Contention-Aware Data Caching in Wireless Multihop Ad Hoc Networks

Xiaopeng Fan, Jiannong Cao
Department of Computing
Hong Kong Polytechnic University
Kowloon, Hong Kong
{csxpfan, csjcao}@comp.polyu.edu.hk

Weigang Wu
Department of Computer Science
Sun Yat-sen University
Guangzhou, China
wuweig@mail.sysu.edu.cn

***Abstract*—Data caching is one of the attractive techniques that can increase the efficiency of data access in wireless multi-hop ad hoc networks. However, it is still a challenging task to achieve an optimal trade-off between the total traffic cost and average access delay by properly selecting cache nodes, i.e. cache placement. In this paper, we address the problem of cache placement in wireless multi-hop ad hoc networks. We consider dynamic networks, in which there is a data source that stores one data item and other client nodes require accessing the data item. We define the cache placement problem on a dynamic network topology as Dynamic Cache Placement (DCP). Most of the existing cache placement algorithms use hop counts to measure the total cost of caching systems. Due to the impact of contentions in wireless networks, one hop delay is different from each other and it varies with the traffic load. Thus the previous algorithms cannot achieve their expected performance. We propose two heuristic cache placement algorithms, named Centralized Contention-Aware Caching Algorithm (CCCA) and Distributed Contention-aware Caching Algorithm (DCCA), which detect the variation of contentions to evaluate the benefit of selecting a node as cache node. Simulation results show that the proposed algorithms achieve better performance than other alternative ones in terms of average query delay, caching overheads, and query success ratio.**

Keywords-contention-aware; cache placement; wireless ad hoc network

I. INTRODUCTION

With the rapid growth of interest in mobile computing, wireless multi-hop ad hoc networks are used to provide radio devices with many services anywhere and anytime, such as Internet access. One of the most important merits of such networks is to enable users to spontaneously form a dynamic communication system. Users can access these services offered by the fixed network through multi-hop communications, without requiring infrastructure in the users' proximity. However, there are several challenging issues to improve the efficiency of data access in wireless ad hoc networks. First, wireless networks are notorious for the scarcity of communication bandwidth and resources. For example, sensor networks can only have at most a gross data rate with 250kbps [1] in the physical layer. Second, more and more applications require transmitting more datum than before, which dramatically increase the traffic load of networks. In wireless mesh networks, streaming services are becoming much more popular, and some real-time applications require quality of services (QoS) guarantees, such as real-time surveillance. Third, if users are mobile, they form a mobile ad hoc network such that dynamic topologies exacerbate the performance of the algorithms designed for static topologies. In such environment, dynamic topology should be considered to improve the efficiency of data access. For example, Internet-based Mobile Ad Hoc Network (IMANET) combines a mobile ad hoc network (MANET) and the Internet to provide universal information accessibility [13].

To overcome the problems above, data caching is one of the attractive techniques that can increase the efficiency of data access in wireless multi-hop ad hoc networks. The data source transfers some copies of data item to some client nodes, called cache nodes. Thus other client nodes can access the copies of the data item from cache nodes instead of the data source. In this way, average access delay is decreased due to the service provided by cache nodes. However, the data source is responsible for updating the copies at cache nodes periodically or aperiodically. The traffic introduced by updates brings more traffic cost. Obviously, there is a trade-off between average access delay and overall traffic cost.

The problem of cache placement, how to select cache nodes, has been widely studied in the context of wired and wireless networks. Normally, these works [6] [7] model this problem as an optimization problem under certain kind of topologies, and build the objective function that describes the total cost. Then they propose solutions to minimize the total cost. However, they measure the cost of a caching system with hop counts, i.e., they assume that there is no difference between any pair of hops. To the best of our knowledge, the previous work [7] is the one with the best performance, which considers the cache placement with single data item in a general static topology. In [7], the cache placement problem is mapped to the rent-or-buy problem, a special case of the connected facility location problem [3], which is known to be NP-hard.

However, due to the shared nature of wireless medium, nodes may contend with each other to access the medium. Therefore, per-hop delay on each node is different from each other [2]. Some previous works conclude that per-hop delay mainly depends on the contention delay in 802.11-based wireless ad hoc networks [2]. Additionally, when more than

978-1-4244-5113-5/09/$25.00 ©2009 IEEE

one nodes access the same data item at one cache node, it increases the probability of access collisions. More collisions bring more retransmissions of data requests such that the access delay is increased sharply. As a result, it is necessary to consider the impact of contentions at wireless nodes on the performance of data caching.

In this paper, our problem is different from the problems above in the following way. There are three important points: 1) each hop has different weight in wireless multi-hop ad hoc networks due to contentions; 2) the strategy of selecting cache nodes changes the contentions of network nodes by the variation of traffic flows; and 3) the contentions can be changed due to the movement of mobile nodes. This problem can be described as the cache placement problem on a topology with dynamic topology. We define such a problem as Dynamic Cache Placement (DCP). Due to its intractability, we address it by proposing a heuristic strategy, which dynamically collects the contention variation on wireless nodes from the MAC layer, and selects proper nodes as cache node to balance the caching overheads with access delay. We try to reduce the length of paths from client nodes to cache nodes and select nodes with fewer contentions as cache nodes. Our goal is to find an algorithm to select a set of cache nodes in order to minimize the total cost, which includes caching overheads and total access delay. To the best of our knowledge, this is the first time to take the contentions into consideration for caching system in wireless multi-hop ad hoc networks.

The proposed heuristic algorithms, called Centralized Contention-Aware Caching Algorithm (CCCA) and Distributed Contention-aware Caching Algorithm (DCCA), provide a sub-optimal solution to the cache placement problem in wireless multi-hop ad hoc networks. Our algorithm have the following desirable properties: 1) it is a polynomial time algorithm; 2) it is a cross-layer designed algorithm because it makes caching decisions based on the contention information collected from the MAC layer; 3) it can be applied and adapted to any arbitrary dynamic network topology; and 4) it can be easily implemented in a distributed and asynchronous fashion.

The rest of the paper is organized as follows: Section 2 discusses the related work. Section 3 describes the network model, the problem formulation, and the calculation of per-hop delay. In Section 4, we present the main ideas of our algorithms, describe the details of the propose algorithms, and analyze their performance. Simulation results are reported in Section 5. Finally, Section 6 concludes this paper.

II. RELATED WORK

The problem of caching placement has been widely studied in the context of both wired networks and wireless networks. In this paper, we focus on the cache placement for a single data item in a wireless multi-hop ad hoc network. The key problem of determining optimal cache placement in an arbitrary network topology has similarity to two problems in graph theory viz. the facility location problem and the k-median problem [3]. As we all known, the two problems have been proved to be NP-Hard. In the facility location problem, setting up a cache at a node incurs a certain fixed cost, and the goal is to minimize the sum of total access cost and the fixed costs of all caches. On the other hand, the k-median problem is to minimize the total access cost under the constraint of the number of cache nodes, i.e., that at most k nodes can be selected as cache nodes. A number of constant factor approximation algorithms have been developed for each of the two problems [4], under the assumption of triangular inequality of edge costs. Without the triangular inequality assumption, either problem is as hard as approximating the set cover [5], and thus can be approximated better than $O(\log|V|)$ unless $P=NP$.

In [6], the authors address the problem of proxy placement and employ dynamic programming to determine the optimal placement. They only consider the case of tree topology. In the context of wireless network, Nuggehailli et al. [7] formulate the cache placement problem in ad hoc wireless networks as a special case of the connected facility location problem [8], named as the rent-or-buy problem. An existing facility is given, along with a set of locations at which further facilities can be built. Every location is associated with a service demand, denoted by p_k, which must be served by one facility. The cost of serving k by using facility j is equal to $p_k c_{kj}$, where c_{kj} is the cost of connecting k to j. The fixed cost of opening a facility at any location is zero. Besides selecting the sites to build the facilities, all of the facilities should be connected by using a Steiner Tree with the given facility as root. To the best of our knowledge, the best solution for this problem is 2.92-approximation algorithm [9]. In [7], the authors propose an algorithm named POACH, which is a 6-approximation algorithm for ad hoc wireless networks. However, POACH is designed for static topology such that it does not work in a mobile ad hoc network. Moreover, the performance of POACH is evaluated by the numerical results, which cannot reflect the reality of wireless ad hoc networks. Although the author mentioned that POACH can be implemented in a distributed fashion, there is still no any real implementation in the original work and the extended version ECHO [14]. It is important to know that ECHO requires global topological information and disseminating such information in a low-cost manner. It is not a good choice in mobile ad hoc networks.

As we mentioned above, the objective function of the cost in the previous caching systems is measured by hop counts. In a real wireless multi-hop ad hoc network, the cost of sending a packet depends on the delay of this hop, which mainly results from the delay caused by contentions [2]. Furthermore, these contentions vary as the change of topology in mobile ad hoc networks. Based on this observation, we try to detect the variation of contentions on a wireless node in order to select cache nodes for the optimal trade-off between the traffic cost and average query delay. The reasons why we choose contentions as the measurement of the traffic cost and the heuristics of selecting cache nodes are that 1) contention is a practical impact in wireless networks, which means we should consider the impact of packet loss and transmission delay on the performance of data caching; 2) contentions on a wireless node can be easily detected from the MAC layer regardless of the mobility model of nodes; and 3) the variation of contentions is sensitive to topology changes, especially in mobile environments.

III. PROBLEM FORMULATION

In this section, we first introduce the system model for our work. Next we formulate the cache placement problem in wireless multi-hop ad hoc networks.

A. System Model

Let $G(V, E)$ be a connected graph representing a wireless multi-hop ad hoc network with n ($n=|V|$) nodes connected by m ($m=|E|$) links. Two network nodes communicate directly with each other, which is represented by an edge in the graph, whereas nodes that cannot communicate with each other directly may still contend directly with each other due to the shared nature of wireless medium. The radius of the contention range r depends on the type of wireless networks. Normally, the radius of the contention range in 802.11-based ad hoc networks is two times as big as the radius of the transmission range. The data rate of channel is defined as W.

For Internet service or other applications, wireless nodes need to access a data item d maintained by the data source s. Let s_D be the size of data item d. The data source updates data item d with some given frequency f_u, i.e., the data item d changes with the mean update time T_u. Node i has the access frequency $f_a(i)$ for the data item d. The size of a data request is defined as s_R. In order to reduce access latency, the data source caches data item d at some nodes in the network. Similar to the model in [7], we define two phases in this paper, *the update phase* and *the access phase*. The first phase is that the server updates cache copies in the network, and the second phase means that the nodes send requests for data item d to the nearest cache nodes. We define a cache placement strategy by the vector $V_C = \{c_1, c_2, \ldots c_n\}$, where $c_i = 1$ if one copy of data item d is cached on the N_i. Otherwise, c_i is equal to 0.

To model our problem, we have the following assumptions.

- The data server s always possesses a cache copy of data item d.
- Links are bidirectional.
- Wireless nodes never fail.

B. Problem Formulation

Given a set of graphs $G^k=\{G_0, G_1, G_2, \ldots, G_k\}$, which describes the change of topology for a wireless multi-hop ad hoc network G. We define a set $V=\{N_1, N_2, \ldots, N_n\}$ as n wireless nodes specified by a piecewise functions $\{g_1, g_2, \ldots, g_n\}$, where g_i, $1 \leqq i \leqq n$ maps time interval $[0,T]$ to G_k. Given k functions $f_1, f_2, \ldots, f_k$, $f_i: G_i \rightarrow [0,1]$ to select cache nodes for G_i such that at given moment $t \in [0,T]$, f_i forms a sup-problem SP_i for all the mobile nodes located by $\{g_1, g_2, \ldots, g_n\}$.

We divide the cache placement problem in wireless multi-hop ad hoc networks into sub-problems. For each sup-problem SP_i, we provide a formal formulation for the cache placement problem. Given a directed connected graph $G_i(V,E)$ and the weight of each link depends on the contentions of the sender. Let C be the set of cache nodes. Each message is sent from the sender i to the receiver j by following a directed path $PATH(i, j)$. The weight of a path, $w(i, j)$, is the sum of the total weight of each hop on this path. The weight of each hop (x, y) is measured by the per-hop delay of the sender x. As we mentioned before, $D(x)$ is mainly determined by the contention delay of node x. Let $d_c(x)$ be the contention delay of node x. Thus we substitute the contention delay $d_c(x)$ for the per-hop delay $D(x)$ in this paper. We can obtain $w(i, j)$ as follows:

$$w(i,j) = \sum_{x} d_c(x) \quad x \in PATH(i,j) \text{ and } x \neq j$$

Next, let $w_{min}(c, s)$ be the minimum weight among all of the paths from cache node c to the data source s. Let $w_{min}(i, c)$ be the minimum weight among all of the paths from N_i to the cache node c.

As we mentioned in Section 3.1, the cost in the update phase is defined as C_u, i.e., the traffic cost that the data source s sends update packets to all of the cache nodes. The cost in the access phase includes: 1) the traffic cost, C_a, i.e. client nodes send data queries to the set of cache nodes, and cache nodes (including the data source) send reply packets to the client nodes, and 2) the access delay D_a that the nodes experience. Given a strategy $V_c(i)$ for the sub-problem SP_i, we define the total cost as follows:

$$Cost(G_i, V_c(i)) = C_u + C_a + D_a \qquad (1)$$

Therefore, the cache placement problem for the sub-problem SP_i is to select a set of cache nodes $V_c(i)$ to minimize the total cost:

$$\begin{aligned} Cost(G_i, V_c(i)) = & \sum_{n \in C} w_{\min}(n,s) \times s_D \times f_u \\ & + \sum_{n \in V} \sum_{c \in C} w_{\min}(n,c) \times s_R \times f_a(i) \\ & + \sum_{n \in V} \sum_{c \in C} w_{\min}(n,c) \times f_a(i) \end{aligned} \qquad (2)$$

Therefore, the total cost for the cache placement is defined as C_{total}, which can be calculated as follows:

$$C_{total} = \sum_{i=0}^{k} Cost(G_i, V_c(i)) \qquad (3)$$

The cache placement problem on a dynamic topology is to find the k functions to minimize the total cost C_{total}. We define this problem as *Dynamic Cache Placement Problem (DCP)*.

Theorem 1: *The computation of DCP is NP-hard.*

PROOF:

If the topology of the network is not changed, it is proved that DCP is the rent-or-buy problem, the special case of the connected facility location problem. It has been proved NP-hard [8].

If the topology is changed due to the mobility of nodes, DCP is one of the special cases of the mobile facility location

problem [15]. The mobile facility location problem is proposed for continuously moving points. Given a set S of n demand points in d-dimensional space (d ≥1), find a set P of k supply points so that the dispersion defined as the sum of the L_p distance between demand points and their nearest supply pints in P is minimized. If we consider the cache nodes as facilities, DCP can be converted to a mobile facility location problem. Similar to the rent-or-buy problem, DCP in mobile networks considers the cost of updates. The mobile facility location problem is NP-hard. Therefore, DCP is also a NP-hard problem. □

As mentioned in [15], the data structures and algorithms that have been developed for the static problems (i.e. network nodes are not moving) are not directly applicable to the setting of moving nodes when the motion of the facilities must satisfy natural constraints. Therefore POACH is not expected as efficient as in mobile networks. Our simulations also demonstrate this point.

IV. A HEURISTIC SOLUTION

In this section, we first explain how to calculate the per-hop delay for each node in order to evaluate the weight of each path. Then we propose our two Contention-aware Caching Algorithms, i.e., CCCA and DCCA, which provides two heuristic rules to minimize the total cost in Eq. 2, i.e., balance caching traffic overheads with the average query delay. In our algorithm, we use two key techniques to implement our heuristic rules, including *Hedging Flow (HF)* and *Contention Coefficient*. At last, we describe the details of the whole algorithm.

A. Per-hop Delay

Since we consider the difference between hops in wireless multi-hop ad hoc networks, we explain how to obtain the per-hop delay at wireless nodes to describe the weight of paths, on which the node sends data packets. To model the per-hop delay, we make use of the previous work proposed in [2]. The per-hop delay consists of the queuing delay, the contention delay, and the transmission delay. However, their work shows that average transmission delay is fixed, and average queuing delay is determined by both the mean and the variance of contention delay. Thus the per-hop delay is determined by the contention delay. The contention delay is the interval between the time that the packet becomes the head of the line (HOL) in the node's queue, and the time that the packet actually starts to be transmitted on the physical medium. In this paper, we focus on the contention delay in 802.11-based wireless ad hoc network, but the method is easily applicable for any contention-based channel access schemes. We define *CR(i)* as the set of nodes in the contention range of mobile node N_i. The contention delay *d(i, c)* can be calculated by the following equation

$$d(i,c) = DIFS + R_b(w_i\varepsilon + m_i T_d) \qquad (4)$$

where *DIFS* is defined in 802.11 as DCF Inter-Frame Space, R_b is the probability that the channel state is busy, w_i is the number of backoff slots at mobile node N_i. ε is the value of bakeoff slot, m_i is the number of data packets transmitted by the neighboring nodes during N_i's bakeoff process, and T_d is the duration of a successful data transmission. Next, we present a simple example to explain how to calculate the contention delay in 802.11 as follows.

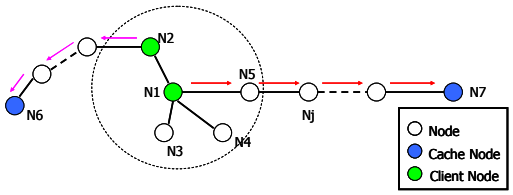

Figure 1. Example to show how to calculate the contention delay of node N1 in 802.11

In Fig. 1, the contention range *CR(1)* is referred to as the node set $\{N_2, N_3, N_4, N_5\}$, each of which is in the contention range of N_1. a_{ji} is the discount factor of concurrent influence to node N_i from node N_j [2] . Thus the contention delay of node N_1 can be calculated as follows:

$$d(1,c) = DIFS + (w_1\varepsilon + m_1 T_d)\sum_{j\in CR(1)} \frac{a_{j,1} f_a(j)}{W} + \frac{f_a(1)}{W} \qquad (5)$$

In our simulations, we collect the related parameters by interposing a detector in the source code of IEEE 802.11 MAC in NS2 [10].

B. Two Heuristic Rules

In this subsection, we explain our heuristic rules in detail. As shown in Eq.3, our objective is to minimize the total cost. When we select a cache node, the first important thing is that how much traffic flows can be saved and added. The best case is that we maximize the total traffics flows if we consider the flows saved as positive and the flows introduced as negative. We define such a kind of flows as *Hedging Flow (HF)*, which is borrowed from the finance. Thus those nodes with higher HFs can be the candidates of cache nodes. The second important thing is how to minimize the cost of sending and receiving data packets. Since we use the contention delay to describe the weight of each path, those nodes with more contentions are not suitable to act as cache nodes. Therefore, we propose *Contention Coefficient* to describe the level of contentions on each node.

Next we explain the two heuristic rules in detail. Before we introduce the first heuristic rule, we clarify traffic flows in a caching system as follow. There are mainly three kinds of traffic flows. The first one is the *Access Flow (AF)*, which is defined as the flows that traverse a node for data access. The second one is the *Reply Flow (RF)*, which is referred to as the flows that traverse a node to reply data requests from cache nodes or the data server *s*. The third one is the *Update Flow (UF)*, which is defined as the flows that traverse a node to update cache copies on cache nodes from the data server *s*. Therefore, it is important to select some special nodes as candidates for cache nodes. By selecting these candidates, we aim to reduce the access flows and the reply flows as many as

possible, and increase the update flows as few as possible otherwise. Consequently we reduce the length of paths for data access and update as much as possible. As we mentioned, *Hedging Flow* describes the variation of the traffic flows if we select some node as a cache node. We denote the hedging flow of node N_i as *HF(i)*, which can be calculated as follows:

$$HF(i) = \Delta AF(i) + \Delta RF(i) - UF(i) \quad (6)$$

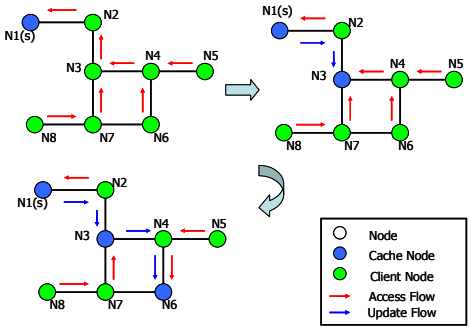

Figure 2. Example to show how to calculate the hedging flow

Given N_i that we intend to select as a cache node, we obtain the reduced access flow $\Delta AF(i)$, the reduced reply flow $\Delta RF(i)$, and the introduced update flow *UF(i)*, three of which traverse N_i. In the above calculation, note that both the access flows and the reply flows include the traffics flows that traverse N_i, not only generated by N_i. Therefore, the first rule of our heuristic algorithm is given as follows:

Rule 1: When selecting the candidates of cache nodes, those nodes with their HF higher than a given threshold Θ_1 should be selected.

To explain this rule, we use the following example shown in Fig. 2. In this example, we combine access flows with reply flows for simplicity. That means we use the access flow represents the two flows, and we assume the reply flow is sent out with the same path from the requestor to its nearest cache node. In this example, N_1 is the data source *s*. If we plan to select N_3 as the cache node, the reduced access flow is calculated by the following equation:

$$\Delta AF(3) = (f_a(4) + f_a(5) + f_a(6) + f_a(7) + f_a(8))(S_R + S_D)w(3,1) \quad (7)$$

This means the reduced access flow is the total of the access flows that traverse N_3, which should be forwarded to N_1 within two hops. Then the introduced update flow can be obtained by the following equation:

$$UF(3) = f_U * S_D * w(1,3) \quad (8)$$

According the definition of hedging flow, the hedging flow of N_3, *HF(3)*, can be obtained as follows:

$$HF(3) = \Delta AF(3) - UF(3) \quad (9)$$

The first heuristic rule aims to optimize the length of the paths from client nodes to its nearest cache nodes, i.e., client nodes access the data item with fewer cost. However, this operation changes the traffic load locally such that the contentions in the network are also changed. Therefore, the second heuristic rule is designed to select cache nodes from the candidates with fewer contentions.

To select cache nodes from the candidates, we detect the variation of contentions on all the candidates. If the contentions are lower enough, it decreases the access delay dramatically. Therefore, we present our second heuristic rule as follows:

Rule 2: When selecting a cache node from the set of the candidates, those nodes with their contention coefficients greater than Θ_2 should be selected.

When we design our algorithms to implement *Rule 2*, we use the *Contention Coefficient* of N_i, *CC(i)*, to describe the level of contentions on N_i. We define *F(i)* as the set of the nodes that have flows traversing N_i. Thus we can obtain *CC(i)* by the following equation:

$$CC(i) = d(i,c) * (f_a(i) + \sum_{j \in F(i)} f_a(j)) \quad (10)$$

In wireless technologies, we can collect the contention information from the MAC layer every T_{con} seconds in order to calculate the variation of contentions on each node. For example, from time *t* to time ($t + T_c$), we count the number of back-offs on N_i, and the number of the packets sending by N_i's neighbors during N_i's back-offs. Then the *Contention*

Algorithm 1: Centralized Contention-aware Caching Algorithm
INPUT:
Graph G[n][n], Contention Coefficient CC[n]
OUTPUT:
Cache Node Set C[n], Nearest Cache Node Set NC[n]
BEGIN
Data server s converts G to a directed weighted Graph G' by using CC[n].
C = {s} and NC = {s}
While (V-C) != Ø)
For each x ∈(V-C), s calculates the hedging flow HF(x).
While (y ∈ RoutingTable(x) & y ∉ C & HF(y) is not calculated before)
If($w_{min}(y, x) < w_{min}(y, NC[y])$)
$HF(x) += (w_{min}(y,NC[y]) - w_{min}(y, x))(s_D + s_R) f_a(y)$
End While
$HF(x) -= w_{min}(s,x) * s_D * f_u$
If HF(x) > Θ_1, add x to cache candidates set H.
End For
If H == Ø, break.
For each node a ∈ H
If CC(a)< Θ_2, add a to cache node set C.
End For
Data server s re-calculates contention coefficients.
Data server s converts G to a directed weighted Graph G' by using CC[n].
For each node z ∈ V and z∉ C
NC[z] = arg $min_{b \in C}$ w(z, b)
End For
End While // V-C
END.

Coefficient CC(i) can be calculated by Eq.10. Normally a dramatic variation between two sequential periods means that traffic flows are changed dramatically, or the topology is changed around nodes. We detect the variation of contentions regardless of the mobility model. This is also another merit of our solution.

C. Centralized Contention-aware Caching Algorithm

In this subsection, we present a centralized algorithm to solve DCP. We refer to the proposed algorithm as CCCA (Centralized Contention-aware Caching Algorithm). CCCA is mainly designed for the case of static topology, such as in a wireless sensor network. CCCA starts with the case that only the data sever *s* has the data item *d*. Other nodes access the data item from *s*. Then other nodes collect their neighboring set and calculate contention coefficients. Next each node sends the information above to *s*. The data server *s* calculates the hedging flows for these nodes and selects the set of cache nodes. Next the data server *s* finds the nearest cache node for each node in an iterative way. Simultaneously, the data server *s* updates the date item *d* periodically. Other client nodes access the data item from its nearest cache node or the data server *s*.

CCCA is divided into two steps. In the first step, we select the candidates for cache nodes. These candidates are selected one by one with *Rule 1*, which means we select nodes whose hedging flows are beyond the threshold Θ_1. In the second step, we select those whose contention coefficients are smaller than Θ_2 as cache nodes with *Rule 2*. Since the set of cache nodes is changed, the traffic flows are also changed consequently. Then we recalculate the contention coefficients with the changed traffic flows. It is noted that Θ_2 is normalized in the interval [0, 1]. Let *TF(i)* be the total flows that traverse node N_i. When the data server *s* recalculates contention coefficients, *TF(i)* is changed to *TF'(i)* and *CC(i)* is changed to *CC'(i)*. We can obtain *CC'(i)* by Eq. 11.

$$CC'(i) = \frac{TF'(i)}{TF(i)} CC(i) \qquad (11)$$

D. Distributed Contention-aware Caching Algorithm

In this subsection, we describe a localized distributed implementation of CCCA, namely DCCA (Distributed Contention-aware Caching Algorithm). The advantage of DCCA is that it adapts itself to dynamic traffic conditions in a mobile wireless ad hoc network with low communication overheads.

Topology Detection. Each network node collects its neighbour set every T_{top} seconds. Then each node broadcasts a *TopologyBroadcast* message to its one-hop neighbours. The *TopologyBroadcast* message for N_i contains the list of its neighbouring node list *NL(i)*, the nearest cache node *NC[i]*, the flag of cache nodes *C[i]* (1 means N_i is one of the cache nodes, otherwise 0), and its own access frequency $f_a(i)$ to data item *d*.

Contention Detection. Each network node collects its contention coefficient every T_{con} seconds. If the contention coefficient *CC[i]* at N_i is greater than Θ_3, N_i calculates the hedging flow *HF(i)*. If *HF(i)* is greater than Θ_4, N_i broadcasts a *ContentionBroadcast* message to its one-hop neighbours. The *ContentionBroadcast* message consists of its contention coefficient *CC[i]* and its hedging flow *HF(i)*. The *ContentionBroadcast* message is broadcast with a timeout T_{cb}. If N_i receives the *ContentionBroadcast* from its neighbours within T_{cb}, N_i compares its own *HF(i)* with that of its neighbours. If *HF(i)* is the maximum one, N_i sends a *CacheRequest* message to the data server *s* with a timeout T_{cr}. If the data server *s* sends back a *CacheReply* message, which includes an acknowledgment and the latest update of the data item *d*, N_i updates its state to open a cache node and updates the cache copy with the latest version. Then N_i broadcasts a *OpenCacheNode* message to its neighbors. If N_i does not receive a *CacheReply message within* T_{cr}, N_i does nothing until the next period of contention detection.

Algorithm 2: Distributed Contention-aware Caching Algorithm
INPUT:
Graph G, Contention Coefficients CC[n]
OUTPUT:
Cache Node Set C[n], Nearest Cache Node Set NC[n]
BEGIN
Every T_u seconds, data server s sends CacheUpdate(i, d, version) to the set of cache nodes C.
Every T_{top} seconds, N_i broadcasts TopologyBroadcast(i, NL(i)).
Every T_{con} seconds, N_i calculates CC[i].
if CC[i] is greater than Θ_3
Then N_i calculates HF(i).
If HF(i) is greater than Θ_4,
Then N_i broadcasts ContentionBroadcast(i, CC[i], HF(i)) with timeout T_{cb}.
On receiving TopolgoyBroadcast(j, NL(j), N[j], C[j], f_a(j))
Store this information to the table T_i.
On receiving ContentionBroadcast(j, CC[j], HF(j))
If HF(i) is smaller than HF(j)
Then N_i send AckContention(TRUE)
Else send AckContentention(FALSE)
On receiving AckContention
If within T_{con} N_i received AckContention(TRUE) from all of its neighbours,
Then N_i send CacheRequest(i) to the data server s and set a timer T_{cr}.
On receiving CacheRequest(j)
If N_i is the data server s,
Then s sends CacheReply(C, d, version) to node j.
On receiving CacheReply(C, d, version)
If T_{cr} is not timed out,
Then N_i updates its cache copy and broadcasts OpenCacheNode(i) to its neighbours.
On receiving OpenCacheNode(j)
If w(i,j)<w(i,NC[i]) Then NC[i]=j.
On receiving AccessRequst(j)
If C[i] == 1
Then N_i sends AccessReply(d) to node j.
On receiving CacheUpdate
If C[i] == 1,
Then N_i compares the local version and the timestamp with the latest ones from the data server s.
If there is a gap or the waiting time is beyond timeout $2T_u$.
Then N_i sends a CloseCacheNode to s.
Else N_i updates the local copy.
On receiving CloseCacheNode(j),
If N_i is the data server s,
Then s updates the set of cache nodes.
If NC[i] ==j, Then N_i resets its new nearest cache node.
END

Cache Update. The data server s sends a *CacheUpdate* message to all the cache nodes every T_u seconds. If N_i is one of cache nodes, N_i receives the *CacheUpdate* message within timeout $2T_u$. Then N_i updates the local copy of data item d with the latest version. If there is a gap between the local version and the received version, or the waiting time is beyond the timeout $2T_u$, N_i sends a *CloseCacheNode* to the data server s. The data sever s receives such messages and updates the set of cache nodes by removing N_i. N_i also broadcast the *CloseCacheNode* message to its neighbours.

V. SIMULATIONS

In this section, we demonstrate through simulations the performance of our designed cache placement algorithms over randomly generated network topologies. We first compare the performance of CCCA with that of POACH in a wireless network with static topology. Then we compare the relatively quality of the solutions returned by NOCACHE, POACH and DCCA in mobile networks. Here NOCACHE is a simple strategy that has no cache node in the network.

Our simulations are carried out by using the NS2 simulator. The NS2 [10] simulator contains models for common ad hoc network routing protocols, IEEE Standard 802.11 MAC layer protocol, and two-way ground reflection propagation models [11]. The DSDV routing protocol [12] is used in our work to provide routing services.

We simulated our algorithm in a network of randomly placed 100 nodes in an area of 2000x500m^2. Note that the normal radio range for two directly communicating nodes in the NS2 is about 250 meters. In our simulations, we set node 0 as the data source and the size of data item d is 1500 bytes.

Each network node is a client node. Each client node in the network sends out a single stream of read-only queries. Each query is essentially the http request for the same data item d.

A. Static Networks

In this subsection, we evaluate the relative performance of CCCA and the previous work POACH [3].

In our simulations, the time interval between two consecutive queries is know as the *query generation time* and follows exponential distribution with the mean value T_q which we vary from 2 to 12 seconds. In our update model, the time between two consecutive updates is defined as the *update generation time* and follows exponential distribution with the mean value T_u, which we vary from 2 to 12 seconds. We set the timeout of the request as 20 seconds.

We measure two performance metrics for comparison of various cache placement strategies, viz., average query delay and caching overheads. Query delay is defined as the interval between the time when a client node sends its data requests out and the time when the client node obtains the reply from its nearest cache node. Average query delay is the average of query delays over all queries. Caching overheads includes all of the packets in our caching system, viz., data requests, data replies, data updates and other messages for caching systems. Note that these packets do not include routing packets because the two strategies use the same routing protocol DSDV. In our CCCA, we set the hedging flow threshold Θ_1 as 100.0 and set the contention threshold Θ_2 as 0.04.

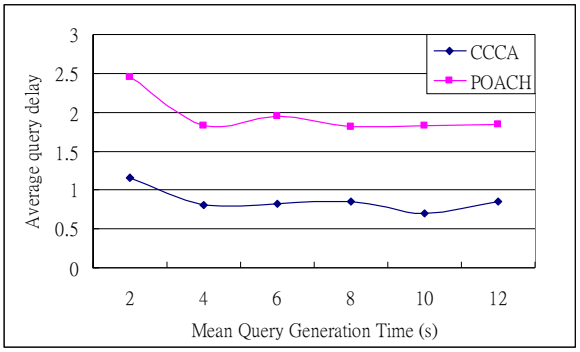

Figure 3. Average query delay vs. Mean query generation time

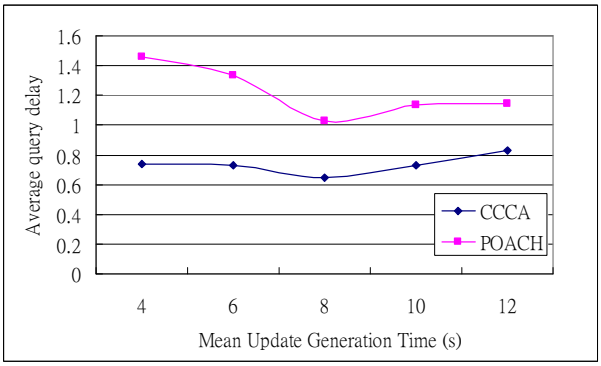

Figure 4. Average query delay vs. Mean update generation time

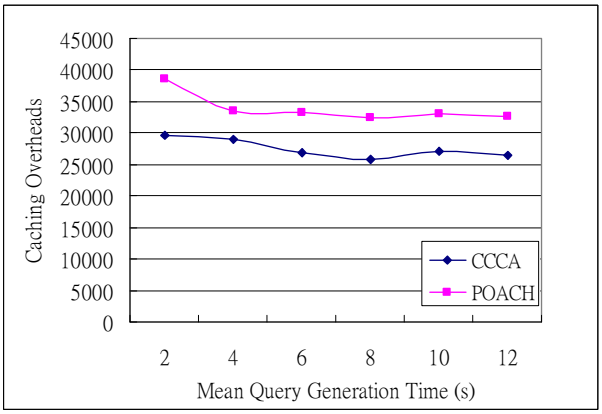

Figure 5. Caching Overheads vs. Mean query generation time

1) Average Query Delay

Figure 3 and Figure 4 show the results of comparison between POACH and CCCA in terms of average query delay. In Figure 3, we vary the mean query generation time while keeping the mean update generation time T_u constant at 2 seconds. We observe that our CCCA outperforms POACH constantly for all mean query generation time. The difference is much more significant for lower mean generation time, which suggests that CCCA is more suitable for higher frequency of data access. In Figure 4, we vary the mean update generation time while keeping the mean query generation time constant at

4 seconds. The results shows that CCCA also outperforms the POACH.

2) Caching Overheads

Figure 5 and Figure 6 are presented to show the performance comparison between POACH and our CCCA in terms of caching overheads. In Figure 5, we vary the mean query generation time while keeping the mean update generation time T_u constant at 2 seconds. We observe that CCCA has a better performance than POACH constantly for all mean query generation time. In Figure 6, we vary the mean update generation time while keeping the mean query generation time constant at 4 seconds. Figure 5 and 6 show CCCA outperforms POACH, especially in the case with higher update and access frequency.

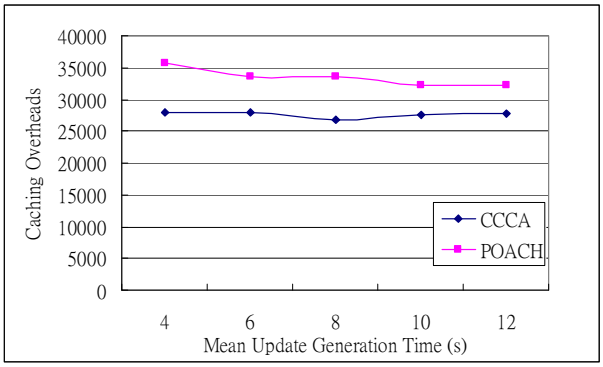

Figure 6. Caching Overheads vs. Mean update generation time

B. Mobile Networks

In this subsection, we evaluate the performance of NOCACHE, POACH and DCCA in mobile networks. Note that we still use POACH in a centralized manner due to the unpractical factor to implement a distributed version in a mobile ad hoc network, as we mentioned in Section 2.

In our simulations, mobile nodes move based on the random waypoint model in NS2. In this model, each node selects a random destination and moves towards the destination with a speed selected randomly from ($0m/s$, $v_{max}m/s$). After the node reaches its destination, it pauses for a period of time (chosen to be 20 seconds) in our simulation and repeat the movement pattern. To focus on the impact of mobile speed on the performance of the cache placement algorithms, we set the mean query generation time as 2 seconds and set the mean update generation time as 30 seconds. We set the contention threshold Θ_3 as 0.04 and set the hedging flow threshold Θ_4 as 100.

Due to the mobility of wireless nodes, data queries may be lost, or the data reply may be lost, or the query delay is beyond the timeout that mobile nodes can endure. Therefore, we introduce another metric named query success ratio besides average query delay and caching overheads in mobile networks. The query success ratio is defined as the percentage of the queries that receive the requested data item within the query success timeout period. We set the timeout of the request as 20 seconds.

1) Average Query Delay

Figure 7 shows simulation results comparing three caching algorithms, viz. NOCACHE, POACH and DCCA in terms of average query delay. In Figure 7, we vary the maximum speed of mobile nodes from 2m/s to 16m/s while keeping the mean update generation time and the mean update generation time. We observe that DCCA outperforms POACH and NOCACHE constantly for all the maximum speed. The results also show that DCCA has a much better performance with higher speed.

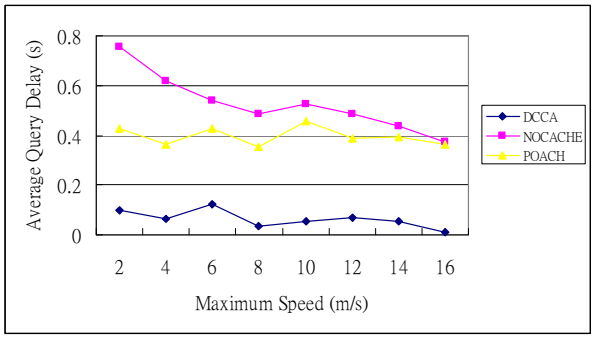

Figure 7. Average Query Delay vs. Maximum Speed

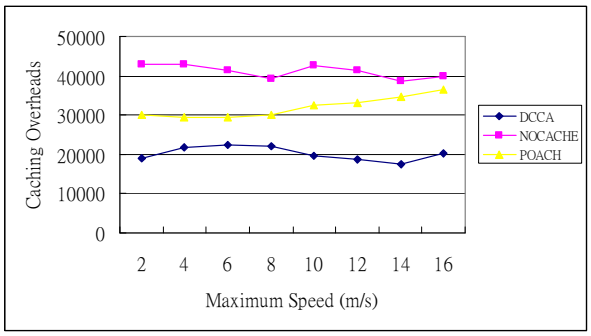

Figure 8. Caching Overheads vs. Maximum Speed

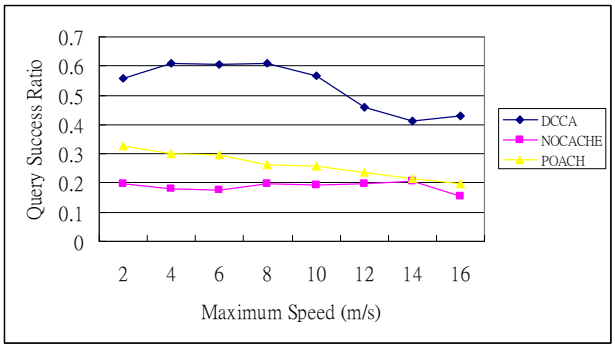

Figure 9. Query Success Ratio vs. Maximum Speed

2) Caching Overheads

Figure 8 shows simulation results in terms of caching overheads. In Figure 8, we vary the maximum speed of mobile nodes from 2 m/s to 16 m/s while keeping the mean update generation time and the mean update generation time. We observe in Figure 8 that DCCA outperforms POACH and NOCACHE constantly for all the maximum speed and the performance of DCCA is stable.

3) Query Success Ratio

Figure 9 shows simulation results in terms of query success ratio. We vary the maximum speed of mobile nodes from 2 m/s to 16 m/s while keeping the mean update generation time and the mean update generation time. Although the query success ratio is decreased with higher speed, we still observe that DCCA outperforms POACH and NOCACHE constantly for all the maximum speed.

VI. CONCLUSIONS

In this paper, we addressed the cache placement problem in wireless multi-hop ad hoc networks. We investigate the impact of contentions on the performance of data caching. To the best of our knowledge, this is the first time to consider the contribution of contentions to the total cost in caching system in wireless ad hoc networks. Since most of the previous work aim to minimize the total cost evaluated by the hops of messages, these strategies cannot achieve their expected performance in real applications in wireless multihop ad hoc networks. We introduce two heuristic rules to optimize the total cost that are described by the hop delay. The first rule aims to reduce the access traffic flows as many as possible and increase the update traffic flows as few as possible. The second rule intends to select cache nodes with fewer contentions from the candidates coming from the first heuristic rule. Based on the two heuristic rules, we present CCCA (Centralized Contention-aware Caching Algorithm) and DCCA (Distributed Contention-aware Caching Algorithm). In our simulations, we evaluate the proposed algorithms in static network and mobile networks respectively. We take caching overheads, average query delay, and success query ratio as our performance metrics. The results demonstrate that our algorithms outperform the previous work, particularly in more challenging scenarios, such as higher query frequency, higher update frequency and higher mobility.

ACKNOWLEDGMENT

This research is partially supported by the Hong Kong RGC under the GRF grant PolyU 5102/08E, the Hong Kong Polytechnic University under the grant 1-BB6C, and National Natural Science Foundation of China (Grant No. 60803137). The authors would like to thank Dr. Bin Tang for providing us with their simulation code and Mr. Chisheng Zhang for helping the simulation code at the MAC layer.

REFERENCES

[1] P. Baronti, P. Pillai, V. Chook, S. Chessa, A. Gotta, and Y. Hu, "Wireless Sensor Networks: A Survey on the State of the Art and the 802.15.4 and ZigBee standards," *Computer Communications*, 30(7): 1655-1695 (2007).

[2] Y. Yang and R. Kravets, "Achieving Delay Guarantees in Ad Hoc Networks through Dynamic Contention Window Adaptation," in *IEEE INFOCOM*, 2006.

[3] C. Swamy and A. Kumar, "Primal-dual Algorithms for Connected Facility Location Problems," In *5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX 2002)*, 2002.

[4] M. Charikar and S. Guha, "Improved Combinatorial Algorithms for the Facility Location and k-median Problems," In *FOCS*, 1999.

[5] K. Jain and V. Vazirani, "Approximation Algorithms for Metric Facility Location and *k*-median Problems using the Primal-dual Schema and Lagrangian Relaxation," *Journal of the ACM*, 48(2), 2001.

[6] B. Li, M. Golin, G. Ialiano, and X. Deng, "On the Optimal Placement of Web Proxies in the Internet," In *IEEE INFOCOM 1999*, March 1999.

[7] P. Nuggehalli, V. Srinivasan, and C. Chiasserini, "Energy-efficient Caching Strategies in Ad Hoc Wireless Networks," In *MobiHoc*, 2003.

[8] C. Swamy and A. Kumar, "Primal-dual Algorithms for Connected Facility Location Problems," In *Intl. Workshop on APPROX*, 2002.

[9] Friedrich Eisenbrand, Farizio Grandoni, Thomas Rothvob, Guido Schafer, "Approximating Connected Facility Location Problems via Random Facility Sampling and Core Detouring," In *Proc. 19th Annu. ACM-SIAM Symp. on Discrete Algorithms (SODA'08)*, San Francisco, California, 20-22.01.2008.

[10] Kevin Fall and Kannan Varadhan, "NS notes and documentation," in *The VINT Project, UC Berkely, LBL, USC/ISI, and Xerox PARC*, 1997.

[11] Broch, D. A. Maltz, D. B. Johnson, Y-C. Hu, and J. Jetcheva, "A Performance Comparison of Multi-hop Wireless Ad Hoc Network Routing Protocols," In *MOBICOM*, 1998.

[12] C. Perkins and P. Bhagwat, "Highly Dynamic Dsdv Routing for Mobile Computers," in *SIGCOMM*, 1994.

[13] M.S. Corson, J.P. Macker, G.H. Cirincione, "Internet-Based Mobile Ad Hoc Networking," in: IEEE Internet Computing, July−August 1999, pp. 63−70.

[14] P.Nuggehalli, V.Srinivasan, C. F. Chiasserini and R. R. Rao, "Efficient Cache Placement in Multi-hop Wireless Networks," in *ACM/IEEE Transactions on Networking*, Volume 14 , Issue 5 (October 2006).

[15] S. Bespamyatnikh, B. Bhattacharya, D. Kirkpatrick, and M. Segal, "Mobile facility location," in 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing & Communications, 2000.