Data Caching Placement based on information density in wireless ad hoc network

C.Srinivas, Samreen Khan

Abstract

Data caching strategy for ad hoc networks whose nodes exchange information items in a peer-to-peer fashion. Data caching is a fully distributed scheme where each node, upon receiving requested information, determines the cache drop time of the information or which content to replace to make room for the newly arrived information. These decisions are made depending on the perceived "presence" of the content in the nodes proximity, whose estimation does not cause any additional overhead to the information sharing system.

We devise a strategy where nodes, independent of each other, decide whether to cache some content and for how long. In the case of small-sized caches, we aim to design a content replacement strategy that allows nodes to successfully store newly received information while maintaining the good performance of the content distribution system. Under both conditions, each node takes decisions according to its perception of what nearby users may store in their caches and with the aim of differentiating its own cache content from the other nodes'.

The result is the creation of content diversity within the nodes neighbourhood so that a requesting user likely finds the desired information nearby. We simulate our caching algorithms in different ad hoc network scenarios and compare them with other caching schemes, showing that our solution succeeds in creating the desired content diversity, thus leading to a resource-efficient information access.

***Key terms**: Data Caching, Distributed Caching Algorithm, Ad hoc network*

I. INTRODUCTION

Ad hoc networks are multi hop wireless networks of small computing devices with wireless interfaces. The computing devices could be conventional computers (for example, PDA, laptop, or PC) or backbone routing platforms or even embedded processors such as sensor nodes. The problem of optimal placement of caches to reduce overall cost of accessing data is motivated by the following two defining characteristics of ad hoc networks. First, the ad hoc networks are multi hop networks without a central base station. Thus, remote access of information typically occurs via multi hop routing, which can greatly benefit from caching to reduce access latency. Second, the network is generally resource constrained in terms of channel bandwidth or battery power in the nodes. Caching helps in reducing communication, this results in savings in bandwidth, as well as battery energy. The problem of cache placement is particularly challenging when each network node has a limited memory to cache data items.

In this paper, our focus is on developing efficient caching techniques in ad hoc networks with memory limitations. Research into data storage, access, and dissemination techniques in ad hoc networks is not new. In particular, these mechanisms have been investigated in connection with sensor networking peer-to-peer networks mesh networks world wide Web and even more general ad hoc networks. However, the presented approaches have so far been somewhat "ad hoc" and empirically based, without any strong analytical foundation. In contrast, the theory literature abounds in analytical studies into the optimality properties of caching and replica allocation problems. However, distributed implementations of these techniques and their performances in complex network settings have not been investigated. It is even unclear whether these techniques are amenable to efficient distributed implementations.

Our goal in this paper is to develop an approach that is both analytically tractable with a provable performance bound in a centralized setting and is also amenable to a natural distributed implementation. In our network model, there are multiple data items; each data item has a server, and a set of clients that wish to access the data item at a given frequency. Each node carefully chooses data items to cache in its limited memory to minimize the overall access cost. Essentially, in this article, we develop efficient strategies to select data items to cache at each node. In particular, we develop two algorithms—a centralized approximation algorithm, which delivers a 4-approximation (2-approximation for uniform size data items) solution, and a localized distributed algorithm, which is based on the approximation algorithm and can handle mobility of nodes and dynamic traffic conditions. Using simulations, we show that the distributed algorithm performs very close to the approximation algorithm. Finally, we show through extensive experiments on ns2 that our proposed distributed algorithm performs much better than a prior approach over a broad range of parameter values. Ours is the first work to present a distributed implementation based on an approximation algorithm for the general problem of cache placement of multiple data items under memory constraint.

Data caching strategy for ad hoc networks whose nodes exchange information items in a peer-to-peer fashion. Data caching is a fully distributed scheme where each node, upon receiving requested information, determines the cache drop time of the information or which content to replace to make room for the newly arrived information. These decisions are made depending on the perceived “presence” of the content in the nodes proximity, whose estimation does not cause any additional overhead to the information sharing system.

We devise a strategy where nodes, independent of each other, decide whether to cache some content and for how long. In the case of small-sized caches, we aim to design a content replacement strategy that allows nodes to successfully store newly received information while maintaining the good performance of the content distribution system. Under both conditions, each node takes decisions according to its perception of what nearby users may store in their caches and with the aim of differentiating its own cache content from the other nodes’. The result is the creation of content diversity within the nodes neighbourhood so that a requesting user likely finds the desired information nearby. We simulate our caching algorithms in different ad hoc network scenarios and compare them with other caching schemes, showing that our solution succeeds in creating the desired content diversity, thus leading to a resource-efficient information access.

II. SYSTEM OUTLINE OVERVIEW:

Hamlet is fully distributed caching wireless ad hoc networks whose nodes exchange information item in a peer to peer fashion. In particular we address a mobile ad hoc network whose nodes might be resource-constrained devices, pedestrian users, or vehicles on city roads. Each node runs an application to request and possibly cache desired information items. Nodes in the network retrieve information items from other users that temporarily cache (part of) the requested items or from one or more gate way nodes, which can store content or quickly fetch it from the internet .

We propose, called Hamlet, aims at creating *content diversity* within the node neighbourhood so that users likely find a copy of the different information items nearby (Regardless of the content popularity level) and avoid flooding the network with query messages. Although a similar concept has been put forward in the novelty in our proposal resides in the probabilistic estimate, run by each node, of the *information presence* (i.e., of the cached content) in the node proximity. The estimate is performed in a cross-layer fashion by overhearing content query and information reply messages due to the broadcast nature of the wireless channel. By leveraging such a local estimate, nodes autonomously decide what information to keep and for how long, resulting in a distributed scheme that does not require additional control messages.

The Hamlet approach applies to the following cases.

• *Information presence estimation.* In this case we define the reach range of a generic node that can receive a query generated by node *n* itself. As an example, in an ideal setting in which all nodes have the same radio range, the reach range is given by the product of TTL and the node radio range. Next we denote by *f* the frequency at which every node estimate the presence of each information item with in the *reach range* , and we define as *1/f* the duration of each estimation step (also called time step hereafter).

The generic node *n* uses the information captured with in its reach range, during the estimation step *j*, to compute the following quantities:1) a provider counter by using application-layer data and 2) a transit counter by using data that are collected through channel overhearing in a cross-layer fashion. These counters are defined as follows:

• *Provider counter* $d_{ic}(n, j)$. This quantity accounts for the presence of new copies of information *i’s* chunks *c*, delivered by *n* to querying nodes within its range during step *j*. Node *n* updates this quantity every time it acts as a provider node.(e.g., like node P in the upper plot of figure 2)

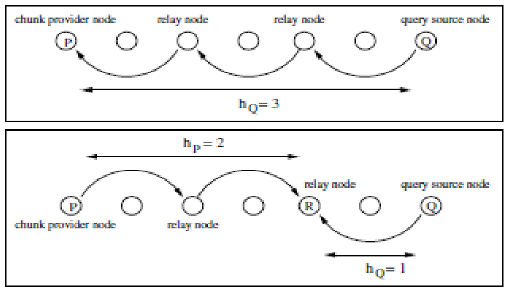

Fig. 1. Q and P denote, respectively, a node issuing a query and a node providing the requested content. Node R in the lower plot is a relay node, overhearing the exchanged messages. The plots represent: case *a)* (upper plot) h_Q value for the provider node P, and case *b)* (lower plot) h_Q and h_P values for relay node R, with respect to the query source Q and the provider P

• *Large-sized caches*. In this case, nodes can potentially store a large portion (i.e., up to 50%) of the available information items. Reduced memory usage is a desirable (if not required) condition, because the same memory may be shared by different services and applications that run at nodes. In such a scenario, a caching decision consists of computing for how long a given content should be stored by a node that has previously requested it, with the goal of minimizing the memory usage without affecting the overall information retrieval performance;

• *Small-sized caches*. In this case, nodes have a dedicated but limited amount of memory where to store a small percentage (i.e., up to 10%) of the data that they retrieve. The caching decision translates into a cache replacement strategy that selects the information items to be dropped among the information items just received and the information items that already fill up the dedicated memory. We evaluate the performance of Hamlet in different mobile network scenarios, where nodes communicate through ad hoc connectivity. The results show that our 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.

III. System Analysis

In this we are going to deal with the caching placement problems and algorithm. Those are as follows:

1. SELF-ORGANIZING
2. SELF-ADDRESSING (MANET)
3. INTEGRATED CACHE-ROUTING
4. LOCALIZED CACHING POLICY
5. DISTRIBUTED CACHING ALGORITHM

1. SELF-ORGANIZING

Multiple Data's deals with the three algorithms for cache placement of multiple data items in ad hoc networks. In the first approach, each node caches the items most frequently accessed by itself; the second approach eliminates replications among *neighboring* nodes introduced by the first approach; the third approach require one or more "central" nodes to gather neighbourhood information and determine caching placements. The first two approaches are largely localized, and hence, would fare very badly when the percentage of client nodes in the network is low, or the access frequencies are uniform. In the third approach, it is hard to find stable groups in ad hoc networks because of frequent failures and movements.

2. SELF-ADDRESSING (MANET):

A multi-hop ad hoc network can be represented as an undirected graph *G(V, E)* where the set of nodes vertices *V* the nodes in the network and *E* are the set of weighted edges in the graph. Two network nodes that can communicate directly with each other are connected by an edge in the graph. The edge weight may represent a link metric such as loss rate, delay, or transmission power. For the cache placement problem addressed in this article, there are multiple data items and each data item is served by its server (a network node may act as a server for more than one data items). Each network node has limited memory and can cache multiple data items subject to its memory capacity limitation. The objective of our cache placement problem is to minimize the overall access cost. Below, we give a formal definition of the cache placement problem addressed in this system.

Problem Formulation. Given a general ad hoc network graph $G(V;E)$ with p data items $D_1, D_{2.........}, D_p$, where a data item Dj is served by a server S_j . A network node may act as a server for multiple data items. *For clarity of presentation*, we assume uniform-size (occupying unit memory) data items for now. Our techniques easily generalize to non-uniform size data items, as discussed later. Each node i has a memory capacity of mi units. We use a_{ij} to denote the access frequency with which a node i requests the data item D_j , and d_{il} to denote the weighted distance between two network nodes i and l. The *cache placement problem* is to select a *set of sets* $M = \{M_1, M_2. . . ,M_p\}$, where M_j is a set of networknodes that store a copy of Dj , to minimize the total access cost

$$\tau(G, M) = \sum_{i\in V}\sum_{j=1}^{p} a_{ij} \times min_{l\in(\{S_j\}\cup M_j)} d_{il},$$

under the memory capacity constraint that

$$|\{M_j | i \in M_j\}| \leq m_i \qquad \text{for all } i \in V,$$

which means each network node i appears in at most mi sets of M. The cache placement problem is known to be NP-hard [3].

EXAMPLE 1: Figure 1 illustrates the above described cache placement problem in a small ad hoc network. In Figure 2, each graph edge has a unit weight. All the nodes have the same memory capacity of 2 pages, and the size of each data item is 1 memory page. Each of the nodes 1, 2, 3, 4, and 12 have one distinct data item to be served (as shown in the parenthesis with their node numbers). Each of the client nodes (9, 10, 11, 13, and 14) accesses each of the data items $D1;D2$, and $D3$ with unit access frequency. Figure 2 shows that the nodes 5, 6, 7, 8 have cached one or more data items, and also shows the cache contents in those nodes. As indicated by the bold edges, the clients use the nearest cache node instead of the server to access a data item. The set of cache nodes of each data item are: $M1 = \{7, 8\}$,$M2 = \{7, 8\}$, $M3 = \{5, 6\}$. One can observe that total access cost is 20 units for the given cache placement.

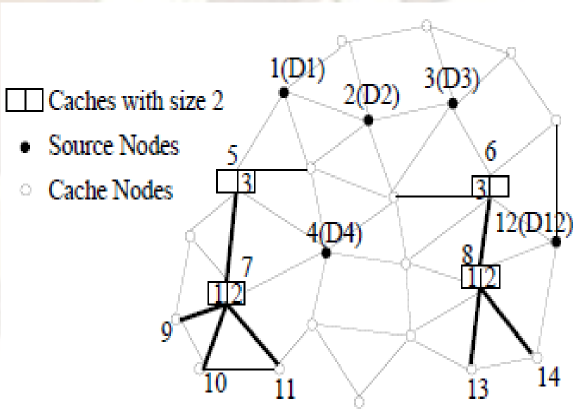

Fig. 2. Illustrating cache placement problem under memoryconstraint

3. Integrated Cache Routing

Integrated cache routing is based on three step procedure those are as follows:

- Nearest-caching tables can be used in conjunction with any underlying routing protocol to reach the nearest cache node, as long as the distances to other nodes are maintained by the routing protocol.

- However, note that maintaining cache-routing tables instead of nearest-cache tables *and* routing tables doesn't offer any clear advantage in terms of number of messages transmissions.

- We could maintain the integrated cache-routing tables in the similar vein as routing tables are maintained in mobile ad hoc networks. Alternatively, we could have the servers periodically broadcast the latest cache lists. In our simulations, we adopted the latter strategy, since it precludes the need to broadcast Add Cache and Delete Cache messages to some extent.

4. LOCALIZED CACHING POLICY

- The caching policy of DGA is as follows. Each node computes benefit of data items based on its "local traffic" observed for a sufficiently long time. The *local traffic* of a node *i* includes its own local data requests, non-local data requests to data items cached at *i*, and the traffic that the node *i* is forwarding to other nodes in the network.

- A node decides to cache the most beneficial (in terms of local benefit per unit size of data item) data items that can fit in its local memory. When the local cache memory of a node is full, the following cache replacement policy is used.

- In particular, a data item is newly cached only if its local benefit is higher than the benefit threshold, and a data item replaces a set of cached data items only if the difference in their local benefits is greater than the benefit threshold.

6. DISTRIBUTED GREEEDY ALGORITHM (DGA)

Distributed Greedy Algorithm (DGA) is also known as Distributed Caching Algorithm (DCA).

The above components of nearest-cache table and cache replacement policy are combined to yield our Distributed Greedy Algorithm (DGA) for cache placement problem. In addition, the server uses the cache list to periodically update the
caches in response to changes to the data at the server. The departure of DGA from CGA is primarily in its inability to gather information about all traffic (access frequencies).

In addition, the inaccuracies and staleness of the nearest cache table entries (due to message losses or arbitrary communication delays) may result in approximate local benefit values. Finally, in DGA, the placement of caches happens simultaneously at all nodes in a distributed manner, which is in contrast to the sequential manner in which the caches are selected by the CGA. However, DGA is able to cope with dynamically changing access frequencies and cache placements. As noted before, any changes in cache placements trigger updates in the nearest-cache table, which in turn affect the local benefit values. Below is a summarized description of the DGA.

Algorithm 1: Distributed Greedy Algorithm (DGA)

Setting

A network graph $G(V;E)$ with p data items. Each node i has a memory capacity of m_i pages. Let □ be the benefit threshold.

```
Program of Node i
BEGIN
    When a data item $D_j$ passes by
    if local memory has available space and ($Bij$ > □)
    then cache $D_j$
    else if there is a set $D$ of cached data items such
        that(local benefit of $D$ < $Bij$ □ □) and (|$D$| □ □ |$D_j$ |),
        then replace $D$ with $Dj$
    When a data item $D_j$ is added to local cache
    Notify the server of $Dj$
    Broadcast an AddCache message with ($i$ ,$Dj$)
    When a data item $D_j$ is deleted from local cache
    Get the cache list $C_j$ from the server of $D_j$
    Broadcast a DeleteCache message with ($i'$,$D_j$ ,$C_j$)
On receiving an AddCache message
    ($i'$,$D_j$)
if $i'$ is nearer than current nearest-
    cache for $D_j$
then update nearest-cache table entry
    and broadcast the AddCache
    message to neighbors
else send the message to the nearest-
    cache of $i$
On receiving a DeleteCache message
    ($i_0$, $D_j$, $C_j$ )
if $i'$ is the current nearest-cache for $D_j$
then update the nearest-case of $D_j$
    using $C_j$ , and broadcast the
    DeleteCache message
else send the message to the nearest-
    cache of $i$
For mobile networks, instead of
    AddCache and DeleteCache
    messages, for each data item, its
    server periodically broadcasts (to
    the entire network) the latest
    cache list.
END.
```

Performance Analysis. Based on the above observation, if we assume that local benefit is reflective of the accurate benefit (i.e., if the local traffic seen by a node *i* is the only traffic that is affected by caching a data item at node *i*), then DGA also yields a solution whose benefit is one fourth of the optimal benefit.

Data Expiry and Cache Updates. We incorporate the concepts of data expiry and cache updates in our overall framework as follows. For data expiry, we use the concept of *Time-to-Live (TTL)* [7], which is the time till which the given copy of the data item is considered valid/fresh. The data item or its copy is considered *expired* at the
end of the TTL time value. We consider two data expiry models, viz. *TTL-per-request* and *TTL-per-item*. In the *TTL-per-request* data expiry model [7], the server responds to any data item request with the requested data item and an appropriately generated TTL value. Thus, *each* copy of the data item in the network is associated with an independent TTL value. In the *TTL-per-item* data expiry model, the server associates a TTL value with each *data item* (rather than each request), and all requests for the same data item are associated with the same TTL (until the data item expires). Thus, in the *TTL-per-item* data expiry model, all the fresh copies of a data item in the network are associated with the same TTL value. On expiry of the data item, the server generates a new TTL value for the data item. For updating the cached data items, we consider two mechanisms. For the case of *TTL-per-request* data expiry model, we use the *cache deletion* update model, where each cache node independently deletes its copy of the expired data item. Such deletions are handled in the similar way as describe before, i.e., by broadcasting a *DeleteCache* request. In the case of *TTL-per-item* data expiry model, all the copies of a particular data item expire simultaneously. Thus, we use the *server multicast* cache update model, wherein the server multicasts the fresh copy of the data item to all the cache nodes, on expiration of the data item (at the server). If the cache list is not maintained at the server, then the above update is implemented using a network wide broadcast

CONCLUSION:

❖ Data caching strategy for ad hoc networks whose nodes exchange information items in a peer-to-peer fashion. Data caching is a fully distributed scheme where each node, upon receiving requested information, determines the cache drop time of the information or which content to replace for the newly arrived information.

❖ We have developed a paradigm of data caching techniques to support effective data access in ad hoc networks. In particular, we have considered memory capacity constraint of the network nodes.

❖ We have developed efficient data caching algorithms to determine near optimal cache placements to maximize reduction in overall access cost. Reduction in access cost leads to communication cost savings and hence, better bandwidth usage and energy savings.

❖ However, our simulations over a wide range of network and application parameters show that the performance of the caching algorithms.

❖ Presents a distributed implementation based on an approximation algorithm for the problem of cache placement of multiple data items under memory constraint.

❖ The result is the creation of content diversity within the nodes neighborhood so that a requesting user likely finds the desired information nearby.

❖ We simulate our caching algorithms in different ad hoc network scenarios and compare them with other caching schemes, showing that our solution succeeds in creating the desired content diversity, thus leading to a resource-efficient information access.

LIMITATIONS & FUTURE ENHANCEMENTS :

Hamlet is caching self contained and is designed to self adapt to network environments with different mobility and connectivity features.

We assume a content distribution system where the following assumptions hold:

1) A number *I* of *information items* is available to the users, with each item divided into a number *C* of *chunks*;

2) User nodes can overhear queries for content and relative responses within their radio proximity by exploiting the broadcast nature of the wireless medium; and

3) User nodes can estimate their distance in hops from the query source and the responding node due to a hop-count field in the messages. Although Hamlet can work with *any* system that satisfies the aforementioned three generic assumptions, for concreteness, we detail the features of the specific content retrieval system that we will consider in the remainder of this paper.

REFERENCE & BIBLIOGRAPHY:

Good Teachers are worth more than thousand books, we have them in Our Department.

[1] Caching Strategies Based on Information Density Estimation in Wireless Ad Hoc NetworksMarco Fiore, *Member, IEEE*, Claudio Casetti, *Member, IEEE*, and Carla-Fabiana Chiasserini, *Senior*

Member, IEEE IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, VOL. 60, NO. 5, JUNE 2011 pg.no.1294 to pg.no.2028

[2] B.-J. Ko and D. Rubenstein, "Distributed self-stabilizing placement of replicated resources in emerging networks," *IEEE/ACM Trans. Netw.*, vol. 13, no. 3, pp. 476–487, Jun. 2005.

[3] "Benefit-based Data Caching in Ad Hoc Networks " Bin Tang, *Member, IEEE*, Himanshu Gupta, *Member, IEEE,* and Samir R. Das, *Member, IEEE*

[4] 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.

[5] C.-Y. Chow, H. V. Leong, and A. T. S. Chan, "GroCoca: Group-based peer-to-peer cooperative caching in mobile environment," *IEEE J. Sel. Areas Commun.*, vol. 25, no. 1, pp. 179–191, Jan. 2007.

[6] T. Hara, "Cooperative caching by mobile clients in push-based information systems," in *Proc. CIKM*, 2002, pp. 186–193.

[7] 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.

[8] N. Dimokas, D. Katsaros, and Y. Manolopoulos, "Cooperative caching in wireless multimedia sensor networks," *ACM Mobile Netw. Appl.*, vol. 13, no. 3/4, pp. 337–356, Aug. 2008.

[9] Y. Du, S. K. S. Gupta, and G. Varsamopoulos, "Improving on-demand data access efficiency in MANETs with cooperative caching," *Ad Hoc Netw.*, vol. 7, no. 3, pp. 579–598, May 2009.

[10] W. Li, E. Chan, and D. Chen, "Energy-efficient cache replacement policies for cooperative caching in mobile ad hoc network," in *Proc. IEEE WCNC*, Kowloon, Hong Kong, Mar. 2007, pp. 3347–3352.

[11] 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.

[12] H. Chen, Y. Xiao, and X. Shen, "Update-based cache replacement policies in wireless data access," in *Proc. BroadNets*, Boston, MA, Oct. 2005, pp. 797–804.

[14] J. Xu, Q. Hu, W.-C. Lee, and D. L. Lee, "Performance evaluation of an optimal cache replacement policy for wireless data dissemination," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 1, pp. 125–139, Jan. 2004.

[14] J. Cao, Y. Zhang, G. Cao, and L. Xie, "Data consistency for cooperative caching in mobile environments," *Computer*, vol. 40, no. 4, pp. 60–66, Apr. 2007.

[15] N. Dimokas, D. Katsaros, and Y. Manolopoulos, "Cache consistency in wireless multimedia sensor networks," *Ad Hoc Netw.*, vol. 8, no. 2, pp. 214–240, Mar. 2010.

[16] M. Fiore, F. Mininni, C. Casetti, and C.-F. Chiasserini, "To cache or not to cache?" in *Proc. IEEE INFOCOM*, Rio de Janeiro, Brazil, Apr. 2009, pp. 235–243.

REFERENCES MADE FROM:

1. Professional Java Network Programming
2. Java Complete Reference
4. Data Communications and Networking, by Behrouz A Forouzan.
5. Computer Networking: A Top-Down Approach, by James F. Kurose.
6. Operating System Concepts, by Abraham Silberschatz.

C.Srinivas *M.Tech Associate Professor , HOD IT Department***,** *Sree Visvesvraya Institute of technology and Sciences* affiliated to JNTUH, approved by AICTE New Delhi, Mahabubnagar. His research includes Information security , Networking

Samreen Khan pursuing *M.Tech CSE final year in Sree Visvesvaraya Institute of Technology and Science* affiliated to JNTUH, approved by AICTE New Delhi. Chowderpally Mahabubnagar.
Research area includes Java computer network, information security .