Mobile Information Systems 3 (2007) 19–37
IOS Press

Cooperative caching in mobile ad hoc networks based on data utility

Narottam Chand[a,*], R.C. Joshi[b] and Manoj Misra[b]
[a]*Department of Computer Science and Engineering, National Institite of Technology Hamirpur, India*
[b]*Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, India*
E-mail: {joshifcc, manojfec}@iitr.ernet.in

Abstract. Cooperative caching, which allows sharing and coordination of cached data among clients, is a potential technique to improve the data access performance and availability in mobile ad hoc networks. However, variable data sizes, frequent data updates, limited client resources, insufficient wireless bandwidth and client's mobility make cache management a challenge. In this paper, we propose a utility based cache replacement policy, *least utility value* (*LUV*), to improve the data availability and reduce the local cache miss ratio. LUV considers several factors that affect cache performance, namely access probability, distance between the requester and data source/cache, coherency and data size. A cooperative cache management strategy, *Zone Cooperative* (*ZC*), is developed that employs LUV as replacement policy. In ZC one-hop neighbors of a client form a cooperation zone since the cost for communication with them is low both in terms of energy consumption and message exchange. Simulation experiments have been conducted to evaluate the performance of LUV based ZC caching strategy. The simulation results show that, LUV replacement policy substantially outperforms the LRU policy.

Keywords: Ad hoc networks, cooperative caching, cache replacement, multi-hop, admission control

1. Introduction

Recent explosive growth in computer and wireless communication technologies has led to the development of Mobile Ad hoc NETworks (MANETs) that are constructed only from mobile hosts. The interest in developing mobile wireless ad hoc networking solutions has been due to their flexibility, ease of deployment and potential applications such as battlefield, disaster recovery, outdoor assemblies, etc. Most of the previous researches [1–5] in MANETs focus on the development of dynamic routing protocols that can improve the connectivity among mobile hosts which are connected to each other by one-hop/multi-hop links. Although routing is an important issue in ad hoc networks, other issues such as data access are also very important since the ultimate goal of using such networks is to provide information access to mobile hosts [6]. One of the most attractive techniques that improve data availability is caching. In general caching results in (i) enhanced QoS at the clients – i.e., lower jitter, latency and packet loss, (ii) reduced network bandwidth consumption, and (iii) reduced data server/source workload. In addition, reduction in bandwidth consumption infers that a properly implemented caching architecture for a MANET environment can potentially improve battery life of mobile clients.

Caching has been proved to be an important technique for improving the data retrieval performance in mobile environments [15–18]. With caching, the data access delay is reduced since data access

*Corresponding author. Tel.: +91 94180 94345; Fax: +91 1972 223834; E-mail: nar@nitham.ac.in.

requests can be served from the local cache, thereby obviating the need for data transmission over the scarce wireless links. However, caching techniques used in one-hop mobile environment (i.e., cellular networks) may not be applicable to multi-hop mobile environments since the data or request may need to go through multiple hops. As mobile clients in ad hoc networks may have similar tasks and share common interest, cooperative caching, which allows the sharing and coordination of cached data among multiple clients, can be used to reduce the bandwidth and power consumption.

To date there are some works in literature on cooperative caching in ad hoc networks, such as consistency [6,7], placement [9,11,12], replacement [23,24], discovery [10,24] and proxy caching [13, 19–22]. However, most of the solutions have one or more limitations.

Cache management in mobile ad hoc networks, in general, includes the following issues to be addressed:

1. The cache discovery algorithm that is used to efficiently discover, select, and deliver the requested data item(s) from neighboring nodes. In a cooperative architecture, the order of looking for an item follows local cache to neighboring nodes, and then to the original server.
2. Cache admission control – this is to decide on what data items can be cached to improve the performance of the caching system.
3. The design of cache replacement algorithm – when the cache space is sufficient for storing one new item, the client places the item in the cache. Otherwise, the possibility of replacing other cached item(s) with the new item is considered.
4. The cache consistency algorithm which ensures that updates are propagated to the copies elsewhere, and no stale data items are present.

In this paper, we investigate the data retrieval challenge of mobile ad hoc networks and propose a novel scheme, called *Zone Cooperative* (*ZC*) for caching that exploits data utility value for cache replacement. The goal of ZC is to reduce the caching overhead and provide optimal replacement policy. Mobile clients belonging to the neighborhood (zone) of a given client form a cooperative cache system for this client since the cost for communication with them is low both in terms of energy consumption and message exchanges. In ZC caching, each mobile client has a cache to store the frequently accessed data items. The cache at a client is a nonvolatile memory such as hard disk. The data items in the cache satisfy not only the client's own requests but also the data requests passing through it from other clients. For a data miss in the local cache, the client first searches the data item in its zone before forwarding the request to the next client that lies on a path towards server. We also develop an analytical model for the ZC caching system to evaluate its performance. To improve the efficiency of ZC caching, a *Least Utility Value* (*LUV*) based replacement policy has been developed. Simulations prove that ZC caching with LUV achieves higher performance than ZC based on LRU replacement.

The rest of the paper is organized as follows. Section 2 reviews the related work. The network model and system environment are presented in Section 3. Section 4 describes the proposed ZC caching scheme for data retrieval. Section 5 describes the LUV cache replacement policy. Section 6 presents an analytical model of ZC. Section 7 is devoted to performance evaluation and presents detailed simulation results. Section 8 concludes the paper and discusses future work.

2. Related work

Caching is an important technique to enhance the performance of both wired and wireless network. A number of studies have been conducted to improve the caching performance in wireless mobile

environment [15–18]. Jayaputera and Taniar [25] have proposed to use an invalidation report mechanism in conjunction with caching of a CORBA (Common Object Request Broker Architecture) object in a mobile environment. The approach caches CORBA object by value and is capable to keep the object updated in a client side during disconnection. Client mobility hampers the caching performance in a mobile environment. It is found that as mobile clients move from one cell to another they experience increasing delay when accessing data items from their home location caches [26]. Lai et al. [26] have proposed two techniques to improve the performance of existing cache relocation methods for mobile networks. The first technique, 2PR, compensates for poor path prediction by temporarily moving data items to a common parent node prior to a handover. Items are moved to the correct destination once the client's new location has been confirmed. The second technique, ROLP, reduces the traffic overhead associated with cache relocation by ensuring duplicate items are not relocated and relocation of items are performed only from the nearest node to the destination.

Cooperative caching has been studied in the Web environment, but little work has been done to efficiently manage the cache in MANETs. Due to mobility and constrained resources (i.e., bandwidth, battery power and computational capacity) in wireless networks, cooperative cache management techniques designed for wired networks may not be applicable to ad hoc networks.

In the context of ad hoc networks, it is beneficial to cache frequently accessed data not only to reduce the average query latency but also to save wireless bandwidth. Hara [14] proposed several replica allocation methods to increase data accessibility and tolerate network partitions in MANETs. In these schemes, the replicated data are relocated periodically based on access frequency and overall network topology. Although replication can improve data accessibility, the overhead for relocating replicas periodically is significantly high. Due to updates at the server, the cost of maintaining the consistent copy of replicas is quite high. Papadopouli et al. [12] suggested the 7DS architecture, in which a couple of protocols are defined to share and disseminate information among users. It operates either on a prefetch mode, based on the information and user's future needs or on an on-demand mode, which searches for data items in a one-hop multicast basis. Depending on the collaborative behavior, a peer-to-peer (P2P) and server-to-client mode are used. Unlike our approach, this strategy focuses on data dissemination, and thus the cache management including cache admission control and replacement is not well explored. Sailhan and Issarny [20] proposed a cooperative caching scheme to increase data accessibility by P2P communication among mobile clients, when they are out of bound of a fixed infrastructure. It is implemented on top of Zone Routing Protocol (ZRP). The authors proposed a fixed broadcast range based on the underlying routing protocol. However, the mobile users' location, data popularity and network density often change in a real mobile environment, so the fixed broadcast scheme is hard to adapt to real mobile applications.

Lau et al. [13] proposed a cooperative caching architecture for supporting continuous media proxy caching. They introduced an application manager to transparently perform data location and session migration of continuous media streams among all proxy caches. Nuggehalli et al. [11] addressed the problem of optimal cache placement in ad hoc wireless networks and proposed a greedy algorithm, called POACH, to minimize the weighted sum of energy expenditure and access delay. S. Lim et al. [22] proposed cache invalidation techniques for Internet based MANETs (IMANETs). However, due to broadcast nature, the cost of maintaining strong cache consistency is very high in ad hoc networks as compared to cellular mobile networks. Authors in [2] proposed a cooperative caching scheme for IMANETs. A broadcast based simple search scheme is proposed to establish cooperation among all clients in the network to share cached data items. Although the broadcast based data search scheme can locate the nearest required data item, the energy and bandwidth cost of the flooding search is significantly high for a mobile ad hoc network. Shen et al. [8] proposed a broadcast based cooperative caching scheme

for hybrid networks where a client shares the caches of clients lying in its proximity. Bandwidth and energy consumption to locate a client having cached the requested data is very high due to flooding of the messages.

Yin and Cao [6,7,23] designed and evaluated three caching algorithms (CacheData, CachePath and Hybrid) to support data access in ad hoc networks. These algorithms mainly focus on the problem of choosing data item or data path for caching in the limited cache space of mobile nodes. The problem with CacheData approach is that it could take a lot of caching space with forwarding clients. The recording path could become obsolete in CachePath and this scheme could introduce extra processing overhead. Hybrid cache decides when to use which of the two schemes based on the properties (size and time-to-live (TTL)) of the passing-by data and requires that the forwarding clients will keep record of which data is more important. Since the forwarding clients can move to somewhere else in the next second, statistics collected by the forwarding clients does not help much. In [23], the authors also proposed a cache replacement policy based on data item size and popularity. But, the policy ignored the important factors such as TTL and distance of the requester from the cache/source satisfying the request. Zang et al. [9] talked of security concerns for cooperative caching in ad hoc networks.

Du and Gupta [24] proposed a cooperative caching scheme called COOP for MANETs with the aim to improve data availability and access efficiency. For cache resolution, COOP uses flooding based approach. As part of cache management, it uses the inter-category and intra-category rules to minimize caching duplications between neighboring nodes. Disadvantage of the scheme is that flooding incurs high discovery overhead and it does not consider factors such as size and consistency during replacement.

3. Network model and system environment

3.1. Network model

This work assumes that an ad hoc network comprises a group of mobile clients communicating through omni-directional antennas with the same transmission range. The topology of an ad hoc network is thus represented by an undirected graph G = (V, E), where V is the set of mobile clients MH_1, MH_2, ..., and E$\subseteq$V$\times$V is the set of links between clients. The existence of a link (u, v)$\in$E also means (v, u)$\in$E, and that clients u and v are within the transmission range of each other, in which case u and v are called one-hop neighbors of each other. The set of one-hop neighbors of a client MH_i is denoted by MH_i^1 and forms a cooperation zone. The combination of clients and transitive closure of their one-hop neighbors forms a MANET. Two clients that are not connected but share at least one common one-hop neighbor are called two-hop neighbors of each other.

As clients can physically move, there is no guarantee that a neighbor at time t will remain in the zone at later time $t + \tau$. The devices might be turned off or on at any time, so the set of live clients varies with time and has no fixed size.

3.2. System environment

The system environment is assumed to be an ad hoc network where mobile clients access data items held as originals by other mobile clients. A mobile client that holds the original value of a data item is called data server/source/center. A data request initiated by a client is forwarded hop-by-hop along the routing path until it reaches the data source and then the data source sends back the requested data. Each mobile client maintains local cache in its hard disk. To reduce the bandwidth consumption and

query latency, the number of hops between the data source/cache and the requester should be as small as possible. Most mobile clients, however, do not have sufficient cache storage and hence the caching strategy is to be devised efficiently. We also make the following assumptions:

- The system has total of M hosts and MH_i ($1 \leqslant i \leqslant M$) is a unique host identifier. The set of one-hop neighbors of a host MH_i is denoted by MH_i^1 and forms a zone.
- The set of all data items is denoted by $D = \{d_1, d_2, \ldots, d_N\}$, where N is the total number of data items and d_j ($1 \leqslant j \leqslant N$) is a data identifier. D_i denotes the actual data of the item with id d_i. Size of data item d_i is s_i. The original of each data item is held by a particular data source.
- Each mobile host has a cache space of C bytes.
- Each data item is periodically updated at data source. After a data item is updated, its cached copy (maintained on one or more hosts) may become invalid.

4. Zone cooperative caching scheme

This section describes our Zone Cooperative (ZC) caching scheme for data retrieval in MANETs. The design rationale of the ZC caching is that it is advantageous for a client to share cached data with its neighbors lying in the zone (i.e., mobile clients that are accessible in one hop). Mobile clients belonging to the zone of a given client then form a cooperative cache system for this client since the cost for communicating with them is low both in terms of energy consumption and message exchange. Figure 1 shows the behavior of ZC caching strategy for a client request. For each request, one of the following four cases holds:

Case 1: Local hit. When copy of the requested data item is stored in the cache of the requester. If the data item is valid, it is retrieved to serve the query and no cooperation is necessary.

Case 2: Zone hit. When the requested data item is stored in the cache of one or more one-hop neighbors of the requester. Message exchange within the home zone of the requester is required during the cache discovery.

Case 3: Remote hit. When the data is found with a client belonging to a zone (other than home zone of the requester) along the routing path to the server.

Case 4: Global hit. Data item is retrieved from the server.

4.1. Cache discovery process

A cache discovery algorithm is needed to determine if and where the requested item is cached when the requester does not have knowledge of the destination. When a data request is initiated at a client, it first looks for the data item in its own cache. If there is a local cache miss, the client broadcasts *request* packet to check if the data item is cached in other clients within its home zone. When a client receives the *request* and has the data item in its local cache (i.e., a zone cache hit), it will send an *ack* packet to the requester to acknowledge that it has the data item. In case of a zone cache miss, the *request* is forwarded to the neighbor along the routing path. Before forwarding a *request*, each client along the path searches the item in its local cache or zone as described above. If the data item is not found on the zones along the routing path (i.e., a remote cache miss), the *request* finally reaches the data source. When a client receives an *ack* packet, it sends a *confirm* packet against first *ack* packet. The *ack* packets for the same item received from other clients are discarded without further processing. When a client/server receives a *confirm* packet, it responds back with the actual data value to the requester.

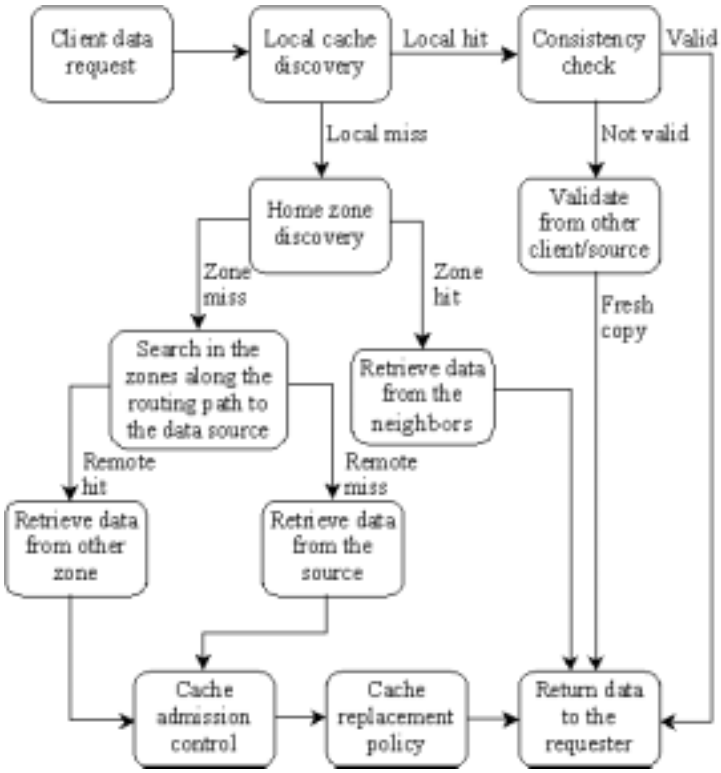


Fig. 1. Service of a client request by ZC caching strategy.

To demonstrate the above idea, we present a cache discovery example in Fig. 2 to determine the data access path to the client having the requested cached data or to the data source. Let us assume that MH_i sends a request for a data item d_x and MH_k is located along the path through which the request travels to the data source MH_s, where k$\in\{$a, c, d$\}$. The discovery process is described as follows:

1. When MH_i needs d_x, it first checks its own cache. If the data item is not available in its local cache (i.e., a local cache miss), it broadcasts a *request* packet to the mobile hosts in its zone (i.e., to MH_j and MH_a). After MH_i broadcasts the *request*, it waits for an acknowledgement. If it does not get any acknowledgement within a specified timeout period, it fails to get d_x within home zone (i.e., zone cache miss). In case of any MH_i^1 (MH_j or MH_a) has the data item d_x, it sends *ack* packet to MH_i.
2. When MH_k receives a *request* packet, it broadcasts the packet to MH_k^1 (i.e., mobile hosts in the zone of MH_k) if it does not have d_x in its local cache. When MH_k receives an *ack* packet, it sends a *confirm* packet to the *ack* packet sender. There may be additional *ack* packets received by MH_k from other hosts in its zone and are discarded as it has already received an *ack* packet from a host closer to it.
3. When $MH_i^1/MH_k^1/MH_s$ receives a *confirm* packet, it sends the *reply* packet to the requester. The *reply* packet containing item *id* d_x, actual data D_x and TTL_x, is forwarded hop-by-hop along the routing path until it reaches the original requester.

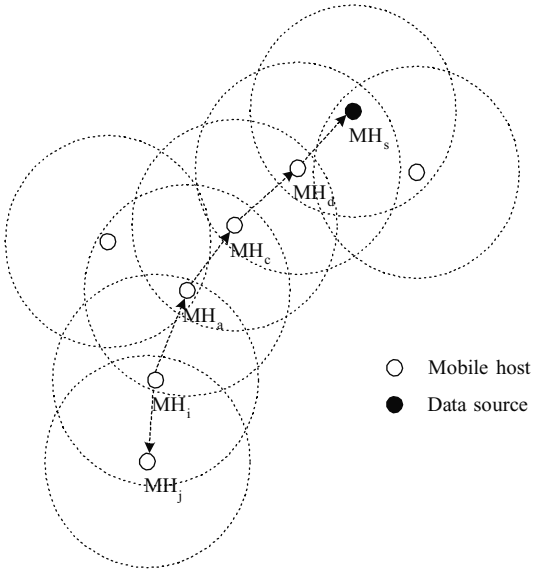


Fig. 2. A request packet from client MH_i is forwarded to the data source MH_s.

4.2. Cache admission control

When a client receives the requested data, a cache admission control is triggered to decide whether it should be brought into the cache. Inserting a data item into the cache might not always be favorable because incorrect decision can lower the probability of cache hits [7]. For example, replacing a data item that will be accessed soon with an item that will not be accessed in the near future degrades performance. In ZC, the cache admission control allows a host to cache a data item based on the distance of data source or other host that has the requested data. If the host or data source is Δ hops away from the requesting host, then it does not cache the data; otherwise it caches the data item. For example, if the origin of the data item resides in the same zone of the requesting client i.e., $\Delta = 1$, then the item is not cached, because it is unnecessary to replicate data item in the same zone since cached data can be used by closely located hosts. In general, same data items are cached at least Δ hops away.

A tradeoff exists between query latency and data accessibility. With a small Δ, the number of replicas for each data item is high and access delay for this data item is low. On the other hand, with a larger Δ, each data item has a small number of replicas, and the access delay can be little longer. Advantage is that mobile clients can cache more distinct data items and still serve requests when the data source is not accessible. However, if a network partition exists, many clients might not be able to access this data.

4.3. Cache consistency

Cache consistency issue must be addressed to ensure that clients only access valid states of the data. Problems related to cache consistency have been studied in many other systems such as multi-processor architectures, distributed file systems, distributed shared memory, and client-server database systems. Two widely used cache consistency models are the weak consistency and the strong consistency model. In the weak consistency model, a stale data might be returned to the client. In the strong consistency model, after a update completes, no stale copy of the modified data will be returned to the client.

Recently, we have done some work [15,16] on maintaining strong cache consistency in the one-hop based mobile environment. However, due to bandwidth and power constraints in ad hoc networks, it is too expensive to maintain strong consistency, and the weak consistency model is more attractive [6,7, 9]. The ZC caching uses a simple weak consistency model based on the time-to-live (TTL), in which a client considers a cached copy up-to-date if its TTL has not expired. The client removes the cached data when the TTL expires. A client refreshes a cached data item and its TTL if a fresh copy of the same data passes by.

5. Cache replacement policy

A cache replacement policy is required when a client wants to cache a data item, but the cache is full, and thus it needs to victimize a suitable subset of data items to evict from the cache. Cache replacement policies have been extensively studied in operating systems, virtual memory management and database buffer management. However, these algorithms might be unsuitable for ad hoc networks for several reasons [7]:

- The data item size may not be fixed in ad hoc environment, the used replacement policy must handle data items of varying sizes.
- The data item's transfer time might depend on the item's size and the distance between the requesting client and the data source (or cache). Consequently, the cache hit ratio might not be the most accurate measurement of a cache replacement policy's quality.
- The replacement algorithm should also consider cache consistency, that is, data items that tend to be inconsistent earlier should be replaced earlier.

5.1. Utility based replacement

We have developed *Least Utility Value* (*LUV*) based cache replacement policy, where documents with the lowest *utility* are those that are removed from the cache. Four factors are considered while computing *utility* value of a data item at a client:

Popularity. The access probability reflects the popularity of a data item for a host. An item with lower access probability should be chosen for replacement. At a host, the access probability A_i for data item d_i is given as

$$A_i = a_i \Big/ \sum_{k=1}^{N} a_k \tag{1}$$

Where a_i is the mean access rate to data item d_i. a_i can be estimated by employing *sliding window* method of last K access times [29]. We keep a sliding window of K most recent access timestamps $(t_i^1, t_i^2, \ldots, t_i^K)$ for data item d_i in the cache. The access rate is updated using the following formula:

$$a_i = \frac{K}{t^c - t_i^K},$$

where t^c is the current time and t_i^K is the timestamp of oldest access to item d_i in the sliding window. When fewer than K samples are available, all the samples are used to estimate a_i. To reduce the computational complexity, the access rates for all cached items are not updated during each replacement,

rather the access rate for an item is updated only when the item is accessed. K can be as small as two or three to achieve the best performance [29]. During simulation experiments, we have used $K = 2$, thus the spatial overhead to store recent access timestamps is relatively small.

Distance. Distance (δ) is measured as the number of hops between the requesting client and the responding client (data source or cache). This policy incorporates the distance as an important parameter in selecting a victim for replacement. The greater the distance, the greater is the *utility* value of the data item. This is because caching data items which are further away, saves bandwidth and reduces latency for subsequent requests.

Coherency. A data item d_i is valid for a limited lifetime, which is known using the TTL_i field. An item which is valid for shorter period should be preferred for replacement.

Size (s). A data item with larger data size should be chosen for replacement, because the cache can accommodate more data items and satisfy more access requests.

Based on the above factors, the *utility*$_i$ function for a data item d_i is computed using the following expression:

$$\text{utility}_i = \frac{A_i.\delta_i.TTL_i}{s_i} \tag{2}$$

The idea is to maximize the total utility value for the data items kept in the cache. For a cache of size C such that the size s_i of each data item d_i is very much less than C, the principle of optimality implies that the mobile client MH_x should always retain a set C_x of data items in its cache such that

$$\sum_{d_i \in C_x} \text{utility}_i \tag{3}$$

is maximized subject to

$$\sum_{d_i \in C_x} s_i \leq C \tag{4}$$

The optimization problem defined by the objective functions Eqs (3) and (4) is termed as ad hoc caching problem.

5.2. NP-Hardness

Here we will prove that ad hoc caching problem is NP-hard.

Definition 1. ADCACHE (ad hoc caching problem)

Instance: Given a set of data items D such that for each item $d_i \in$D, a size $s_i > 0$ and utility *utility*$_i > 0$. Also given that cache capacity of each client is C.

Question: Find a cache set $C_x \subset$D for client MH_x

that maximizes

$$\sum_{d_i \in C_x} \text{utility}_i$$

subject to

$$\sum_{d_i \in C_x} s_i \leqslant C$$

Theorem 1. The ADCACHE is NP-hard.

Proof. The proof follows transformation from 0/1 Knapsack problem [28] that is known as NP-complete problem. The instance and question of 0/1 Knapsack are presented as follows:

Definition 2. KNAPSACK (0/1 Knapsack problem)

Instance: Given a set of items *Itemset* $= \{item_1, item_2, \ldots, item_n\}$, such that each item $item_i$ is associated with a value $v_i > 0$ and weight $w_i > 0$.

Question: Select a set of items from *Itemset* (denoted by *Selectset*)

that maximizes

$$\sum_{\mathrm{item_i} \in \mathrm{Selectset}} v_i$$

subject to

$$\sum_{\mathrm{item_i} \in \mathrm{Selectset}} w_i \leqslant W$$

The KNAPSACK can be transformed into ADCACHE by mapping W to C, *Itemset* to D, *Selectset* to C_x, v_i to $utility_i$ and w_i to s_i. By solving ADCACHE, we get $\sum_{d_i \in C_x} \mathrm{utility}_i$ and the size of items in the cache $\sum_{d_i \in C_x} s_i \leqslant C$. Therefore, the KNAPSACK can be solved by solving ADCACHE. Hence, the ADCACHE is NP-hard.

5.3. Implementation issues

Maximizing the objective function defined by Eqs (3) and (4) implies a minimization of the response time per reference. The task of LUV is to make this optimal decision for every replacement. When the data size is relatively small compared to the cache size [28], we can use heuristics to obtain sub-optimal solutions. The heuristic we use is: through out the cached data item d_i with the minimum $utility_i/s_i$ value until the free cache space is sufficient to accommodate the incoming data. A binary min-heap [28] data structure is used to implement the LUV policy. The key field for the heap is the $\mathrm{utility}_i/s_i$ value for each cached data item d_i. When the events of cache replacement occur, the root item of the heap is deleted. This operation is repeated until sufficient space is obtained for the incoming data item. Let N_c denotes the number of cached items and N_v the victim set size. Every deletion operation has a complexity of $O(logN_c)$. An insertion operation also has a $O(\mathrm{logN}_c)$ complexity. Thus the time complexity for every cache replacement operation is $O(N_v \mathrm{logN}_c)$. In addition, when an item's *utility* value is updated, its position in the heap needs to be adjusted. The time complexity for every adjustment operation is $O(\mathrm{logN}_c)$. In most of the cases, the maximum value of N_v is three. Therefore, the overall time complexity of LUV is $O(\mathrm{logN}_c)$.

6. Analytical model

To analyze the performance of the proposed ZC caching technique, we develop an analytical model. The performance is measured with the following metrics:

Table 1
Symbols used in the analysis

Symbol	Definition
S	Data source/server
H_i	Number of hops between S and MH_i
P_{ij}	Probability of data item d_j cached in MH_i
r	Transmission range of an MH (meters)
ρ	Node density of the network
H	Average number of hops between an MH and server S
L_j	Number of hops to retrieve data item d_j
L_{avg}	Average number of hops to retrieve a data item
T_{local}	Average time to retrieve a data item from the local cache
T_{zone}	Average time to retrieve a data item from the zone
$T_{discovery}$	Average overhead for cache discovery
T_s	Average time to retrieve a data item from S
T_j	Query latency to retrieve data item d_j
T_{avg}	Average query latency to retrieve a data item

1. **Average number of hops** a request is expected to travel before it reaches the data cache/source. Reducing the hop count can reduce the query latency, bandwidth and the power consumption since fewer clients are involved in the query process [6]. Reduction in the hop count can also alleviate the load at data source since some of the requests are satisfied by the cache.
2. **Average query latency** is the time elapsed between the query is sent and the data is transmitted back to the requester averaged upon all the successful queries.

Table 1 shows the symbols and notations used in the analysis.

Theorem 2. If P is the probability of cache hit at a client, then the probability of cache hit in a cooperation zone is $1-(1-P)^{\rho\pi r^2-1}$.

Proof. Area of a cooperation zone $= \pi r^2$

Number of nodes in a cooperation zone, $N_{zone} = \rho\pi r^2$

Number of nodes contributing for zone hit (i.e., number of one-hop neighbors of a node) $= \rho\pi r^2 - 1$

Probability of cache miss for a node $= (1-P)$

Probability of cache miss in a zone $= (1-P)^{\rho\pi r^2-1}$

Probability of cache hit in a cooperation zone $= 1-(1-P)^{\rho\pi r^2-1}$

For each client request, four cases (local hit, zone hit, remote hit or global hit) are possible to retrieve the requested data item as described in Section 4. To compute the number of hops L_j and query latency T_j for retrieving the data item d_j by client MH_i, the probability of data hit for each case along with number of hops is shown below:

Case	Probability of data hit	Number of hops
Local hit	P_{ij}	0
Zone hit	$1-(1-P_{ij})^{\rho\pi r^2-1}$	1
Remote hit	$(1-P_{ij})^{\rho\pi r^2 k}[1-(1-P_{ij})^{\rho\pi r^2}]$	k, $1 \le k < H_i$
Global hit	$(1-P_{ij})^{\rho\pi r^2 H_i} P_{ij}$	H_i

So, L_j is given as

$$L_j = P_{ij}.0 + [1-(1-P_{ij})^{\rho\pi r^2-1}].1 + \sum_{k=1}^{H_i-1} (1-P_{ij})^{\rho\pi r^2 k}.[1-(1-P_{ij})^{\rho\pi r^2}].k$$

$$+(1-P_{ij})^{\rho\pi r^2 H_i}.P_{ij}.H_i \tag{5}$$

The Eq. (5) above can be explained as follows. When there is a local cache hit, the requested data item d_j can be retrieved locally (0 hops) at client MH_i with a probability P_{ij}. In case of zone hit (1 hop distance from the requester) the access probability is $1-(1-P_{ij})^{\rho\pi r^2-1}$. When there is a remote data hit, the item is retrieved from a client lying on a zone (other than home zone of the requester) along the routing path at a distance of k hops from the requester with a probability $(1-P_{ij})^{\rho\pi r^2 k}[1-(1-P_{ij})^{\rho\pi r^2}]$. If none of the clients has cached the requested item d_j, it is retrieved from the data server (global hit) with a probability $(1-P_{ij})^{\rho\pi r^2 H_i}P_{ij}$.

To simplify the model, assume that P is the probability that a data item is cached by an MH. The average number of hops ($\mathrm{L_{avg}}$) is given as

$$L_{\mathrm{avg}} = 1-(1-P)^{\rho\pi r^2-1} + \sum_{k=1}^{H-1}(1-P)^{\rho\pi r^2 k}.[1-(1-P)^{\rho\pi r^2}].k + (1-P)^{\rho\pi r^2 H}.P.H \tag{6}$$

This equation is an approximation of $\mathrm{L_{avg}}$ since in practice P may differ for different data items.

If there is no cooperation within a zone i.e., a client does not share caches with its one-hop neighbors, then $\rho\pi\mathrm{r}^2 = 1$. The average number of hops becomes

$$L_{\mathrm{avg,\ CacheData}} = \sum_{k=1}^{H}(1-P)^k.P.k \tag{7}$$

The Eq. (7) above is same as developed for CacheData scheme in [6,23].

Query latency is given as follows:

$$T_j = P_{ij}.T_{\mathrm{local}} + [1-(1-P_{ij})^{\rho\pi r^2-1}].[T_{\mathrm{zone}} + T_{\mathrm{discovery}}] + \sum_{k=1}^{H_i-1}(1-P_{ij})^{\rho\pi r^2 k}.$$
$$[1-(1-P_{ij})^{\rho\pi r^2}].[T_{\mathrm{zone}} + T_{\mathrm{discovery}}] + (1-P_{ij})^{\rho\pi r^2 H_i}.P_{ij}.T_s \tag{8}$$

Similar to the explanation of Eq. (5) described above, the Eq. (8) can be justified.

$$T_{\mathrm{avg}} = P.T_{\mathrm{local}} + [1-(1-P)^{\rho\pi r^2-1}].[T_{\mathrm{zone}} + T_{\mathrm{discovery}}] + \sum_{k=1}^{H-1}(1-P)^{\rho\pi r^2 k}.$$
$$[1-(1-P)^{\rho\pi r^2}].[T_{\mathrm{zone}} + T_{\mathrm{discovery}}] + (1-P)^{\rho\pi r^2 H}.P.T_s \tag{9}$$

The effect of data popularity, node density and transmission range can be studied by varying the values of P, ρ and r respectively in Eqs (6) and (9).

7. Performance analysis

In this section, we evaluate the performance of ZC caching through simulation experiments under LUV and LRU replacements.

21	22	23	24	25
20	19	18	17	16
11	12	13	14	15
10	9	8	7	6
1	2	3	4	5

Fig. 3. Division of MANET into grids.

7.1. Simulation model

During the simulation AODV [5] has been used as underlying routing algorithm. The node density is changed by choosing the number of nodes between 50 and 100 in a fixed area. The wireless bandwidth is assumed to be 2 Mbps with transmission range of 250 m.

7.2. Client model

The time interval between two consecutive queries generated from each client follows an exponential distribution with mean T_q. Each client generates a single stream of read-only queries. After a query is sent out the client does not generate new query until the pending query is served. Each client generates accesses to the data items following Zipf distribution [27] with a skewness parameter θ. In Zipf distribution, the access probability of the i^{th} ($1 \leqslant i \leqslant N$) data item is represented as follows

$$A_i = \frac{1}{i^\theta \sum_{k=1}^{N} \frac{1}{k^\theta}},\ 0 \leqslant \theta \leqslant 1$$

If $\theta = 0$, clients uniformly access the data items. As θ is increasing, the access to the data items becomes more skewed. Similar to other studies [6,8] we chose θ to be 0.8.

The simulation area is assumed of size 1500 m $\times$ 1500 m. The clients move according to the random waypoint model. Initially, the clients are randomly distributed in the area. Each client selects a random destination and moves towards the destination with a speed selected randomly from $[v_{\min}, v_{\max}]$. After the client reaches its destination it pauses for a period of time and repeats this movement pattern.

The access pattern of mobile clients can be location dependent, that is, clients that are around the same location tend to access similar data. In order to model this kind of access pattern, the whole network area is equally divided into 5×5 square grids. These grids are named grid 1, 2, ..., 25 in a column wise fashion as shown in Fig. 3. The clients in the same grid have the same Zipf data popularity and clients in different grids have different shift values for the Zipf pattern. For a client in grid i, the id of data access is shifted by i so that id = (id + i)mod N. This access pattern ensures that clients in neighboring grids have similar, although not same, access pattern.

7.3. Server model

The data server is placed at center of network area i.e., at location (750 m, 750 m). There are N data items at the server. Data item sizes vary from $s_{\min}$ to $s_{\max}$ such that size s_i of item d_i is,

Table 2
Simulation parameters

Parameter	Default Value	Range
Database size (N)	1000 items	
$S_{\min}$	1 KB	
$s_{\max}$	10 KB	
Number of clients (M)	70	50∼100
Client cache size (C)	800 KB	200 ∼ 1400 KB
Client speed ($v_{\min} \sim v_{\max}$)	2 m/s	
Bandwidth (b)	2 Mbps	
TTL	5000 sec	
Pause time	300 sec	
Mean query generate time (T_q)	5 sec	2 ∼ 100 sec
Transmission range (r)	250 m	
Δ	1	1 ∼ 5
Skewness parameter (θ)	0.8	

$s_i = s_{\min} + \lfloor \text{random}().(s_{\max} - s_{\min} + 1) \rfloor$, $i = 1, 2, \ldots N$, where random( ) is a random function uniformly distributed between 0 and 1. The data are updated only by the server. The server serves the requests on FCFS (first-come-first-serve) basis. When the server sends a data item to a client, it sends the TTL value along with the data. The TTL value is set exponentially with a mean value. After the TTL expires, the client has to get the new version of the data item either from the server or from other client (having maintained the data item in its cache) before serving the query.

The system parameters are given in Table 2. During the simulation, the parameters are changed to study their impacts.

7.4. Simulation metrics

We evaluate two performance parameters here: average query latency (T_{avg}), and cache hit ratio including local cache hit (H_{local}), zone cache hit (H_{zone}) and remote cache hit (H_{remote}). The average query latency (T_{avg}) is the time elapsed between the query is sent and the data is transmitted back to the requester. Hit ratio is used as a measure of the efficiency of the cache management.

If n_{local}, n_{zone} and n_{remote} denotes the number of local hits, zone hits and remote hits respectively out of total n_{total} requests, then

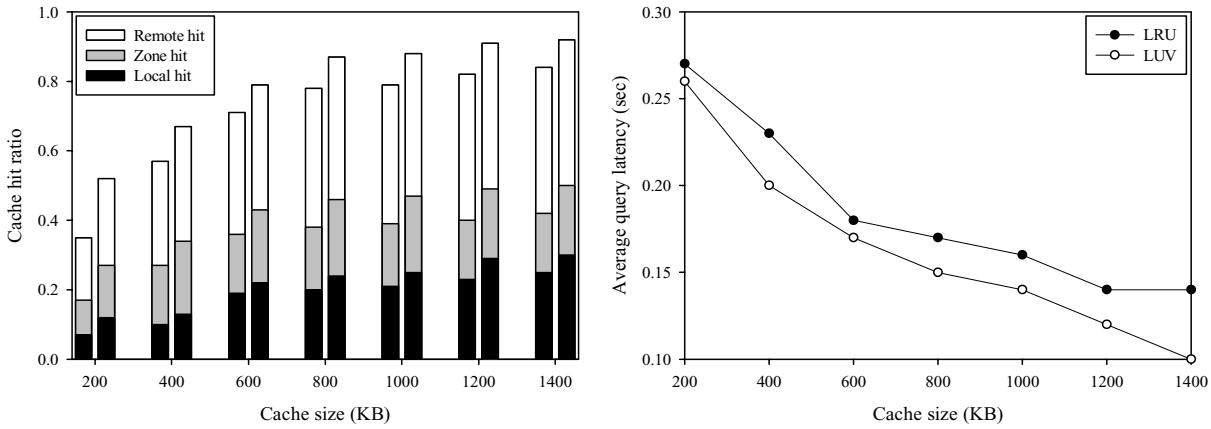
$$H_{\text{local}} = n_{\text{local}}/n_{\text{total}} \times 100\%, H_{\text{zone}} = n_{\text{zone}}/n_{\text{total}} \times 100\%, \text{ and } H_{\text{remote}} = n_{\text{remote}}/n_{\text{total}} \times 100\%$$

7.5. Simulation results

Here we examine the impact of different workload parameters such as cache size, query generate time, node density and Δ on the performance of proposed ZC caching strategy. To demonstrate the effectiveness of LUV policy, we compare the ZC scheme under LUV replacement and LRU based replacement.

7.6. Effects of cache size

Figure 4 shows the effect of cache size on the hit ratio and query latency by varying the cache size from 200 KB to 1400 KB. The cache hit comprises of local hit, zone hit and remote hit. Figure 4a shows



(a) (b)

Fig. 4. Effect of cache size on cache hit ratio and average query latency.

that the local cache hit increases with the increasing cache size because with large cache size more data can be shared locally. The local hit ratio for LRU policy is always lower. LUV policy considers the data size during replacement therefore more items may be cached thus improving the hit ratio. As the cache size increases, the improvement of LUV over LRU becomes more significant. This implies that LUV can utilize the cache space more efficiently. Due to sharing of data among one-hop neighbors, the zone hit and remote hit also improve with the cache size.

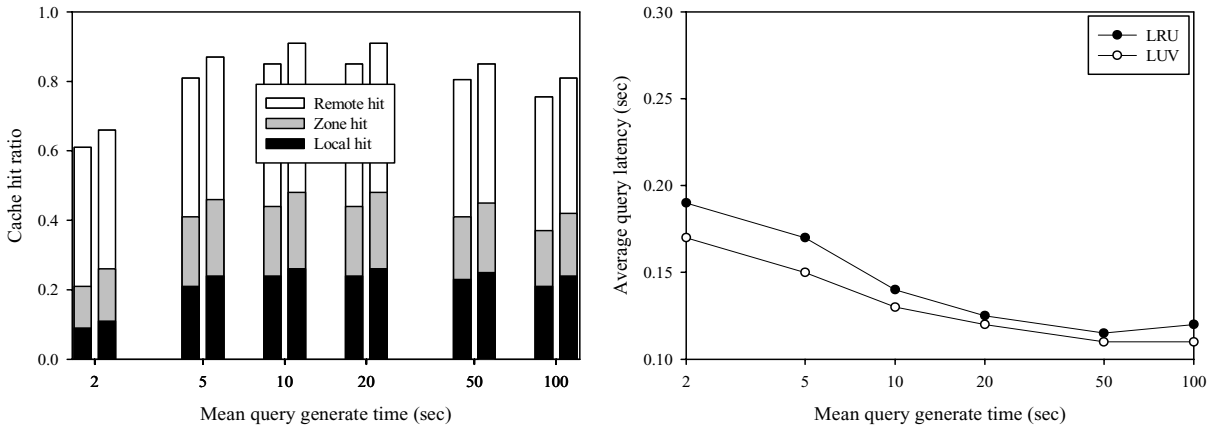
From Fig. 4b, we can see that the proposed LUV policy performs much better than LRU policy. As the cache size increases more data can be found in local + zone cache, thus, the need for accessing the remote and global cache alleviated. Because the hop count of zone data hit is one and is lower than the average hop count of remote/global data hit, the query latency decreases. As the cache size is large enough, the MHs can access most of the required data items from local and zone cache, so it reduces query latency.

Comparing these two policies, we can see that LUV performs much better than LRU. Because of high overall hit ratio, LUV achieves the best performance compared to LRU at all cache sizes.

7.7. Effects of query generate time

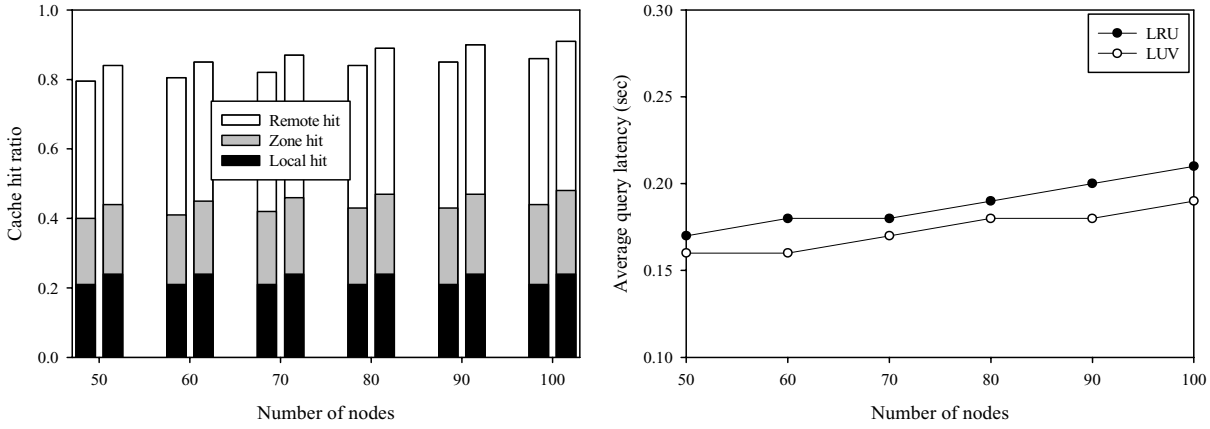
Figure 5 shows the effect of mean query generate time T_q. At small T_q, more queries are generated and hence more cache replacements take place. Higher number of cache replacements at lower T_q increases the number of cache misses at a client. This results in low cache hit ratio at small T_q. The LUV based replacement behaves better than LRU as LUV is more realistic for ad hoc environment. The cache hit ratio improves with an increase in T_q because of lower number of generated queries and hence number of replacements decreases. At very large value of T_q, the cache hit ratio is low because query generate rate is so low that the number of cached data is small and many cached data items are not usable because their TTL have already expired before queries are generated for them. Figure 5a verifies this trend.

Figure 5b shows the average query latency as a function of the mean generate time T_q. At small value of T_q, the query generate rate is high and system workload is more. This results in high value of average query latency. When T_q increases, less queries are generated and average query latency drops. If T_q keeps increasing, the average query latency drops slowly or even increases slightly due to decrease in



(a) (b)

Fig. 5. Effect of mean query generate time on cache hit ratio and average query latency.



(a) (b)

Fig. 6. Effect of node density on cache hit ratio and average query latency.

cache hit ratio. Under extreme high T_q, most of the queries are served by the remote data server and both the replacement policies perform similarly.

7.8. Effects of node density

We vary the number of mobile nodes from 50 to 100 in network area to study the performance under different node densities. As shown in Fig. 6a, the LUV based replacement has better local hit ratio than LRU policy at all the node densities. When the node density is high, the number of MHs in a cooperation zone increases which leads to an improvement in zone hit ratio and remote hit ratio. LUV also performs better than LRU in terms of zone and remote hit under different node densities.

Figure 6b shows the average query latency as a function of the node density. As node density increases, the latency increases because more nodes compete for limited bandwidth. The query latency for LRU

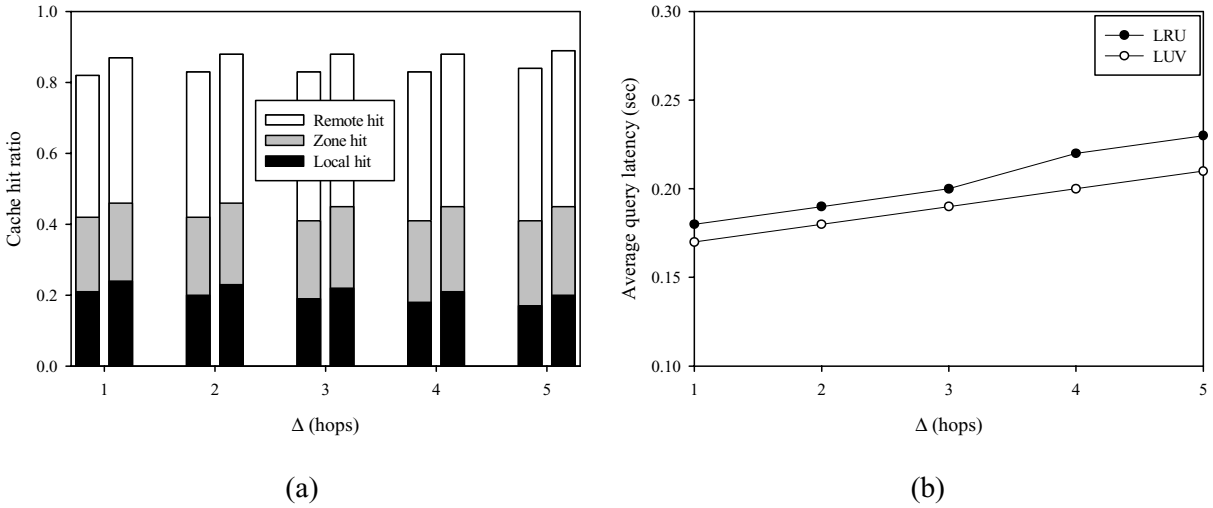


Fig. 7. Effect of Δ on cache hit ratio and average query latency.

policy increases much faster than LUV. This can be explained by the fact that LUV considers various factors to make more intelligent replacement decision.

7.9. Effects of admission control

In this subsection, we evaluate the replacement policies in terms of admission control. We examine the effect of parameter Δ that determines which data item can be cached. Although a high Δ value enables more data items to be distributed over the entire MANET, so that more distinct items will be cached, the average query latency will increase. Figure 7b verified this trend.

In Fig. 7a, local cache hit ratio decreases with increase in Δ because an item will be cached at a client only if no neighbor of the client at distance lower than Δ hops have cached it. Due to caching of distinct items by neighboring clients, there is improvement in zone and remote cache hit with increasing Δ. The decrease in local cache hit ratio causes higher average query latency with increasing Δ.

8. Conclusions

This paper presents LUV based replacement policy for ZC caching scheme in mobile ad hoc networks. The ZC scheme enables clients in a zone to share their data which helps alleviate the longer average query latency and limited data accessibility problems in ad hoc networks. An analytical model of ZC is also developed. The LUV policy considers several factors such as access probability, distance between the requester and data source/cache, coherency and data size, and thus is more realistic for cooperative caching in ad hoc networks. A simulation based performance study was conducted to evaluate the ZC scheme under LUV and LRU policies. Results show that the ZC caching scheme with LUV policy performs better in terms of cache hit ratio and average query latency in comparison with LRU policy. Our future work includes development of a cooperative caching scheme where each client has cache state information within a zone so that replacement is performed on unified zone cache rather than single local cache.

References

[1] M. Frodigh, P. Johansson and L. Larsson, Wireless Ad Hoc Networking – The Art of Networking Without a Network, *Ericsson Review* **77**(4) (2000), 248–263.

[2] S. Das, C. Perkins and E. Royer, Performance Comparison of Two On-Demand Routing Protocols for Ad Hoc Networks, *IEEE INFOCOM* (2000), 3–12.

[3] D. Johnson and D. Maltz, Dynamic Source Routing in Ad Hoc Wireless Networks, *Mobile Computing* (1996), 158–181.

[4] C. Perkins and P. Bhagwat, Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers, *ACM SIGCOMM* (1994), 234–244.

[5] C. Perkins and E.M. Royer, Ad Hoc On-Demand Distance Vector Routing, *IEEE Workshop on Mobile Computing Systems and Applications* (1999), 90–100.

[6] L. Yin and G. Cao, Supporting Cooperative Caching in Ad Hoc Networks, *IEEE INFOCOM* (March 2004), 2537–2547.

[7] G. Cao, L. Yin and C. Das, Cooperative Cache Based Data Access Framework for Ad Hoc Networks, *IEEE Computer* (February 2004), 32–39.

[8] H. Shen, S.K. Das, M. Kumar and Z. Wang, Cooperative Caching with Optimal Radius in Hybrid Wireless Networks, *NETWORKING* (2004), 841–853.

[9] W. Zhang, L. Yin and G. Cao, Secure Cooperative Cache Based Data Access in Ad Hoc Networks, *NSF International Workshop on Theoretical and Algorithmic Aspects of Wireless Ad Hoc, Sensor, and Peer-to-Peer Networks*, June 2004, 11–16.

[10] M. Takaaki and H. Aida, Cache Data Access System in Ad Hoc Networks, *Vehicular Technology Conference* (*VTC*) (April 2003), 1228–1232.

[11] P. Nuggehalli, V. Srinivasan and C.-F. Chiasserini, Energy-Efficient Caching Strategies in Ad Hoc Wireless Networks, *MobiHoc* (2003), 25–34.

[12] M. Papadopouli and H. Schulzrinne, Effects of Power Conservation, Wireless Coverage and Cooperation on Data Dissemination Among Mobile Devices, *ACM SIGMOBILE Symposium on Mobile Ad Hoc Networking and Computing* (*MobiHoc*), *California* (October 2001), 117–127.

[13] W.H.O. Lau, M. Kumar and S. Venkatesh, Cooperative Cache Architecture in Support of Caching Multimedia Objects in MANETs, 5th *ACM International Workshop on Wireless Mobile Multimedia*, (2002), 56–63.

[14] T. Hara, Replica Allocation Methods in Ad Hoc Networks with Data Update, *Kluwer Journal of Mobile Networks and Applications* **8**(4) (2003), 343–354.

[15] N. Chand, R.C. Joshi and M. Misra, Broadcast Based Cache Invalidation and Prefetching in Mobile Environment, *International Conference on High Performance Computing* (*HiPC*), Springer-Verlag LNCS, **3296** (2004), 410–419.

[16] N. Chand, R.C. Joshi and M. Misra, Energy Efficient Cache Invalidation in a Disconnected Wireless Mobile Environment, *International Journal of Ad Hoc and Ubiquitous Computing* (*IJAHUC*) **1**(3) (2005), 184–192.

[17] G. Cao, On Improving the Performance of Cache Invalidation in Mobile Environments, *ACM/Kluwer Mobile Networks and Applications* **7**(4) (2002), 291–303.

[18] G. Cao, A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments, *IEEE Transactions on Knowledge and Data Engineering* **15**(5) (2003), 1251–1265.

[19] R. Friedman, M. Gradinariu and G. Simon, Locating Cache Proxies in MANETs, 5th *ACM International Symposium on Mobile Ad Hoc Networking and Computing* (2004), 175–186.

[20] F. Sailhan and V. Issarny, Cooperative Caching in Ad Hoc Networks, *International Conference on Mobile Data Management* (*MDM*) (2003), 13–28.

[21] S. Lim, W.-C. Lee, G. Cao and C.R. Das, A Novel Caching Scheme for Improving Internet-Based Mobile Ad Hoc Networks Performance, *Elsevier Ad Hoc Networks Journal* **4**(2) (March 2006), 225–239.

[22] S. Lim, W.-C. Lee, G. Cao and C.R. Das, Performance Comparison of Cache Invalidation Strategies for Internet-Based Mobile Ad Hoc Networks, *IEEE International Conference on Mobile Ad Hoc and Sensor Systems* (*MASS*) (October 2004), 104–113.

[23] L. Yin and G. Cao, Supporting Cooperative Caching in Ad Hoc Networks, *IEEE Transactions on Mobile Computing* **5**(1) (2006), 77–89.

[24] Y. Du and S. Gupta, COOP – A Cooperative Caching Service in MANETs, *Proceedings of the IEEE ICAS/ICNS* (2005), 58–63.

[25] J. Jayaputera and D. Taniar, Invalidation for CORBA Caching in Wireless Devices, *Embedded and Ubiquitous Computing*, Lecture Notes in Computer Science, (Vol. 3207), Springer-Verlag, 2004, 460–471.

[26] K.Y. Lai, Z. Tari and P. Bertok, Supporting User Mobility Through Cache Relocation, *Mobile Information Systems*, IOS Press, **1**(4) (2005), 275–307.

[27] L. Breslau, P. Cao, L. Fan, G. Phillips and S. Sheker, Web Caching and Zipf-Like Distributions: Evidence and Implications, *IEEE INFOCOM*, March 1999, 126–134.

[28] T.H. Cormen, C.E. Leiserson, R.L. Rovest and C. Stein, *Introduction to Algorithms*, Prentice Hall, 2003.

[29] J. Shim, P. Scheuermann and R. Vingralek, Proxy Cache Design: Algorithms, Implementation and Performance, *IEEE Transactions on Knowledge and Data Engineering* **11**(4) (July/Aug 1999), 549–562.

Narottam Chand received his PhD degree from Indian Institute of Technology Roorkee in Computer Science and Engineering. He is currently a faculty at the Department of Computer Science and Engineering, National Institute of Technology Hamirpur, India. His current research interests include data dissemination techniques in mobile, ad hoc and sensor networks.

R.C. Joshi received his PhD degree in Electronics and Computer Engineering from IIT Roorkee. Since September 1987 he has been a Professor in the Department of Electronics and Computer Engineering, IIT Roorkee. He has served as Head of the Department twice; first from January 1991-January 1994 and then from January 1997-December 1999. Dr. Joshi has published several research papers in referred journals/conferences. His current research interests include database management systems, data mining and knowledge discovery, mobile and distributed computing, sensor networks and security. He has served as the main organizing chair for the International Conference ADCOM, 1999.

Manoj Misra obtained his PhD degree from Newcastle upon Tyne, UK in Computer Science and Engineering. He is currently an Associate Professor at the Department of Electronics and Computer Engineering, IIT Roorkee. Dr. Misra's current research interests are in mobile computing, performance evaluation and distributed computing. Currently he is working as Principal Investigator for Data Caching on PlanetLab Project from Intel.