COACS: A Cooperative and Adaptive Caching System for MANETs

Hassan Artail, *Member*, *IEEE*, Haidar Safa, *Member*, *IEEE*, Khaleel Mershad, Zahy Abou-Atme, *Student Member*, *IEEE*, and Nabeel Sulieman, *Student Member*, *IEEE*

Abstract—This paper introduces a cooperation-based database caching system for Mobile Ad Hoc Networks (MANETs). The heart of the system is the nodes that cache submitted queries. The queries are used as indexes to data cached in nodes that previously requested them. We discuss how the system is formed and how the requested data is found if cached or retrieved from the external database and then cached. Analysis is performed, and expressions are derived for the different parameters, including the upper and lower bounds for the number of query caching nodes and the average load they experience, generated network traffic, node bandwidth consumption, and other performance-related measures. Simulations with the ns-2 software were used to study the performance of the system in terms of average delay and hit ratio and to compare it with the performance of two other caching schemes for MANETs, namely, CachePath and CacheData. The results demonstrate the effectiveness of the proposed system in terms of achieved hit ratio and low delay.

Index Terms—Cache management, distributed cache, mobile ad hoc networks, cache indexing, mobility, database queries.

1 Introduction

As Mobile Ad Hoc Networks (MANETs) are becoming increasingly widespread, the need for developing methods to improve their performance and reliability increases. One of the biggest challenges in MANETs lies in the creation of efficient routing techniques [6], but to be useful for applications that demand collaboration, effective algorithms are needed to handle the acquisition and management of data in the highly dynamic environments of MANETs.

In many scenarios, mobile devices (nodes) may be spread over a large area in which access to external data is achieved through one or more access points (APs). However, not all nodes have a direct link with these APs. Instead, they depend on other nodes that act as routers to reach them. In certain situations, the APs may be located at the extremities of the MANET, where reaching them could be costly in terms of delay, power consumption, and bandwidth utilization. Additionally, the AP may connect to a costly resource (e.g., a satellite link) or an external network that is susceptible to intrusion. For such reasons and others dealing with data availability and response time, caching data in MANETs is a topic that deserves attention.

MANETs are dynamic in nature, and therefore, a reliable caching scheme is more difficult to achieve. Links between nodes may constantly change as nodes move around, enter, or leave the network. This can make storing and retrieving cached data particularly difficult and unreliable. The use of mobile devices adds even more complexity due to their relatively limited computing resources (e.g., processing power and storage capacity) and limited battery life. It follows that an effective caching system for MANETs needs to provide a solution that takes all of these issues into consideration. An important policy of such a solution is not to rely on a single node but to distribute cache data and decision points across the network. With distribution, however, comes a new set of challenges. The most important of which is the coordination among the various nodes that is needed in order to store and find data.

A preliminary system was proposed in [2] to cache database responses to queries in given nodes and uses the queries as indexes to the responses. This paper builds on the same general idea but introduces several significant changes at the design level, in addition to elaborate studies and simulations that were made to examine the system and prove its usefulness. Briefly, the architecture of the proposed system is more flat when compared to the one in [2] (a review of this system is given at the end of Section 2), as it eliminates the role of the service manager, which is responsible for performing management duties, and instead distributes such duties to the nodes that will perform the low-level functions themselves. In short, the aim of the proposed framework is to provide efficient and reliable caching in MANET environments.

The rest of this paper is organized as follows: In Section 2, a survey of related work is given, followed by Section 3, which describes the proposed system. Section 4 provides an analysis and derives expressions for the system parameters and performance measures. Section 5 is dedicated to describing the simulation experiments and discussing the results. Finally, Section 6 concludes the paper and proposes

- *H. Artail, K. Mershad, Z. Abou-Atme, and N. Sulieman are with the Electrical and Computer Engineering Department, American University of Beirut, Beirut 1107-2020, Lebanon. E-mail: {hartail, kwm03, zla00, nss04}@aub.edu.lb.*
- *H. Safa is with the Computer Science Department, American University of Beirut, Beirut 1107-2020, Lebanon. E-mail: hs33@aub.edu.lb.*

Manuscript received 8 June 2006; revised 12 Mar. 2007; accepted 23 Jan. 2008; published online 28 Jan. 2008.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-0158-0606.
Digital Object Identifier no. 10.1109/TMC.2008.18.

future works related to fault tolerance and improving the system response.

2 Literature Overview

Few caching schemes for MANETs have been proposed in the literature. In this section, we provide a review of some of those that provide serious solutions to the caching problem.

In [19], three related caching schemes were discussed: CachePath, CacheData, and HybridCache. The main idea behind these schemes is to analyze passing requests and cache either the data or the address of the node in which it is stored. Later, if the same request for that data passes through the node, it can provide the data itself or redirect the request to its location. CachePath saves space by storing locations where data should be stored, while CacheData saves time by storing the data instead of the path. The third scheme, HybridCache, is a middle solution where queries are cached by path or by data, depending on what is optimal.

The success of the above systems, in terms of delay and hit ratio, is highly dependent on node positions relative to each other and relative to the AP. A node that is caching data will only be accessed if it is on the path of the request to the external data source. Even when each node is given infinite caching space, the system can only reach a high hit ratio when one node serves as the exclusive connection between other nodes and the AP. In this case, however, the load on this node may be prohibitively high. If there is more than one node that can reach the external data source, the load on the nodes will decrease, but performance will suffer because there is no coordination between the nodes. This lack of coordination also has an adverse effect on space since nodes cache the paths or data independently. Additionally, it should be noted that the above schemes rely on the modification of the routing protocol in that every time a packet passes through a node, it is checked to see if it is a data request. If it is, the cache is checked for a copy of the data or the path to it, and the request is processed accordingly.

In [13], a caching algorithm is suggested to minimize the delay when acquiring data. In order to retrieve the data as quickly as possible, the query is issued as a broadcast to the entire network. All nodes that have this data are supposed to send an acknowledgment back to the source of the broadcast. The requesting node (RN) will then issue a request for the data (unicast) to the first acknowledging node it hears from. The main advantage of this algorithm is its simplicity and the fact that it does achieve a low response delay. However, the scheme is inefficient in terms of bandwidth usage because of the broadcasts, which, if frequent, will largely decrease the throughput of the system due to flooding the network with request packets [9]. Additionally, large amounts of bandwidth will also be consumed when data items happen to be cached in many different nodes. Such situations occur because the system does not account for controlling redundancy, thus allowing many nodes to cache the same data.

In [15] and [14], two strategies for indexing and caching Web pages were proposed. In the method in [15], several nodes are chosen as proxy servers to cache proxies and use multicasting to cooperate and form a single distributed cache in which any server can handle the client's request. When a server receives a client request, it redirects it to the server that holds the proxy (page). In [14], on the other hand, a caching software named *SQUIRREL* was integrated into Internet nodes to allow several nodes within a given region to share their caches. The basic idea is to cache each page at a certain node according to a hashing key obtained from the page URL. When a new request is issued, either it is served by the local cache or the key of this request is calculated, and the request is forwarded to the node that might have the page in its cache.

Our system shares several basic similarities with [15] in terms of assigning certain nodes to be *guides* to cached objects and making these nodes cooperate to answer a request. In fact, requests in our system could be changed from query IDs to Web page URLs without changing the design of the system. However, in our case, we process the request at each server (or query directory (QD)) and then pass it on to the next one instead of multicasting it as in [15]. Also, the server that finds the cached object does not redirect the client to the other node that contains the object but forwards the request directly to this node. These strategies reduce the network traffic significantly without imposing a large increase on the network response time or hit ratio.

A related scheme for caching, locating, and streaming multimedia objects (MOs) in mobile networks was proposed in [11]. The basic idea is to rely on an application manager (APGR) that is interposed between multimedia applications and the network layer at each node. The APGR determines the best source caching the required MO, opens a session with this source's APGR, and organizes data streaming by sending control messages at constant time periods. APGRs communicate with each other to make the process of finding and downloading MOs much faster.

As was mentioned earlier, the same general idea of caching database queries and their corresponding responses was introduced in [2]. In this scheme, the queries were used as indexes to the responses when searching for data. The described architecture is hierarchical in the sense that it has an elected *service manager* (SM) that oversees the caching operations and is responsible for assigning the roles of QDs and *caching nodes* (CNs) to specific mobile nodes for caching queries and their responses, respectively. The QDs decide on which CNs to cache external (noncached) data and maintain one-way hash tables to quickly locate the responses of the locally cached queries. The limitations of this scheme include relying heavily on the SM (and its backup), which, just like any other mobile node, may move around, leave the network, or run low on battery power. Second, the system employs random forwarding of requests when searching for a cached query that matches the one being requested. This does not exploit the locations of the QDs with respect to the RN and with respect to each other, which may result in unnecessary delays and resource consumptions.

3 Proposed Framework

This section describes the proposed system, COACS, which stands for Cooperative and Adaptive Caching System. The idea is to create a cooperative caching system that minimizes delay and maximizes the likelihood of finding data that is cached in the ad hoc network, all without inducing excessively large traffic at the nodes. First, we

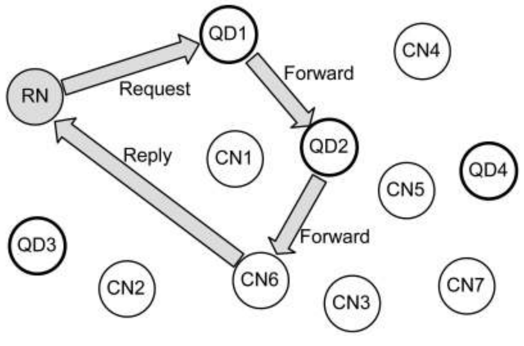


Fig. 1. An example of a COACS scenario: a request submitted to QD_1 and forwarded to QD_2, where a hit occurred. The response of the query, stored in CN_6, is sent to the RN.

cover the basic concepts that distinguish COACS from other systems, as they are essential to the rest of this section.

3.1 Basic Concepts

COACS is a distributed caching scheme that relies on the indexing of cached queries to make the task of locating the desired database data more efficient and reliable. Nodes can take on one of two possible roles: CNs and QDs. A QD's task is to cache queries submitted by the requesting mobile nodes, while the CN's task is to cache data items (responses to queries). When a node requests data that is not cached in the system (a miss), the database is accessed to retrieve this information. Upon receiving the response, the node that requested the data will act as a CN by caching this data. The nearest QD to the CN will cache the query and make an entry in its hash table to link the query to its response.

The CachePath and CacheData schemes that were discussed earlier have nodes with functions similar to a CN, but they do not offer functionality for searching the contents of all the CNs. In order to find data in a system of only CNs, all the nodes in the network would need to be searched. This is where QDs come into play in the proposed system. QDs act as distributed indexes for previously requested and cached data by storing queries along with the addresses of the CNs containing the corresponding data. In this paper, we refer to the node that is requesting the data as the RN, which could be any node, including a CN or a QD.

The QD nodes make up the core of the caching system. To cope with the limited resources of mobile devices and to decrease the response time of the system, several QDs are used to form a distributed indexing system. If one QD receives a request that it has not indexed, the request is passed on to another QD. The desired number of QDs is a function of the various system parameters, which are addressed in Section 4. Fig. 1 shows a simplified example of a scenario in COACS, where the requested query is stored in QD_2, and its response is stored in CN_6.

Since queries typically occupy much less space than the corresponding data, QDs will be able to store far more entries compared to the number of entries a CN can hold. This helps decrease the number of nodes that need to be queried in order to find data in the MANET. The capacity of QDs can be further enhanced by using query compression techniques similar to the one proposed in [10], which may prove helpful in large networks.

Since COACS is a mobile distributed system, deciding which QD to send a request to and the next QD to forward it to (if the first one does not return a hit) is a crucial issue. It would be inefficient for a request to be forwarded from one end of the network to another in order to reach one QD and then have it forwarded back across the network to reach a second QD. For this, we propose the Minimum Distance Packet Forwarding (MDPF) algorithm. Many routing protocols such as DSDV have tables that keep track of how to reach all known nodes in the network and record the next neighboring node and the number of hops (metric) needed to reach a destination. An RN can use this data to send the request to the nearest QD. If a QD does not have a matching query, it uses MDPF and forwards the request to a nearby QD. In COACS, MDPF is used in all scenarios that involve iteratively searching through nodes. To avoid sending a packet back to a node that was already visited, a list of visited nodes is maintained in the packet being forwarded.

3.2 Caching System Formation

The QDs are the central component of the system and must be selected carefully. Preference should be given to nodes that are expected to stay the longest in the network and have sufficient resources. Nodes have to calculate and store a special score that summarizes their resource capabilities, including the expected time during which the device is in the MANET (*TIME*), the battery life (*BAT*), the available bandwidth (*BW*), and the available memory for caching (*MEM*). To be considered a candidate QD, the device must meet a minimum criterion in each category. That is

$$\{D_k\}|R_X^k > \theta_X, \quad \forall X \in \{TIME, BAT, BW, MEM\}, \qquad (1)$$

where $\{D_k\}$ is the set of candidate devices, R_X^k is a resource for device k, and θ_X is an empirically set threshold for resource X. If $\{D_k\}$ includes more than one device, then the one with the maximum weighted score is selected. That is, if device j is the selected one, then

$$SC_j = \max\{SC_k\}|SC_k = \sum \alpha_X R_X^k, \qquad (2)$$

where SC_k is node k's score, k refers to one of the devices satisfying the condition in (1), and α_X is the weight associated with resource X such that $\sum \alpha_X = 1$.

Although the QD selection factors in (1) may be very dynamic in a MANET environment, the configuration of the system (list of QDs) will not necessarily change when these factors vary. The QD list will usually only change if a significant number of nodes join the network, thus requiring additional caching capacity, or a QD leaves, in which case another node takes its place.

As illustrated in Section 4, the addition of a QD decreases the average load on existing QDs and potentially increases the available cache space (given that nodes can cache additional results) and, in turn, the hit ratio. At the same time though, this increases the response time of the system. Hence, the number of QDs should be chosen prudently in order to restrict the average delay of the system from increasing indefinitely while maintaining an acceptable load on QDs.

When nodes join the network, they send HELLO packets so that other nodes know about them. After the HELLO packets are exchanged, the first node that needs to cache

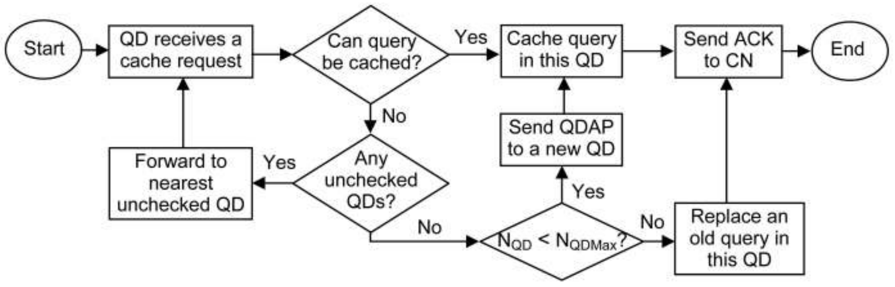


Fig. 2. Procedure for caching queries in QDs.

a data item (i.e., after submitting a data request and getting the reply) sends a *COACS Score Packet* (CSP) containing its score, address, and caching capacity and an empty *exhausted node list* to one of its neighbors. When a node receives a CSP, it adds its score, address, and caching capacity to the table in the CSP and then chooses one of the nearest nodes from its routing table that is not contained in the list of addresses and the exhausted node list in CSP (implying that this node has not received this CSP yet) and sends the CSP to it. If the node receiving the CSP finds that all nodes from its routing table are present in the list of addresses in CSP, it adds itself to the exhausted node list and sends the CSP to a node that is in the list of addresses but not in the exhausted list. This strategy insures that the CSP will traverse all nodes in the network sequentially. Each node checks if the CSP table includes the scores and caching capacities of all nodes in the network excluding itself (the list of nodes can be retrieved from the underlying routing protocol). If yes, it plays the role of a QD Assigner (QDA) by sending the node that corresponds to the highest score a *QD Assignment Packet* (QDAP) containing the CSP table. The identified node (QD_1) computes the starting and maximum numbers of QDs, N_{QD}^{strt} and $N_{QD}^{\max}$, using the data from the list in the QDAP. This node then sends a QDAP to the $N_{QD}^{strt} - 1$ nodes with the highest scores in the list (excluding itself). Assuming that all candidate QDs returned an acknowledgment, QD_1 broadcasts a *COACS Information Packet* (CIP) with the addresses of all assigned QDs, thus informing all nodes in the network of the complete list of QDs. The CIP is broadcast only when the list of QDs changes, and nodes that join the network later get the list of QDs by requesting a CIP from nearby nodes, which reply by sending a unicast CIP.

Generally, a new QD is added to the system when a query needs to be cached but no QD agreed to cache it. The last QD to receive the caching request will initiate a CSP. Also, COACS needs to account for nodes going offline, and it depends on the routing protocol to detect such occurrences and update the routing tables. If a QD goes offline, the first node to discover this will initiate a CSP in order to find a new candidate QD. In both cases, the first node to compute the highest score from the CSP will be the QDA and sends a QDAP to the highest score candidate. If this node accepts, it broadcasts a CIP with itself added to the QD list; else, it replies with a negative acknowledgment to the QDA. To protect against situations in which this candidate takes no action, a timer is started at the QDA after each QDAP is sent. If the QDA receives a NACK or if it waits a period of T, it sends a QDAP to the second-highest-score candidate, and so on, until a candidate accepts the assignment. As discussed below, the CN holds for each cache entry, in addition to the response data, the query itself and a reference to the QD that caches it. This added information is used to rebuild QD entries when a QD goes offline. Upon receiving the CIP from the replacement QD, the concerned CNs will send it the queries that used to reference the lost QD using a *Query Caching Request Packet* (QCRP). The CIP will also serve to inform nodes about the change and prompt them to update their QD lists. If a CN goes offline, the QDs will detect its departure after the routing protocol updates their routing tables and will delete the associated entries in their cache. Note that in case an on-demand routing protocol is in place, each QD could send a special message to all other QDs periodically to discover if a QD has gone offline. Furthermore, every CN could be set up to return an acknowledgment when it is forwarded a request from a QD, so it could eventually be discovered when it goes offline.

Additionally, the score of a QD may fall below the set threshold at any time due to its participation in the network and its use by the user. When it detects that its score is about to become low, it broadcasts a CSP packet, and upon receiving the CIP from the new QD, it transfers its cache to it, broadcasts a CIP not including itself, and then deletes its cached queries.

3.3 Caching Data

The RN will only act as a CN for the submitted request if the reply comes from the data source. This, however, does not preclude the RN from caching the reply for its own use even if it is cached elsewhere in the ad hoc network but will only cache it for the purpose of servicing it to another node if this reply and its associated query are not cached. When an RN becomes a CN for a particular request, it stores the data item and the corresponding query and then sends a QCRP to the nearest QD. If this QD has no memory available, it forwards the request to its nearest QD, and so on, until a QD that will store the query is reached or a new QD is added to the system. If a QD ends up caching the query, it sends a *Cache Acknowledgement Packet* (CACK) to the CN, which will in turn store a reference that links the cached data with this QD. Fig. 2 shows how a QD processes a request.

3.4 Search Algorithm

Given that all nodes in the MANET have knowledge of all QDs, then when a node requires certain data, it sends a *Data Request Packet* (DRP) to the nearest QD. If this QD does not have a matching query, it adds its address to the DRP to indicate that it has already received the request and then sends this modified DRP to the nearest QD that has not been

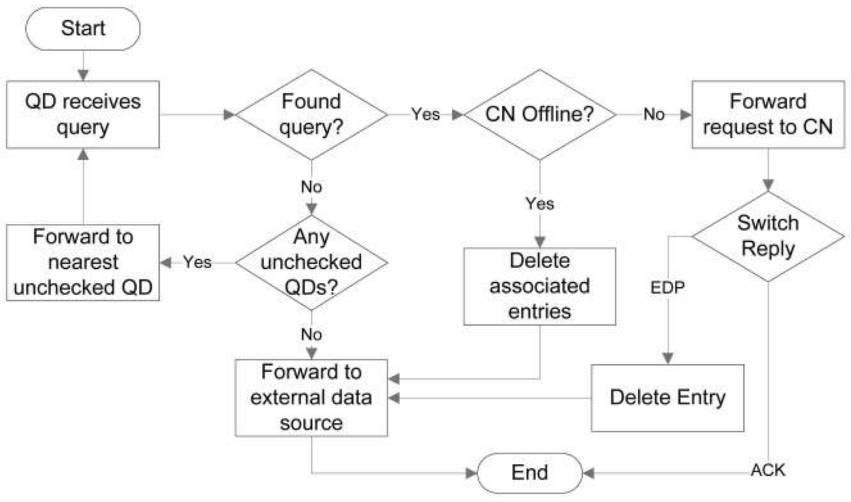


Fig. 3. Sequence of events when a QD receives a query.

checked yet. This continues until a hit occurs or until all the QDs have been checked, in which case an attempt is made to contact the data source. If a hit occurs at a QD, the QD visit list is removed from the DRP before the latter is sent to the CN that is caching the corresponding data, which will in turn send the reply data via a *Data Reply Packet* (DREP) directly to the RN, whose address is in the source field of the request packet. If a CN has gone offline, all QDs will be able to detect this when their routing tables get updated by the proactive routing protocol and will delete all the related entries in their memories. On the other hand, if a CN ever decides to replace an old cache item with a newer one, it informs the corresponding QD about it by sending an *Entry Deletion Packet* (EDP). In this case, the QD will delete the related entry from its cache to prevent misdirected requests for the data. The whole process is depicted in Fig. 3.

This section ends with Table 1, which gives a summary of the packets discussed.

4 Analysis

In this section, we show how varying some of the system parameters affects its performance. It is demonstrated later that as the number of QDs in the system increases, the average delay to receive a response for a request increases, while the load on the individual QDs decreases. Conversely, a smaller number of QDs results in a smaller average delay but a higher load on the QDs. Hence, an "optimal" number of QDs ought to be found that provides the maximum possible system caching capacity *while* resulting in a tolerable average delay and acceptable load on the individual QDs. For this purpose, we develop an upper limit on the average delay and also an upper limit on the individual QD load and then select the number of QDs accordingly for different values of the desired hit ratio. We start with the average delay and later treat the load.

4.1 Average Delay Limit

One way to derive the limit on the average delay τ is to make it equal to the delay of having no caching $\tau_{NoCaching}$. The maximum number of QDs $N_{QD}^{\max}$ is set accordingly as follows:

$$N_{QD}^{\max} = \max(N_{QD})\big|E[\tau] < E[\tau_{NoCaching}]. \tag{3}$$

Next, we derive several parameters that are involved in computing $N_{QD}^{\max}$:

1. the expected number of hops between any two nodes,
2. the expected number of hops within the QD system (different because of MDPF routing),
3. the expected number of hops to the external network (i.e., AP, assumed to be in the corner of the topography), and
4. the response time of the system.

TABLE 1
Packets Used in COACS

Packet	Function	Description
HELLO	New node's arrival	Broadcasted by a newly arriving node
CSP	Node score request	Broadcasted when a new QD is needed
CIP	COACS Info	Broadcasts QD list to all nodes
DRP	Data Request	Submitted by an RN to request data
DREP	Data Reply	Submitted by an CN or DB to RN
QCRP	Query Caching Request	Sent by a CN to a QD to cache one or more queries
EDP	Entry Deletion	Deletes a QD entry for a particular CN
CACK	Cache Add Acknowledge	Sent from QD to CN to Acknowledge caching
QDAP	QD Assignment Packet	Invitation to a node to become a QD

4.1.1 Expected Number of Hops between Two Nodes

Similar to [4], we assume a rectangular topology with area $a \times b$ and uniform distribution of nodes. Two nodes can form a direct link if the distance S between them is $\leq r_0$, where r_0 is the maximum node transmission range. We seek to compute the expected number of hops between any two nodes. Using stochastic geometry, the probability density function of S is given in [4] as

$$f(s) = \frac{4s}{a^2b^2}\left(\frac{\pi}{2}ab - as - bs + 0.5s^2\right) \tag{4}$$

for $0 \leq s < b < a$. It is concluded that if two nodes are at a distance s_0 from each other, the number of hops between them, when there are a sufficient number of nodes to form a connected network, would tend toward s_0/r_0. Hence, $E[H]$, the expected minimum number of hops between any two nodes in the network, is equivalent to dividing $E[S]$, the expected distance, by r_0. It should be noted that the value of $E[H]$ represents a lower bound because when nodes are sparse in the network, the number of hops will inevitably increase due to having to route through longer distances to reach a certain node. When $a = b$, the expected number of hops, as is depicted in [4], is

$$E[H] = 0.521a/r_0. \tag{5}$$

4.1.2 Expected Number of Hops within the System of Query Directories

The previously determined $E[H]$ represents the expected number of hops when only one destination choice is available. However, using MDPF, an RN or a QD picks the nearest QD, and hence, the expected number of hops is anticipated to be lower. When there are more choices, it is more likely for an RN or a QD to find an unchecked QD that is closer to it than when having fewer choices. We develop a recursive algorithm, where the analysis for two choices depends on the expected distance of one choice (i.e., $E[H]$), the solution for three choices depends on the expected number of hops when having two choices, and so on. First, we define three functions:

$$E[H|n = N],$$

$$P(H < h) = P(S < h \times r_0) = \int_0^{h\times r_0} f(s)ds, \text{ and}$$

$$E[H|H < h] = \int_0^{h\times r_0} sf(s)ds \Big/ \int_0^{h\times r_0} f(s)ds,$$

which are respectively the expected number of hops given N choices, the probability that a node is within h hops, and the expected distance to a node within h hops away.

To understand the analysis, first, suppose we place two nodes $O1$ and $O2$ randomly in a square and pick one of them, say, $O1$, as a reference (the node that has to forward the request). The expected distance (hops) between this node and $O2$ is as determined before:

$$E[H|n = 1] = E[H] = 0.521a/r_0. \tag{6}$$

A third node, $O3$, will either be closer to $O1$ than $O2$ or farther. If we always send to the nearest choice, the expected number of hops to one of the two choices from the reference node is

$$\begin{aligned} E[H|n = 2] = {} & P(H > E[H|n = 1])E[H|n = 1] \\ & + P(H < E[H|n = 1])E[H|H < E[H|n = 1]]. \end{aligned} \tag{7}$$

The above is the probability that $O3$ is farther times the expected number of hops to reach $O2$ plus the probability that $O3$ is closer times the expected number of hops to reach $O3$. Similarly, after adding node $O4$, the expected number of hops from $O1$ to the nearest of the three choices is

$$\begin{aligned} E[H|n = 3] = {} & P(H > E[H|n = 2])E[H|n = 2] \\ & + P(H < E[H|n = 2])E[H|H < E[H|n = 2]]. \end{aligned} \tag{8}$$

Hence, we can write the following general expression:

$$\begin{aligned} E[H|n=i+1] = {} & P(H > E[H|n=i])E[H|n=i] \\ & + P(H < E[H|n=i])E[H|H < E[H|n=i]]. \end{aligned} \tag{9}$$

In an N_{QD} system, the expected number of hops from the RN to the nearest QD, say, QD_1, is denoted by $E[H_1|N_{QD}]$ and is equal to $E[H|n = N_{QD}]$. To calculate the number of hops to the nearest QD from QD_1, denoted by $E[H_2|N_{QD}]$, we add the expected number of hops when there are $N_{QD} - 1$ choices; that is, $E[H|n=N_{QD}-1]$. Hence, the expected number of hops from the RN to QD_i (going through $\text{QD}_1, \text{QD}_2, \ldots, \text{QD}_{i-1}$), when there are N_{QD} choices and using MDPF, is

$$E\left[H_i|N_{QD}\right] = \sum_{j=N_{QD}+1-i}^{N_{QD}} E[H|n = j]. \tag{10}$$

To calculate the average number of hops to get to the QD with the desired data from an RN, we multiply the probability P_i that QD_i has the data with the average number of hops to contact each QD and then take the sum. Hence, the expected number of hops to get to the QD with the data is

$$E\left[H_{QD_{Data}}\right] = \sum_{i=1}^{N_{QD}} P_i E\left[H_i|N_{QD}\right]. \tag{11}$$

We can use the above to derive the expected number of hops to get to the last QD in the system:

$$E\left[H_{QD_{Last}}\right] = E\left[H_{N_{QD}}|n = N_{QD}\right] = \sum_{j=1}^{N_{QD}} E[H|n = j]. \tag{12}$$

Assuming a uniform cache size, the probability that the data is located in QD_i, P_i, is equal to $1/N_{QD}$.

4.1.3 Expected Number of Hops to the External Network

Assuming an $a \times a$ topography filled with a sufficient number of nodes that are uniformly distributed, the expected distance to the AP, assumed to be at the corner, is

$$E[S_{AP}] = \int_0^a \int_0^a \frac{1}{a^2}\sqrt{x^2 + y^2}dxdy. \tag{13}$$

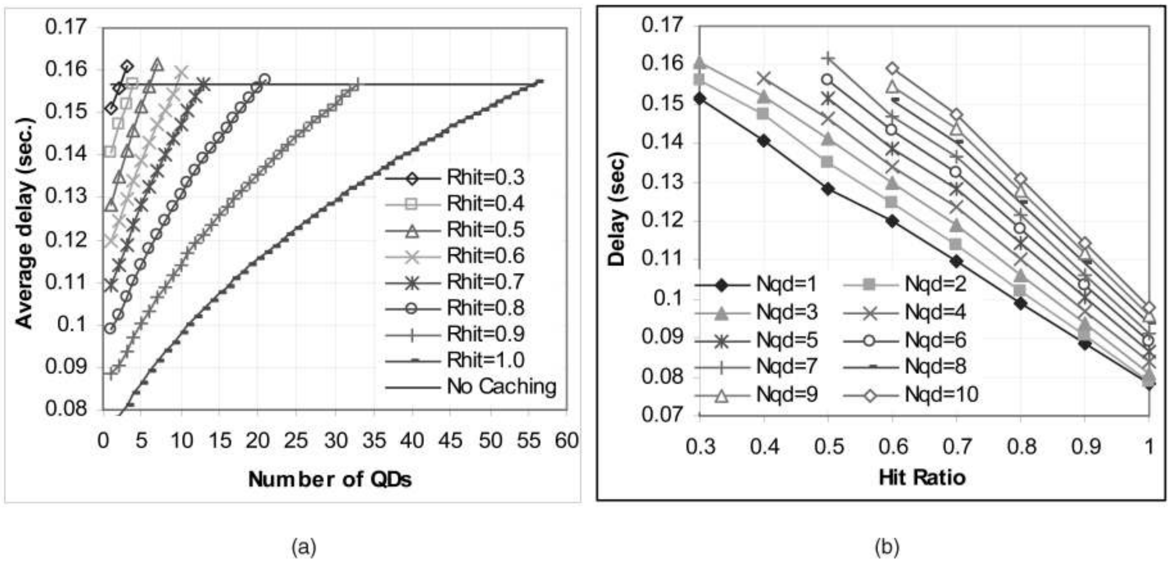


Fig. 4. (a) Average delay for different hit ratio values versus N_{QD} plotted against the delay of no caching. (b) Delay versus hit ratio for different N_{QD} values.

This seemingly trivial problem has a complex solution, which is illustrated in Appendix A. After dividing the result by the transmission range r_0, we get the expected number of hops $E[H_{AP}]$:

$$E[H_{AP}] = E[S_{AP}]/r_0 = 0.7652a/r_0. \tag{14}$$

4.1.4 Query Directory Access and Delay

To compute the average system delay, we must account for the hit ratio R_{hit}, since with a low R_{hit}, the average number of accessed QDs will be near the total number of QDs, while with a high R_{hit}, it will be near the median number of QDs. We also need to account for the delay for transmitting packets between nodes inside the network T_{in} and the delay for accessing a node outside the network (data source) T_{out}. The delay will vary depending on several factors, such as node location, packet size, the number of hops to reach a node, and network congestion. For simplicity, however, T_{in} and T_{out} are assumed to be average delays that account for these factors. Finally, it is noted that the processing delays at the CNs and QDs were neglected since the process of searching for the query ID takes very small time when compared to T_{in} and T_{out}.

In the case of a hit and after a delay of $T_{in}E[H_{QD_{Data}}]$ (to get to the QD with the data), an additional delay of 2 $T_{in}E[H]$ is incurred to access the CN and transmit the reply back to the RN. For a miss and after a delay of $T_{in}E[H_{QD_{Last}}]$ (to traverse all QDs), a delay of $T_{in}E[H_{AP}]$ is first incurred to forward the request to the AP. This is followed by a delay of $2T_{out}$ for accessing the DB and getting its reply and a further $T_{in}E[H_{AP}]$ to send it back to the RN. Considering the above terms and ignoring the processing delays of the database, QDs, and CNs, the following expression gives the average delay for the COACS model:

$$\begin{aligned} E[\tau] =& R_{hit}T_{in}\left(E\left[H_{QD_{Data}}\right]+2E[H]\right) \\ &+ (1-R_{hit})\left(T_{in}\left(E\left[H_{QD_{Last}}\right]+2E[H_{AP}]\right)+2T_{out}\right). \end{aligned} \tag{15}$$

From the above, the upper and lower limits of delay can be determined by setting R_{hit} to zero and one, respectively. The best and worst performances are respectively the hit and miss ratio delays:

$$E[\tau_{hit}] = T_{in}\left(E\left[H_{QD_{Data}}\right] + 2E[H]\right), \tag{16}$$

$$E[\tau_{Miss}] = T_{in}\left(E\left[H_{QD_{Last}}\right] + 2E[H_{AP}]\right) + 2T_{out}. \tag{17}$$

One can observe that when there is a miss, the average delay of the system is greater than $2(T_{in}E[H_{AP}] + T_{out})$, which means that the node would have a lower delay collecting data from the database server itself. However, one of the advantages of this system is that the average hit ratio increases rapidly, which, in turn, decreases the average response time.

4.1.5 Determining the Maximum Number of Query Directories

With the above information, we are ready now to apply the expression in (3) and determine the upper limit of N_{QD}. First, however, we specify the delay for going straight to the database as

$$E[\tau_{NoCaching}] = T_{in}(2E[H_{AP}]) + 2T_{out}. \tag{18}$$

Next, we can plug in the expressions of $E[H]$, $E[H_{QD_{Data}}]$, $E[H_{QD_{Last}}]$, and $E[H_{AP}]$ in (15) and then use the resultant expression along with (18) in (3). We get the following inequality:

$$E[H_{QD_{Data}}]+(1/R_{hit}-1)E\left[H_{QD_{Last}}\right]-0.4884a/r_0-2T_{out}/T_{in}<0. \tag{19}$$

Since $E[H_{QD_{Data}}]$ and $E[H_{QD_{Last}}]$ are recursively derived, the inequality in (19) is evaluated for different values of R_{hit} and N_{QD}. We want to determine the maximum N_{QD} value $N_{QD}^{\max}$ that satisfies (19) for each R_{hit} value. For example, by setting a/r_0 to 10 for a 1 Km^2 area and T_{out}/T_{in} to eight, we obtain the results shown in Fig. 4a, which illustrates the

TABLE 2
Values for the Maximum Values of N_{QD} to Avoid Excessive Delays

$R_{hit} \rightarrow$	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
$N_{QD}^{\max} \rightarrow$	2	3	5	9	12	20	32	55

value of $N_{QD}^{\max}$ for different R_{hit} values, as shown in Table 2. The right graph shows the delay versus R_{hit} when N_{QD} varies. It demonstrates that as R_{hit} increases (fewer trips are made to the data source), the resultant time savings are increasingly greater than the additional delay incurred by adding a QD.

4.2 Load Balancing on Query Directories

Since QDs are ordinary nodes themselves, an objective would be to minimize the number of requests handled by each node without degrading the system's performance. Given that MDPF calls for forwarding the request to the nearest QD and that the RN may be any one in the network, the initial QD may then be any of the QDs. Similarly, the second QD may be any of the remaining QDs, and so on. Hence, the order in which the QDs are accessed will be uniformly random. We define the load ratio on QD$_i$, λ_i, as the ratio of the number of accesses on QD$_i$ to the total number of requests issued and assume that the QDs have varying cache sizes. Having a cache size C_i for QD$_i$ with no replication, the probability P_i of finding a random query in QD$_i$ is

$$P_i = \frac{C_i}{\sum_{j=1}^{N_{QD}} C_j} = \frac{C_i}{C_{total}}. \tag{20}$$

Using this probability, we calculate the load ratio on each QD in the system (λ_i). The derivation of λ_i is given in Appendix B and is found to be equal to

$$\lambda_i = R_{hit}\frac{P_i - 1}{2} + 1. \tag{21}$$

Note that in the case of a uniform cache size, $p_i = 1/N_{QD}$, and hence, (21) becomes

$$\lambda_i = \frac{R_{hit}}{2} \times \frac{1 - N_{QD}}{N_{QD}} + 1. \tag{22}$$

The expression in (21) is plotted in Fig. 5, where one QD has twice the cache size with respect to the others that have the same size. The curves illustrate the load trends for the QD with double the capacity and for any of the other QDs, as the number of QDs increases. As shown, the load starts high, especially for the double-capacity QD and then decreases toward lower bounds associated with different hit ratios. The curves illustrate that beyond a certain number of QDs, the benefit in terms of lessening the load becomes insignificant and also show that the lower limit of the load per QD is 0.5 under the best circumstances (100 percent hit ratio and large N_{QD}).

As expected, the largest QD will experience the largest load, but as N_{QD} increases, this load gets increasingly closer to that of the other nodes. We find the above results motivating in two ways. First, a QD donating twice the amount of cache size is not penalized with much more load. Second, all other nodes will generally benefit in terms of load. Hence, the load balancing property of the system is not greatly disturbed when the caching capacities of the QDs fluctuate.

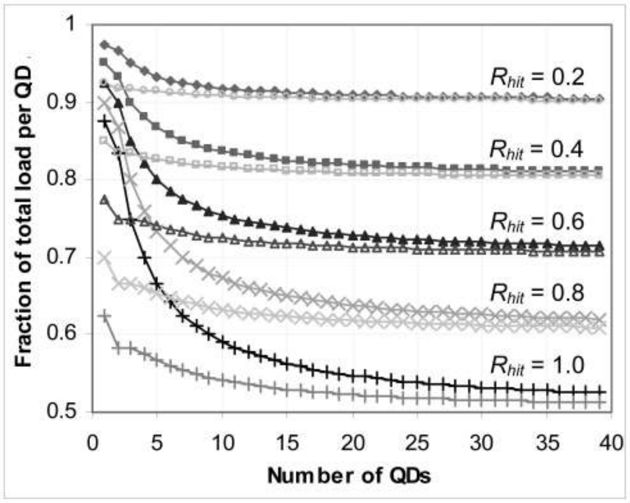


Fig. 5. Fraction of load per QD for different hit ratios. The top curve in each pair is for the QD with double the size while the lower curve is for each of the remaining QDs.

4.3 Determining the Starting Number of Query Directories

The starting number of QDs should be chosen so as to minimize both the delay and the load on QDs. To build the system of QDs, there are two possibilities: 1) start with one QD and then add QDs on demand as the need for more caching capacity arises or 2) start with N_{QD} QDs such that $N_{QD}^{strt} \leq N_{QD} \leq N_{QD}^{\max}$ for a desired hit ratio and add QDs as needed. We start with N_{QD}^{strt} since this leads to better performance and reduced overhead: the load on QDs will initially be lower, and the overhead plus delay involved in adding a QD while a request is pending will be reduced.

To determine N_{QD}^{strt}, we look for the value of N_{QD} after which the effect of adding one QD will offer a load fraction relief that is less than a threshold K_L. First, we multiply the load fraction λ_i for QD$_i$ by the request rate (RR) per node R_{req} and by the number of nodes N to get the total number of requests handled by this QD. Next, we take the derivative of the resulting function with respect to N_{QD} and set it to K_L. Given the infinite number of possible relative cache sizes within the system of QDs, we assume that all QDs have nearly the same cache size (i.e., use (22)). The derived number of QDs will then become the lower bound (starting number) of N_{QD}:

$$N_{QD}^{strt} = \sqrt{\frac{NR_{req}R_{hit}}{2K_L}}. \tag{23}$$

Taking $K_L = 0.1$ (10 percent), $R_{req} = 0.1$, and $N = 100$, Table 3 gives the N_{QD}^{strt} for different hit ratio values.

After the first QD is assigned, it will have a complete list of the nodes' scores, number of hops from the routing table, and cache sizes, all sorted by score. This first QD will

TABLE 3
Values for the Starting Values of N_{QD} for a Uniform Cache Size

$R_{hit} \rightarrow$	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
$N_{QD}^{strt} \rightarrow$	4	5	5	6	6	7	7	7

estimate the network size by multiplying the number of hops by the transmission range (r_0). It then estimates the maximum achievable hit ratio by summing the node cache sizes and dividing by the size cost of caching an item $(2 \times \text{average query size} + \text{average result size})$ and then by the total number of queries (n_q). Using this data, the node then calculates $N_{QD}^{\max}$ in accordance with (19) and N_{QD}^{strt} using (23) and then sets N_{QD} to the smaller of the two (at low hit ratios $N_{QD}^{\max}$ may be smaller). Finally, it sends a QDAP to the other $N_{QD}^{strt} - 1$ nodes at the top of the score list using MDPF and broadcasts a CIP to the network to inform the nodes about the list of QDs.

4.4 Network Traffic and Bandwidth Consumption

The traffic generated by the COACS system may be categorized into two types: traffic that is due to submitting requests and getting results and traffic that results from performing system management functions (e.g., replacing disconnected QDs). The second type requires an estimate of the node disconnection rate, and hence, we start this section with a study to derive an approximation of this rate. Next, we investigate the overall traffic generated in the network and compare it to the case of Cache Path, Cache Data, and no caching. Finally, we derive expressions to estimate the bandwidth consumption per node.

4.4.1 Expected Node Disconnection Rate

In this study, we assume that nodes will leave the network mostly because of mobility, and the network has a sufficient number of nodes to stay internally connected. Hence, nodes leave the network only if they are at the edge. We base the analysis on the random waypoint (RWP) mobility model that is used in the network simulator software ns-2. We will therefore refer to the movement epoch, or simply epoch, which is a random movement from one point to another at a constant speed and in a random direction. In [3], the probability density function (pdf) and the expected value of the length L of an epoch for the RWP model in an area of size $a \times b$ were derived. The pdf is identical to that in (4), while the $E(L)$ for a square area is equal to $0.521a$. Using these derivations, we calculate the expected number of nodes that will disconnect per second, as illustrated in Appendix C, and obtain

$$E[N_{disc}] = \frac{N \times P_d}{E(L)/v}, \tag{24}$$

where N is the total number of nodes, v is the constant speed of the epoch movement, and P_d is the probability that a node will exit the network. As an example, for $a = 1{,}000$, $r_0 = 100$, and $P_d = 0.175$, then, given 100 nodes and an average speed of 2 m/s, the expected time of one movement epoch is about 260 seconds, and the expected number of nodes that will leave the network per second (disconnection rate) is 0.067.

TABLE 4
Packet Fields and Their Sizes

Field	Description	Value
thu	Unicast headers (TCP+ IP+ 802.11 MAC)	74 bytes
thb	Broadcast headers (UDP+ IP+ 802.11 MAC)	62 bytes
pt	Packet type (packet field)	1 byte
id	Node address (packet field)	4 bytes
csize	Node cache size in KB (packet field)	2 bytes
sco	Node score (packet field)	1 bytes
S_Q	Query size	200 bytes
S_R	Result size	10 Kbytes

4.4.2 Overall Network Traffic

We start by defining the different packets fields and their sizes in Table 4. The packet header sizes for IP, TCP, UDP, and 802.11 are 20, 20, 8, and 34 bytes, respectively. These fields are used to derive the sizes of the messages used in COACS and the number of times these messages are sent during a given time (Table 5). Moreover, this data is also used to compute the number of requests and replies in Cache Path, Cache Data, and No Caching (Table 6).

With respect to the data in Table 5, the terms n_{req} and T_{total} denote the total number of requests per node and the network lifetime, respectively, while P_{full}^{QD} and P_{full}^{CN} represent the average probabilities that a QD's cache and a CN's cache are full. It is noted that P_{full} is a dynamic value that changes, for example, when queries are cached in QDs or when a new QD is added. The term $1/(1 - P_{full}^{QD})$ is an approximation of $\sum_{n=0}^{N_{QD}} \left(P_{full}^{QD}\right)^n$, which is one plus the probability that the first contacted QD is full and the probability that both the first and second QDs are full, and so on, including the probability that all QDs are full. This expression is used to determine the number of transmissions of the QCRP to cache a single query. This packet is sent initially by the CN and then from one QD to another until the query is cached or all QDs are checked. The term $(P_{full})^{N_{QD}}$ alone is the probability that all QDs are full and is used to determine the number of times a QDAP is sent. Next, the term $E[QD]$ is the expected number of traversed QDs to get a hit and is equal to $E[H_{QDdata}]/E[H]$. On the other hand, when N_{QD} is divided by N, it gives the probability that a node is a QD, while when the number of cached queries in the system $n_q \times R_{hit}$ is divided by N_{QD}, it yields the average number of queries that a QD caches.

If we multiply the entries in the third and fourth columns for each packet type in Table 5 and take the sum, we get an estimate of the total network traffic during T_{total}. This and the data in Table 6 were plugged into Matlab to generate plots of the traffic for COACS, CachePath, CacheData, and No Caching. We consider a 1 Km^2 area with 100 nodes generating 100 requests each. For CAOCS, N_{QD} was varied with the hit ratio while considering both $N_{QD}^{\max}$ and N_{QD}^{strt}. Taking $T_{total} = 3{,}000$ seconds, we can then use the results in Section 4.4.1 to compute the expected number of times nodes go offline and get a value that is very close to 200 (most of, if not all, the nodes that leave the network will rejoin soon after or eventually, and hence, the network will not drain out of nodes). Assuming that $P_{full}^{QD} = P_{full}^{CN} = 0.7$

TABLE 5
COACS Packet Sizes and the Number of Times Sent during the Network Lifetime

Packet	Content	Size (bytes)	Number of times Sent
HELLO	ID	$thb+pt+id$	N^2
CSP	List of IDs, scores, and cache sizes	$thb+pt+N(id+sco+csize)$	$(N_{QD}^{strt} + N\times n_{req}\times (P_{full}^{QD})^{N_{QD}} \times(1\text{-}R_{hit}))\times E[H]$
QDAP	None	$thu+pt$	$(N_{QD}^{strt} + N\times n_{req}\times (P_{full}^{QD})^{N_{QD}} \times(1\text{-}R_{hit}))\times E[H]$
CIP	QD list	$thb+pt+N_{QD}\times id$	$N\times(N_{QD}+T_{total}\times E[N_{disc}]\times N_{QD}/N)$
DRP	ID, query, and QDvisit list	$thu+pt+id+id\times E[QD]+S_Q$	$N\times n_{req}\times(R_{hit}\times(E[H_{QDdata}]+1) + (1\text{-}R_{hit})\times(E[H_{QDlast}]+ E[H_{AP}]))$
DREP	Query, result	$thu+pt+S_Q+S_R$	$N\times n_{req}\times(R_{hit}\times E[H]+(1\text{-}R_{hit})\times E[H_{AP}])$
QCRP	ID, query, and QDvisit list	$thu+pt+id+S_Q+id\times(1/(1\text{-} P_{full}^{QD}))$	$N\times n_{req}\times(1\text{-}R_{hit})\times(1/(1\text{-} P_{full}^{QD}))\times E[H]$
	ID, queries	$thu+pt+id+((n_q\times R_{hit})/(N \times N_{QD}))\times S_Q$	$T_{total}\times E[N_{disc}]\times E[H]$
CACK	ID, query	$thu+pt+ id+S_Q$	$N\times n_{req} \times(1\text{-}R_{hit})\times E[H]$
EDP	ID, query	$thu+pt+ id+S_Q$	$N\times n_{req}\times(1\text{-}R_{hit})\times E[H]\times P_{full}^{CN}$

(70 percent cache full), Fig. 6 compares the overall traffic for the four systems and presents the control packet overhead for COACS.

In the figure, the curves that correspond to N_{QD}^{strt} and $N_{QD}^{\max}$ represent the lower and upper bounds for the overall traffic, respectively. As shown, with N_{QD}^{strt} QDs, the traffic for COACS is higher than that of CachePath but gets closer to it as R_{hit} increases. Moreover, as we shall observe from the simulation results, COACS operates at a R_{hit} close to 0.8 (given the availability of cache capacity), while in CachePath, R_{hit} can get up to 0.4. Thus, the effective network traffic generated under COACS is less than that generated in a CachePath system. Relative to Fig. 6b, all packets except DRP and DREP are considered control packets.

4.4.3 Network Traffic per Node

To estimate the bandwidth consumption per node, we multiply the average packet size by the rate of the packets sent, received, or passing through a specific node. The traffic induced by the setup packets, namely, HELLO, CIP, and QDAP, are excluded since they are only sent at system start-up or when a QD is added/replaced, and therefore, they have transient effects. The main packet types involved in calculating the average traffic per node are shown in Table 7, together with the sending and receiving rates for a CN or RN and the added rates for a QD. The traffic on a QD is the sum of both because a QD also requests and caches queries just like any other node.

TABLE 6
Number of Times a Packet Is Sent in No Caching, CachePath, and CacheData

System	Request	Reply
No Caching	$E[H_{AP}]$	$E[H_{AP}]$
Cache Data	$R_{hit}\times E[H]+ (1\text{-}R_{hit})\times E[H_{AP}]$	$R_{hit}\times E[H]+ (1\text{-}R_{hit})\times E[H_{AP}]$
Cache Path	$R_{hit}\times 2E[H]+(1\text{-}R_{hit})\times E[H_{AP}]$	$R_{hit}\times E[H]+ (1\text{-} R_{hit})\times E[H_{AP}]$

The above expressions do not include forwarding traffic. To account for it, we calculate the overall traffic as above but replace n_{req} with R_{req} (RR per node) and exclude the originating and receiving node traffic (by subtracting one from the values of $E[H]$, $E[H_{AP}]$, $E[H_{QDdata}]$, and $E[H_{QDlast}]$) and then divide the result by the total number of nodes. The average forwarding traffic per node versus N_{QD} is shown in Fig. 7 for different hit ratios.

The average bandwidth consumption per node can be calculated by multiplying the values in Table 7 by their respective packet sizes, summing the results, and then adding the forwarding traffic. The average traffic at a QD and at a CN plus the penalty that a QD pays are all plotted in Fig. 8 while setting the RR per node to 9 requests per minute $(R_{req} = 0.15)$. Figs. 9a and 9b, on the other hand, show the effect of varying the RR on the bandwidth.

Finally, Fig. 9c illustrates the impact of changes in the number of nodes on the average traffic at a QD node. As illustrated in this graph, this traffic increases linearly with the increase of N. The increase in traffic is justified since more nodes will be submitting requests, and these requests must be forwarded through the existing network nodes, which will increase the traffic on each node and, consequently, on the whole network.

5 Performance Evaluation

To assess the performance of COACS and compare it to other systems, namely, CachePath and CacheData, all three systems were implemented using the ns-2 software with the CMU wireless extension [16]. The underlying ad hoc routing protocol chosen was DSDV, which is suitable for MDPF. However, we also present a scenario implemented using the AODV routing protocol. The wireless bandwidth and the transmission range were assumed to be 2 Mbps and 100 m, respectively, while the topography size was set to $1{,}000 \times 1{,}000$ m, and the AP that connects the database to the wireless system was placed near one of the corners. The

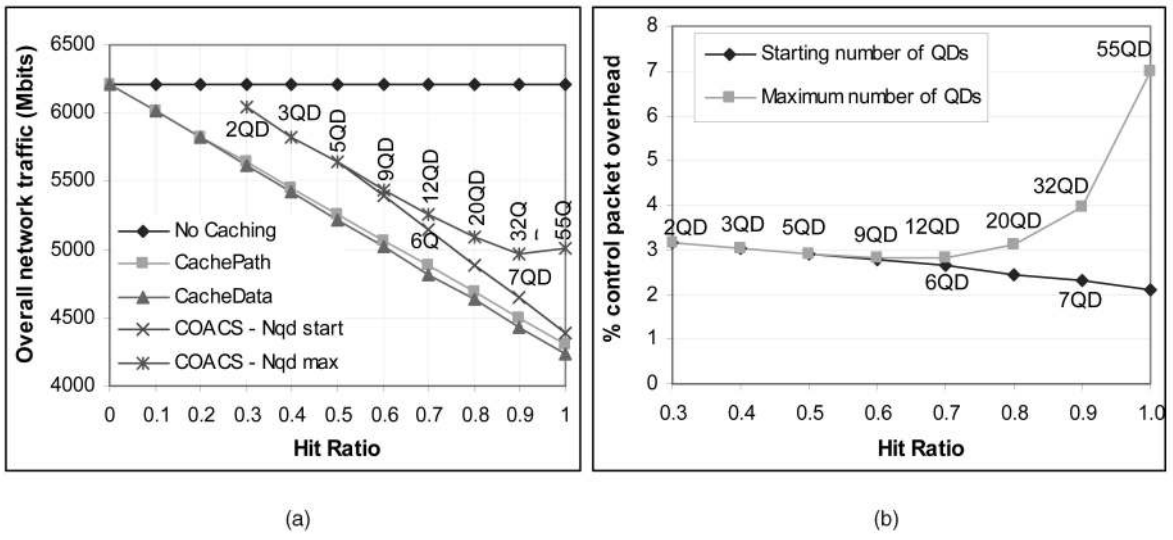


Fig. 6. (a) Total traffic generated during 3,000 seconds. (b) Control packet overhead.

TABLE 7
COACS Packet Send/Receive Rate per Node

Packet	Rate for RN/CN		Added Rate for QDs	
	Send	Receive	Send	Receive
DRP	R_{req}		$(N\times R_{req})\times (R_{hit}\times E[QD]/N_{QD}+(1\text{-}R_{hit}))$	$(N\times R_{req})\times (R_{hit}\times E[QD]/N_{QD}+(1\text{-}R_{hit}))$
DREP	$R_{req}\times R_{hit}$	$R_{req}+R_{req}\times R_{hit}$	$N\times R_{req}\times R_{hit}/N_{QD}$	
QCRP	$R_{req}\times(1\text{-}R_{hit})$			$N\times R_{req}\times(1\text{-}R_{hit})/N_{QD}$
CACK		$R_{req}\times(1\text{-}R_{hit})$	$N\times R_{req}\times(1\text{-}R_{hit})/N_{QD}$	
EDP	$R_{req}\times(1\text{-}R_{hit})\times P_{full}$			$N\times R_{req}\times(1\text{-}R_{hit})\times P_{full}^{CN}/N_{QD}$

nodes were randomly distributed in the topography and followed the RWP movement model. We set the minimum and maximum speeds of the nodes ($V_{\min}$ and $V_{\max}$) to 0.01 and 2.00 m/s, respectively, and the pause time to 100 seconds. However, in order to study the effect of high mobility on the network, we present a scenario with $V_{\min}$ set to 10 m/s and $V_{\max}$ to 20 m/s, and accordingly, the average velocity V_{avg} was found to be equal to 13.8 m/s. Finally, the delay at the data source link (T_{out}) was set to 40 ms, which is relatively low by Curran and Duffy's [8] standards. The rest of the simulation parameters are listed with their values in Table 8 and are chosen so as to run the different caching systems on scenarios that were as identical as possible.

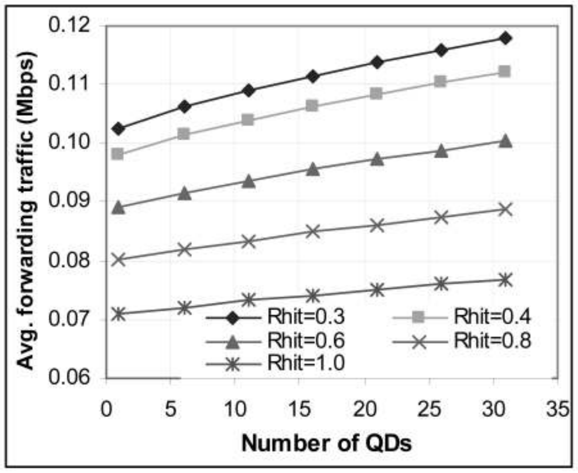


Fig. 7. Average forwarding traffic passing by a node in a COACS system.

In order to calculate the number of runs required for achieving at least a 90 percent confidence level, we ran a scenario with default parameter values 10 times. For each simulation run, the pseudorandom generator seed (based on the clock of the *ns*-2 simulator's scheduler) and the node movement file were changed. The average hit ratio and delay of the system were computed starting from $T = 500$ seconds, and the mean and standard deviation for each set were calculated. Next, the number of runs required to achieve the 90 percent confidence was computed using the central limit theorem, as discussed in [1]. The $+/-$ precision values for the hit ratio and the delay were chosen as 0.2 and 10 ms, respectively (approximately 20 percent of the measured maximum value). The required number of runs was found to be equal to nine for the hit ratio and four for the delay. As a consequence, we computed the result of each scenario from the average of nine simulation runs.

In the simulation, each node sends a request every 10 seconds selected from 10,000 different ones following a biased strict Zipf-like access pattern [20], which has been used frequently to model nonuniform distributions [5]. In Zipf's law, an item ranked i $(1 \leq i \leq n_q)$ is accessed with probability $1/\left(i^{\theta}\sum_{k=1}^{n_q} 1/k^{\theta}\right)$, where θ ranges between

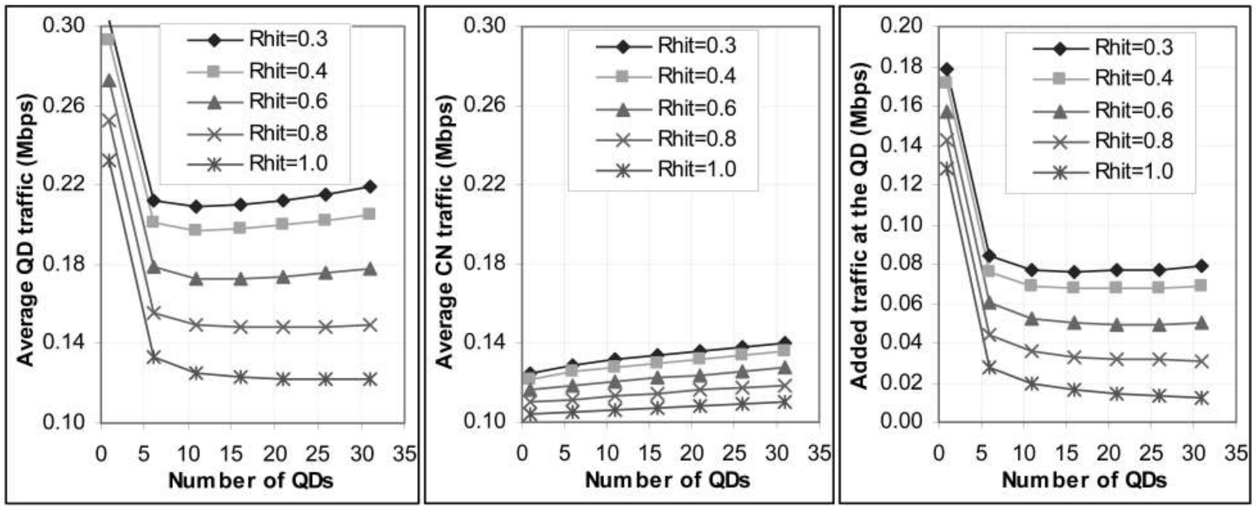


Fig. 8. Average bandwidth consumption for a QD, CN, and penalty for being a QD.

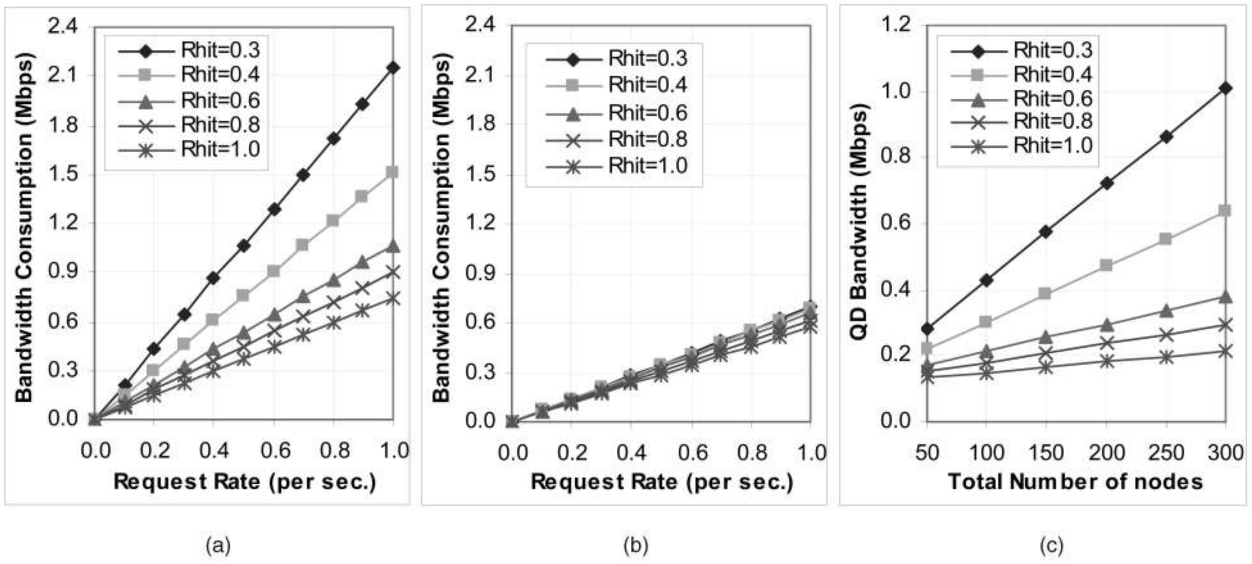


Fig. 9. Average bandwidth consumption versus RR for (a) a QD and (b) a CN and (c) the effect of increasing the number of nodes on the QD bandwidth.

zero (uniform distribution) and one (strict Zipf distribution). The access pattern is also location dependent in the sense that nodes around the same location tend to access similar data (i.e., have similar interests). For this purpose, the square area was divided into 25 zones 200 m $\times$ 200 m each. Clients in the same zone follow the same Zipf pattern, while nodes in different zones have different offset values. For instance, if a node in zone i generated a request for data id following the original Zipf-like access pattern, then the new id would be set to $(id + n_q \bmod (i)) \bmod (n_q)$, where n_q is the database size. This access pattern can make sure that nodes in neighboring grids have similar, although not the same, access pattern. The time-out mechanism for COACS was implemented as follows: After each node sends a request, it waits for $T = 1$ second before sending the same request again if it does not receive an answer within T, then it waits for another T second, and so on. After 10 seconds, the node checks if it has not received an answer for this request; if it has not, it considers this request as failed and generates a new request. The percentage of successfully answered requests is stated in Section 5.1.

TABLE 8
Parameters Used in the Simulations

Parameter	Value
Database size, n_q	10,000 items
query size, S_Q	512 bytes
result size, S_R	10 Kb
Client cache size	200 Kb
Number of nodes	100
Simulation time	2000 sec

The starting number of QDs was set to seven in accordance with the above study and were randomly distributed in the topography. The times taken to reply to the requests, in addition to other relevant results, were logged and were used to generate the output that is discussed next.

5.1 Hit Ratio

The effective cache size in CachePath is not the sum of the caching capacity of all the nodes, since significant amounts

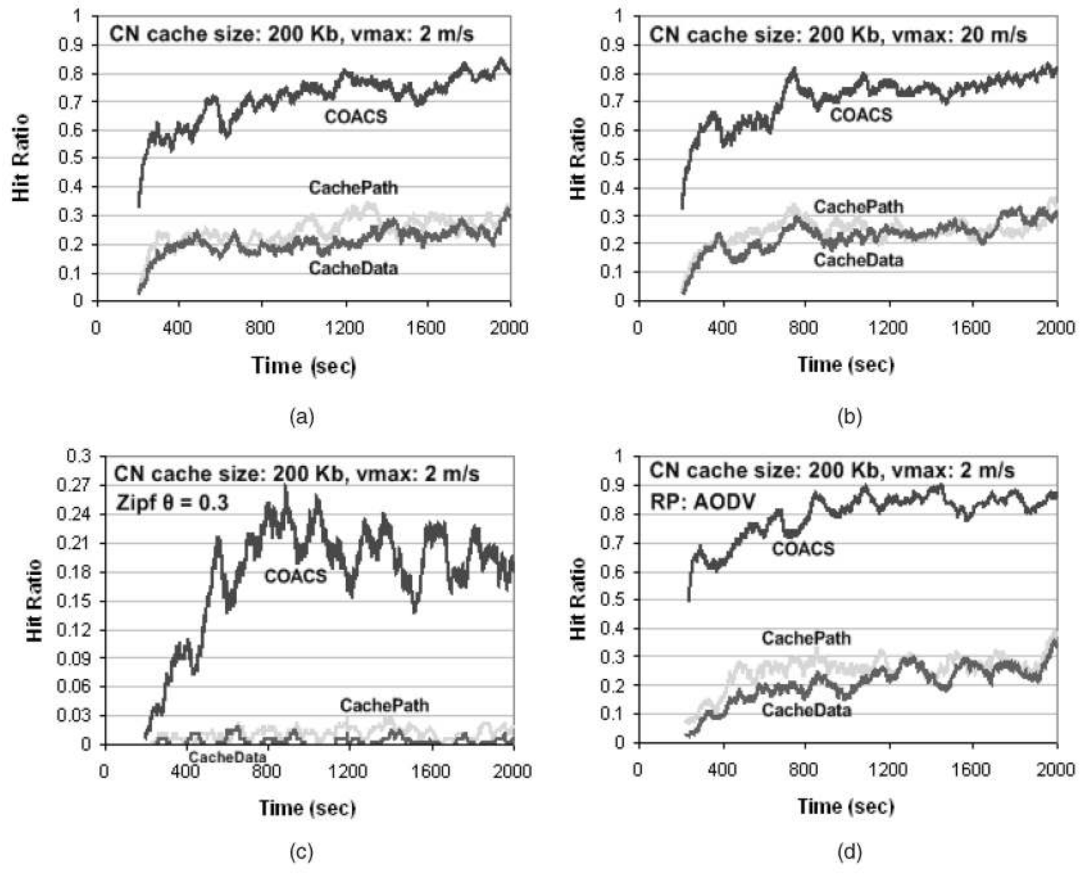


Fig. 10. Hit ratio for COACS, CachePath, and CacheData versus time in four scenarios.

of redundancy can easily occur in the system. Additionally, for a request to be found using this system, it must "accidentally" pass through a node that is caching the path to the data or a miss will occur. On the other hand, because there is coordination between the QDs in COACS, redundancy is eliminated, and the effective caching capacity is the total cache size of all the nodes. It can be concluded that for the same amount of resources, COACS uses these resources more efficiently, thus achieving a much higher hit ratio compared to CachePath and CacheData. This is shown in Fig. 10 and is illustrated using four scenarios.

The first scenario corresponds to Fig. 10a, which represents our normal simulation setup. The second scenario is depicted in Fig. 10b, which shows that the hit ratio of the three systems slightly decreases before it stabilizes at the end of the simulation. In the third scenario (Fig. 10c), the Zipf parameter θ was set to 0.3. In this scenario, all data items are requested with high probabilities. Since the total number of requests made during the simulation time is 10,000, which is equal to the number of data items, it is less likely to find a new request in the cache, and hence, the hit ratio significantly decreases. Last, the routing protocol was changed to AODV in the fourth scenario. Although the hit ratio did not change as compared to the first scenario, the percentage of answered queries dropped for the three systems. Using DSDV, the percentage of answered queries was between 80 percent and 88 percent for COACS and 79 percent and 91 percent for CachePath and CacheData. Using AODV, on the other hand, the percentage of answered queries was 77 percent for COACS, 82 percent for CachePath, and 78 percent for CacheData.

5.2 Average Delay

Fig. 11 compares the average delay of COACS to that of No Caching, CachePath, and CacheData for the same four scenarios discussed above. In all scenarios, COACS clearly outperforms all three systems. As shown in all graphs, the delay of all systems follows a decreasing trend (due to increasing hit ratio) until nodes start moving after 500 seconds from the start of the simulation. It can be observed that at higher mobility, the delay of COACS and CachePath increases. Moreover, CachePath performs better than CacheData at low mobility, which is consistent with the results presented in [19]. The delay of the three systems is approximately the same when the Zipf parameter θ was set to 0.3. Finally, when AODV is used, the delay of all systems significantly increases since AODV requires an overhead delay for discovering new routing paths when they are needed.

5.3 Varying the Cache Size and the Request Rate

The default cache size of a node was set to 200 Kbit, thus giving a total cache size in the network of approximately 20 Mbit, in contrast to 100 Mbit for the database size. Fig. 12 shows the effect of varying the cache size of a node on the hit ratio and delay of the three systems. The average hit ratio of CachePath and CaheData increases with increasing cache size, while that of COACS does not change because

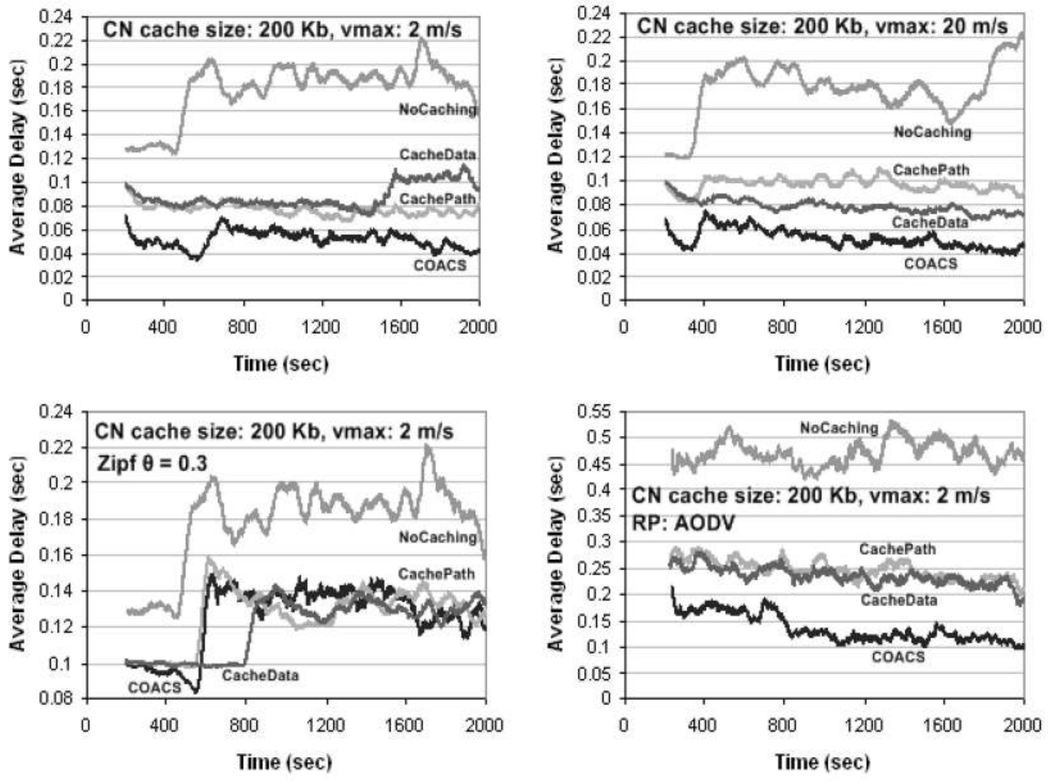


Fig. 11. Average delay for No Caching, CachePath, CacheData, and COACS.

the effective hit ratio in COACS depends on the total cache size of QDs. The average delay of the three systems decreases with increasing cache size because fewer cache replacements are encountered.

Fig. 13 shows the behavior of the hit ratio, delay, and total network traffic in response to varying the RR, which was varied between 0.67 and 6 requests per minute. The average hit ratio of the three systems increases with increasing RR because more requests are cached. The delay of COACS decreases with increasing RR up to 1.5 requests per minute, where it starts increasing again. At low RRs, CacheData has less delay than CachePath but the situation is reversed starting from an RR of 2 requests per minute, where CachePath starts to perform better. Finally, Fig. 13c shows the overhead traffic of COACS that is caused by the control packets and request forwarding.

6 Concluding Remarks and Future Work

This paper presented a novel architecture for caching database data in MANETs that works by caching queries in special nodes (QDs) and using them as indexes to their responses. A key feature of the system is its ability to increase the hit ratio rapidly as more requests for data are submitted. Lower and upper bounds were derived for the number of QDs to control the load on these nodes and the system response time. The design assumed a proactive routing protocol and relied on updates to the routing table for detecting nodes leaving and coming into the network. However, it can be easily adapted to reactive protocols like AODV through the addition of few messages to invoke certain system maintenance functions. Simulations results showed that the system performs significantly better than other recently proposed systems in terms of achieving

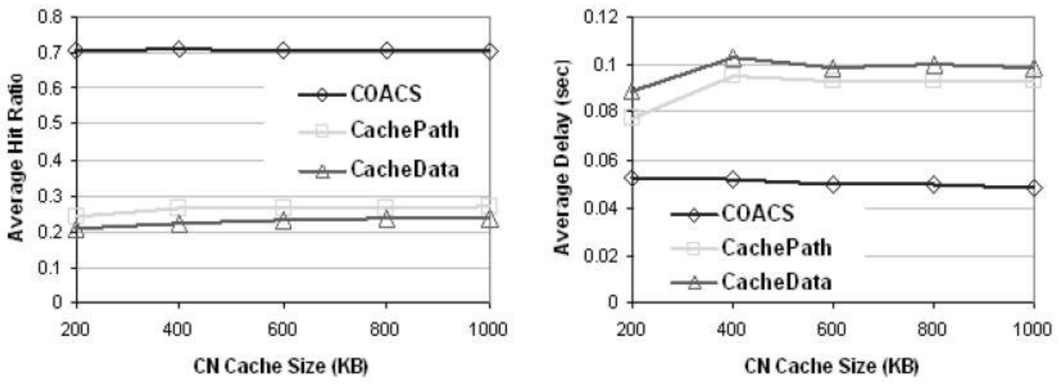


Fig. 12. Average hit ratio and delay for the three systems when varying the CN cache size.

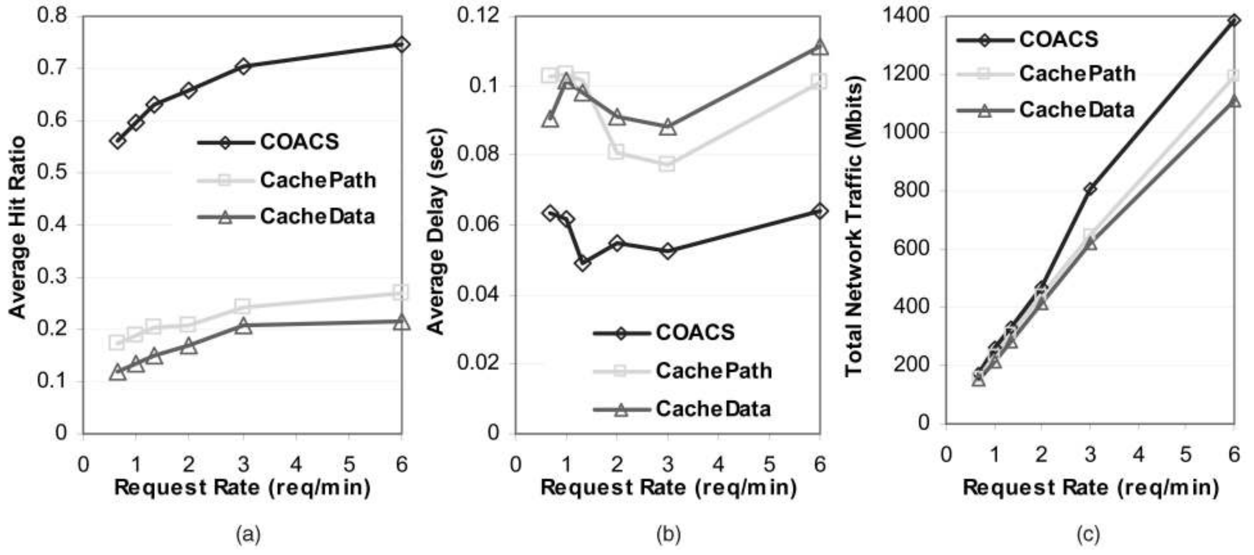


Fig. 13. (a) Average hit ratio, (b) delay, and (c) total network traffic for different RRs.

better hit ratios and smaller delays but at the cost of a slightly higher bandwidth consumption.

This research focused on the design of the COACS system and analyzed its performance. There are, however, several enhancements that can be introduced for improving the system's performance and reliability. At the top of the list are cache invalidation, cache replication, partial cache reuse, and the graceful shutdown of devices. Cache invalidation aims to keep the cache consistent with the data source, and for this, the design of COACS allows for improving on current methods that use the invalidation report (IR)-based cache management scheme [7], [18]. Instead of broadcasting the IRs to all nodes, the server could just send them to the QDs, and these could in turn disseminate them to the concerned CNs. This will notably cut down on network traffic and make the update process more efficient. As to replication, a simple algorithm could be implemented in which a QD that gets a request for caching a query would send a QCRP to a distant QD that is more than a certain number of hops away. Moreover, the result of the query may also be replicated by having selected RNs that request such data become CNs even if the data comes from the cache. This would distribute the replicas as much as possible in the network and would help in reducing the average delay. With partial cache reuse, the system would make more use of the data in the cache and reduce the network traffic with the external data source. Semantic caching, in which the client maintains in the cache both semantic descriptions and the results of previous queries [17], [12], could be applied by allowing a query to be answered partially or completely across the QDs. Finally, relating to graceful shutdown, if a device outage can be predicted, an attempt could be made to back up part or all of the data before it is lost.

APPENDIX A
DERIVING THE EXPECTED DISTANCE TO THE ACCESS POINT

The solution for calculating the expected number of hops to reach the AP is the outcome of using tables of integration and the *Mathematica* software:

$$\begin{aligned}
E[S_{AP}] &= \frac{1}{a^2}\int_0^a \frac{1}{2}\Big(x\sqrt{x^2+y^2}+\log\Big(x+\sqrt{x^2+y^2}\Big)\Big]_0^a\Big)dy\\
&= \frac{1}{a^2}\int_0^a \frac{1}{2}\Big(a\sqrt{a^2+y^2}+\log\Big(a+\sqrt{a^2+y^2}\Big)-\log|y|\Big)dy\\
&= \frac{1}{2a^2}\Bigg(\frac{2}{3}ay\sqrt{a^2+y^2}-\frac{1}{3}y^3\log(y)\\
&\qquad +\frac{1}{3}y^3\log\Big(a+\sqrt{a^2+y^2}\Big)\\
&\qquad +\frac{1}{3}a^3\log\Big(y+\sqrt{a^2+y^2}\Big)\Big]_0^a\Bigg)\\
&= \frac{1}{2a^2}\left(\frac{2}{3}a^2\sqrt{2a^2}-\frac{1}{3}a^3\log(a)+\frac{2}{3}a^3\log\Big(a+\sqrt{2a^2}\Big)\right)\\
&\qquad -\frac{1}{2}\left(\frac{2}{3}a^3\log(a)\right)\\
&= \left(\frac{1}{3}\sqrt{2a^2}-\frac{1}{3}a\log(a)+\frac{1}{3}a\log\Big(a+\sqrt{2a^2}\Big)\right)\\
&= \frac{1}{3}\Big[a\sqrt{2}-a\log(a)+a\log a\Big(1+\sqrt{2}\Big)\Big]\\
&= \frac{1}{3}\Big[\sqrt{2}+\log\Big(1+\sqrt{2}\Big)\Big]a = 0.7652\,a.
\end{aligned}$$

APPENDIX B
CALCULATING THE LOAD RATIO ON EACH QD

When calculating the load ratio on QD_i (λ_i), all possible positions of QD_i should be taken into account, since the list of QDs may be accessed in any order. For this purpose, we define the function $P_A(a_i|n)$, which is the probability that QD_i will be accessed (or have a request forwarded to) given that it is in position n (where $1 \leq n \leq N_{QD}$). As explained in Section 4.2, this probability depends on the cache size of all nodes that follow QD_i. However, since the next nodes are considered to be random, an expected total cache size must be determined. Now, since there is no a priori knowledge of the positions of each of the other nodes in the sequence, their size is estimated using the expected cache size of other nodes. This is determined as follows (N stands for N_{QD}):

$$E[C|C_i] = \frac{C_{total} - C_i}{N-1}.$$

We then multiply this value by the number of nodes that follow QD$_i$ and add C_i to get the total expected cache size of node QD$_i$, as well as all the nodes that follow it. Dividing the resultant value by the total cache size of the system gives us $P_A(a_i|n)$ as follows:

$$\begin{aligned} P_A(a_i|n) &= \left(C_i + \frac{(N-n)(C_{total} - C_i)}{N-1}\right)\frac{1}{C_{total}} \\ &= \frac{(n-1)C_i + (N-n)C_{total}}{(N-1)C_{total}}. \end{aligned}$$

Finally, since the position of QD$_i$ is assumed to be uniformly random, the probability of it being accessed (λ_i) is given by taking the average of $P_A(a_i|n)$ for all values of n:

$$\begin{aligned} P_A(a_i) = \lambda_i &= \frac{1}{N}\sum_{n=1}^{N_{QD}} P_A(a_i|n) \\ &= \frac{1}{N}\sum_{n=1}^{N_{QD}} \frac{(n-1)C_i + (N-n)C_{total}}{(N-1)C_{total}} \\ &= \frac{(0+1+\ldots+(N-1))C_i}{N(N-1)C_{total}} \\ &\quad + \frac{((N-1)+(N-2)+\ldots+1+0)}{N(N-1)} \\ &= \frac{(0+1+\ldots+(N-1))(C_i + C_{total})}{N(N-1)C_{total}} \\ &= \frac{N(N-1)(C_i + C_{total})}{2N(N-1)C_{total}} = \frac{1}{2} + \frac{C_i}{2C_{total}} = \frac{1+P_i}{2}. \end{aligned}$$

The value of λ_i is modified to account for the hit ratio (all nodes will be accessed upon a miss):

$$\lambda_i = R_{hit}\frac{1+P_i}{2} + (1 - R_{hit}) = R_{hit}\frac{P_i - 1}{2} + 1.$$

Appendix C

Calculating the Expected Node Disconnection Rate

A node that is at the edge of the network will surely disconnect if it moves away from the network and travels a distance that is at least r_0. The probability that a node is at the edge of the network (with an $a \times b$ area) can be shown to be $4(ar_0 - r_0^2)/(ab)$, while the probability of moving away from the network has an upper bound of 0.5 (in [3], it is reported that nodes starting from the edge tend to move back toward the middle of the area). Finally, the probability of moving a distance r_0 is $P(L > r_0)$, and for an $a \times a$ area, it is obtained from the pdf as follows:

$$\begin{aligned} P(L > r_0) &= 1 - \int_0^{r_0} \frac{4l}{a^4}\left(\frac{\pi a^2}{2} - 2al + 0.5l^2\right)dl \\ &= 1 - \frac{\pi r_0^2}{a^2} + \frac{8r_0^3}{3a^3} - \frac{r_0^4}{2a^4}. \end{aligned}$$

Now, we can define the probability that a node will disconnect (exit the network) as

$$P_d = \frac{2(ar_0 - r_0^2)}{ab}\left(1 - \frac{\pi r_0^2}{a^2} + \frac{8r_0^3}{3a^3} - \frac{r_0^4}{2a^4}\right).$$

Then, we compute the expected number of nodes that will disconnect per second by dividing P_d by the expected time of one movement epoch and multiplying by the total number of nodes:

$$E[N_{disc}] = \frac{N \times P_d}{E(L)/v}.$$

References

[1] T. Andrel and A. Yasinsac, "On Credibility of Manet Simulations," *Computer*, pp. 48-54, 2006.

[2] H. Artail, H. Safa, and S. Pierre, "Database Caching in Manets Based on Separation of Queries and Responses," *Proc. IEEE Int'l Conf. Wireless and Mobile Computing, Networking and Comm. (WiMob '05)*, pp. 237-244, Aug. 2005.

[3] C. Bettstetter, H. Hartenstein, and X. Perez-Costa, "Stochastic Properties of the Random Waypoint Mobility Model: Epoch Length, Direction Distribution, and Cell Change Rate," *Proc. Fifth ACM Int'l Workshop Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM '02)*, pp. 7-14, Sept. 2002.

[4] C. Bettstetter and J. Eberspacher, "Hop Distances in Homogeneous Ad Hoc Networks," *Proc. 57th IEEE Vehicular Technology Conf. (VTC-Spring '03)*, vol. 4, pp. 2286-2290, Apr. 2003.

[5] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-Like Distributions: Evidence and Implications," *Proc. IEEE INFOCOM '99*, pp. 126-134, 1999.

[6] J. Broch, D. Maltz, D. Johnson, Y. Hu, and J. Jetcheva, "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols Source," *Proc. ACM MobiCom '98*, pp. 85-97, 1998.

[7] G. Cao, "A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments," *IEEE Trans. Knowledge and Data Eng.*, vol. 15, pp. 1251-1265, 2003.

[8] K. Curran and C. Duffy, "Understanding and Reducing Web Delays," *Int'l J. Network Management*, vol. 15, no. 2, pp. 89-102, 2005.

[9] P. Gupta and P. Kumar, "The Capacity of Wireless Networks," *IEEE Trans. Information Theory*, vol. 46, no. 2, pp. 388-404, 2000.

[10] A. Idris, H. Artail, and H. Safa, "Query Caching in Manets for Speeding Up Access to Database Data," *Proc. Third Int'l Symp. Telecomm. (IST '05)*, pp. 987-992, Sept. 2005.

[11] W. Lau, M. Kumar, and S. Venkatesh, "A Cooperative Cache Architecture in Supporting Caching Multimedia Objects in Manets," *Proc. Fifth Int'l Workshop Wireless Mobile Multimedia (WoWMoM)*, 2002.

[12] K. Lee, H. Leong, and A. Si, "Semantic Query Caching in a Mobile Environment," *Mobile Computing and Comm. Rev.*, vol. 3, no. 2, pp. 28-36, 1999.

[13] S. Lim, W. Lee, G. Cao, and C. Das, "A Novel Caching Scheme for Internet Based Mobile Ad Hoc Networks Performance," *Ad Hoc Networks*, vol. 4, no. 2, pp. 225-239, 2006.

[14] S. Lyer, A. Rowstron, and P. Druschel, "Squirrel: A Decentralized Peer-to-Peer Web Cache," *Proc. 21st ACM Symp. Principles of Distributed Computing (PODC)*, 2002.

[15] R. Malpani, J. Lorch, and D. Berger, "Making World Wide Web Caching Servers Cooperate," *World Wide Web J.*, vol. 1, no. 1, 1996.

[16] *NS-2 Simulator*, http://www.insi.edu/nsnam/ns, Apr. 2002.

[17] Q. Ren, M. Dunham, and V. Kumar, "Semantic Caching and Query Processing," *IEEE Trans. Knowledge and Data Eng.*, vol. 15, no. 1, pp. 192-210, 2003.

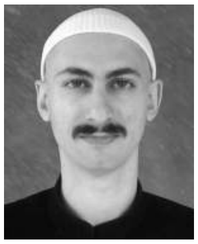
[18] K. Tan, J. Cai, and B. Ooi, "Evaluation of Cache Invalidation Strategies in Wireless Environments," *IEEE Trans. Parallel and Distributed Systems*, vol. 12, no. 8, pp. 789-807, 2001.

[19] L. Yin and G. Cao, "Supporting Cooperative Caching in Ad Hoc Networks," *IEEE Trans. Mobile Computing*, vol. 5, no. 1, pp. 77-89, 2006.

[20] G. Zipf, *Human Behavior and the Principle of Least Effort*. Addison-Wesley, 1949.

Hassan Artail received the BS and MS degrees (with high distinction) in electrical engineering from the University of Detroit in 1985 and 1986, respectively, and the PhD degree from Wayne State University in 1999. Before joining the American University of Beirut (AUB), Beirut, Lebanon, at the end of 2001, he was a system development supervisor at the Scientific Laboratories of DaimlerChrysler, where he worked for 11 years in the field of software and system development for vehicle testing applications. He is currently an associate professor in the Electrical and Computer Engineering Department, AUB, and is doing research in the areas of Internet and mobile computing, distributed systems, and mobile ad hoc networks. During the past six years, he has published more than 70 papers in top conference proceedings and reputable journals. He is a member of the IEEE.

Haidar Safa received the BSc degree in computer science from the Lebanese University, Lebanon, in 1991, the MSc degree in computer science from the University of Quebec at Montreal (UQAM), Canada, in 1996, and the PhD degree in computer and electrical engineering in 2001 from the Ecole Polytechnique de Montreal, Canada. He joined ADC Telecommunications, Shelton, Connecticut, in 2000 and SS8 Networks in 2001, where he worked on designing and developing networking and system software. In 2003, he joined the Department of Computer Science, American University of Beirut, Beirut, Lebanon, where he is currently an assistant professor. His main research interests include mobile and wireless networks, quality of service, and routing and network security. He is a member of the IEEE.



Khaleel Mershad received the BE degree (with high distinction) in computer engineering and informatics from Beirut Arab University, Lebanon, in July 2004, and the ME degree in computer and communications engineering from the American University of Beirut, Beirut, Lebanon, in February 2007. He is currently a PhD student in the Electrical and Computer Engineering Department, American University of Beirut. His research interests include mobile ad hoc networks, data management, knowledge discovery, and distributed computing.

Zahy Abou-Atme received the bachelor's degree in electrical engineering at the American University of Beirut, Beirut, Lebanon, in 2005 and the master's degree from the Technical University of Munich, Germany, in 2007. His master's thesis topic was on MIMO signal processing for next-generation VDSL technology with spatially colored alien noise while reasonably reducing complexity. He is currently with the Electrical and Computer Engineering Department, American University of Beirut. He is a student member of the IEEE.

Nabeel Sulieman received the degree in computer and communications engineering (minor in philosophy) from the American University of Beirut, Beirut, Lebanon, and the master's degree in communications engineering from the Technical University of Munich. He is currently a development engineer at Qimonda AG. His research interests include mobile networks, computer architecture, compiler theory, and bioinformatics. He is a student member of the IEEE.