCaching, CacheData [1],[12] and ZoneCooperative [11] and compared with the proposed GroupCaching. In SimpleCaching, only requester caches the replied data object for itself. All schemes use LRU as the cache replacement policy. The performance metrics is introduced in Section 4.3. Section 4.4 shows the results in performance evaluation.

4.1 The simulation model

The simulation is performed on NS2 [9] with the CMU wireless extension. In our simulation, the AODV routing protocol [14] was tested as the underlying ad hoc routing algorithm. The simulation time is set to 6000 seconds. The number of mobile hosts is set to 100

Table 2: Average hop count under different node cache size (a) and leave/join rate (b).

Cache size	200KB	400KB	600KB	800KB	1000KB	1200KB	1400KB
SimpleCache	4.75	4.79	4.92	4.95	4.81	4.86	5.21
CacheData	4.22	4.15	3.82	3.93	3.81	3.8	3.77
ZoneCooperative	4.1	3.63	3.95	3.67	3.51	3.32	3.39
GroupCaching	3.94	3.59	3.69	3.51	3.26	3.03	3.255

(a) Experiment under different cache size

Leave/Join rate	1/20sec.	1/40sec.	1/60sec.	1/80sec.	1/100sec.	1/120sec.
SimpleCache	4.72	4.64	4.737	4.99	4.64	4.81
CacheData	4.51	4.46	4.27	3.955	3.7	3.81
ZoneCooperative	4.61	4.37	4.05	4.05	3.62	3.51
GroupCaching	4.49	4.3	3.93	3.77	3.59	3.26

(b) Experiment under different join/leave rate

in a fixed area. We assume that the wireless bandwidth is 2MB/s and the radio range is 100m. There are totally 1000 data items distributed uniformly among all MHs. The number of hot data objects is set to 200 and all hot data objects are distributed uniformly among all MHs. The probability of queries for the hot data is set to 80%. The query rate of MHs is set to 0.2/second. In order to simulate the node join and leave operations, we set a join/leave rate. If the value of join/leave rate is 20, there will be ten MHs randomly joining and leaving the network every 20 seconds. If an MH joins or leaves the network, its content of cache will be cleared.

4.2 The node movement model

We model the movement of nodes in a 1500m x 500m rectangle area. The moving pattern follows the random way point mobility model [15]. Initially, nodes are placed randomly in the area. Each node selects a random destination and moves toward the destination with a speed selected randomly from (0 m/s, 10 m/s). After the node reaches its destination, it pauses for a random period of time and repeats this movement pattern. The detail of other simulation parameters is shown in Table 1.

4.3 Performance metrics

The evaluated performance metrics are average hop count, cache hit ratio (includes remote cache hit ratio in remote caching nodes), and average latency of data objects.

Average hop count: The number of hop counts between the source and the destination or caching nodes.

Cache hit ratio: The combined cache hit ratio in the requester and its group members

Average latency: The time interval between the time of generating a query in the requester and the time of receiving requested data object from the data source.

4.4 Simulation results

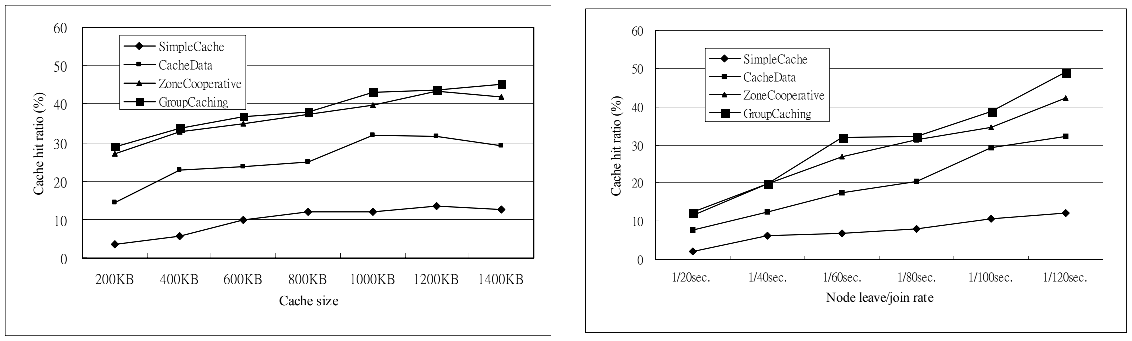
4.4.1 Average hop count

We first measure the hop counts in all schemes. Table 2 shows the average hop count between the source and the destination when a requester wants to retrieve a data object. The destination can be the data source or intermediate caching nodes. The simulation is run under different cache sizes and different join/leave rates. In all schemes, when the cache size is large, the average hop count is reduced. In GroupCaching, the average hop count is the lowest because the GroupCaching improves the cache hit ratio and then reduce the average hop count.

4.4.2 Cache hit ratio

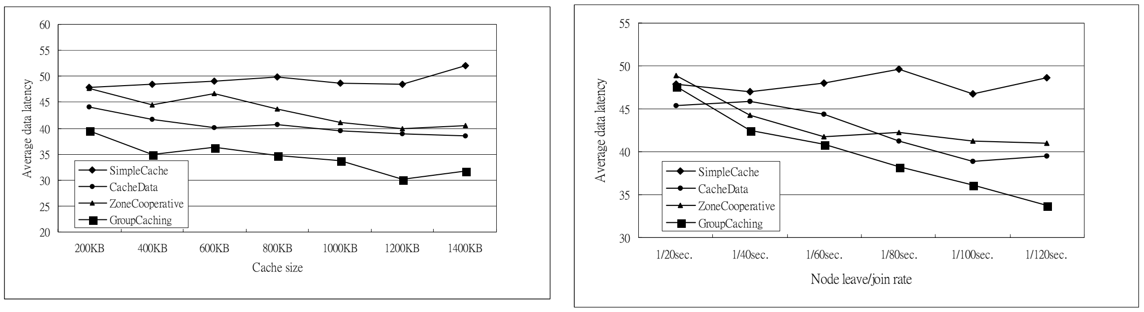
Figure 4 shows the cache hit ratios of MHs under different cache sizes and join/leave rates. The measured cache hit ratio includes the cache hit (local cache hit) in the requester and cache hit in the other MHs (remote cache hit) except the data source. The cache size is set to 200KB, 400KB, 600KB, 800KB, 1000KB, 1200KB and 1400KB. The size of a data item is set to 10KB. The pair of source and destination nodes is randomly selected in the simulation. In general, the cache hit ratio increases while the cache size increases. Figure 4 (a) shows the GroupCaching has a higher cache hit ratio than others because both the MH and its group members can store data objects. These cached data objects improve the cache hit ratio. Figure 4 (b) shows the experimental results under the dynamic topology. In every 20, 40, 60, 80, 100, and 120 seconds, ten MHs are selected randomly for joining and leaving the network. When an MH leaves the network, it removes all cached data objects. When the MH joins the network, the content of its cache is set to empty.

GroupCaching shows the highest cache hit ratio because it utilizes all the available cache space of neighbors (group members). When an MH joins the network, its available cache space can be utilized by other MHs. Therefore, In GroupCaching, the cache hit ratio is higher than other schemes. In ZoneCooperative



(a) Cache hit ratio verus cache size (b) Cache hit ratio verus leave/join rate

Figure 4: Cache hit ratio under different node cache size (a) and leave/join rate (b).



(a) latency verus cache size (b) latency verus leave/join rate

Figure 5: Average data latency under different node cache size (a) and leave/join rate (b).

and CacheData schemes, there is no cooperative caching protocol among MHs. So the MH can not efficiently integrate their neighbor's cache space.

4.4.3 Average latency

Figure 5 shows the average latency under different cache sizes and different join/leave rates. We know that ZoneCooperative scheme has no cooperative protocol among the MHs. Therefore, when an MH receives a data request, it needs to send a request to its zone and waits for the response. As a result, it leads to the long latency if there is no cache record in a zone along the routing path. In GroupCaching, the MH and its one-hop neighbors form a group. If a data request is received, the MH can check its *self_table* and *group_table* immediately. No communication with its neighbors is needed to know the caching status in other group members. As a result, the average latency is reduced. Also, due to the placement and replacement algorithms executed in a group, all the MHs in a group member can cache more data objects and then reduce the redundancy of the cached data. Therefore, the average latency is reduced compared with other schemes.

5. Conclusion

In this paper, we propose a cooperative caching scheme (GroupCaching) for mobile ad hoc networks. MHs maintain the localized caching status among the group members. Therefore, the MHs can cooperative to store different data objects. Furthermore, if an MH has available cache space, it can be utilized by its neighbors as soon as it joins a group. It improves the cache hit ratio and reduces the average lateny compared with existing schemes. In the future work, we will investigate the integration of broadcasting and cooperative caching.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Council, Republic of China, under Grant NSC-95-2221-E-006-163 -MY2.

Reference

[1] L. Yin and G. Cao, "Supporting Cooperative Caching in Ad Hoc Networks,"IEEE INFOCOM, pp. 2537-2547, March 2004.
[2] C. Aggarwal, J. Wolf, and P. Yu, "Caching on the World Wide Web," IEEE Trans. Knowledge and Data Eng., vol. 11, no. 1, Jan./Feb. 1999.
[3] W. Lau, M. Kumar, and S. Venkatesh, "A Cooperative Cache Architecture in Supporting Caching Multimedia Objects in MANETs," Proc. Fifth Int'l Workshop Wireless Mobile Multimedia, 2002.
[4] J. Shim, P. Scheuermann, and R. Vingralek, "Proxy Cache Algorithms: Design, Implementation, and Performance," IEEE Trans. Knowledge and Data

Eng., vol. 11, no. 4, July/Aug. 1999.

[5] D. Wessels and K. Claffy, "ICP and the Squid Web Cache," IEEE J. Selected Areas in Comm., pp. 345-357, 1998.

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

[7] F. Sailhan and V. Issarny, "Cooperative Caching in Ad Hoc Networks,"International Conference on Mobile Data Management (MDM), pp. 13-28, 2003.

[8] C.-Y. Chow, H.V. Leong and A. Chan, "Peer-to-Peer Cooperative Caching in Moible Environments," Proceedings of the 24th Intl. Conf. on Distributed Computing Systems Workshop (ICDCSW), 2004.

[9] K. Fall and K. Varadhan, "The NS2 manual", the VINT Project, http://www.isi.edu/nsnam/ns/. Apr. 2002.

[10] J. Broch, D. A. Maltz, D. B. Johnson, Y. C. Hu, and J. Jetcheva, "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols," Proc. of ACM/IEEE Mobicom'98, Dallas, TX, 85-97.

[11] Chand, N. Joshi, R.C., and Misra, M., "Efficient Cooperative Caching in Ad Hoc Networks Communication System Software and Middleware," 2006. Comsware 2006. First International Conference on 08-12 Jan. 2006 Page(s):1 - 8

[12]Liangzhong Yin; Guohong Cao, "Supporting cooperative caching in ad hoc networks," IEEE Transactions on Mobile Computing, Volume 5, Issue 1, Jan. 2006 Page(s):77-89

[13] Joonho Cho, Seungtaek Oh, Jaemyoung Kim, Hyeong Ho Lee, Joonwon Lee, "Neighbor caching in multi-hop wireless ad hoc networks," IEEE Communications Letters, Volume 7, Issue 11, Nov. 2003 Page(s):525 – 527

[14] C. Perkins, E. Belding-Royer, and I. Chakeres, "Ad Hoc On Demand Distance Vector (AODV) Routing," IETF Internet draft, draft-perkins-manet-aodvbis-00.txt, Oct. 2003.

[15] T. Camp, J. Boleng, and V. Davies, 2002. A Survey of Mobility Models for Ad Hoc Network Research. Wireless Comm. & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications, 2 (5): 483-502.