An Asymmetric Approach for Cooperative Caching in Multi Hop Wireless Ad hoc Networks

Dr B. Vijaya Babu,PhD
Professor, CSE Department
K L University
Guntur, India
vijay_gemini@kluniversity.in

M. Sudheer Raja
M. Tech Student, CSE Department
K L University
Guntur, India
sudheerraja@kluniversity.in

***Abstract*— Most previous research in ad hoc networks focused on the developing of dynamic routing protocols that can effectively find routes between two communicating nodes. Although routing is important issue, the ultimate goal of ad hoc network to provide mobile nodes with access to information. Some recent studies have shown that cooperative cache can improve the system performance in wireless P2P networks such as ad hoc networks and mesh networks. Cooperative caching which allows the sharing and coordination of cached data among multiple nodes has been applied to improve the system performance in wireless P2P networks. We propose a novel asymmetric cooperative cache approach, where the data requests are transmitted to the cache layer on every node, but the data replies are only transmitted to the cache layer at the intermediate nodes that need to cache the data. This solution not only reduces the overhead of copying data between the user space and the kernel space, it also allows data pipelines to reduce the end-to-end delay.**

***Keywords*- ad hoc network, cooperative caching.**

1. INTRODUCTION

First, there comes the need to communicate while on the move, or away from the fixed infrastructure. Ad hoc network is a wireless network where nodes are movable and exchange the information in peer-to-peer fashion. This type of network was borne with goal to setup communication for specialized, customized applications in area where there no pre-existing infrastructure (e.g. Jungle explorations, battlefield) or where the infrastructure has failed (e.g. earth quakes). Cooperative caching in wireless networks, where the nodes may be mobile and exchange information in a peer-to-peer manner.

For example, in a battlefield, a wireless P2P network may consist of several commanding officers and a group of soldiers. Each officer has a relatively powerful data center, and the soldiers need to access the data centers to get various data such as the detailed geographic information, enemy information, and new commands. The neighbouring soldiers tend to have similar missions and thus share common interests. If one soldier has accessed a data item from the data center, it is quite possible that nearby soldiers access the same data some time later. It will save a large amount of battery power, bandwidth, and time if later accesses to the same data are served by the nearby soldier who has the data instead of the far away data center. As another example, people in the same residential area may access the Internet through a wireless P2P network. After one node downloads a MP3 audio or video file, other people can get the file from this node instead of the far away Web server.

Through these examples, we can see that if nodes are able to collaborate with each other, bandwidth and power can be saved, and delay can be reduced. Actually, cooperative caching which allows the sharing and coordination of cached data among multiple nodes has been applied to improve the system performance in wireless P2P networks.

In this paper, we present our design and implementation of cooperative cache in wireless P2P networks. Through real Implementations, we identify important design issues and propose an asymmetric approach to reduce the overhead of copying data between the user space and the kernel space, and hence to reduce the data processing delay.

Cooperative Caching

Cooperative caching [1], [5], [6], which allows the sharing and coordination of cached data among multiple nodes, has been widely used to improve the Web performance. Although cooperative caching and proxy techniques have been extensively studied in wired networks, little has been done to apply this technique to ad hoc networks. Due to mobility and resource constraints, techniques designed for wired networks may not be applicable to ad hoc networks.

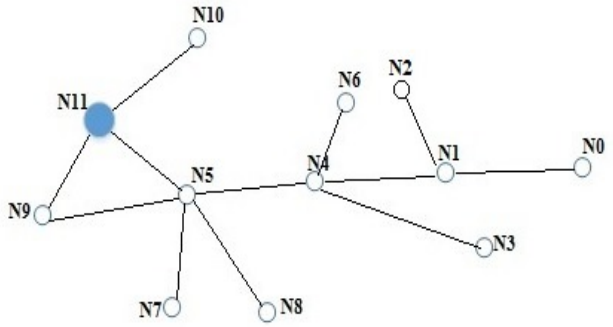


Fig.1. an ad hoc network

Fig. 1 illustrates the Cache Path concept. Suppose node N_0 requests a data item d_i from N_{11}. When N_1 forwards d_i to N_0; N_1 knows that N_0 has a copy of the data. Later, if N_2 requests d_i; N_2 knows that the data source N_{11}. Is three hops away whereas N_0 is only one hop away. Thus, N_1 forwards the request to N_0 instead of N_4. Many routing algorithms (such as AODV [7] and DSR [8]) provide the hop count information between the source and destination. Caching the data path for each data item reduces bandwidth and power consumption because nodes can obtain the data using fewer hops. However, mapping data items and caching nodes increase routing overhead, and the following techniques are used to improve CachePath's performance.

In Cache Path, a node need not record the path information of all passing data. Rather, it only records the data path when it is closer to the caching node than the data source. For example, when N_{11} forwards d_i to the destination node N_0 along the path N_5 - N_4 - N_1; N_4 and N_5 won't cache di path information because they are closer to the data source than the caching node N_0. In general, a node caches the data path only when the caching node is very close. The closeness can be defined as a function of the node's distance to the data source, its distance to the caching node, route stability, and the data update rate. Intuitively, if the network is relatively stable, the data update rate is low, and its distance to the caching node is much shorter than its distance to the data source, then the routing node should cache the data path. In CacheData, the intermediate node caches the data instead of the path when it finds that the data item is frequently accessed. For example, in Fig. 1, if both N_7 and N_8 request d_i through N_5; N_5 may think that di is popular and cache it locally. N_5 Can then serve N_4's future requests directly. Because the CacheData approach needs extra space to save the data, it should be used prudently. Suppose N_1 forwards several requests for di to N0. The nodes along the path N_1, N_4, and N_5 may want to cache di as a frequently accessed item. However, they will waste a large amount of cache space if they all cache d_i. To avoid this, CacheData enforces another rule: A node does not cache the data if all requests for the data are from the same node. In this example, all the requests N_5 received are from N4, and these requests in turn come from N_1. With the new rule, N4 and N_5 won't cache di. If N_1 receives requests from different nodes, for example, N_0 and N_2, it caches the data. Certainly, if N_5 later receives requests for di from N_7 and N_8, it can also cache the data. CachePath and CacheData can significantly improve system performance. Analytical results [2] show that CachePath performs better when the cache is small or the data update rate is low, while CacheData performs better in other situations. To further improve performance, we can use HybridCache, a hybrid scheme that exploits the strengths of CacheData and CachePath while avoiding their weaknesses. Specifically, when a node forwards a data item, it caches the data or path based on several criteria discussed in [2].

We evaluate the performance of caching frame work in different mobile network scenarios, where nodes communicate through ad hoc connectivity. The results show that the solution ensures a high query resolution ratio while maintaining the traffic load very low, even for scarcely popular content, and consistently along different network connectivity and mobility scenarios.

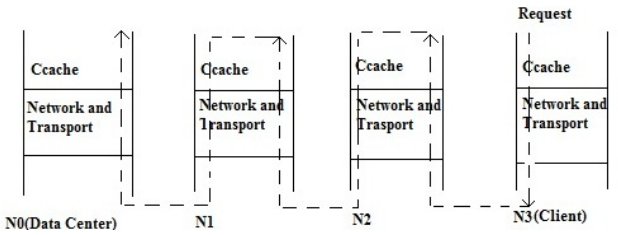
2. DESIGN AND IMPLENTATION OF COOPERATIVE CACHING

In this paper, we focus on design and implementation of the CacheData scheme discussed in the above section. To realize the benefit of cooperative cache, intermediate nodes along the routing path need to check every passing-by packet to see if the cached data match the data request. This certainly cannot be satisfied by the existing ad hoc routing protocols. Next, we look at two design options.

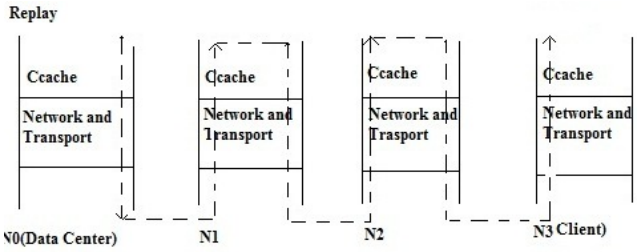
2.1. Layered Design

There are two options for the layered design. One naïve solution uses cross-layer information, where the application passes data request (search key) to the routing layer, which can be used to match the local cached data. However, this solution not only violates the layered design, but also adds significant complexity to the routing protocol which now needs to maintain a local cache table. Further, if an intermediate node needs to cache the data based on the cooperative cache protocol, it has to deal with fragmentation issues since some fragments of the data may not go through this node. Thus, this naive solution does not work in practice. Another solution is to strictly follow the layered approach, where the cooperative cache layer is on top of the network layer (TCP/IP). Fig. 2 shows the message flow (dashed line) in the layered design. In the figure, N_3 sends a request to N_0. Based on the routing protocol, N_3 knows that the next hop is N_2 and sends the request to N_2 encapsulating the original request message. After N_2 receives the request, it passes the message to the cache layer, which can check if the request can be served locally. This process continues until the request is served or reaches N_0. After N_0 receives the request, it forwards the data item back to N_3 hop by hop, which is the reverse of the data request, as shown in Fig. 2b. Note that the data has to go up to the cache layer in case some intermediate nodes need to cache the data. Although this solution can solve the problems of the naïve solution, it has significant overhead.

For example, to avoid caching corrupted data, reliable protocols such as TCP are needed. However, this will significantly increase the overhead, since the data packets have to move to the TCP layer at each hop. Note that the data packets only need to go to the routing layer if cooperative cache is not used. Further, this solution has a very high context switching overhead. At each intermediate node, the packets have to be copied from the kernel to the user space for cache operations, and then reinjected back to kernel to be routed to the next hop.



(a) The request packet flow



(b) The replay packet flow

Fig.2. Layered design

In cooperative cache, the caching granularity is at the data item level. Although a large data item is still fragmented by the transport layer, there is no pipeline due to the layered design. This is because the cache layer is on top of the transport layer, which will reassemble the fragmented packets. Since all packets have to go up to the cache layer hop by hop, the network runs like "stop and wait" instead of "sliding window." This will significantly increase the end -to- end delay, especially for data with large size.

2.2. The Asymmetric Cooperative Cache Approach

To address the problem of the layered design, we propose an asymmetric approach. We first give the basic idea and then present the details of the scheme.

In our solution, data requests and data replies are treated differently. The request message still follows the path shown in Fig. 2(a) however, the reply message follows a different path. If no intermediate node needs to cache the data, N_0 sends the data directly to N_3 without going up to the cache layer. Suppose N_2 needs to cache the data based on the cooperative cache protocol, as shown in Fig. 3. After N_1 receives the request message, it modifies the message and notifies N_0 that the data should be sent to N_1. As a result, the data are sent from N_0 to N_1 through the cache layer, and then sent to N_3. Note that the data will not go to the cache layer in intermediate nodes such as N_2 in this example. In this way, the data only reach the routing layer for most intermediate nodes, and go up to the cache layer only when the intermediate node needs to cache the data. Although the request message always needs to go up to the cache layer, it has a small size, and the added overhead is limited.

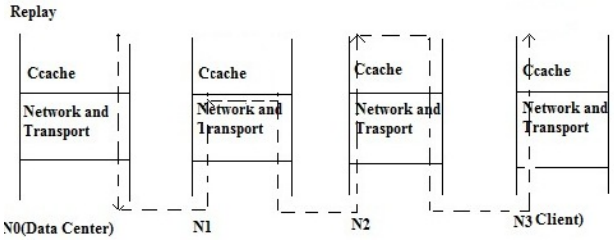


Fig.3. In the asymmetric approach, the data reply only goes up to the cache layer at the intermediate nodes that need to cache the data.

If the requested data item is large, this asymmetric approach allows data pipeline between two caching nodes, and hence reduces the end-to-end delay. The cache layer processing overhead, especially data copying between kernel and user spaces, is also minimized because the data item is not delivered to the cache layer at the nodes that are unlikely to cache the data. Next, we discuss the details of our asymmetric approach.

2.2.1. The Asymmetric Approach

Our asymmetric caching approach has three phases:

Phase 1: Forwarding the request message

. After a request message is generated by the application, it is passed down to the cache layer. To send the request message to the next hop, the cache layer wraps the original request message with a new destination address, which is the next hop to reach the data server (real destination). Here, we assume that the cache layer can access the routing table and find out the next hop to reach the data center. This can be easily accomplished if the routing protocol is based on DSR or AODV. In this way, the packet is received and processed hop by hop by all nodes on the path from the requester to the data server. For example, in Fig. 2a, when N_3 requests d_i from N_0, it adds a new header where the destination of the data request becomes N_2, although the real destination should be N_0. After N_2 receives and processes the packet, it changes the destination to be N_1, and so on, until the request packet arrives at N_0. When an intermediate node receives the request message and delivers to the cache layer, the cache manager performs two tasks: First, it checks if it has the requested data in its local cache; if not, it adds its local information to the request packet. The local information includes the access frequency (number of access requests per time unit) of the requested data, channel used, and throughput of the link where the request is received. Its node *id* will also be added to *path_list*, which is a linked list encapsulated in the cache layer header. When the request message reaches the node who has the data, Path List in the message will include all the intermediate nodes along the forwarding path.

Phase 2: Determining the caching nodes.

When a request message reaches the data the cache manager decides the caching nodes on the forwarding path. Then, the ids of these caching nodes are added to a list called Cache List, which is encapsulated in the cache layer header.

Phase 3: Forwarding the data reply.

Unlike the data request, the data reply only needs to be processed by those nodes that need to cache the data. To deliver the data only to those that will cache the data, tunneling techniques [8] are used. The data reply is encapsulated by the cache manager and tunneled only to those nodes appearing in Cache List. As shown in Fig. 3, suppose the intermediate node N_1needs to cache data di. Then, N_1 and N_2 are the nodes to process the data at the cache layer. N_0 Includes N_1 and N_2 in the cache header of the data item di, and first sets the destination address of d_i to be N_1. When N_1 receives any fragmented packet of di, the routing layer of N_1 will deliver the packet upward to the transport layer and then to the cache layer. After the whole data item di has been received by N_1, it caches the data item, sets the next destination using the next entry in Cache List, which is N_2, and then passes the data down to the routing layer. After N_2 receives the data, it delivers the data to the application layer.

2.3. System Implementation

2.3.1 Architecture Overview

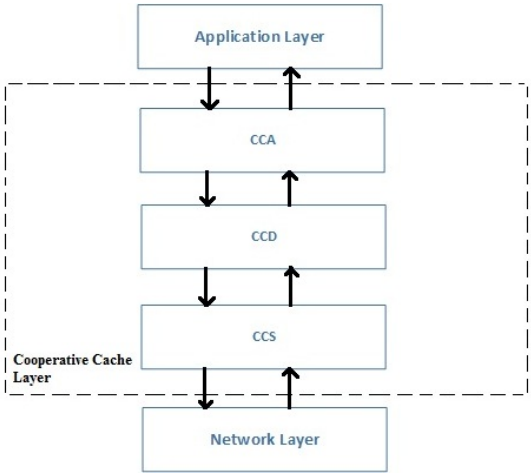


Fig.4. System architecture

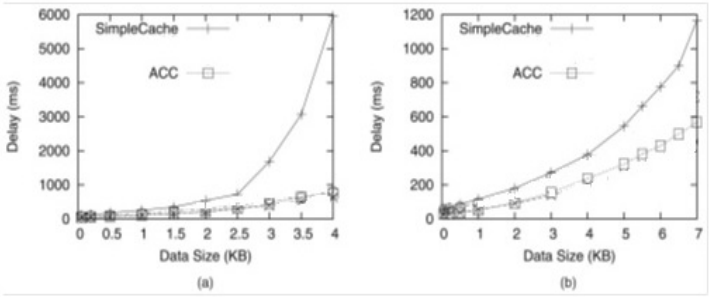
Fig 4 shows the architecture of our cooperative cache middleware, which consists of three parts: Cooperative Cache Supporting Library (CCS), Cooperative Cache Daemon (CCD), and Cooperative Cache Agent (CCA). CCS is the core component to provide primitive operations of the cooperative cache, e.g., checking passing by packets, recording data access history, and cache read/ write/replacement primitives. A data cache buffer is maintained at every node to store the cached data items. There is an interface between CCS and the routing daemon, from which CCS obtains the routing distance to a certain node. All these primitive cache operations are enclosed as CCS API to provide a uniform interface to the upper layer. CCD is the component that implements different cooperative cache mechanisms, namely, CacheData, CachePath, and HybirdCache. There is one cache daemon for each cooperative cache scheme. It extends the basic CCS API to accomplish the characteristic of each scheme. CCA is the module that maps application protocol messages to corresponding cooperative cache layer messages. It is a connector between CCD and user applications. There is one CCA for each user application.

3. EXPERIMENTAL SETUP

In this section, we compare the performance of the Simple Cache approach, the Symmetric Cooperative Cache (SCC) approach, and the Asymmetric Cooperative Cache (ACC) approach in various network environments. Simple Cache is the traditional cache scheme that only caches the received data at the query node. We also compare these schemes to an Ideal Cooperative Cache (ICC) approach, which does not have processing delay at the cache layer. Further, upon receiving each packet, the cache manager makes a copy of the packet and buffers it, and then forwards the original one immediately. Thus, an intermediate node can immediately forward the packet without waiting until the whole data item is received, which can maximize the pipeline effect. It is easy to see that ICC sets up a performance upper bound that a cooperative cache scheme may achieve.

TABLE 1: SIMULATION PARAMETERS

Parameters	Value
MAC Layer	IEEE 802.11
Number of Nodes	100,200
Node Mobility Speed	0 – 50 m/sec
Simulation Area	600*600
Simulation Duration	200 sec
Traffic Flow	TCP,CBR
Mobility Pattern	Random wave point
Packet Size	512
Transmission Range	100m



4. CONCLUSION

In this paper, we presented our design and implementation of cooperative cache in wireless P2P networks, and proposed solutions to find the best place to cache the data. In our asymmetric approach, data request packets are transmitted to the cache layer on every node; however, the data reply packets are only transmitted to the cache layer on the intermediate nodes which need to cache the data. This solution not only reduces the overhead of copying data between the user space and the kernel space, but also allows data pipeline to reduce the end-to-end delay. We have developed a prototype to demonstrate the advantage of the asymmetric approach. Since our prototype is at a small scale, we evaluate our design for a

large scale network through simulations. Our simulation results show that the asymmetric approach outperforms the symmetric approach in traditional 802.11-based ad hoc networks by removing most of the processing overhead.

5. REFERENCE

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

[2]. 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.

[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]. M. K. Denko and J. Tian, "Cross-layer design for cooperative caching in mobile ad hoc networks," in Proc. IEEE CCNC, Las Vegas, NV, Jan. 2008, pp. 375–380.

[5]. B. Tang, H. Gupta, and S. Das, "Benefit-Based Data Caching in Ad Hoc Networks," IEEEE Trans. Mobile Computing, vol. 7, no. 3, pp. 289-304, Mar. 2008.

[6]. J. Zhao, P. Zhang, and G. Cao, "On Cooperative Caching in Wireless P2P Networks," Proc. IEEE Int'l Conf. Distributed Computing Systems (ICDCS), pp. 731-739, June 2008.

[7]. 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.

[8]. A. Raniwala and T. Chiueh, "Architecture and Algorithms for an IEEE 802.11-Based Multi-Channel Wireless Mesh Network," Proc. IEEE INFOCOM, 2005.

.