# A Review of Cooperative Cache Management Techniques in MANETs

Prashant Kumar
prashantkumar32@gmail.com

Naveen Chauhan
naveen@nitham.ac.in

Department of Computer Science and Engineering
National Institute of Technology, Hamirpur (H.P.) INDIA

***Abstract- Caching of frequently accessed data in ad hoc networks is a potential technique that can improve the data access, performance and availability. A cooperative cache-based data access framework lets mobile node cache the data or the path to the data to reduce query delays and improve data accessibility. Due to mobility and resource constraints of ad hoc networks, cooperative caching techniques designed for wired network may not be applicable to ad hoc networks. The objective of cooperative caching is to improve data availability and access efficiency by collaborating local resources of mobile devices. This paper reviews the various cooperative cache management techniques in the mobile ad-hoc networks.***

**Keywords: Mobile adhoc networks, cooperative caching, cache resolution**

## 1. INTRODUCTION

Mobile Ad hoc Network are autonomously structured multi-hop wireless links in peer to peer fashion without aid of any infrastructure network. Due to lack of infrastructure support, each node in network act as router, coordinating to forward data packets to other nodes. Rapid progress in portable computer technologies allows MANET to be used in number of areas such as military application, industrial and commercial areas.

Example of Ad hoc Network is a battlefield. Several Commanding Officers and group of soldiers form an Ad hoc Network. Each higher Officer have relatively powerful data center all Officers under them have to access data centers of higher officers to get various data needed by them. Soldiers under these lower rank Officers access data from stores of these Officers. If one Soldier access some data, it may be possible that nearby soldiers share common operation and require same data sometimes later. Such scheme saves large amount of bandwidth, time and battery power.

Mobile host cooperate with each other to forward data and mobile host have peer to peer connection among themselves. There are several characteristics of Mobile Ad Hoc network. Firstly, Mobile devices are frequently disconnected due to mobility or the need to conserve power. Secondly, Devices employ multi-hop communication through unreliable links, which may cause long Communication delay. Third, Broadcast in Mobile Ad Hoc Network is costly thus traditional cache consistent scheme are not suitable for these network.

Mobile Ad hoc networks are ideal in situations where installing an infrastructure is not possible because the infrastructure is too expensive or too vulnerable. This type of network can communicate with external networks such as Internet through a gateway [10]. However, MANETs are limited by intermittent network connections, restricted power supplies, and limited computing resources. These restrictions raise several new challenges for data access applications with the respects of data availability and access efficiency. In ad hoc networks, mobile nodes communicate with each other using multihop wireless links. Due to a lack of infrastructure support, each node acts as a router, forwarding data packets for other nodes. Most previous research in ad hoc networks focused on the development of dynamic routing protocols that can efficiently find routes between two communicating nodes. Although routing is an important issue, but the ultimate goal of ad hoc networks is to provide mobile nodes with access to information.

In ad hoc networks, due to frequent network partition, data availability is lower than that in traditional wired networks. This problem can be solved by caching data items on mobile hosts. However, the movement of nodes, limited storage space and frequent disconnections limit the availability. By the caching of frequently accessed data in ad hoc networks we can improve the data access, performance and availability. A data management in adhoc network that is based on cooperative caching data access framework lets mobile node to cache the data or the path to the data to reduce query latency and improve data accessibility. Due to mobility and resource constraints of ad hoc networks, caching techniques designed for wired network may not be applicable to ad hoc networks.

## 2. WHY CACHING?

Let consider a scenario in which mobile devices always retrieve data from the data center. This may result in a large amount of traffic in the MANET. This, apparently, is undesirable as traffic directed to

the data center consumes wireless bandwidth as well as power of mobile devices. In addition, a mobile host suffers from high access latency if it is distant from the data center, and packet loss probability for long-distance data access is high. Furthermore, traffic near the data center will be heavy, and this leads to a potential performance bottleneck. These problems are more pronounced when the network size is large, which results in poor scalability of the system. The above observations motivate researchers to investigate data caching techniques for MANETs. With data cached in mobile nodes, a data request may be satisfied by a nearby caching site, instead of being serviced by the data center.

In many applications, mobile nodes in a MANET share common interests. In this scenario, sharing cache contents between mobile nodes offers significant benefits. Typically, nodes cache data items for serving their own needs. Cache sharing, however, allows geographically neighboring mobile nodes to access each other's cache contents. By doing so, the number of long-distance data accesses to the data center can be reduced. The key to this technique is that a node has to know if there is some node in its vicinity that has cached the data it requires and where it is, if any. One approach to deal with this requirement is to let a mobile node record the caching information about a nearby node while forwarding the data requested by the node. The caching information can subsequently be used to direct requests for the same data to the caching site.

If mobile users around infostations, which have limited coverage, form an ad hoc network, a mobile user who moves out of the range of a particular infostation can still access the data it contains. If one of the nodes along the path to the data source has a cached copy of the requested data, it can forward the data to the mobile user, saving bandwidth and power. Thus, if mobile nodes can work as request-forwarding routers, they can save bandwidth and power and reduce delays. Since MANETs are mobile and constrained by limited energy, bandwidth, and computation power, which is a big concern when designing protocols for such networks.

## 3. COOPERATIVE CACHING IN MOBILE ADHOC NETWORK

As we have seen that cooperative caching is helpful to reduce the use of network bandwidth and access time to retrieve the data from the data center. Many researchers provide various techniques in order to retrieve the data more efficiently. Some of the techniques are described here.

### 3.1 PUSH AND PULL APPROACH

The two basic types of cache sharing techniques are push based and pull based. With push-based cache sharing, when a node acquires and caches a new data item, it actively advertises the caching event to the nodes in its neighborhood. Mobile nodes in the vicinity will record the caching information upon receiving such an advertisement and use it to direct subsequent requests for the same item. This scheme enhances the usefulness of the cached contents. The cost we have to pay is the communication overhead for the advertisement; an advertisement is useless if no demands for the cached item arise in the neighborhood.

In the push-based scheme, the caching information known to a node may become obsolete due to node mobility or cache replacement. The pull-based approach may overcome this problem. With pull-based cache sharing, when a mobile node wants to access a data item that is not cached locally it will broadcast a request to the nodes in its vicinity. A nearby node that has cached the data will send a copy of the data to the request originator (a pull operation). Unlike pushing, pulling allows the node to utilize the latest cache contents. However, in contrast with the pushing technique, the pulling scheme has two drawbacks:

**1.** In case the requested data item is not cached by any node in the vicinity, the requester node will wait for the time-out interval to expire before it proceeds to send another request to the data center. This will cause extra access latency, and the pulling effort is in vain.

**2.** Pulling resorts to broadcast to locate a cached copy of an item. In addition, more than one copy will be returned to the request originator if multiple nodes in the neighborhood cache the needed data. This introduces extra communication overhead. [1]

Another issue of concern is the limited cache space that is available in a mobile node. Hence, a cache replacement mechanism must be in place for evicting data items from the cache to make room for a newly acquired one, when the cache is full. Since cache contents of a node are shared by other nodes, a good cache replacement policy should take into consideration the access demands from the entire neighborhood.

### 3.2 COOP – A cooperative caching service in MANETs

Yu Du [2, 3] et. al. presents COOP, a novel cooperative caching scheme for on-demand data access applications in MANETs. The objective is to improve data availability and access efficiency by collaborating local resources of mobile nodes. The cooperation of caching nodes is twofold. First, a caching node can answer the data requests from other

nodes. Second, a caching node stores the data not only on behalf of its own needs, but also based on other nodes' needs. COOP addresses two basic problems for cooperative caching in MANETs:
1. Cache resolution – how does a mobile device decide where to fetch a data item requested by the user?
2. Cache management – how does a mobile device decide which data item to place/purge in its local cache?

For cache resolution, COOP tries to discover a data source which induces less communication cost by utilizing historical profiles and forwarding nodes. For cache management, COOP minimizes caching duplications between neighbor nodes and allows cooperative caches to store more distinctive data items to improve the overall performance.

### 3.2.1. CACHE RESOLUTION

Cache resolution addresses how to resolve a data request with minimal cost of time, energy, and bandwidth. In cooperative caching, the emphasis of cache resolution is to answer how nodes can help each other in resolving data requests to improve the average performance. In COOP the authors give three cache resolution schemes:
1. Hop-by-hop cache resolution
2. Zone-based cache resolution
3. The cocktail resolution scheme

For on-demand data access applications, the traditional way of resolving a data request is to check the local cache first and send the request to the server after local cache misses. This scheme is referred to as SimpleCache in [5]. This scheme works well as long as the connection to the server is reliable and not too expensive; otherwise, it results in failed data requests or request timeouts. To increase data availability and reduce the cost in terms of increased data access latency and increased energy consumption, hop-by-hop cache resolution allows a node on the forwarding path to serve as a proxy for resolving the request. If a forwarding node caches an unexpired copy of the requested data, it can send a reply to the requester and stop forwarding the data request.

The second approach is zone-based cache resolution. This scheme is the extension of the hop-by-hop resolution scheme. If a forwarding node does not have the data locally but it knows a closer data source (e.g. by proactive data discovery in its cooperation zone), it can also redirect the request to the closer data source, which also reduces the travel distance of data messages and hence minimizes the energy cost and response delay.

COOP uses a cocktail approach based on the basic approaches described above. COOP uses profile-based resolution after the local cache misses. If no matching cache is found or the request fails, COOP uses reactive approach to discover the data in its cooperation zone. If this again fails, COOP forwards the data request to the data server, and hop-by-hop resolution is used to resolve the request along the forwarding path.

### 3.2.2 CACHE MANAGEMENT

For cooperative caching, the emphasis of cache management is how to manage an individual cache not only from the local node's point of view, but also from the view of the overall cooperative caching system. To maximize the capacity of cooperative caches, COOP tries to reduce duplicated caching within the cooperation zone, such that the cache space can be used to accommodate more distinct data items. In this paper the authors categorize cached data copies based on whether they are already available in the cooperation zone or not. A data copy is primary if there is no other primary copy within the zone. Otherwise, the data copy is secondary. To decide caching priorities of primary and secondary data the inter- and intra-category rules are used.

**1. The Inter Category Rule:** The idea of inter-category rule is to put primary items at a priority level, i.e. secondary items are purged to accommodate primary items, but not vice versa. The problem in implementation is how to determine whether a data item is primary or secondary. Here the authors use a simplified approach to address this problem. Once a node fetches a data item, it labels the item as primary copy if the item comes from a node beyond the zone radius. Otherwise, if a data item comes from within the zone radius, then check whether the data provider labels the item as primary or secondary. If the provider already labels its copy as primary, the new copy would be secondary since there will not be duplicated primary copies in the same cooperation zone. On the other hand, if the provider tags its own copy as secondary, the provider needs to attach the information of the primary copy holder. If the primary copy holder is beyond the zone radius, the new copy is primary copy; otherwise, the new copy is a secondary copy.

**2. The Intra Category Rule:** The intra-category rule is used to evaluate the data items within the same category. For this purpose, here the authors simply adopt the LRU (least recently used) algorithm.

### 3.2.3 LIMITAION OF COOP

To improve data availability and access performance, COOP addresses two basic problems of cooperative caching. For cache resolution, COOP uses the cocktail approach which consists of two basic schemes: hop-by-hop resolution and zone-based

resolution. By using this approach, COOP discovers data sources which have less communication cost. For cache management, COOP uses the inter- and intra-category rules to minimize caching duplications between the nodes within a same cooperation zone and this improves the overall capacity of cooperated caches. The disadvantage of the scheme is that flooding incurs high discovery overhead and it does not consider factors such as size and consistency during replacement.

### 3.3 CacheData, CachePath and HybridCache

In [4, 5] Yin and Cao propose three schemes: CachePath, CacheData, and HybridCache. In CacheData, intermediate nodes cache the data to serve future requests instead of fetching data from the data center. In CachePath, mobile nodes cache the data path and use it to redirect future requests to the nearby node which has the data instead of the faraway data center. To further improve the performance, we design a hybrid approach (HybridCache), which can further improve the performance by taking advantage of CacheData and CachePath while avoiding their weaknesses.

#### 3.3.1 CacheData and CachePath

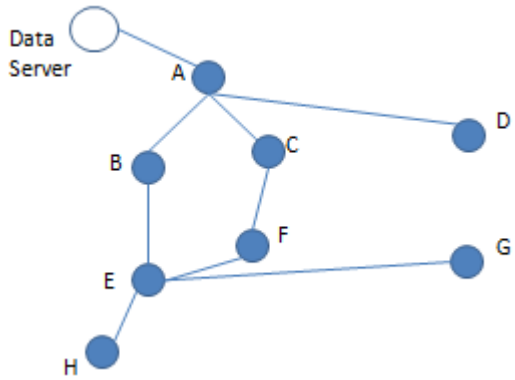


Figure 1: A Mobile Adhoc Network

In CacheData, if a node finds many requests for a particular data item d then data item is cached by the node. For example, in figure 1 both node B and node C request d through node A, node A knows that d is popular and cache it locally. Future request by node D can be served by node A. Suppose the data center receives several requests for d forwarded by node F. Nodes along the path F-C-A may all think that d is a popular item and should be cached. However, it wastes a large amount of cache space if three of them all cache d. To avoid this, authors proposed a conservative rule. That states: A node does not cache the data if all requests for the data are from the same node. As in the previous example, all requests received by node F are from node C, which in turn are from node A. With the new rule, node C and node A do not cache d. If the requests received by node A are from different nodes such as node C and node D, node A will cache the data.

The idea of CachePath can be explained by using figure 1. Suppose node G has requested a data item d from server. When node E forwards the data d back to node G, node E knows that node G has a copy of d. Later, if node H requests d, node E knows that the data server is three hops away whereas node G is only one hop away. Thus, node E forwards the request to node G instead of node B. When saving the path information, a node need not save all the node information along the path. Instead, it can save only the destination node information, as the path from current router to the destination can be found by the underlying routing algorithm.

In CachePath, a node does not need to record the path information of all passing-by data. For example, when d flows from data server to destination node G along the path A-B-E, node A and node B need not cache the path information of d since node A and node B are closer to the data center than the caching node G. Thus, a node only needs to record the data path when it is closer to the caching node than the data center.

#### 3.3.2 Hybrid Cache

In HybridCache, when a mobile node forwards a data item, it caches the data or the path based on some criteria. These criteria include the data item size and the time-to-live (TTL) of the item. For a data item d, the following heuristics are used to decide whether to cache data or path:

1. If size of d is small, CacheData should be adopted because the data item only needs a very small part of the cache; otherwise, CachePath should be adopted to save cache space. The threshold value for data size is denoted as $\square_s$.
2. If TTL of d is small, CachePath is not a good choice because the data item may be invalid soon. Using CachePath may result in chasing the wrong path and end up with resending the query to the data center. Thus, CacheData should be used in this situation. If TTL of d is large, CachePath should be adopted. The threshold value for TTL is a system tuning parameter and denoted as $\square_{TTL}$.

#### 3.3.3 Limitations of CacheData and CachePath

As we seen in CacheData, forwarding nodes check the passing-by data requests. If a data item is found to be frequently requested, forwarding nodes cache the data, so that the next request for the same data can be

answered by forwarding nodes instead of travelling further to the data server. A problem for this approach is that the data could take a lot of caching space in forwarding nodes. To overcome this problem the authors present another cache resolution scheme CachePath. In CachePath forwarding nodes cache the path to the closest caching node instead of the data and redirect future requests along the cached path. This scheme saves caching spaces compared to CacheData, but since the caching node is dynamic, the recorded path could become obsolete and this scheme could introduce extra processing overhead. Trying to avoid the weak points of those two schemes the authors proposed HybridCache. In HybridCache, when a mobile node forwards a data item, it caches the data or the path based on some criteria. These criteria include the data item size and the time-to-live (TTL) of the item. Because due to the mobility of nodes the collected statistics about the popular data may become useless. One another drawback of these schemes is that if the node does not lie on the forwarding path of a request to the data center the caching information of a node cannot be shared.

### 3.4 IXP and DPIP Protocols

Chiu et. el. [1] proposed two cooperative caching schemes IXP and DPIP. Index Push (IXP) is push based in the sense that a mobile node broadcasts an index packet in its zone to advertise a caching event. The Data Pull/Index Push (DPIP) is a pull based one. DPIP is offers an implicit index push property by exploiting in-zone request broadcasts.

#### 3.4.1The IXP Protocol

The idea of IXP is based on having each node share its cache contents with the nodes in its zone. To facilitate exposition, authors call the nodes in the zone of a node M the buddies of M. A node should make its cache contents known to its buddies, and likewise, its buddies should reveal their contents to the node. IXP requires that, whenever a node caches a data item, it broadcasts an index packet to its buddies to advertise the caching event.

Each node maintains an index vector, denoted as IV. An IV has N elements, where N is the number of data items in the data set. Each element of IV corresponds to a different data item and consists of three entries that are used to record caching information of the corresponding item. Consider the IV of a node M.

- The first entry associated with a data item x is of type binary and is represented by IV[x].cached. This entry indicates whether x is cached locally. If the entry is TRUE, it means that x is locally available; otherwise, x has to be acquired from the data center or some other node.
- The second entry, denoted as IV[x].cachednode, is used to record a nearby node that has cached x. For the sake of saving storage space, M only records the last buddy that has broadcasted an index packet associated with x.
- The third entry, represented by IV[x].count, maintains a count of M's buddies that are known to have cached x, after x was last cached by M.

Initially, the IV[x].cached is set to FALSE, IV[x].cachednode is set to NULL, and IV[x].count is set to zero.

Consider that a node M wants to access a data item x. M first checks its IV[x].cached to see if x is cached locally. If the entry is FALSE, M proceeds to examine IV[x].cachednode, expecting someone in the vicinity may offer a cached copy of x. If the entry is NULL, M sends a request packet toward the data center. An intermediate node I on the path to the data center can redirect the request to a buddy node that I knows has cached the item according to its IV[x].cachednode entry. If the entry of M is non-NULL, M issues a request to the node, sayM1, indicated by the entry, ifM1 is still in the zone. In case that M1 no longer stays in M's zone due to mobility, M sends the request toward the data center.

When M eventually receives a copy of x, it caches x. In doing so, it may possibly need to evict another cached item, say y, if its cache is full. M will set its IV[x].cached to TRUE and IV[y].cached to FALSE in this case. Then M notifies its buddies of both caching and the accompanied replacement events. Knowing what has been replaced enhances the accuracy of the caching information. Upon receiving the index packet, M's buddies update their IV[x].cachednode entries by setting them to M, increase IV[x].count by one, and decrease IV[y].count by one. Furthermore, if a buddy has recorded M in its IV[y].cachednode, it has to set the entry to NULL because y is no longer cached by M.

Here the authors propose a count-based scheme, denoted as CV, which employs IV[x].count entry for cache replacement. Basically, CV attempts to replace the items whose removals from the cache induce least impact on satisfying data requests from the buddy nodes.

Consider a mobile node M. Recall that IV[x].count indicates the number of M's buddies that have cached the item x after M last cached x. CV replaces the item that has the maximum IV[x].count among all cached ones. Replacing such an item tends to induce less impact on M's buddies because there will be less buddies relying on M for accessing the item when the count becomes bigger. Moreover, doing so has the

effect of limiting cache duplicates. Notice that once x is chosen by M for replacement, M's buddies will decrement their IV[x].count by one. Consequently, there will be less chance for these buddies to have x replaced. This can ameliorate the problem of concurrently replacing the same item by all the nodes in the same neighborhood.

### 3.4.2 The DPIP Protocol

IXP is essentially push based in the sense that a caching node "advertises" the caching information to the surrounding buddies. Each node has a view of the caching status in its zone only. However, due to node mobility and some limitations of mobile devices such as transient disconnections, the caching status represented by IV may become obsolete or not up-to-date.

For example, suppose that, according to M's IV, none of M's buddies caches x. If a new node that has cached x moves into M's zone, the cache status cannot be captured by M's IV with IXP. In the following, we propose a more sophisticated protocol, called DPIP, to deal with this problem. DPIP is basically a pull-based protocol. However, it also exploits an implicit index push property. We now describe the details of DPIP.

Similar to IXP, each node maintains an IV vector. When a node M wants to access a data item x that is not cached by itself, it first examines the entry IV[x].cachednode to see if some buddy node in its zone has cached x. If such a buddy node exists, M issues a request to the node to ask for a copy of x in the same way as IXP. However, unlike IXP, if IV[x].cachednode entry is NULL, M broadcasts a special data-pull packet, data_pull (dp for short), to its buddies. The dp packet carries the IDs of both x and the data item that will be replaced if the cache space is full.

Upon receiving the dp packet, a buddy node M1 will reply to M if either of the following conditions is met:

1) It has cached x and
2) It knows some of its buddies has cached x (as per its IV).

If the first case is true, M1 returns a copy of x to M. Otherwise, if the second case is true, M1 returns to M a location_reply packet, which contains the node ID recorded in M1's IV[x].cachednode.

In contrast with IXP, DPIP increases the chance for M to obtain a copy of x from the nodes in its neighborhood. This is argued as follows: In addition to the fact that M can acquire x from its buddies if the first condition specified previously is met, it may possibly obtain the cache status of the nodes that are beyond its zone but within the zones of its buddies as specified by the second condition. In addition, the in-zone dp broadcast, which initiates the "data pulling" operation, allows DPIP to use the latest cache contents.

### 3.4.3 LIMITATIONS OF IXP and DPIP PROTOCOLS

In the IXP protocol when a node M enters in a new zone, the nodes of the new zone are not aware about M's update. In their approaches the authors use a cache replacement policy that based on the count vector. According to the policy the data item with higher count vector is replaced. A node with a Count Vector 0 will never be replaced. This may cause the waste of cache memory space.

### 3.5 Some Other Approaches

Moriya et.al. [11] proposed a "self-resolver" paradigm, in which a client user itself queries and measures which node it should access. In this method if a node M requests the data D then it forwards a query packet to its neighbor nodes. If some node has the data D then it returns a REPLY packet to S. Otherwise it recursively sends QUERY packets to its neighboring nodes. The disadvantage of this approach is flooding which introduce high discovery overhead. Furthermore in this paper this issue is not discussed that how the request of M is fulfilled if the requested data is not cached any neighbor node.

Chow et.al. [7, 8] have proposed a cooperative caching protocol, called CoCa, for mobile computing environments. In this protocol, mobile nodes share their cache contents with each other to reduce both the number of server requests and the number of access misses. Further, built upon the CoCa framework, a group-based cooperative caching scheme, called GroCoCa, has been proposed in [9], in which a centralized incremental clustering algorithm is adopted by taking into consideration node mobility and data access pattern. GroCoCa improves system performance at the cost of extra power consumption.

Lim et al in [6], a caching algorithm is suggested to minimize the delay when acquiring data. In order to retrieve the data as quickly as possible, the query is issued and broadcast to the entire network. All nodes that have this data are supposed to send an acknowledgment back to the source of the broadcast. The requesting node will then issue a request for the data (unicast) to the first acknowledging node it hears from. The main advantage of this algorithm is its simplicity and the fact that it does achieve a low response delay. However, the scheme is inefficient in terms of bandwidth usage because of the broadcasts, which, if frequent, will largely decrease the throughput of the system due to flooding the network with request packets [12]. Additionally, large amounts of bandwidth will also be consumed when

data items happen to be cached in many different nodes because the system does not account for controlling redundancy.

## 4. CONCLUSIONS

In this paper we have discussed cache sharing issues related to mobile adhoc network environment and give analysis of some popular cooperative caching schemes. These caching schemes are useful in MANET environment. Here we present how these schemes are advantageous in order to find a data item in a MANET by using less resources (e.g. network bandwidth, energy etc.) and improves the performance(data availability and latency time). We also discussed the limitations of these techniques. As the cooperative caching is a useful technique to improve the data availability in the MANET so these analyses will be helpful for the future research.

## 5. REFERENCES


[1] Ge-Ming Chiu and Cheng-Ru Young, Exploiting In-Zone Broadcasts for Cache Sharing in Mobile Ad Hoc Networks IEEE TRANSACTIONS ON MOBILE COMPUTING, VOL. 8, NO. 3, MARCH 2009.

[2] Y. Du and S. Gupta, COOP – A Cooperative Caching Service in MANETs, *Proceedings of the IEEE ICAS/ICNS* (2005), 58–63.

[3] Yu Du, Sandeep K.S. Gupta and Georgios Varsamopoulos, Improving on-demand data access efficiency in MANETs with cooperative caching, *Ad Hoc Networks*, 7 (3), p.579-598, May 2009.

[4]L. Yin and G. Cao, "Supporting Cooperative Caching in Ad Hoc Networks," Proc. IEEE INFOCOM '04, pp. 2537-2547, 2004.

[5] L. Yin and G. Cao, "Supporting Cooperative Caching in Ad Hoc Networks," IEEE Trans. Mobile Computing, vol. 5, no. 1, pp. 77-89, Jan. 2006.

[6] S. Lim, W. Lee, G. Cao, and C. Das, "A Novel Caching Scheme for Internet Based Mobile Ad Hoc Networks Performance," Ad Hoc Networks, vol. 4, no. 2, pp. 225-239, 2006.

[7] C.-Y. Chow, H.V. Leong, and A. Chan, "Peer-to-Peer Cooperative Caching in Mobile Environments," Proc. 24th Int'l Conf. Distributed Computing Systems Workshops (ICDCSW '04), pp. 528-533, 2004.

[8] C.-Y. Chow, H.V. Leong, and A. Chan, "Cache Signatures for Peer-to-Peer Cooperative Caching in Mobile Environments," Proc. 18th Int'l Conf. Advanced Information Networking and Applications (AINA '04), pp. 96-101, 2004.

[9] C.-Y. Chow, H.V. Leong, and A.T.S. Chan, "Group-Based Cooperative Cache Management for Mobile Clients in Mobile Environments," Proc. 33rd Int'l Conf. Parallel Processing (ICPP '04), pp. 83-90, 2004.

[10] Y. Sun et al. Internet connectivity for ad hoc mobile networks. *International Journal of Wireless Information Networks*, 9(2), April 2002.

[11] T. Moriya and H. Aida, "Cache Data Access System in Ad Hoc Networks," Proc. Vehicular Technology Conf. (VTC '03), vol. 2, pp. 1228-1232, Apr. 2003.

[12] P. Gupta and P. Kumar, "The Capacity of Wireless Networks," IEEE Trans. Information Theory, vol. 46, no. 2, pp. 388-404, 2000

[13] G. Cao, L. Yin, and C.R. Das, "Cooperative Cache-Based Data Access in Ad Hoc Networks," Computer, vol. 37, no. 2, pp. 32-39, Feb. 2004.

[14] Y. Du, S. Gupta, Handbook of Mobile Computing, CRC Press, 2004. Chapter 15, pp. 337–360.