COOP – A cooperative caching service in MANETs

Yu Du and Sandeep K. S. Gupta
Department of Computer Science and Enginnering
Ira A. Fulton School of Engineering at Arizona State University
Tempe, AZ 85287
{*Yu.Du, Sandeep.Gupta*}*@asu.edu*

Abstract

Data access applications in Mobile Ad Hoc Networks (MANETs) suffer from intermittent network connections and restricted power supplies. While most of the researches focus on Media Access Control (MAC) and routing layer solutions, we explore the possibility of application cooperation. In this paper, we propose COOP, a novel cooperative caching service for data access applications in MANETs. The objective is to improve data availability and access efficiency by collaborating local resources of mobile devices. COOP addresses two basic problems of cooperative caching: cache resolution and cache management, i.e. how to find the requested data efficiently and how to manage local cache to improve the capacity of cooperated caches. The simulation results show that the service significantly reduces response delay and improves data availability for data access applications in MANETs.

1. Introduction

Mobile Ad-hoc networks (MANETs) provide an attractive solution for networking in the situations where network infrastructure or service subscription is not available. This type of network can communicate with external networks such as Internet through a gateway[13]. However, MANETs are limited by intermittent network connections, restricted power supplies, and limited computing resources. These restrictions raise several new challenges for data access applications with the respects of data availability and access efficiency. While most researches focus on MAC and routing layer solutions, we study how to use application level cooperative caching to address these challenges. Example scenarios are illustrated in the following:

- At a rescue site, the personnel communicate with each other's mobile device through a spontaneous MANET due to lack of network infrastructure support. The devices that have interfaces to external networks serve as gateways to allow other devices communicate with the outside world. Suppose Ashley has retrieved a patient's medical history information from the outside data server. Now Bob needs the information for the same patient. If Ashley can share the information which currently resides in her local cache, it would save significant amount of delay, bandwidth, and battery power for Bob, as the communication cost to the remote server is much higher than local communications within MANETs.
- MANETs can be used to extend the coverage of infrastructure-based networks such as cellular networks [2, 5]. The basic mechanism is to let in-range devices relay packets for out-range devices. The relay can involve multiple intermediate devices until reach the service area. For example, if tourist Charles is out of range of the service area, his request for the area map can be relayed by other tourists. However, if some tourist has already cached the requested information, it would save some time and energy to reply the request directly instead of continue forwarding.

From these examples, we can see that for the information of common interests, cooperative caching can help to save time, energy, and bandwidth by localizing communications. However, one problem is security and privacy, for example, users may not want to share all the cached contents. In this case, we consider a separated cache which contains the data publicly available and share-able, such as the vast information provided on Internet. To further improve privacy and security, we can use mechanisms like source encryption, but this is out of the scope of this paper.

In this paper, we present COOP, a cooperative caching service for MANETs, which aims to improve data availability and access efficiency. COOP addresses two basic problems for cooperative caching in MANETs:

- Cache resolution – how does a mobile device decide where to fetch a data item requested by the user?
- Cache management – how does a mobile device decide which data item to place/purge in its local cache?

For cache resolution, COOP tries to discover a data source which induces less communication cost by utilizing historical profiles and forwarding nodes. For cache management, COOP minimizes caching duplications between neighbor nodes and allows cooperative caches to store more distinctive data items to improve the overall performance.

A cooperative caching scheme specifies how multiple nodes share and manage cached data in a collaborative manner. This paper focuses on caching of regular objects such as text files and pictures. For multimedia caching in mobile computing environments, readers can refer to [11]. Cooperative caching schemes can be classified into hierarchical, directory-based, and hashtable-based ones. Harvest[1] uses the hierarchical approach. A user's request is forwarded up the cache hierarchy until cache hits at some level. Summary[9] uses a directory-based approach and keeps information of which cache has what content. When cache miss happens, the request is forwarded to the caches which contain the requested data potentially. For hashtable-based approach, Squirrel[12] uses distributed hash tables to map data items to different home nodes where the data items are cached. Those schemes have been evaluated and demonstrated performance improvement for Web accessing. However, these schemes are designed for Internet caching, and do not suit MANETs with dynamic topologies.

Cooperative caching in MANETs received recent attentions from research efforts. Yin and Cao presented three cache resolution schemes: CacheData, CachePath, and HybridCache in [4]. In CacheData, forwarding nodes check the passing-by data requests. If a data item is found to be frequently requested, forwarding nodes cache the data, so that the next request for the same data can be answered by forwarding nodes instead of travelling further to the data server. A problem for this approach is that the data could take a lot of caching space in forwarding nodes. CachePath is similar with CacheData, except that forwarding nodes cache the path to the closest caching node instead of the data and redirect future requests along the cached path. This scheme saves caching spaces compared to CacheData, but since the caching node is dynamic, the recorded path could become obsolete and this scheme could introduce extra processing overhead. Trying to avoid the weak points of those two schemes, HybridCache decides when to use which scheme based on the properties (size and the Time-to-Live value) of the passing-by data. Here we question that why the forwarding nodes shall keep record of which data is more popular? Since the forwarding nodes can move to somewhere else in the next second, how much do the statistics collected by the forwarding nodes reflect future needs? In COOP, we also allow forwarding nodes to reply data requests. But we do not advocate that forwarding nodes shall collect statistics based on the forwarded requests. As a summary, the comparison of existing cooperative caching schemes and COOP is listed in Table 1. For other caching schemes, readers can refer to [14].

2. Proposed Approaches

In this section we first present the system model, followed by an overview of COOP, and then describe the details of cache resolution and cache management of COOP.

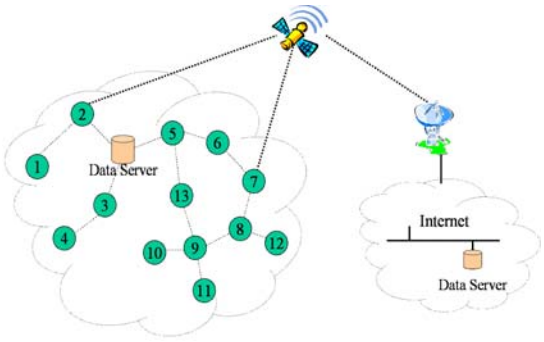

Figure 1. System Model

2.1. System Model

In this paper, we consider a MANET environment where some of the mobile nodes have interfaces to other networks such as satellite networks(Fig. 1). Those nodes serve as gateways to allow other nodes to communicate with the outside networks. A data source can reside inside or outside the MANET. The mobile nodes can read data from the data sources. But only the sources have the privilege to write or update the data they host. The mobile nodes can cache a copy of the data in its local storage. We assume the TTL(Time-To-Live) scheme is used to maintain consistency between client-side cache and the data source. The basic idea of TTL is that each data item is assigned a TTL value, and the data item is considered valid as long as the caching time has not exceeded the TTL value. The TTL scheme is popularly used, for example, in HTTP 1.1 the server can specify the TTL value using the "max-age" directive.

2.2. Overview of COOP

The system architecture of COOP is illustrated in Fig. 2. COOP sits on the middleware level, acts as a proxy for user's applications, and uses underlying network stack to communicate with COOP instances running on other nodes. A running COOP instance receives data requests from user's applications, and resolves the requests using the cache resolution scheme described later in Section 2.3. The cache management (Section 2.4) scheme decides what data to place/purge in the local cache of a mobile device. It discriminates primary data (non-duplicated copy) and secondary data (duplicated copy) with primary data at a higher caching priority, so that cooperated caches can store more distinctive data items to improve the overall performance.

2.3. Cache resolution

Cache resolution addresses how to resolve a data request with minimal cost of time, energy, and bandwidth. COOP's cache resolution is a cocktail scheme, which consists of three basic schemes. The first one is "Adaptive Flooding", which calculates proper flooding range based on the cost to fetch the requested data. Limited flooding is used for cache

Table 1. Comparison of cooperative caching schemes

Schemes	Cache Resolution	Cache management
Harvest [1]	Hierarchical	No specification
Summary [9]	Directory-based	LRU
Squirrel [12]	Hash-based	LRU: all accesses, from the local node and from the neighbors, contribute to the weight of a data item.
Yin04[4, 10]	CacheData, CachePath, HybridCache	LRU
COOP	A cocktail scheme (Adaptive flooding, Profiled based resolution, Roadside Resolution)	Inter-category and intra-category rules

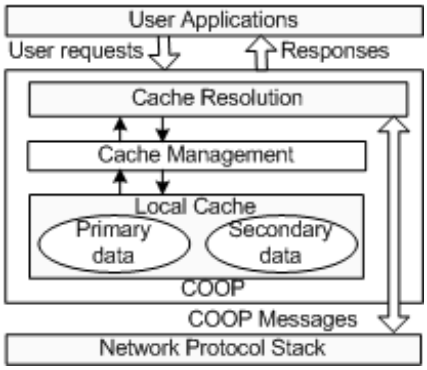

Figure 2. System Architecture

resolution, not only because it has potential to discover the closest cache around the requester, but also because flooding can serve as an announcement in the neighborhood and effectively segment the whole network into clusters, within which they can share and collaborate managing cached contents. The second scheme is "Profile-based Resolution", which maintains a historical profile of previously received data requests, and determines a closer data source for user's requests based on the profile. If a data request cannot get resolved using those two schemes, the data request is forwarded to the original data source, and the third scheme, "Roadside Resolution", is used to resolve the data request along the forwarding path. The details of the basic schemes and the cocktail scheme are described as follows.

2.3.1 Adaptive Flooding

As explained previously, flooding can help to discover the closest cache around the requester, as well as to serve as an announcement in the neighborhood such that the surrounding nodes will know who has the data next time. However, network flooding generates significant amount of traffic, and it shall be used on a restricted basis. In adaptive flooding, we study how to determine a proper flooding range based on the cost of fetching data from the original data source. In the following analysis, we study the average data fetching cost with an x-hop flooding and determine the optimal flooding range to minimize this cost.

We assume that the data fetching cost is proportional to the travel distance of the data request. If a cache for the requested data is discovered with x-hop flooding, the cost is proportional to x. Otherwise, the data request is forwarded to the original data source after x-hop flooding, and the cost is proportional to $(x + L_s)$, where L_s is the distance to the original data source. To calculate the average data fetching cost in all situations, the remaining problem is how likely an x-hop flooding will discover a satisfying cache? To estimate this possibility, we made the following assumptions:

- P_d: the possibility of each node to cache the data d. We assume that every node has the same value of P_d.
- λ: the average node density. Therefore, there are $\lambda\pi x^2$ nodes within x-hop range of the studied node.

Then the probability to discover a cache for data d within x-hop flooding ($0 \leq x < L_s$)is $P(x) = 1 - (1 - P_d)^{\lambda\pi x^2}$. The average cost of fetching d is $\overline{C} = x + L_s(1 - P_d)^{\lambda\pi x^2}$.

Fig. 3 shows the relationship between average data fetching cost $\overline{C}$ and flooding range x($\lambda = 1$, $P_d = 0.1, L_s = 3, 5, 7, 10$). We can see that the lowest average cost happens when $x = 3$ for $L_s = 5, 7, 10$ hops, and $x = 4$ when $L_s = 100$ hops. As L_s increases, the optimal flooding range increases very slowly to achieve minimal average cost. However, note that we assume $\lambda = 1$. If node density becomes higher, the optimal flooding range shall be reduced accordingly. The conclusion of this analysis is that limited flooding helps to minimize average data fetching cost. The flooding range is dependent on the cost of fetching data from the original data source which is proportional to the distance of the data source. For most of the cases, the flooding range shall be restricted within 4 hops.

2.3.2 Profile-based Resolution

Profile-based Resolution maintains a historical profile of previously received data requests. The purpose is to avoid duplicated flooding for same data items and to determine a closer data source for user's requests based on the profile. In this scheme, previously received data requests are recorded in a table which is called RRT (Recent Requests Table). Each entry of RRT records the information of one previously received request, i.e. the sender, the target, and the timestamp of the request. The lifetime of each entry

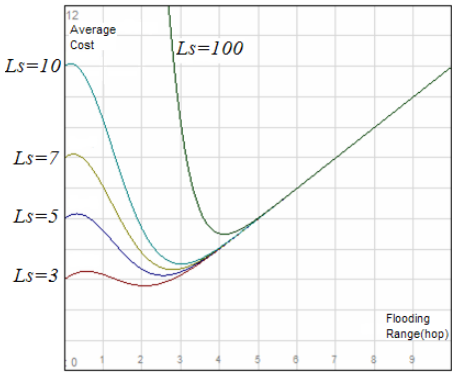

Figure 3. Average data fetching cost.

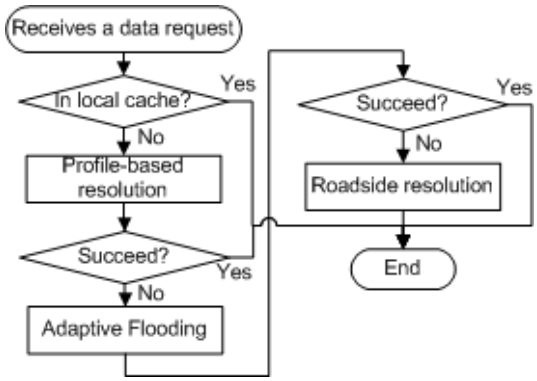

Figure 4. The Cocktail Resolution Scheme

is initialized to a default value which could be the average TTL value of requested data. In implementation, RRT is realized using a circular array with user-specified size. If RRT is full, the new coming request over-writes the oldest request entry.

In operation, a node checks RRT after its local cache misses and before flooding a request. If matching entries are found, the node compares its distances to these matching caches and the original data source, and selects the closest one to resolve the data request. If the requested data is successfully returned, the node updates the lifetime of the RRT entry based on the TTL value in the reply. Otherwise if the data request fails, the table entry is removed, and adaptive flooding is used to resolve the data request. In this way, COOP can avoid duplicated request flooding and select a closer data source with previously learned knowledge.

In the future, this scheme can be further extended by profiling and recognizing reliable long-term cooperation partners. Unlike P2P networks though, to choose a partner not only needs to consider the common interest, but also needs to consider other factors like communication cost, network link reliability, and storage space constraints.

2.3.3 Roadside Resolution

If no cache is found using previous schemes, the data request is forwarded to the original data source. Roadside Resolution allows a node on the forwarding path to serve as a proxy to resolve the request. That is, if a forwarding node caches a copy of the requested data, it can respond to the request and stop forwarding the data request. For example, assume node 11 issues a request for data D in Fig. 1. Before node 9 forwards the request, it checks if it has a valid copy of D in its local cache. If so, node 9 stops forwarding the request and sends the copy back to node 11. This scheme can be further extended. If the forwarding node finds a closer data source in RRT, it can redirect the request to the closer data source. Hence this approach minimizes response delay and energy consumption by reducing the travel distance of data requests and replies.

As explained in Section II, this approach is similar with the approaches in [4] in that both allow forwarding nodes to respond data requests. However, the difference is how forwarding nodes decide what to cache. In [4], forwarding nodes decide what to cache based on statistics collected upon forwarded requests. In COOP, we do not advocate to base caching decisions upon statistics of forwarded requests, and the intuitive reason is that previously collected statistics on forwarding nodes are difficult to reflect future needs because of node mobility and link breakage.

2.3.4 The Cocktail Resolution Scheme

These three basic schemes benefit different phases of cache resolution as described previously. COOP uses a cocktail approach based on the basic approaches. As shown in Fig. 4, COOP uses Profile-based Resolution after the local cache misses. If no matching cache is found or the request fails, COOP uses Adaptive Flooding to discover a cache in the neighborhood. If this again fails, COOP forwards the data request to the data server, and Roadside Resolution is used to resolve the request along the forwarding path.

2.4. Cache management

COOP cache management studies how to decide which data item to keep in a node's local cache. The goal is to increase cache hit ratio, which largely depends on the capacity of the cache. To maximize the capacity of cooperative caches, COOP tries to reduce duplicated caching within short-distance neighborhood, such that the cache space can be used to accommodate more distinct data items.

We categorize cached data copies based on whether they are already available in the neighborhood or not. A data copy is primary if it is not available within the neighborhood. Otherwise, the data copy is secondary. The range of neighborhood is provided as a customizable option. The reason of discriminating primary and secondary data is that cache miss cost is proportional to the travel distance of a data request, and primary data usually incur higher cache miss cost than secondary data. The inter-category and intra-category rules are used to decide caching priorities of primary and second data, which are described in the following.

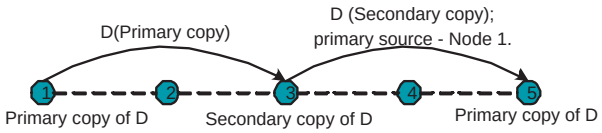

Figure 5. Primary and secondary copy

2.4.1 The inter-category rule

The idea of inter-category rule is to put primary items at a priority level, i.e., secondary items are purged to accommodate primary items, but not vice versa. But the problem in implementation is how to determine whether a data item is primary or secondary. We can do this by broadcasting in the neighborhood, and those who have the data item sends a reply so that we know if the item is available in the neighborhood or not. But this would add extra overhead and probably is not worthwhile since the neighborhood is dynamic.

Here we use a simplified approach to address this problem. Once a node fetches a data item, it labels the item as primary copy if the item comes from a node beyond the neighborhood range. Otherwise, if a data item comes from within the neighborhood, we need to further consider whether the data provider labels the item as primary or secondary. If the provider already labels its copy as primary, the new copy would be secondary since we do not intend to have duplicated primary copies for the neighborhood. On the other hand, if the provider tags its own copy as secondary, the provider needs to attach the information of the primary copy holder. If the primary copy holder is beyond the neighborhood range, the new copy is primary copy, otherwise, the new copy is a secondary copy.

An example is illustrated in Fig. 5. We assume that the neighborhood range is 3 hops and node 1 holds the primary copy of data D initially. When node 3 requests D, node 1 replies and indicates that it has the primary copy of D. Since D already has a primary copy on node 1, which is within 3 hops, node 3 labels it copy of D as secondary copy. Later, when node 5 requests D, node 3 replies and indicates it has a secondary copy and the primary copy is on node 1. Even though D comes from within the neighborhood, node 5 labels its copy of D as a primary copy because the original primary copy is beyond the neighborhood range for node 5.

2.4.2 The intra-category rule

The intra-category rule is used to evaluate the data items within the same category. For this purpose, we simply adopt the LRU (Least Recently Used) algorithm.

Fig. 6 shows an example of the application of the inter-category and intra-category rules. When the node first initializes, all its cache space is free. Then the primary copies $A1$ - $A4$, and secondary copies $B1$-$B5$ are entered to the cache one by one, at when all the space is taken. When another secondary copy $B6$ comes, $B1$, the secondary copy

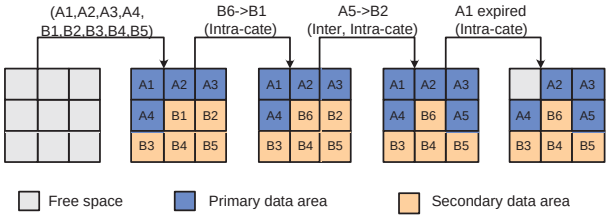

Figure 6. Cache management example

which is least recently used, is replaced by $B6$. When the primary copies $A5$ and $A6$ come, since there is no spare space, they took the space of the least recently used secondary copy $B2$ and $B3$. Finally when the TTL of $A1$ expires, its space is released and become spare space again.

3. Performance Evaluation

The simulation is based on the NS-2[3] which is popularly adopted in networking research. In the simulation, COOP is implemented in ns-2.28 on a Linux platform. COOP uses AODV as the underlying routing protocol which is included in the simulator. The following metrics are used to evaluate COOP from the perspectives of data availability and time efficiency:

- Request success ratio: the percentage of succeeded requests. This reflects the resulted data availability.
- Average response delay: the average delay from sending a request till receiving the response, which reflects the time efficiency of studied approaches.

The settings of important simulation parameters are presented in Table 2. The mean values of exponentially distributed parameters are noted in the parentheses. To simulate user's requests, we assume that each node generates a sequence of data requests with exponentially distributed time intervals. The request pattern follows Zipf-like distribution [15]. Zipf-like distribution has been used to model various behaviors such as the Web page request pattern. The normalized mathematical representation is $P(i) = \frac{1}{i^{\alpha} \sum_{k=1}^{n} \frac{1}{k^{\alpha}}}$, which calculates the access probability of the i-th popular data item. In this formula, the parameter n is the total number of data items, and the parameter α indicates how skew is the distribution. The value of α varies for different applications. For Web page accesses, it is in the range of [0.64, 0.83] according to the studies in [8].

We compared COOP with the traditional cache scheme, which is referred to as SimpleCache [10]. In SimpleCache, each node has a local cache, and user's requests are forwarded to the original data source if local cache misses.

Fig. 7 shows average response delay of SimpleCache and COOP. For different number of nodes we have tried, COOP performs better than SimpleCache. The improvement ratio is from 15% to 26%. The reason is that user's requests can only get satisfied from the original data source after local cache misses in SimpleCache. In this case, response delay

Table 2. Simulation Parameters

Parameters	Settings
Simulation area	1500m*300m
Total #nodes	100
Bandwidth	2Mb/s
Radio range	250m
Mobility pattern	random two way point
Speed	1 – 20m/s
Pause time	120sec
Neighborhood range	2hops
Request trace	zipf-distributed ($\alpha = 0.8$)
Request interval	exponentially distributed (5 sec)
Total #data items	2000
Server failure rate	0.1%
Client cache size	50 items
TTL	exponentially distributed (5000 sec)

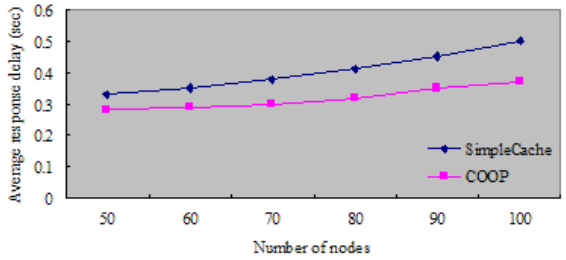

Figure 7. Average response delay.

largely depends on the network condition and the distance to the source. On the other hand, COOP can resolve user's requests using some neighbor cache without travelling further to the original data source, which can save significant amount of communication time.

Fig. 8 shows request success ratio of SimpleCache and COOP. A data request can be dropped because of server failure or network congestions. For different number of nodes we have tried, COOP performs better than Simple cache. The reason is that COOP tends to localize the communication within the neighborhood range, which has less risk of packet loss[6]. While SimpleCache needs to travel further to resolve user's requests after local cache misses, and this significantly increases the possibility of packet loss.

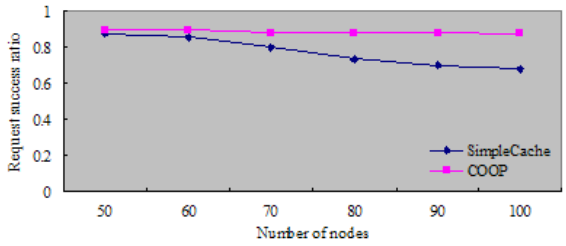

Figure 8. Request success ratio.

4. Conclusions & Future Work

In this paper, we have presented COOP, a novel cooperative caching service for MANETs. To improve data availability and access performance, COOP addresses two basic problems of cooperative caching. For cache resolution, COOP uses a cocktail approach which consists of three basic schemes: Adaptive Flooding, Profile-based Resolution, and Roadside Resolution. By using this approach, COOP discovers data sources which induce less communication cost. For cache management, COOP uses the inter-category and intra-category rules to minimize caching duplications between neighbor nodes and to improve the overall capacity of cooperative caches. Finally, simulations show that COOP can significantly improve data request success ratio and ameliorate average response delay compared to the SimpleCache scheme. In future, we plan to implement COOP in the Ayushman project[7] to demonstrate the application of cooperative caching on the wireless sensor network based health monitoring test-bed.

Acknowledgements: The authors would like to thank the anonymous reviewers for their comments and suggestions. This work is supported in part by NSF grants ANI-0123980, ANI-0196156, and ANI-0086020.

References

[1] A. Chankhunthod et al. A hierarchical internet object cache. In *USENIX Annual Technical Conference*, 1996.
[2] D. Tacconi et al. Ad hoc enhanced routing in umts for increased packet delivery rates. In *IEEE WCNC*, 2004.
[3] K. Fall and K. Varadhan Ed. The ns manual.
[4] G. Cao et al. Cooperative cache based data access in ad hoc networks. *IEEE Computer*, 37(2):32–39, Feburary 2004.
[5] H. Wu et al. Integrated cellular and ad hoc relaying systems: icar. *IEEE JSAC*, 19(10), October 2001.
[6] J. Li et al. Capacity of ad hoc wireless networks. In *Mobile Computing and Networking*, 2001.
[7] K. Venkatasubramanian et al. Poster - ayushman: A wireless sensor network based health monitoring infrastructure and testbed. In *IEEE DCOSS*, 2005.
[8] L. Breslau et al. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM*, 1999.
[9] L. Fan et al. Summary cache: a scalable wide-area web cache sharing protocol. In *Sigcomm*, 1998.
[10] L. Yin et al. Supporting cooperative caching in ad hoc networks. In *INFOCOM*, 2004.
[11] Quoc Le et al. Multimedia caching in mobile computing environments. A chapter submitted to Handbook of Research on Mobile Multimedia, 2005.
[12] S. Lyer et al. Squirrel: A decentralized peer-to-peer web cache. In *PODC*, 2002.
[13] Y. Sun et al. Internet connectivity for ad hoc mobile networks. *International Journal of Wireless Information Networks*, 9(2), April 2002.
[14] Yu Du et al. *Handbook of Mobile Computing*, chapter 15, pages 337–360. CRC Press, 2004.
[15] G. Zipf. *Human behavior and the principle of least effort*. Addison-Wesley, 1949.