IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 3, No 8, May 2010
ISSN (Online): 1694-0784
ISSN (Print): 1694-0814

Proactive Approach for Cooperative Caching in Mobile Adhoc Networks

Prashant Kumar[1], Naveen Chauhan[2], LK Awasthi[3], Narottam Chand[4]

Department of Computer Science, National Institute of Technology
Hamirpur, INDIA

Abstract

In Mobile Adhoc Networks (MANETs), due to frequent network partition, data availability is lower than that in traditional wired networks. Cooperative caching provides an attractive solution for this problem. In this paper we propose a new proactive approach for cooperative caching in MANETs, in which we will cache the data of leaving node. Here each mobile node will broadcast a "LEAVE" message when it moves out from its zone. Based upon its Caching Information Table (CIT) zone manager will decide which data is to be cached. This will help to improve the data availability and overall performance of the network.

Keywords: *Mobile adhoc networks, cooperative caching, proactive approach.*

1. INTRODUCTION

In recent years there has been a rapid growth in mobile communication. Mobile Adhoc Networks (MANETs) are very popular solution in the situation where network infrastructure is not available. MANETs can be extended by connecting with some other wired or wireless networks like Internet [10]. In adhoc networks, mobile nodes communicate with each other using multihop wireless links. As there is no infrastructure support, mobile nodes cooperate with each other to forward data. Each node acts as a router, forwarding data packets for other nodes and mobile nodes have peer to peer connection among themselves. 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 adhoc networks is to provide mobile nodes with access to information. 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 adhoc networks, due to frequent network partition, data availability is lower than that in traditional wired networks. Cooperative caching provides an attractive solution for this problem. Cooperative caching is a technique that allows the sharing and coordination among the mobile nodes. However, the movement of nodes, limited storage space and frequent disconnections limit the availability. By the caching of frequently accessed data in adhoc networks we can improve the data access, performance and availability. Due to mobility and resource constraints of adhoc networks, caching techniques designed for wired network may not be applicable to ad hoc networks.

In general, a good cooperative cache management technique for MANETs should address these issues:

1. A cache discovery algorithm that is efficient to discover and deliver requested data items from the neighbors node and able to decide which data items can be cached for future use. In cooperative caching this decision is taken not only on the behalf of the caching node but also based on the other nodes need.
2. There should be a cache replacement algorithm to replace the cached data items when the caching space is not enough to cache the new ones.
3. A cache consistency algorithm to ensure that the cached data items are updated.

In this paper we considered all these issues and proposed a new cooperative cache algorithm. The ultimate goal of this approach is to improve the data availability that means data is available in minimum time and by utilizing fewer resources. In this algorithm we consider that each node is associated with a zone and each zone has a zone manager. Each node will maintain a Caching Information Table (CIT). When a node caches a new data item or updates its CIT it will broadcasts these updates to all its neighbors and to the zone manager. Further when a node wants to move out from its zone then it will broadcasts a "LEAVE" message to its neighbors. Now neighbors will decide whether the data of leaving node can be cached for future use. These decisions will be based on the information stored in the CIT of the neighbor's node. For cache replacement we will use an access count policy with the TTL (Time-to-Live) value. The cache consistency algorithm will be based upon the TTL value.

The rest of this paper is organized as follows: We review the related work in section 2. Section 3

ISSN (Online): 1694-0784
ISSN (Print): 1694-0814

describes about the system model. In section 4 we describe our proposed algorithm. Cache replacement related issues will be discussed in section 5. Section 6 will concludes the paper.

2. RELATED WORKS

Caching is an important technique to enhance the performance of both wired and wireless network. A lot of researches have been done to improve the caching performance in mobile adhoc network environment. The two basic types of cache sharing techniques are push based and pull based. In the push based cache sharing, when a node caches a new data item, it will proactively broadcast the caching updates to its neighbors nodes. The neighboring nodes update the caching information for the future use. Push based scheme improves the data availability at the cost of communication overhead. The disadvantage of the scheme is that an advertisement may become useless if no demand for the cached data items occurs in the vicinity. One more problem with the push based scheme is that caching information may not be longer used if the node moves out from the zone or due to the cache replacement. This drawback may be overcome with the pull based approach. In the pull based cache scheme when a node wants to access a data item, it broadcast a request packet to all its neighbors node. A nearby node who has cached the data item will send data item to the requester. There are two drawbacks associated with this scheme. First if the requested data item is not cached by any node in the neighborhood then the request originator will wait for the time out interval to expire before it resend the request to the data center. This will cause extra access latency. Secondly if more than one node have cached the requested data item then multiple copies will returns to the request originator and this will cause extra communication overhead.

Moriya et al. [12] 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 that it uses 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 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. A caching algorithm is suggested by Lim et al. in [6], 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.

Yin and Cao [4, 5] presented three cache resolution schemes: *CacheData*, *CachePath*, and *HybridCache*. 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. 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.

ISSN (Online): 1694-0784
ISSN (Print): 1694-0814

Du et al. [2, 3] proposed a cooperative caching scheme called *COOP* for MANETs. 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. Disadvantage of the scheme is flooding that introduce extra discovery overhead.

Chiu et al. [1] proposed to protocol *IXP* and *DPIP*. 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. 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* offers an implicit index push property by exploiting in-zone request broadcasts. The disadvantage of the *IXP* protocol is that when a node *M* enters in a new zone, the nodes of the new zone are not aware about *M*'s update. Further 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 data item with a Count Vector 0 will never be replaced. This may cause the waste of cache memory space.

3. SYSTEM MODEL

Let us consider a mobile adhoc network shown in fig. 1. This network has no fixed infrastructure and node are free to move anywhere in the network. Since nodes are mobile so the topology is dynamic and temporary. There exists a data server that contains the database of n items $D_1, D_2,..., Dn$. This data server may be connected to some external wired or wireless network like Internet. When a node requires some data item it sends request to data server. When a node receives a data item it cache the data item locally for future use. A zone is associated with each mobile node and refers to the set of nodes that can be reached by the node within the given number of hops, called the radius of the zone [1]. When a node wants to moves out from the zone it will broadcast a "*LEAVE*" message before leaving the zone.

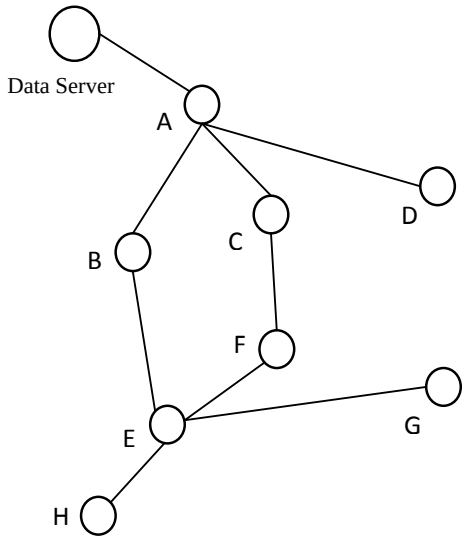

Fig. 1 System Model

Each node in the zone will maintain a Caching Information Table denoted as CIT. This CIT will contain *n* elements where *n* is the number of the data items. There will be four entries related to each element.

Let consider the entries related of node *A* for the data item *d*. The first entry is *d.available* that shows whether *d* is locally cached at node *A*. This is binary type value and is *TRUE* if data is locally available. The second entry is *d.nnode* and shows which neighbor node has cached *d*. The third entry maintained to access count that show how many times *d* is cached by neighbors node of node *A* after *d* is cached by node *A*. This entry is denoted by *d.accesscount*. The final entry is *d.TTL* shows the *TTL* (time-to-live) value that after how much time *d* is expired. This value is assigned by the data server. Initially *d.available* is set to *FALSE*, *d.nnode* is set to *NULL* and *d.accesscount* is set to zero. For example node *A* will contain some data (in its local cache) maintained a CIT that contains the details about the data which is maintained by its neighbors. The entries of CIT are summarized in table 1.

Table 1: Entries of CIT

S No.	Entry Name	Meaning	Initial Value

ISSN (Online): 1694-0784
ISSN (Print): 1694-0814

1	d.available	shows whether *d* is locally cached at node	FALSE
2	d.nnode	shows which neighbor node has cached *d*	null
3	d.accesscount	shows how many times *d* is cached by neighbors node of node *A* after *d* is cached by node *A*	zero
4	d.TTL	shows after how much time *d* is expired.	assigned by the data server

Here we assume that each mobile node has limited cache space and only some data items can be cached. When the cache space of a node is full then the node will select some data items to remove from the cache, when it has to cache the new one. Before forwarding the request of a data item from a node, each intermediate node will check it local cache if it has the data then it will send directly and stop the forwarding otherwise it redirect the to some neighbor that it knows has cached the data item.

4. PROPOSED ALGORITHM

The idea of our proposed algorithm is based upon the fact that each node in the network is willing to share its cache contents with its neighbors. When a node updates its local cache it broadcasts these updates to all neighbors node in the zone. Each node in the zone will maintain a Caching Information Table (CIT).

4.1 Cache Discovery

When a data item *d* is requested by a node *A*, first the node will check whether *d.available* is *TRUE* or *FALSE* to see the data is locally available or not. If this is *FALSE* then the node will check *d.nnode* to see whether the data item is cached by a node in its neighbor. If the matching entry found then the request is redirect to the node otherwise the request is forwarded towards the data server. However the nodes that are lying on the way to the data center checks their own local cache and *d.nnode* entry in their CIT. If any node has data in its local cache then the data is send to requester node and request forwarding is stop and if the data entry is matched in the CIT then the node redirect the request to the node. Here we checks who is close: data center or the node and closer one is selected for the data item. The complete process of cache discovery is shown in fig. 2.

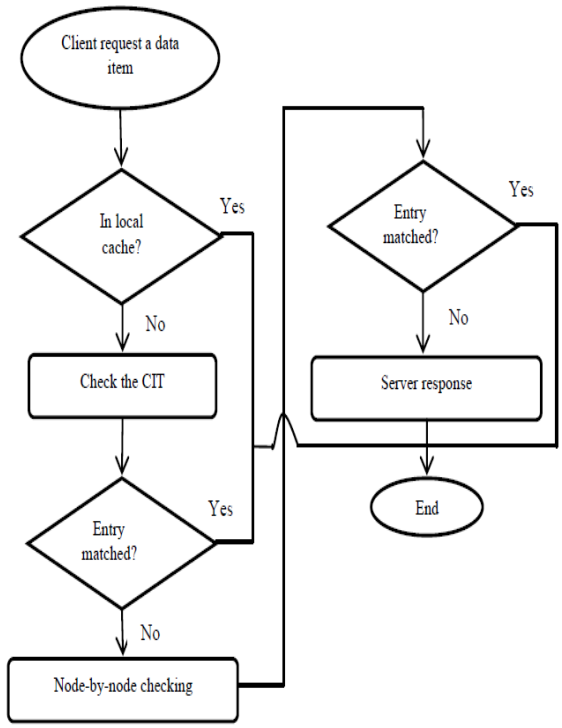

Fig. 2 Cache Discovery Process

Now we consider the movement of the nodes. There are two situations one is that a node enters in a new zone and second is a node leaves its zone. We discuss both the cases separately.

1. When a node enters in a new zone: When a node will enter in a new zone then it will proactively broadcast its updates to all its neighbors of the new zone. In other words we can say that its sends the information of its local cache. This will improve the data availability in the zone.

2. When a node leaves a zone: When a node wants to leave its zone then the data cached by this node will not be available for future use. So before leaving its zone it will broadcast a "*LEAVE*" message to its neighbors. Then the neighbors will decide whether the data held by the leaving node can be cached by them. These decisions will be based on following criteria:-

1. First the nodes will check their local cache if the same data item is found then this data will not cached by these nodes.

2. After the local cache the nodes will check their CIT to see whether the same data is cached by some node in their vicinity. If the entries matched then also this data will not be cached by these

ISSN (Online): 1694-0784
ISSN (Print): 1694-0814

nodes because this data is already available in the zone.

3. Now if the data is not present in the local cache or in the vicinity then this data will be cached by the node. If the free space is not available in the node then by using cache replacement algorithm some data is removed from the cache to save the new one.

Let us consider that node *B* wants to leave its zone then before leaving it will broadcasts a "*LEAVE*" message to its neighbors i.e. to node *A* and node *E*. Now these neighbor nodes will decide whether the data held by node *B* can be cached by them for future use. Then first these nodes will check their local cache and if the same data is found then they will not cache the data. If the data is not in their local cache then they will check *d.nnode* entry in their CIT to see whether the data is present or not in their vicinity. If the then neither of these node will cache the data as the data is available in the zone. Now if the data is not found in their local cache or in the vicinity then either of the nodes will cache the data items. This will be depending on the fact which node has enough space to cache the data. If either node has not enough space to cache the data then we will choose a node which has to remove less content from its cache in order to make the space free. Both nodes will cache the different data.

```
//A is the node requesting a data item d
//P is an intermediate node on the forwarding path of the data request packet

(A) When a node P receives a data request packet for item d
if (d.available==TRUE)
  send d to A;
elseif (d.nnode!=NULL)
  redirect the request to d.nnode;
else
  forward the request to the data center;

(B) When node A cache a new data item d
 dc=d                // dc refers to data item cached
if (the cache space is full)
  {
    select a victim item c whose c.accesscount is maximum for the cache replacement;
    c.available =FALSE;
    dr= c;          // dr refers to data item removed
  }
else
  dr = NULL;
cache d;
d.available = TRUE;
d.accesscount= 0;
Broadcast the cache update packet <A, dc, dr> to neighbors node;

(C) When a neighbors node receives the cache update packet <A, dc, dr>
Dc.nnode=A;
if (dc.available== TRUE)
  dc.accesscount++;
if (dr!=NULL)
  {
    if (dc.available==TRUE && dr.accesscount>0)
   Dr.accesscount--;
    if (dr.nnode==A)
   dr.nnode= NULL;
  }
```

Fig. 3 Pseudocode of Algorithm

4.2 Data Caching and Data Consistency

When a node *A* receives the data item *d*, it caches it. During this process it may happen the node need

ISSN (Online): 1694-0784
ISSN (Print): 1694-0814

to remove another data item from its cache. Now we assume that the victim item is *c*. After caching *d* node *A* will set *d.available* is TRUE and *c.available* is FALSE. Now A will reset the *d.accesscount* to 0. Now node will broadcast its updates to its neighbors. These updates will include information about both (cached item *d* and removed item *c*) data items. By doing so the neighbors will make changes in their CIT, this will improve the information accuracy.

Now when *A*'s neighbors receives the update from *A*, they will set their *d.available* to *A*. Further if these nodes have already cached these data items (*d* and *c*) then they will increase *d.accesscount* by 1 and decrease the *c.accesscount* by 1. And also if a node has set *c.nnode* to *A* then they will set it to NULL because the data item *c* is no longer with *A*. Cache consistency algorithm ensures that the cached data items are updated and clients get fresh copy of data items. There are two widely used cache consistency models one is the weak consistency and the second is strong consistency model. In the weak consistency model, a stale data might be returned to the client. In the strong consistency model, after an update completes, no stale copy of the modified data will be returned to the client [11]. In MANETs due to bandwidth and power constraints it is too expensive to maintain strong consistency, and the weak consistency model is done well. Here in our proposed algorithm we use a simple weak consistency model based on the time-to-live (*TTL*) value. For this we maintain an entry (*d.TTL*) in the CIT. The data is considered as a fresh copy until its *TTL* value is not expired. When *TTL* is expired the node removes the cached data and broadcast updates to its neighbors so that they can updates their CIT.

5. CACHE REPLACEMENT

As in cooperative caching in MANETs the data is not stored only on behalf of caching node but interest of the neighbor's node is also considered. For this reason LRU, LFU [16] and SXO [5] are not suitable for the MANETs. Here our emphasis to remove such data items whose removals introduce the least effect on the requirement of the neighbor's node and also on the data availability in the zone. Here we use a cache replacement policy based on the access count.

As we have maintains an entry *[d].accesscount* in the CIT. in our cache replacement policy we replace a data item which has maximum *[d].accesscount* among all the caching items. We use this replacement policy because by removing a data item with highest access count has least effect on the data availability as other neighbors node has cached the same data item. Because as many times a data item will access by a neighbor's node its *[d].accesscount* will be increased by one. By removing such an item we have the least cache duplicacy in the zone. This is the reason that's why we initially set the *[d].accesscount* to zero. This policy is better than the LRU, LFU and SXO because this is based upon the recent behavior of the nodes.

Now what happens if some data item is not accessed by any of the neighbor node in the zone. Then that item will not be replaced and this will cause the waste of storage. To overcome this problem we will check the *TTL* values of the data items. When the *TTL* value is expired then the data item will be removed from the cache.

6. CONCLUSIONS

In this paper we discuss the cooperative cache algorithm based on proactive approach. This algorithm is unique because this caches the data of leaving node. In order to cache discovery in this algorithm first the node will check its local cache if cache miss occurs then it will check its CIT to see whether the data is available in the neighborhood. If the matching entries are found then the data is returned to the requester otherwise the request is forward to data server. However nodes those are lying on the way to the data server check their own local cache and there CIT. If any node has data in its local cache then the data is send to requester node and request forwarding is stop and if the data entry is matched in the CIT then the node redirect the request to the node. For the cache replacement we use a policy based on the access count in such a way that removing data leave least impact on data availability in the zone. Cache consistency is maintained by using the approach based upon the *TTL* value.

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.

ISSN (Online): 1694-0784
ISSN (Print): 1694-0814

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. N. Chand, R.C. Joshi, and M. Misra, “Cooperative Caching in Mobile Ad Hoc Networks Based on Data Utility,” Mobile Information System, vol. 3, no. 1, pp. 19-37, 2007.
12. 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.
13. P. Gupta and P. Kumar, “The Capacity of Wireless Networks,” IEEE Trans. Information Theory, vol. 46, no. 2, pp. 388-404, 2000
14. 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.
15. Y. Du, S. Gupta, Handbook of Mobile Computing, CRC Press, 2004. Chapter 15, pp. 337–360.
16. A. Silberschatz, P.B. Galvin, and G. Gagne, Operating System Concepts. John Wiley and Sons, 2004.