Energy-Efficient Caching Strategies in Ad Hoc Wireless Networks*

Pavan Nuggehalli
ECE Department
University of California at San Diego
La Jolla, CA
pavan@cwc.ucsd.edu

Vikram Srinivasan
ECE Department
University of California at San Diego
La Jolla, CA
vikram@cwc.ucsd.edu

Carla-Fabiana Chiasserini
CERCOM-Dip. di Elettronica
Politecnico di Torino
Torino, Italy
chiasserini@polito.it

ABSTRACT

In this paper, we address the problem of energy-conscious cache placement in wireless ad hoc networks. We consider a network comprising a server with an interface to the wired network, and some nodes requiring access to the information stored at the server. In order to reduce access latency in such a communication environment, an effective strategy is caching the server information at some nodes distributed across the network. Caching, however, can considerably impact the system energy expenditure; for instance, disseminating information incurs additional energy burden. Since wireless devices have limited amounts of available energy, we need to design caching strategies that optimally trade-off between energy consumption and access latency. We pose our problem as an integer linear program. We show that this problem is the same as a special case of the connected facility location problem, which is known to be NP-hard. We devise a polynomial time algorithm which provides a sub-optimal solution. The proposed algorithm applies to any arbitrary network topology and can be implemented in a distributed and asynchronous manner. In the case of a tree topology, our algorithm gives the optimal solution. In the case of an arbitrary topology, it finds a feasible solution with an objective function value within a factor of 6 of the optimal value. This performance is very close to the best approximate solution known today, which is obtained in a centralized manner. We compare the performance of our algorithm against three candidate caching schemes, and show via extensive simulation that our algorithm consistently outperforms these alternative schemes.

*This work was partially funded by the Center for Wireless Communications, San Diego and CERCOM, Politecnico di Torino, Torino, Italy.

MobiHoc'03, June 1–3, 2003, Annapolis, Maryland, USA.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication

General Terms

Algorithms, Performance

Keywords

Ad hoc networks, caching

1. INTRODUCTION

In the last few years, there has been an explosive growth of interest in mobile computing, as well as in delivering World Wide Web content and streaming traffic to radio devices. There is a huge potential market for providing palmtops, laptops, and personal communication systems with access to airline schedules, weather forecasts, or location dependent information, just to name a few. Wireless ad hoc networks can be used to provide radio devices with such services, anywhere and anytime. Ad hoc networks enable users to spontaneously form a dynamic communication system. They allow users to access the services offered by the fixed network through multi-hop communications, without requiring infrastructure in the user proximity. However, in order to offer high quality and low cost services to ad hoc network nodes, several technical challenges still need to be addressed. First, wireless networks are plagued by scarcity of communication bandwidth, therefore a key issue is to satisfy user requests with minimum service delay. Second, since network nodes have limited energy resources, the energy expended for transferring information across the network has to be minimized.

In this work, we focus on caching as an attractive technique to efficiently meet these challenges. Two straightforward caching approaches are as follows. The first technique employs a centralized server to deliver information to the network nodes as and when they require it. The drawback is that the *access latency* experienced by the users, i.e., the time elapsed from the moment a user requests a data item until that item is received, may be considerable due to the multi-hop nature of information transfer. Moreover, both the server load and the network traffic increase with the increase in the number of requests. The second solution consists in placing a copy of the server content onto all net-

work nodes. The advantage of such an approach is that it minimizes access latency, besides decreasing the radio channel contention and the network traffic load [16, 4]. The drawback is that distributing the server content to all nodes implies a high energy cost for the nodes that transmit and receive such information. It is easy to see that, whenever access latency and energy cost of data transfer are high, the best approach is to cache the requested information at a limited number of nodes distributed across the network. Caching, in fact, allows us to optimally trade-off between access latency and system energy expenditure.

In our study, we address the problem of cache placement. We consider a wireless ad hoc network with V stationary nodes and assume that the network comprises a server where all the information needed by the users is originally stored. Such information changes every T units of time and needs to be updated. By anticipating that the user nodes will demand this information, the server has to determine at which network nodes caches have to be placed, in order to optimize system performance. The assumption of users being stationary is justified, at least in the context of wireless LANs, by large scale experiments such as [8]. To find an efficient caching strategy, we formulate the problem as follows. Distributing information to the network nodes implies a certain energy expenditure, which increases with the number of cached nodes. Every node is associated with a service demand, which must be served by a cache. A node receiving service from a cache incurs a cost that depends on its distance from the cache and on the node's demand. Such a cost takes into account both the access latency and the energy expenditure for the node to access the cache. The objective is to find the optimal cache placement which minimizes the total cost. As we will show later, the problem we have posed is equivalent to a special case of the *connected facility location problem* [10, 14], which is known to be NP-hard.

We propose a greedy algorithm, called *POware Aware Caching Heuristic (POACH)*, which provides a sub-optimal solution to the cache placement problem. POACH has the following desirable properties: (i) it is a polynomial time algorithm, (ii) it applies to any arbitrary network topology, and (iii) it can be implemented in a distributed and asynchronous fashion. The algorithm finds the optimal solution in the case of tree topologies, while it provides an approximate solution for an arbitrary topology. To evaluate the performance of the proposed algorithm, we derive a bound on its performance. We show that POACH always provides a solution that is within a factor of 6 of the optimal solution. This bound is very close to the best approximation of 5 known today, which requires a centralized implementation [14]. Then, we compare the behavior of POACH against three simple caching strategies, namely, *no-caching*, *depth caching*, and *flooding*. Numerical results obtained through extensive simulations show that POACH significantly outperforms these three alternative schemes.

The rest of the paper is organized as follows. Section 2 reviews previous work on caching strategies for wired as well as wireless networks. In Section 3, we present the network model considered in this study and formulate the cache placement problem as a linear programming problem. In Section 4, we describe our distributed greedy algorithm, the so-called POACH. Through examples, we compare POACH to the optimal solution in Section 5, and derive a bound on its performance in Section 6. Section 7 discusses the features of the proposed scheme, while Section 8 describes our simulation setup and presents some numerical results. Finally, Section 9 concludes the paper and points to some aspects that will be the subject of future research.

2. RELATED WORK

The problem of data replication and caching has been widely studied in the context of wired networks. In [9], the authors address the problem of proxies placement and employ dynamic programming to determine the optimal placement. They consider, however, the case of networks with a tree topology only. A similar approach is used in [3]. This work describes a distributed algorithm which solves the allocation of electronic content over a distribution tree, when storage cost is considered. A solution to the placement problem, which minimizes the data transfer cost for tree topologies, is also proposed in [19]. More general network topologies are considered in [12], where algorithms for Web servers placement are presented. The problem addressed there is, however, significantly different from ours. The goal of [12] is to minimize the cost for clients to access copies of the server information, thus it neglects the cost of distributing cache copies to the network nodes. Moreover, the maximum number of copies that can be created is restricted to a fixed value.

In the context of wireless networks, a cooperative cache management scheme for a cellular environment is proposed in [18]. The authors present a simulation study of the problem of cache replication at the base stations, for the case of streaming services. An ad hoc network scenario is considered in [13]. The focus there is on a distributed application software, which implements cooperative Web caching among mobile terminals. Strategies for cache management are also studied with the objective of minimizing energy consumption and network load. [7] analyzes the performance of various data allocation methods, by focusing on the communication cost between a mobile computer and a stationary computer storing all data.

The work closest to ours is presented in [14], which proposes a centralized approximation algorithm to solve the connected facilities location problem. This can be seen as a more general formulation of our cache placement problem. Our work differs from [14] in specifically modeling an ad hoc wireless network, and, even more importantly, in proposing a distributed algorithm.

3. SYSTEM MODEL

Let $G = (\mathcal{V}, \mathcal{E})$ be a connected graph representing a multihop ad hoc network with V ($V = |\mathcal{V}|$) nodes connected by E ($E = |\mathcal{E}|$) links. Assume that the network has a static topology and one of the nodes is the network server with an interface to the wired network. Then, suppose that there is some information (I) which is changing every T units of time. The server node can disseminate I to the other nodes via multi-hop routes. During a T unit interval, a node k desires I with probability p_k, $1 \leq k \leq V$. In order to reduce user access latency, at the beginning of a T unit interval, the server caches I at a few nodes in the network. We call this the *dissemination phase*. We define a caching strategy by the vector $Z = (z_1, z_2, \ldots, z_E)$, where $z_e = 1$ if a copy of I is transmitted on edge $e \in \mathcal{E}$ during the *dissemination phase*, and $z_e = 0$ otherwise. At the end of the *dissemination phase*,

nodes desiring I access the cache location corresponding to the minimum *access cost*. We call this the *access phase*.

3.1 Model Assumptions

1. Links are bidirectional.
2. The server always possesses a cached copy of I.
3. The energy required to transmit and receive I along any link is constant and is equal to 1.
4. We model the access latency cost for a node k ($k = 1, \ldots V$) by the minimum number of hops required to reach a cached copy of I.
5. We assume that storage costs are insignificant.

Assumption 3 is motivated as follows. Our assumption of stationary users implies that fast fading phenomena due to mobility can be ignored. This in turn allows us to assume that the wireless channel is time-invariant. Moreover, in existing ad hoc wireless systems such as IEEE 802.11b WLAN and IEEE 802.15 (Bluetooth) WPAN, there is no provision for dynamic power control [2, 1]. In other words, power levels cannot be changed on the fly. Also, once a transmitter gains access to the channel, the CSMA/CA MAC protocol in IEEE 802.11b ensures that there is almost no interference. Assumption 5 implies that nodes transmitting a copy of I during the dissemination phase retain I in their buffers. Thus, the terminal nodes of any edge e over which I is transferred (i.e., with $z_e = 1$) posses a copy of I. It follows that the subgraph induced by Z is a *connected* subgraph containing the server. Assumption 5 is motivated by the low cost of memory.

3.2 Calculation of Costs

Let $\mathcal{C}$ denote the set of cached nodes and $d_k, k = 1, \ldots V$, be the minimum distance in hops from any node k to $\mathcal{C}$. Since the energy required to transfer I over each link is the same, the energy cost incurred during the *dissemination phase* is given by $K_{diss-energy} = \sum_{e \in \mathcal{E}} z_e$. The cost incurred during the *access phase* includes: (i) the energy cost for the user nodes to retrieve I from a cache, and (ii) the access latency that the user experiences. For a strategy Z, the expected access energy cost, $K_{acc-energy}$, as well as the expected access latency cost, $K_{latency}$, are given by $\sum_{k \in \mathcal{V}} p_k d_k$. Therefore, the average energy expended and the average access latency experienced by the network system are given by

$$K_{energy} = K_{diss-energy} + K_{acc-energy} = \sum_{e \in \mathcal{E}} z_e + \sum_{k \in \mathcal{V}} p_k d_k \quad (1)$$

$$K_{latency} = \sum_{k \in \mathcal{V}} p_k d_k \quad (2)$$

As an example, assume that in the *dissemination phase* the server caches I at all the nodes in the network. Then, during the *access phase*, each node will experience zero access latency and zero energy consumption. However, the network ends up wasting valuable energy in relaying I to those nodes who do not desire I in the *access phase* of a particular interval. On the other hand, if, in the *dissemination phase*, the server decides not to cache I in any of the other nodes in the network, access latency and energy costs will be substantial in the *access phase*.

We define the total cost as

$$K_{tot} = K_{energy} + \lambda K_{latency} = \sum_{e \in \mathcal{E}} z_e + (1 + \lambda) \sum_{k \in \mathcal{V}} p_k d_k, \quad (3)$$

with λ indicating the relative importance of access latency and energy consumption. Notice that, by varying λ, we can model different network scenarios and quality of service requirements.

Our objective is to find a strategy Z which minimizes the total cost, thus providing the optimal trade-off between the average energy expenditure and the average access latency.

3.3 Problem Formulation

Here, we provide an integer linear programming (LP) formulation for the problem described above. Let c_{ke} be the minimum distance in hops from a node k to an edge e. In other words, c_{ke} is the minimum of the shortest path distance between node k and the terminal nodes of edge e. Observe that c_{ke} also represents the access latency cost for node k to access I from the nearest terminal node of edge e. For ease of formulation, we augment graph G by attaching a virtual node v to the server via an edge e_v, and, without loss of generality, we refer to the server node as node 1. By virtue of Assumption 2, we have: $z_{e_v} = 1$, for any strategy Z.

Consider the following LP formulation.

$$\mathbf{P}: \quad \text{Minimize} \quad \sum_{k \in \mathcal{V}} \sum_{e \in \mathcal{E}} p_k c_{ke} x_{ke} + M \sum_{e \in \mathcal{E}} z_e \quad (4)$$

$$\text{s.t.} \quad \sum_{e \in \mathcal{E}} x_{ke} \geq 1 \quad \forall k \in \mathcal{V} \quad (5)$$

$$z_e - x_{ke} \geq 0 \quad \forall k \in \mathcal{V}, e \in \mathcal{E} \quad (6)$$

$$\sum_{e \in \delta(\mathcal{S})} z_e - \sum_{e \in \mathcal{S}} x_{ke} \geq 0 \quad \forall \mathcal{S} \subseteq \mathcal{E}, e_v \notin \mathcal{S}, \forall k \in \mathcal{V} \quad (7)$$

$$x_{ke}, z_e \in \{0, 1\} \quad \forall k \in \mathcal{V}, e \in \mathcal{E} \quad (8)$$

where: $M = 1/(\lambda+1)$; $\{x_{ke}\}$ is a set of assignment variables such that $x_{ke} = 1$ if node k accesses information I from the nearest terminal node of edge e, else $x_{ke} = 0$; $z_e = 1$ if I is disseminated along edge e. From (3), it is easy to see that solving (4) is equivalent to minimizing K_{tot}. Constraint (5) ensures that each node is assigned to at least one cache location from which it can access I. Constraint (6) ensures that each node is assigned to an edge whose terminal nodes possess I. Constraint (7) is a connectivity constraint on the subgraph induced by strategy Z. In particular, $\delta(\mathcal{S})$ represents the cut-set of $\mathcal{S}$ (set of edges adjacent to edges in $\mathcal{S}$ but not belonging to $\mathcal{S}$). If $\sum_{e \in \mathcal{E}} x_{ke} = 1$ for any $k \in \mathcal{V}$, then each node is assigned to exactly one edge, and constraint (7) ensures connectivity of the subgraph induced by Z. If, however, a node is assigned to multiple edges, then constraint (7) ensures that each of these edges has an edge disjoint path to e_v on the subgraph induced by Z. Note that in the optimal solution, each node k will be assigned to only one edge e; otherwise, the objective function in (4) could be further optimized.

As already mentioned in Section 1, our optimization problem is a special case of the *connected facility location problem*, called the *rent-or-buy* problem, whose formulation is as follows [14]. An existing facility is given, along with a set

of locations at which further facilities can be built. Every location k 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 connection cost of a location hosting a facility is zero. Besides selecting the sites to build the facilities, we also want to connect them by a Steiner Tree [17]. Connecting the facilities incurs a cost which is proportional to the weight of the Steiner Tree. The objective is to find a solution (i.e., to select the locations to build facilities and connect them by a Steiner Tree) which minimizes the total cost. It is easy to see that, in our setting, the facility that is already available represents the server node, new facilities correspond to caches, while locations correspond to the network nodes at which caches (i.e., facilities) can be created.

The rent-or-buy problem is NP-hard [14], however several approximation algorithms have been developed. A ρ- approximation algorithm is defined as a polynomial time algorithm that always finds a feasible solution with an objective function value within a factor of ρ of the optimal cost. The best approximation algorithm known today for the rent-or-buy problem was developed by Swamy *et al.* [14], who gave a 5-approximation algorithm. The solution in [14] requires a centralized implementation, therefore it is not suitable for an ad hoc wireless environment. In the following, we design a polynomial-time algorithm, that approximates the optimal solution for any arbitrary graph within a factor of 6. Our algorithm allows for a distributed and asynchronous implementation.

4. A GREEDY SOLUTION

We propose a greedy algorithm, called POACH, which provides an approximate solution to problem **P** in (4). POACH decides which nodes should cache I during the dissemination phase, and is such that it can be implemented in a distributed and asynchronous manner. To motivate our scheme, we first consider the special case of tree topologies, then we describe the algorithm for the general case of connected graphs.

4.1 A Special Case: Tree

For a node k, let $f(k)$ be the number of progeny of k (i.e., all descendent nodes of k) given by $s_1, s_2, \ldots, s_{f(k)}$. Let s_0 be the node k itself. Clearly $f(1) = V-1$, since node 1 is the server. Suppose node k's parent node (say m_k) is a cached node. If m_k disseminates I to k, it will incur an additional energy cost of 1 unit (see Assumption 3). However k and its progeny will experience a decrease in access latency and energy cost given by $(1+\lambda)\sum_{j=0}^{f(k)} p_{s_j}$. Therefore, m_k will disseminate I to k if

$$(1+\lambda)\sum_{j=0}^{f(k)} p_{s_j} \quad \geq \quad 1 \tag{9}$$

i.e.,

$$\sum_{j=0}^{f(k)} p_{s_j} \quad \geq \quad M. \tag{10}$$

This dissemination strategy can be implemented in polynomial time and, in the special case of a tree topology, it can be easily shown by construction to be optimal. In general, however, the optimal solution is not unique. When $\sum_{j=0}^{f(k)} p_{s_j} = M$, the cost of the solution remains the same, irrespective of m_k's decision.

Assume that I is cached at node k. Let $c_1, c_2, \ldots c_l$ be the children of node k. Since the graph is a tree, the progeny of c_i, $1 \leq i \leq l$ are disjoint sets. Thus, the decision to cache at node c_i can be made independently of the decision with respect to node c_j, $i \neq j$. The disjointedness of progeny is a desirable property, since it lends itself to an asynchronous and distributed implementation of our algorithm; however, this property does not hold for a general graph. In the next section, we extend the algorithm described here for a tree topology to the case of networks with a generic topology, in such a way that the progeny sets are disjoint.

4.2 The General Case

We now describe the dissemination algorithm in the case of a network with a generic topology.

For ease of description, we divide the algorithm into stages. We say that an edge is open if its terminal nodes are chosen as caches. Initially, all edges are closed, except for e_v. An edge e is said to be a neighbor of an edge l (denoted by $e \perp l$) if e and l share a common node. Let $\mathcal{O}$ be the set of opened edges. Let $\mathcal{N}_e$ be the set of neighbor edges of edge e. At any stage n, let $\mathcal{B}_e$ be the set of neighbor edges of e, which have not been opened yet ($\mathcal{B}_e \subseteq \mathcal{N}_e$).

We associate a variable α_k with each node. At each stage n, α_k can be viewed as the price that node k is willing to pay to any available cache to access I. We say that node k is *tight* with an edge e if $\alpha_k \geq c_{ke} p_k$. When a node goes tight with a cache, it implies that the price the node pays compensates for the cost incurred by the cache in disseminating I to the node. A node can be in two states: *frozen* and *unfrozen*. We say that a node k is frozen when we stop increasing the associated variable α_k. When a node is frozen, it cannot become tight with any new edge.

For each edge e, we define $\mathcal{R}_e$ as the set of nodes, who belong to the progeny of the terminal nodes of edge e, and receive their copy of I either through e or one of its descendent edges. Note that $\mathcal{R}_{e_v}=\mathcal{V}\backslash\{1\}$. Let $\mathcal{F}_n$ be the set of edges opened at stage n. Let $\mathcal{T}_e$ be the set of nodes tight with edge e. Then the demand, D_e, for edge e is given by $\sum_{k\in\mathcal{T}_e} p_k$.

The algorithm is as follows.

Step 1. Initialize all variables: Set $n=0$; $\mathcal{O}=\{e_v\}$; $\mathcal{B}_{e_v}=\mathcal{N}_{e_v}$; $\alpha_k = 0, \forall k \in \mathcal{V}$; $\mathcal{R}_{e_v}=\mathcal{V}\backslash\{1\}$; $\mathcal{F}_{-1} = \{e_v\}$.
The algorithm begins by opening edge e_v, and initializing $\mathcal{B}_{e_v}$ to the set of neighbor edges of the server node, i.e., the set of edges along which I can be disseminated.

Step 2. $\mathcal{F}_n = \emptyset$, $\alpha_k = 0, \forall k \in \mathcal{V}$.
For any $e \in \mathcal{F}_{n-1}$,

Step 2.a. *During this step, we shall tentatively open some edges belonging to $\mathcal{B}_e$, and freeze all progeny of edge e.*
While all nodes in $\mathcal{R}_e, e \in \mathcal{F}_{n-1}$, are not frozen

1. Freeze node $k \in \mathcal{R}_e$ if it is tight with edge e or with a tentatively open edge $l \in \mathcal{B}_e$.

2. For any edge $l \in \mathcal{B}_e$, if $D_l \geq M$, then declare l to be tentatively open.

An edge, which is not already open and has

at least M units of demand, is tentatively opened. This is because, as highlighted in (10), if I is disseminated along this tentatively open edge, then the decrease in access cost outweighs the increase in dissemination cost.

3. Define $\tilde{\mathcal{R}}_e = \{k : k \in \mathcal{R}_e, k \text{ not frozen}\}$
Increase α_k for all nodes in $\tilde{\mathcal{R}}_e$ by

$\min_{\{k \in \tilde{\mathcal{R}}_e, l \in \mathcal{B}_e \cup \{e\}\}} c_{kl} p_k - \alpha_k$.
Variables α_k's are raised by the minimum amount which is sufficient to let some node $k \in \tilde{\mathcal{R}}_e$ become tight with some edge $l \in \mathcal{B}_e \cup \{e\}$. Notice that many nodes may become tight at the same time.

End While

Step 2.b. Let $\mathcal{L}_e$ be the set of tentatively open edges in $\mathcal{B}_e$.
If $\mathcal{L}_e = \emptyset$
Stop dissemination beyond edge e.
Else
Let $\mathcal{H}_l, l \in \mathcal{L}_e$ be the set of nodes tight with tentatively open edge l.
Let $\mathcal{H}$ be the collection of sets $\{\mathcal{H}_l : l \in \mathcal{L}_e\}$.
At this step, a further optimization is achieved as follows. Suppose l_1 and l_2 are two tentatively open edges originating from the same parent edge. Suppose that edge l_1 has n_1 nodes tight with it and node l_2 has $n_2 < n_1$ nodes tight with it. Also let the number of nodes which are tight with both l_1 and l_2 be n_c. If we disseminate to edge l_1, then the number of additional nodes which edge l_2 can serve is $n_2 - n_c$. If, the demand created by these $n_2 - n_c$ nodes is less than M, then disseminating to edge l_2 is sub-optimal. The operations described in the following avoid such redundant edges from being opened.
While $\mathcal{H} \neq \emptyset$
$l^* = \arg\max_{l \in \mathcal{L}_e} \sum_{k \in \mathcal{H}_l} p_k$
$\mathcal{Q}_{l^*} = \mathcal{H}_{l^*}$
For each $\mathcal{H}_l \in \mathcal{H}$
$\mathcal{H}_l = \mathcal{H}_l \backslash \mathcal{H}_{l^*}$
If $\sum_{k \in \mathcal{H}_l} p_k < M$
$\mathcal{H} = \mathcal{H} \backslash \mathcal{H}_l$
End If
End For
End While Let $\mathcal{A} = \{l : l \in \mathcal{L}_e, \sum_{k \in \mathcal{Q}_l} p_k \geq M\}$. $\mathcal{O} = \mathcal{O} \bigcup \mathcal{A}$, $\mathcal{F}_n = \mathcal{F}_n \bigcup \mathcal{A}$, $\mathcal{R}_l = \mathcal{Q}_l$.
End If

End For
For any $e \in \mathcal{F}_n$, $\mathcal{B}_e = \{l : l \in \mathcal{N}_e, l \not\perp l' \; \forall l' \in \mathcal{O} \backslash \{e\}\}$
$\mathcal{B}_e$ is updated with all neighbor edges of e which are not already opened and are not neighbors of any open edge, other than e.

Step 3. If $\mathcal{F}_n \neq \emptyset$, $n = n + 1$ and go to Step 2.
Otherwise, terminate algorithm.

We observe that the above algorithm allows us to create disjoint progeny sets. In fact, at the end of each stage n, we have that: $\mathcal{R}_e \bigcap \mathcal{R}_{e'} = \emptyset$, $\forall\, e, e' \in \mathcal{F}_n$.

5. COMPARING THE PERFORMANCE OF POACH WITH THE OPTIMAL SOLUTION

Here, we would like to compare the performance of the proposed algorithm with the optimal solution. Since the optimization problem we have posed is NP-hard, we compute the optimal solution by using the following brute force approach. We list all spanning trees of the graph, and, for each spanning tree, we derive the optimal cache placement using (10). Finally, we choose the spanning tree with the least cost. Clearly, we need to consider simple topologies so that the computation of the optimal solution does not become exceedingly cumbersome. In the following, two examples are reported.

Example 1. Consider the topology in Fig. 1, with $\lambda = 1$, $p_k = 0.25$, $\forall k \in \mathcal{V}$, resulting in $M = 0.5$. At the beginning of stage 0, only the virtual edge, e_1 is open (*only the server has information I*), as shown by the left most diagram in Fig. 1. The progeny list for the server includes all the nodes in the network. Edges e_2, e_3 and e_4 are the edges belonging to $\mathcal{B}_{e_1}$ (i.e., nodes 2, 3 and 4 are potential cache locations). Also, initially we have that all $\alpha_k = 0$, for $k = 1, \ldots 7$. Node 1 becomes frozen with $\alpha_1 = 0$ since it is tight with edge e_1. Since all access probabilities are the same and equal to 0.25, α_k's are increased by 0.25 (see Step 2.a.3). When α_k becomes equal to 0.25, for $k = 2, \ldots 7$, nodes 2, 3 and 4 get frozen with edge e_1, node 5 becomes tight with edge e_2, node 6 becomes tight with edge e_3, and node 7 becomes tight with edge e_4. Since all the edges that could be opened have demand less than $M = 0.5$, none of them are tentatively opened. In the next step, $\alpha_k = 0.5$, for $k = \{5, 6, 7\}$. As a result, nodes 5 and 6 become tight with edge e_2, nodes 5, 6 and 7 become tight with edge e_3 and nodes 6 and 7 become tight with edge e_4. Since all the edges that could be opened have demand of at least $M = 0.5$, they are all tentatively opened and all nodes in the network get frozen. It is easy to see that only edge e_3 will be opened in Step 2.b.

At the beginning of stage 1, edges e_1 and e_3 are open, as shown in the middle diagram in Fig. 1. The progeny list for edge e_3 are nodes 5, 6 and 7. The edges that can be opened are e_5, e_6 and e_8. It is easy to see that of these edges only edge e_8 will be opened. The algorithm then terminates with the cache locations as shown by the right most diagram in Fig. 1.

In this example, it turned out that this cache placement is also the optimal one. However, if $\lambda = 0$, POACH results in the cache location shown in the middle diagram in Fig. 1, while the optimal strategy would be not to cache at all.

Example 2. Here we aim at showing the computational savings gained by using our algorithm. We consider the topology shown in Fig. 2. As before, we assume $\lambda = 1$, $p_k = 0.25$, $\forall k \in \mathcal{V}$, thus resulting in $M = 0.5$. In this case, it can be seen that POACH achieves the optimal cache placement with a computation effort that is polynomial in the size of the graph. While, to compute the optimal solution for the same topology, we need to consider 135 spanning trees. Clearly, for larger networks, the number of spanning trees becomes astronomically large.

6. ANALYSIS

In this section, we provide a bound on performance for

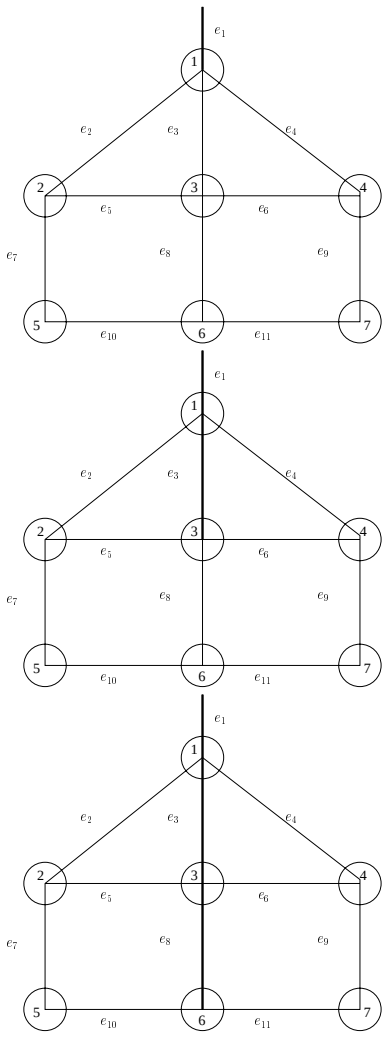

Figure 1: Topology for network 1. The thick lines indicate open edges. (a) Stage 1: In the beginning only the virtual edge is open. (b) Stage 2: Edges e_1 and e_3 are open. (b) Stage 3: Edges e_1, e_3 and e_8 are open.

the POACH algorithm. In Section 4.1, we noticed that the POACH algorithm achieves the optimal value for trees. Now, we show that, for an arbitrary topology, our algorithm approximates the optimal solution within a factor of 6.

Recall from Section 3.3 that we obtain a caching strategy Z by setting $z_e = 1$ for all the open edges. We can prove the following property.

LEMMA 1. *The graph induced by the strategy Z, obtained from the POACH algorithm, is a tree.*

PROOF. By induction. At any stage n of the algorithm, consider the set of edges which are opened at the end of the stage. The only way a cycle can be formed is if a tentatively opened edge forms a cycle with already opened edges, or two tentatively opened edges which are neighbors are opened simultaneously. The former can never happen since Step 2 ensures so. The latter also is infeasible since the sets of progeny are disjoint. □

We have the following theorem.

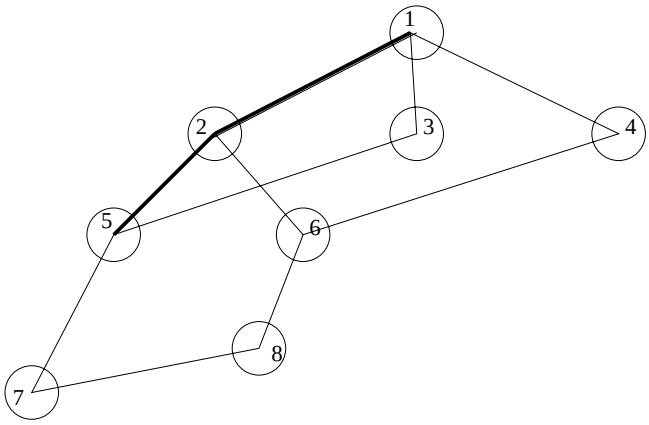

Figure 2: Topology for network 2. The thick lines indicate open edges.

THEOREM 1. *Let OPT be the optimum value of the primal integer problem $\mathbf{P}$ in (4). The solution obtained by the algorithm is at most $6 \times OPT$.*

PROOF. See the Appendix. □

This bound is very close to the best bound in the literature [14] for the optimization problem that we have posed. [14] gives a 5-approximation by using a centralized approach, while our algorithm can be implemented in a distributed and asynchronous fashion.

7. IMPLEMENTATION OF POACH

In order to implement POACH, each node needs to maintain some progeny and distance information. This kind of information is readily available from current routing mechanisms [11]. We show in this section that the POACH algorithm can be implemented for a network with arbitrary topology in a distributed and asynchronous manner.

Consider the set of open edges at the end of stage n. For simplicity, let e_1 and e_2 be the only two edges which were opened in stage n. Let G_1 and G_2 be the subgraph of G induced by the progeny sets $\mathcal{R}_{e_1}$ and $\mathcal{R}_{e_2}$, respectively. In subsequent stages, caches are opened on edges belonging to G_1 and G_2. Opened edges in G_1 (G_2) will inherit progeny which are subsets of $\mathcal{R}_{e_1}$ ($\mathcal{R}_{e_2}$). From the algorithm description, we know that the progeny of e_1 and e_2 are disjoint. This implies that, caching decisions in the subgraphs, G_1 and G_2, are independent. The disjointedness of progeny therefore allows for a distributed implementation of POACH. This argument easily extends to the case when multiple edges are opened in a particular stage.

We now show that the POACH algorithm can be implemented without the need for any synchronization. Consider the situation when a cached node k has to decide to which of its neighbor nodes should I be disseminated. If each of k's progeny communicates its α value during every stage of the dissemination algorithm, then the number of messages exchanged becomes very large resulting in a large overhead. Moreover, if nodes update their α's asynchronously, then some errors may occur. For example, it could happen that a node still increases its α after being frozen with an edge. In this case, some edges would be erroneously opened. To avoid such problems, we propose the following scheme. We

know that any cached node k has knowledge of its immediate neighbors, as well as the progeny set that it serves. Node k multicasts to its progeny the set of potential cache locations. The progeny, in turn, respond with the shortest path distance to each of the potential cache locations that can be opened, as well as their access probabilities. Node k can then locally run the algorithm to determine which of the possible cache locations should be opened. This avoids all timing issues and obviates the need for synchronization in the network. We stress that by distributed we refer to the fact that each node takes independent decisions to propagate the cache. POACH does need global topological information and disseminating such information in a bandwidth and energy efficient manner is a challenging problem. However, efficient means of route discovery have been devised [11] which are beyond the scope of our work. Also, this routing burden is imposed only when the topology changes, not every time the information is updated.

8. NUMERICAL RESULTS

In this section, we present some results showing the sensitivity of the placement algorithm to the system parameters introduced in the previous sections. The performance of the proposed algorithm is compared with the results obtained using three simple schemes, namely *no-caching (NC)*, *depth caching (DC)*, and *flooding (FLD)*. In the NC scheme, I is stored at the server only. The DC scheme corresponds to disseminating I to all nodes that are up to h hops away from the server. Parameter h is chosen heuristically as a function of the access probabilities and λ: as λ increases, so does h. Finally, in the FLD scheme, I is disseminated to all the nodes in the network.

We construct random topologies by choosing $V = 30$ random points in a square of unit area. Each node is assumed to have a communication range of D_{max}, which is taken as a varying parameter of the system. Given the set of random points and D_{max}, one can construct the graph $G = (\mathcal{V}, \mathcal{E})$, where two nodes are connected by an edge if their distance is less than D_{max}. We ensure that the resulting graph G is connected. If not, we generate another topology until we get a connected graph. One of the nodes is arbitrarily identified as the server. We consider two scenarios: one with uniform access probabilities (p_k's), the other with non-uniform access probabilities.

We first compare the performance of POACH versus NC, FLD and DC by fixing λ at 1, D_{max} at 0.3, and the value of the access probabilities. The comparison is obtained by averaging the results over 30 different topologies. The performance is derived in terms of average energy expenditure, access latency, and total cost, as defined in (1)-(3). Table 1 shows the results for the case where all access probabilities are the same and equal to 1/6, while Table 2 presents the results when we have non-uniform access probabilities. Non-uniform access probabilities are set as follows; we divide the 30 nodes into three groups of equal size, and assign to each of them a different value of access probability (namely, 1/4, 1/6, and 1/9).

As expected, the results presented in the two tables show that the highest energy expenditure is obtained with the FLD scheme, while the highest access latency is given by the NC scheme. However, it is interesting to notice that the FLD and the NC scheme have almost the same total cost in both the scenarios. Also, observe that the energy expenditure for the NC scheme is greater than zero, since we have to take into account the energy cost for the nodes to access I from the server. The DC algorithm has access latency performance close to POACH, but it implies a much higher energy consumption. Finally, compared to the other three schemes, POACH allows for a reduction in the total cost ranging between 36% to 43%.

Table 1: Comparison of caching strategies averaged over 30 runs, for V=30, D_{max}=0.3, uniform access probabilities equal to 1/6, and λ=1.

	Energy expenditure	Access latency	Total cost
NC	13.5667	13.5667	27.1333
FLD	29	0	29
DC	20.2833	6.9167	27
POACH	11.9889	5.3222	17.311

Table 2: Comparison of caching strategies averaged over 30 runs, for V=30, D_{max}=0.3, access probabilities equal to: 1/4, 1/6, 1/9, and λ=1.

	Energy expenditure	Access latency	Total cost
NC	14.0741	14.0741	28.1482
FLD	29	0	29
DC	25.7787	5.8120	31.5907
POACH	12.2787	5.6787	17.9574

Next, we investigate the impact of λ on performance, when D_{max} is equal to 0.3 and the access probabilities are all the same and equal to 1/6. Fig.s 3, 4 and 5 show the values of access latency, energy expenditure and total cost as λ varies, for a fixed network topology. Looking at the plots, we see that POACH consistently outperforms the other three caching strategies. (The only exception is for the FLD scheme, which always gives zero access latency as shown in Fig. 3.) For instance, for $\lambda = 1$, POACH gives a reduction in the total cost of 38% with respect to DC and of 50% relative to NC. Also, it is interesting to notice that the access latency and energy plots behave like step functions for POACH and DC. This is because, for both the schemes, the caching strategy changes only at certain break points.

Finally, we investigate the cost as D_{max} changes. We consider a fixed topology and assume that the output transmit power increases as D_{max}^{γ}, where γ is a path loss exponent between 2 and 4. We highlight that, as before, power control is not considered: for a fixed value of D_{max}, all nodes use the same power level, proportional to D_{max}^{γ}. As D_{max} increases, the graph becomes more connected, decreasing access latency but increasing the dissemination energy and access energy costs. The results are shown in Fig. 6. As we can see, the optimal operational point (i.e., optimal value of output transmit power) lies somewhere between the minimum power to ensure connectivity and the maximum power. Notice that the curve is not a smooth function. This is because the caching strategy changes only at certain discrete power points. In between these points, the caching strategy remains the same, but since the power is increasing, the energy cost increases while the access latency cost remains the same, thereby increasing the total cost.

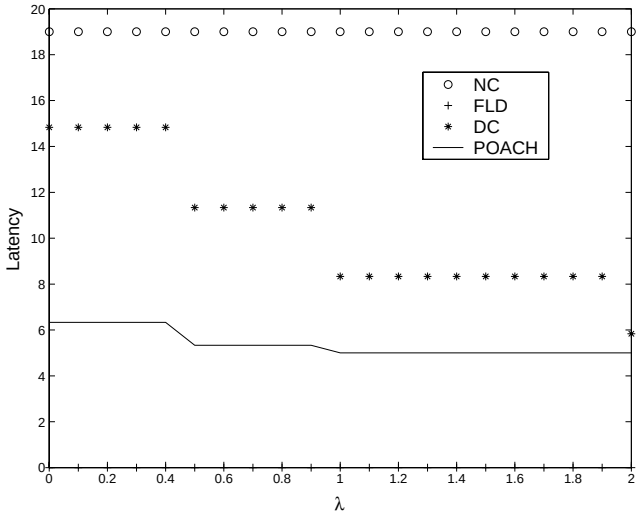

Figure 3: Access latency vs. λ, for a fixed topology, V=30, D_{max}=0.3, and uniform access probability equal to 1/6.

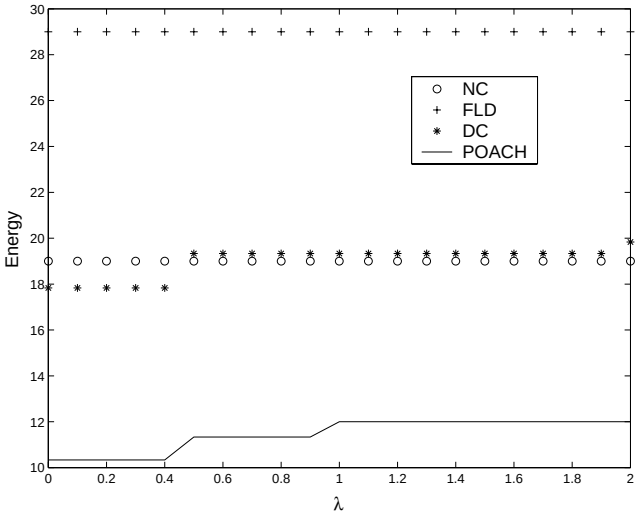

Figure 4: Energy expenditure vs. λ, for a fixed topology, V=30, D_{max}=0.3, and uniform access probability equal to 1/6.

9. CONCLUSIONS AND DISCUSSION

We addressed the problem of optimal cache placement in ad hoc wireless networks. We considered energy consumption and access latency as performance indices. In particular, we defined the system energy consumption to include both the energy expenditure in distributing information to various nodes across the network, and the energy expenditure for nodes to access the desired information from the nearest cache. To minimize the weighted sum of energy cost and access latency, we formulated an integer linear programming problem. This turned out to be an NP-hard problem. Thus, we proposed a polynomial time algorithm, which determines at which nodes to place caches so that performance is maximized. Our algorithm has the desirable property that it can be implemented in a distributed and asynchronous fashion, and it applies to any arbitrary network topology. It provides an optimal solution in the case of tree topologies, and an approximate solution in the case of an arbitrary topology. Nevertheless, the bound that we derived on the algorithm performance shows that we always obtain a solution within a factor of 6 of the optimal solution. This is very close to the best approximation known today for the cache placement problem, which can be achieved by employing a centralized approach. Numerical results show that, when compared against three candidate caching schemes, our algorithm significantly outperforms these alternative caching strategies.

We would also like to stress that our problem formulation is general enough to include various notions of cost. The cost of each link could account for both for energy cost and bandwidth cost. In our work we focused on a stationary wireless network, however future research work shall explore the impact of user mobility on caching strategies and performance. In [6], Grossglauser and Tse found that mobility leads to significant improvements in the performance of ad hoc networks, when packet delivery delays of a few minutes to hours are acceptable. This result however does not directly apply to our system scenario, and a careful study is needed to explore the possible benefits of mobility. The effect of mobility is a function of both the update interval T and the user mobility pattern. If the mobility pattern is predictable, then caching may be useful. Clearly, if topology changes occur at a much slower rate compared to T, then mobility will be of little use. On the other hand, if user mobility is unpredictable, it is not clear how caching will impact performance.

In our paper, we have assumed that the delay is linear in the number of hops. Research [5] has shown that delay increases exponentially with hop length. We have assumed a linear relationship as a first approximation. Incorporating more realistic models of delay is a subject of future research. Another angle to pursue would be to compare *reactive* dissemination schemes with the pro-active caching scheme that we have proposed.

10. ADDITIONAL AUTHORS

Additional authors: Ramesh R. Rao (ECE Department, U.C. San Diego, email: `rao@cwc.ucsd.edu`).

11. REFERENCES

[1] *Bluetooth core specification.* http://www.bluetooth.com/dev/specifications.asp.

[2] *Local and metropolitan area networks: Wireless LAN.* ANSI/IEEE Standard 802.11, 1999.

[3] I. Cidon, S. Kutten, and R. Soffer. Optimal allocation of electronic content. In *IEEE INFOCOM 2001*, April 2001.

[4] B. D. Davison. A web caching primer. *IEEE Internet Computing*, 4(4):38–45, July-August 2001.

[5] M. Gerla, R. Bagrodia, L. Zhang, K. Tang, and L. Wang. Tcp over wireless multihop protocols: Simulation and experiments. In *IEEE ICC 1999*, June 1999.

[6] M. Grossglauser and D. N. C. Tse. Mobility increases the capacity of ad-hoc wireless networks. In *IEEE INFOCOM 2001*, pages 1360–1369, 2001.

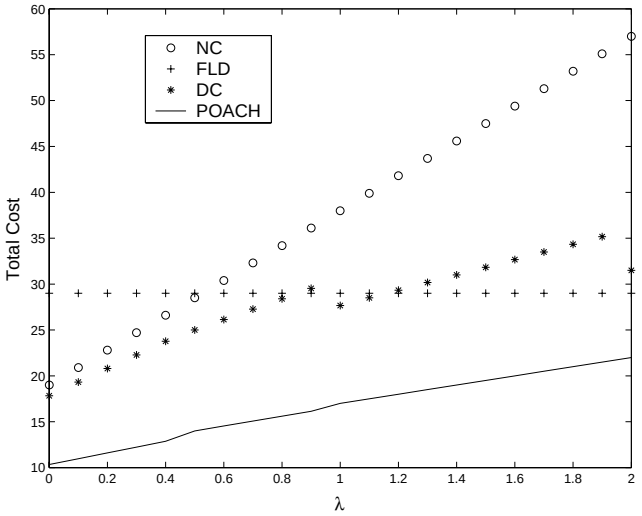

Figure 5: Total cost vs. λ, for a fixed topology, V=30, and uniform access probability equal to 1/6.

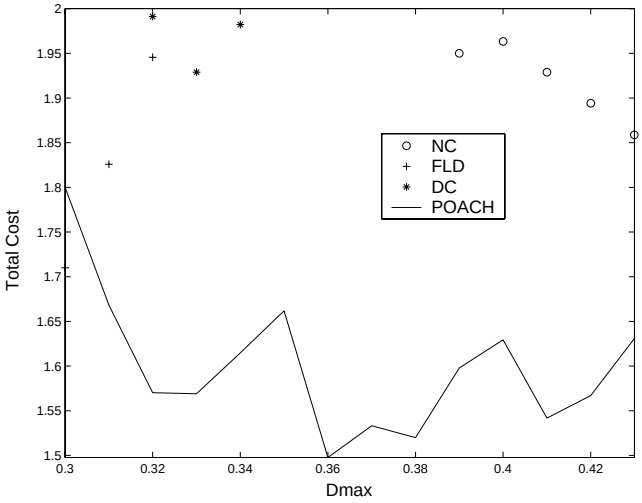

Figure 6: Total cost vs. the nodes radio range (D_{max}), for a fixed topology, V=30, uniform access probability equal to 1/6, and λ=1.

[7] Y. Huang, P. Sistla, and O. Wolfson. Data replication for mobile computers. In *International Conference on Management of Data, ACM SIGMOD*, pages 13–24, 1994.

[8] D. Kotz and K. Essien. Analysis of a campus-wide wireless network. In *ACM MOBICOM 2002*, September 2002.

[9] B. Li, M. J. Golin, G. F. Ialiano, and X. Deng. On the optimal placement of web proxies in the internet. In *IEEE INFOCOM 1999*, March 1999.

[10] P. Mirchandani and R. Francis. *Discrete location theory*. John Wiley and Sons, New York City, NY, 1990.

[11] C. E. Perkins and E. M. Royer. Ad hoc on demand distance vector (aodv) routing. pages 90–100, February 1999.

[12] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the placement of web server replicas. In *IEEE INFOCOM 2001*, pages 1587–1596, 2001.

[13] F. Sailhan and V. Issarny. Energy-aware web caching for mobile terminals. In *22nd International Conference on Distributed Computing Systems Vienna University of Technology (ICDCS 2002)*, 2002.

[14] 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.

[15] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[16] J. Wang. A survey of web caching schemes for the internet. *ACM SIGCOMM Computer Communications Review*, 29(5):35–46, October 1999.

[17] P. Winter. Steiner problem in networks: A survey. *ACM Networks*, 17(2):129–167, 1987.

[18] Z. Xiang, Z. Zhong, and Y. Zhong. A cache cooperation management for wireless multimedia streaming. In *IEEE International Conferences on Info-tech and Info-net ICII 2001*, pages 328–333, 2001.

[19] J. Xu, B. Li, and D. L. Lee. Placement problems for transparent data replication proxy services. *IEEE Journal on Selected Areas in Communications*, 20(7):1383–98, September 2002.

APPENDIX

THEOREM 2. *Let OPT be the optimum value of the primal integer problem* **P** *in (4). The solution obtained by the algorithm is at most* $6 \times OPT$.

We obtain a caching and access strategy by setting $z_e = 1$ for all the open edges, and $x_{ke} = 1$ for the open edge nearest to k such that $k \in R_e$. This ensure that constraints (5) and (6) are met. By using Lemma 1, constraint (7) is also satisfied. An edge e is given by the set $\{r_e, t_e\}$, where nodes r_e and t_e are the terminal nodes of edge e. In order to provide a bound, we will construct a dual feasible solution for our primal problem (4). By introducing the dual variables β and θ, we obtain the following formulation for the dual problem.

$$\text{Maximize} \quad \sum_{k \in \mathcal{V}} \alpha_k \quad (11)$$

$$\text{s.t.} \quad \alpha_k - \beta_{ke} - \sum_{\mathcal{S} \subseteq \mathcal{E}, e \in \mathcal{S}, e_v \notin \mathcal{S}} \theta_{\mathcal{S},k} \leq c_{ke} p_k \quad \forall k \in \mathcal{V}, e \in \mathcal{E} \quad (12)$$

$$\sum_{k \in \mathcal{V}} \beta_{ke} + \sum_{k \in \mathcal{V}} \sum_{\mathcal{S} \subseteq \mathcal{E}: e \in \delta(\mathcal{S}), e_v \notin \mathcal{S}} \theta_{\mathcal{S},k} \leq M \quad \forall e \in \mathcal{E} \quad (13)$$

$$\alpha_k, \beta_{ke}, \theta_{\mathcal{S},k} \geq 0 \; . \quad (14)$$

PROOF. We first assume that all access probabilities (p_k's) are equal. Consider the last stage of POACH in which no new cache locations are opened and the algorithm terminates. At this point, for any node k, $\alpha_k = \sum_{e \in \mathcal{E}} c_{ke} p_k x_{ke}$. Note that $\alpha_k = 0$ at the end of POACH for all nodes lying on open edges. For all k with $\alpha_k = 0$, set $\alpha_k = M$. As a result, we have

$$\sum_{k \in V} \sum_{e \in E} p_k c_{ke} x_{ke} \leq \sum_{k \in \mathcal{V}} \alpha_k \quad (15)$$

$$M \sum_{e \in E} z_e \leq \sum_{k \in \mathcal{V}} \alpha_k \quad (16)$$

The inequality in (16) follows from the fact that each node k, which lies on an open edge, has $\alpha_k = M$.

Set all other dual variables, β and θ, to zero. For an edge e, define $\mathcal{S}_e = \mathcal{E} \backslash \{e\}$. For a node k, define $\theta_{\mathcal{S}_e,k}$ as follows.

$$\theta_{\mathcal{S}_e,k} = \begin{cases} M & k \in e \\ 0 & \text{otherwise.} \end{cases} \quad (17)$$

Consider an edge $e \in \mathcal{E}$ and a node k such that $k \notin e$. Suppose $k \in l$. Clearly, at least one such l exists. Then,

$$\sum_{\mathcal{S} \subseteq \mathcal{E}, e \in \mathcal{S}, e_v \notin \mathcal{S}} \theta_{\mathcal{S},k} \geq \theta_{\mathcal{S}_l,k} = M \quad (18)$$

The last equality follows from (17). Consider now a node $k \in e$. If there exists an edge $l \neq e$ such that $k \in l$, then $\sum_{\mathcal{S} \subseteq \mathcal{E}, e \in \mathcal{S}, e_v \notin \mathcal{S}} \theta_{\mathcal{S},k} \geq M$. This follows from the same logic as in (18). For a node k, which only lies on edge e and no other edge, define $\beta_{ke} = M$. Since, $\alpha_k \leq M, \forall k \in \mathcal{V}$ (otherwise the algorithm would not have terminated), the first set of dual feasibility constraints in (12) is satisfied for the assigned values of β and θ.

Consider the second set of feasibility constraints in (13). For any edge e,we have

$$\sum_{k \in \mathcal{V}} \beta_{ke} \leq M. \quad (19)$$

This follows directly from the values assigned to the dual variables. For any node k, $\sum_{\mathcal{S} \subseteq \mathcal{E}: e \in \delta(\mathcal{S}), e_v \notin \mathcal{S}} \theta_{\mathcal{S},k} = \theta_{\mathcal{S}_e,k}$. Therefore,

$$\sum_{k \in \mathcal{V}} \sum_{\mathcal{S} \subseteq \mathcal{E}: e \in \delta(\mathcal{S}), e_v \notin \mathcal{S}} \theta_{\mathcal{S},k} = \theta_{\mathcal{S}_e,r_e} + \theta_{\mathcal{S}_e,t_e} = 2M. \quad (20)$$

Adding equations (19) and (20), we have

$$\sum_{k \in \mathcal{V}} \beta_{ke} + \sum_{k \in \mathcal{V}} \sum_{\mathcal{S} \subseteq \mathcal{E}: e \in \delta(\mathcal{S}), e_v \notin \mathcal{S}} \theta_{\mathcal{S},k} \leq 3M. \quad (21)$$

From above, it is clear that we can obtain a dual feasible solution by dividing all dual variables (α, β, θ) by 3. Denote these dual feasible variables by $(\hat{\alpha}, \hat{\beta}, \hat{\theta})$. From (15) and (16), we have

$$\sum_{k \in V} \sum_{e \in E} p_k c_{ke} x_{ke} \leq \sum_{k \in \mathcal{V}} \alpha_k = 3 \times \sum_{k \in \mathcal{V}} \hat{\alpha_k} \quad (22)$$

$$M \sum_{e \in E} z_e \leq \sum_{k \in \mathcal{V}} \alpha_k = 3 \times \sum_{k \in \mathcal{V}} \hat{\alpha_k}. \quad (23)$$

Adding equations (22) and (23), we have

$$\sum_{k \in \mathcal{V}} \sum_{e \in \mathcal{E}} c_{ke} x_{ke} + M \sum_{e \in \mathcal{E}} z_e \leq 6 \times \sum_{k \in \mathcal{V}} \hat{\alpha_k}. \quad (24)$$

By the theory of duality [15] and (24), we have

$$\sum_{k \in \mathcal{V}} \hat{\alpha_k} \leq OPT \leq \sum_{k \in \mathcal{V}} \sum_{e \in \mathcal{E}} c_{ke} x_{ke} + M \sum_{e \in \mathcal{E}} z_e \leq 6 \times OPT \; . \quad (25)$$

The proof can be extended to the general case when access probabilities are not uniform, by arguments similar to [14].

□