Hierarchical Cooperative Caching in Mobile Opportunistic Social Networks

Yunsheng Wang
Dept. of Computer Science
Kettering University
Flint, MI 48504

Jie Wu
Dept. of Computer and Info. Sciences
Temple University
Philadelphia, PA 19122

Mingjun Xiao
School of Computer Science and Tech.
University of Science and Tech. of China,
Hefei, China

***Abstract*—A mobile opportunistic social network (MOSN) is a new type of delay tolerant network (DTN), in which the mobile users contact each other opportunistically. While cooperative caching in the Internet has been studied extensively, cooperative caching in MOSNs is a considerably different and challenging problem due to the probabilistic nature of contact among the mobile users in MOSNs. In order to reduce the total access delay, we let the mobile users cooperatively cache these data items in their limited buffer space. We balance between selfishness (caching the data items according to its own preference) and unselfishness (helping other nodes to cache). The friends with higher contact frequency may share similar interests, hence, caching the data items for friend users can lead to some benefit. In this paper, we present a hierarchical cooperative caching scheme, which divides the buffer space into three components: self, friends, and strangers. In the self component, mobile users cache the data items according to their preference. In the friends component, mobile users help their friends to cache some data items. In the strangers component, mobile users randomly cache the remaining data items. We formally analyze the access delay of the proposed scheme. The effectiveness of our approach is verified through extensive real world trace-driven simulations.**

***Index Terms*—Access delay, cooperative caching, mobile opportunistic social networks (MOSNs), Zipf-like distribution.**

I. Introduction

Delay tolerant networks (DTNs) are characterized by intermittent connectivity and limited network capacity, in which most of the time there does not exist an end-to-end path between some or all of the nodes in the network. With the popularization of smart phones, mobile opportunistic social networks (MOSNs), a new type of DTN, becomes popular. In MOSNs, the individuals carrying smart phones walk around and communicate with each other via Bluetooth or WiFi, when they are in each other's transmission range.

Because of the short contact duration and limited bandwidth, only a small amount of data can be transferred during each contact in MOSNs. Also, the slow development of the battery and limited cache space of the mobile devices restricts the message flooding between the mobile devices. Cache placement is an important factor in improving the performance of data access in such a network environment.

Mobile users may cache data items in a cooperative way to improve the efficiency of data access. Recently, there are some literatures focusing on the cooperative caching problem in DTNs [1–6]. A typical strategy in cooperative caching works as follows. Data sources transfer some data copies to some nodes called *cache nodes*. Each cache node selects a subset of all the data items to cache, due to its limited storage. Other nodes can access data items from cache nodes instead of data sources. Consequently, access delay can be reduced because of the service provided by these cache nodes, while cache costs are increased; consistency should also be maintained if data items change.

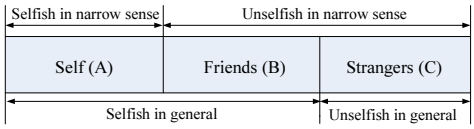


Fig. 1. An illustration of hierarchical cooperative caching: A, B, and C illustrate the size of each component.

In MOSNs, the social relationship between the mobile users becomes much more important. The individuals with higher contact frequency may have similar interests [7], which means that they have similar high-frequency access data items. Therefore, designing an efficient cooperative caching strategy, by considering the social relationship between the mobile users, can improve the performance dramatically.

In this paper, we propose a hierarchical cooperative caching scheme, which divides the buffer storage into three key components: *self*, *friends*, and *strangers*, as shown in Fig. 1. In the *self* component, the mobile nodes will cache its most frequently accessed data items. The mobile nodes with higher contact frequency are considered as friends to the cache nodes. The cache nodes will help the friends to store the friends' most frequently accessed data items in its *friends* component. Finally, each mobile node randomly selects a subset of the remaining data items into its *strangers* component.

Our detailed contributions are listed as follows: (1) To address the problem of cooperative caching, we exploit social relations among nodes. We define the relationship between pairwise nodes based on the contact frequency among the mobile nodes. (2) In order to reflect the selfishness and unselfishness, we divide the cache space into three key components: self, friends, and strangers. In each component, we investigate different data caching and replacement policies. (3) We formally analyze the total access delay of all mobile nodes for the data items. (4) We develop a novel hierarchical cooperative caching scheme in MOSNs, and demonstrate that

it can significantly improve the performance of data access through trace-driven simulations.

The remainder of this paper is organized as follows. In Section II, we review the related work. Section III describes our scheme in detail. Section IV analyzes the the average delay for the node to request the data item in our scheme. Section V focuses on the simulation and evaluation. We summarize the work in Section VI.

II. Related Work

Research in DTNs has attracted a lot of attention in the research community recently. Several solutions have been proposed to handle storage congestion control problems in DTNs, most of which are based on message dropping policies [8, 9] or simply message migration policies [10, 11]. However, in these kinds of mechanisms, the data access delay will increase dramatically. Therefore, cooperative caching schemes have been proposed for the DTN environment [1–3, 12]. They improve the data accessibility from infrastructure networks, such as WiFi Access Points (APs) [2], or the Internet [1], as well as peer-to-peer (P2P) data sharing among the mobile users [3].

Social-based cooperative caching has been studied recently in DTNs [4, 5, 13]. In [4], Zhuo et al. propose a centrality metric to evaluate the caching capability of each node within a community, which is used to determine where to cache. They also consider the impact of the contact duration limitation on cooperative caching. In [5], Zhang and Zhao use social network analysis to classify and study different diffusion schemes based on the "homophily" phenomenon in social networks. In this paper, we consider the mobile users' social role from its own perspective locally, to hierarchical cooperative cache the data items in different components of its buffer.

III. Hierarchical Cooperative Caching

In this section, We first introduce the network model for our work, then we highlight our motivation of cooperative caching in MOSNs. The data item placement scheme is presented next. Finally, we describe the cache replacement policy.

A. Network Model

In MOSNs, the opportunistic contacts are described by network contact graph $G(V, E)$, where V is the set of mobile nodes in the network, and E is the set of edges, with each edge in E representing the opportunistic contact between pairwise nodes. In this paper, there are n mobile nodes in MOSNs ($V = \{N_1, N_2, ..., N_n\}$), and m data items ($D = \{d_1, d_2, ..., d_m\}$). We assume that the sizes (l) of all data items are the same. Also the size (S) of the buffer space of each node is the same. The data item can be updated when the mobile nodes access the Internet via WiFi or WiMAX networks. Mobile node N_i requests the data item d_j with the frequency $f(i, j)$. Let $t(i, j)$ denote the waiting time for node N_i receiving the data item d_j from its own cache ($t(i, j) = 1$), or another node N_k which caches this data item ($t(i, j)$ is the inter contact time between N_i and N_k).

The objective of the cache placement problem is to determine which data items should be cached in which nodes, in order to minimize the total access delay of all nodes in the network. Therefore, the cache placement problem becomes how to select a set of cache node sets $M = \{\{M_1\}, \{M_2\}, ..., \{M_m\}\}$, where each mobile node in $\{M_j\}$ caches a copy of data item d_j. Hence, we represent the cache placement problem as the following optimization problem:

$$\min \quad \sum_{i=1}^{n}\sum_{j=1}^{m}(f(i,j) \times t(i,j)) \tag{1}$$

$$\text{s.t.} \quad |\{\{M_j\}|N_i \in \{M_j\}\}| \leq \left\lfloor \frac{S}{l} \right\rfloor, \forall N_i \in V.$$

The constraint of this optimization problem is the buffer space constraint, which means that N_i can appear in, at most, $\lfloor \frac{S}{l} \rfloor$ sets of M.

This optimization problem can be viewed as a *facility location problem* [14], which has been proved to be an NP-hard problem. In the general facility location problem, there consists a set of potential facility sites where a facility can be opened, and a set of demand points must be serviced. The goal is to pick a subset of facilities to open, to minimize the sum of distances from each demand point to its nearest facility, and plus the sum of opening costs of the facilities. In our problem, the caching nodes are considered as the facility sites. The mobile nodes, which require access to the data, are considered as demand points. The distances from each demand point to its nearest facility are the waiting times required by the nodes to receive the data. The limited buffer space is the constraint of our optimization problem.

B. Motivation

In MOSNs, there is a key concept: *community*. The nodes in the same community tend to contact more frequently, for which we call them "*friends*." Friends tend to have similar interests [7]. Hence, we take advantage of such friends' behavior to perform the cooperative caching, which enables data sharing within a community. As we discussed before, the objective of our work is to minimize the data access delay for the requesters. If the nodes can help their friends to cache the data items, the data access delay can be reduced according to the community property and their opportunistic contacts.

In this paper, we introduce a novel idea that the cache is divided into three hierarchical components: *self*, *friends*, and *strangers*, as shown in Fig. 1. From the narrow sense, the self component reflects the selfishness of the nodes. The friends and strangers components are distributed for other nodes, which shows the side of its unselfishness. In mobile opportunistic social networks, friends group together as a community. Therefore, in general, self and friends components are considered to show the selfishness of the nodes.

C. Data Item Placement

In our proposed hierarchical cooperative caching scheme, we treat the data items with different caching schemes, in terms of selfishness and unselfishness, locally.

Algorithm 1 Hierarchical Cooperative Cache Replacement

/* When cache node N_i is full, and receives a new data item d_j. */
if d_j belongs to the self component. **then**
  if $f(i,j) > f(i,x)$, d_x is the data item cached in N_i with the smallest access frequency. **then**
    d_j replaces d_x.
else
  if d_j belongs to the friends component. **then**
    if $\sum f(k,j) > \sum f(k,x)$, d_x is the data item cached in N_i with the smallest access frequency for all its friends N_k. **then**
      d_j replaces d_x.
else
  if d_j belongs to the strangers component. **then**
    Randomly replace a data item.

- In the self component, each mobile node caches the $\lceil \frac{A}{l} \rceil$ most frequently accessed data items, so that it can access these data items with the minimum delay (we assume this delay is 1). A is the size of the self component.
- In the friends component, each node will store the $\lceil \frac{B}{l} \rceil$ most frequently accessed data items from its friends' point of view, so that its friends can access these items from this cached node with short delay. B is the size of the friends component.
- In the strangers component, each mobile node randomly selects $\lceil \frac{C}{l} \rceil$ data items from the remaining ones to cache, where C is the size of the strangers component.

D. Cache Replacement

Caching locations of the data items are dynamically adjusted by cache replacement. When the buffer is full, after the new data item is received, we first check which component it belongs to. If it is the most frequently accessed data item for itself, it will compare the access frequency with the data items in the self component and replace the lowest one. If this received data item belongs to the friends component, it will replace the data item in its friends component by comparing its access frequency to the friend nodes. Otherwise, if the data item belongs to the strangers component, it randomly replaces one data item in the strangers component. The algorithm of the cache replacement policy is shown in Algorithm 1.

IV. Analysis

A. Request Frequency

In [15], the authors found that the web request follows a Zipf-like distribution, with a small portion of the webs getting the most requests. Therefore, in this paper, we use the Zipf-like distribution as the data request pattern.

We only consider the frequency for a node to request all data items. Moreover, we let $f(i,j)$ be described by the probability that node N_i requests the data item d_j in each data request. Let all the data items be ranked in order of their popularity, where data item d_j is the j'th most frequently requested data item. From [15], we know that $f(i,j)$ has a "cut-off" Zipf-like distribution given by

$$f(i,j) = \frac{\Omega}{j^{\beta}}, \tag{2}$$

where

$$\Omega = \left(\sum_{k=1}^{m} \frac{1}{k^{\beta}} \right)^{-1}. \tag{3}$$

Hence, the probability $f(i,j)$ of a request for the j'th popular data item by N_i is proportional to $\frac{1}{j^{\beta}}$, where $0 < \beta \leq 1$.

B. Data Access Delay

Here, we will focus on calculating the average data access delay time ($t(i,j)$). We assume that the system has run many rounds, and it has entered into a stable state. Moreover, we let I_i denote the set of data items that node N_i is interested in. The set I_i can be determined by the Zipf-like distribution of the frequencies for node N_i to request the data items. We let $F(i)$ denote the set of friends of node N_i, and let $F^+(i)$ denote the set of friends of node N_i, as well as node N_i, itself. We also let $\bar{r}$ denote the average repeated number of a data item in the nodes, and assume that the average request frequency for each data item to be requested is $\bar{p}$. These values also can be determined by the Zipf-like distribution of the frequencies for each node to request the data items. In addition, we assume that the inter contact time of a mobile node to its friends and strangers follows exponential distribution with mean time $\frac{1}{\Lambda}$ and $\frac{1}{\lambda}$, respectively [16]. Now, we consider the calculation in four cases as follows.

The first case is that the data item d_j has been cached in node N_i. There are three subcases:

1) Data item d_j is cached in the self component of N_i as an interested data item. Here, the probability that d_j is cached in N_i is $\frac{a}{|I_i|} \cdot 1_{j \in I_i}$, where $1_{j \in I_i}$ is an indicator function to indicate whether d_j is an interested data item of N_i, and $\frac{a}{|I_i|}$ is the average probability of an interested data item for N_i to be cached in its self component.

2) Data item d_j is cached in the friends component of N_i as an interested data item of N_i's friends. In this subcase, the probability that d_j is cached in the friends component of N_i is $\frac{b}{|\cup_{k \in F(i)} I_k|} \cdot 1_{j \in \cup_{k \in F(i)} I_k}$, where $1_{j \in \cup_{k \in F(i)} I_k}$ is an indicator function to indicate whether d_j is an interested data item for the friends of N_i, and $\frac{b}{|\cup_{k \in F(i)} I_k|}$ is the average probability of an interested data item for the friends of N_i to be cached in its friends component.

3) Data item d_j is cached in the strangers component of N_i. In this subcase, the probability that d_j is cached in the strangers component of N_i is $\frac{c}{|\cup_{k \notin F^+(i)} I_k|} 1_{j \notin \cup_{k \in F^+(i)} I_k}$, where $1_{j \notin \cup_{k \in F^+(i)} I_k}$ is an indicator function to indicate whether d_j is not an interested data item for the nodes in $F^+(i)$, and $\frac{c}{|\cup_{k \notin F^+(i)} I_k|}$ is the average probability of an interested data item for the friends of N_i to be cached in its strangers component. In addition, the delay for node N_i to access the data item d_j in this case is 1. Denote the total probability

and average data access delay for this case as P_1 and t_1, respectively. Then, we have:

$$\begin{aligned} P_1 &= \frac{a}{|I_i|}\cdot 1_{j\in I_i} + \frac{b}{|\cup_{k\in F(i)} I_k|}\cdot 1_{j\in \cup_{k\in F(i)} I_k} + \\ &\quad \frac{c}{|\cup_{k\notin F^+(i)} I_k|}\cdot 1_{j\notin \cup_{k\in F^+(i)} I_k}, \\ t_1 &= 1. \end{aligned} \tag{4}$$

The second case is that the data item d_j has been cached in a friend node of N_i. There are also three subcases, i.e., d_j is cached in the self component, the friends component, and the strangers component of the friend nodes, respectively. Denote the total probability and average data access delay for this case as P_2 and t_2, respectively. By using the same analysis we discuss in the first case, we have:

$$\begin{aligned} P_2 &= \frac{(a+b)|F(i)|}{|\cup_{k\in F(i)} I_k|}\cdot 1_{j\in \cup_{k\in F(i)} I_k} + \frac{|F(i)|c}{|\cup_{k\notin F^+(i)} I_k|}\cdot 1_{j\notin \cup_{k\in F^+(i)} I_k}, \\ t_2 &= \frac{1}{\bar{r}\Lambda}. \end{aligned} \tag{5}$$

Note that the data item is repeated by $\bar{r}$ times. The delay for this case is for N_i to access any one of them. Thus, the average data access delay is $\frac{1}{\Lambda}$ multiplied by $\frac{1}{\bar{r}}$.

The third case is that the data item d_j has been cached in a stranger node of N_i. Since the average request frequency for each data item is $\bar{p}$, the average probability for the data item to be cached in the strangers of N_i is $\frac{\bar{p}(n-|F^+(i)|)}{n}$. Denote the total probability and average data access delay for this case as P_3 and t_3, respectively. Then, we have:

$$P_3 = \frac{\bar{p}(n-|F^+(i)|)}{n}, \quad t_3 = \frac{1}{\bar{r}\lambda}. \tag{6}$$

The fourth case is that the data item d_j has been cached in any node. N_i receives d_j through WiFi or WiMAX from the APs, if its request has not been satisfied in a predefined waiting time T. Then, the corresponding total probability and average data access delay, P_4 and t_4, are:

$$P_4 = 1 - P_1 - P_2 - P_3, \quad t_4 = T, \tag{7}$$

Based on the above analysis, we can get the average data access delay time $t(i,j) = P_1\cdot t_1 + P_2\cdot t_2 + P_3\cdot t_3 + P_4\cdot t_4$.

V. Simulation

A. Comparison Scheme and Evaluation Metrics

In the simulation, we compare our proposed scheme with the following caching mechanisms in MOSNs:

Random Cache, in which every request node caches the received data items in its cache space to facilitate data access in the future.

Selfish Cache, which is similar as CacheData [17] in mobile ad-hoc networks. The mobile nodes cache the pass-by data items, according to their popularity in its own point of view.

Unselfish Cache, in which every mobile node only caches the data items for other nodes, according to their knowledge about the data items' request frequency of their encountered nodes.

The cache replacement policies are different for these caching schemes above. For the Random Cache scheme, the data item will be replaced randomly when the cache space is full. For Selfish Cache and Unselfish Cache schemes, the cache replacement policy will vary, according to the popularity of the data items.

In this paper, the following metrics are used for evaluations:

- **Data access delay**: the average delay for receiving the request data item.
- **Successful ratio**: the ratio of queries being satisfied with the requested data item within the deadline.
- **Overhead**: the average number of data copies being cached in the whole network.

B. Simulation Setting

We compare the performance of our proposed hierarchical cooperative caching scheme with other three schemes we discussed above, on the *Infocom2006 trace*, which is collected by the computer laboratory at the University of Cambridge in the Haggle project [18], and *MIT reality mining trace*, which is gathered by the MIT reality mining project [19].

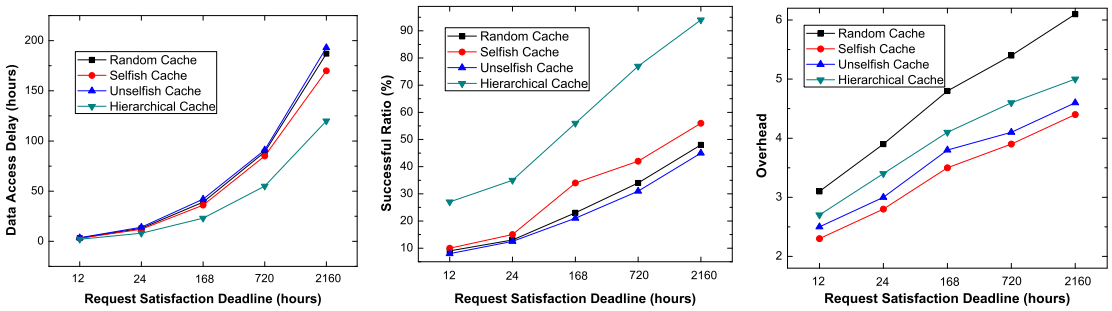
In all experiments, the first half of the trace is used for the learning process, which is for the accumulation of the network information, the process of distinguishing friends from strangers, and the process of learning data item preference of the other nodes. The data generation and requests happen during the second half of both traces.

In the Infocom2006 trace, there are 78 participants with Bluetooth embedded iMote devices to record their contacts during a 4-day conference. There are 128,979 internal contacts among these participants. In the MIT reality mining trace, there are 97 participants with Bluetooth embedded cellphones to record their contacts. The duration of the MIT reality mining trace is 246 days. It records 822,626 internal contacts among these participants.

In the simulation, we assume that the lifetime of the generated data item is infinite. The period for data item generation is set to $2T$, which is according to the request satisfaction deadline in Section IV-B. The data request pattern of each mobile node follows Zipf-like distribution, as we discussed in Section IV-A. Each request has a finite time constraint T. Every time T, each node N_i determines whether to request data item d_j in probability $P_i(j)$, according to Eq. 2, and we set β to 0.5. After time T, if the requested data item has not been received, we suppose an unsuccessful data request, and the mobile node will download this data item from the APs directly. Hence, in this situation, the data access delay is T.

We assume that the data item size is the same as 20MB, and the caching buffer size of the mobile nodes is adjusting in range $[180MB, 900MB]$ for comparison purposes. We compare the performance of the comparison schemes in the following four categories:

1) **Varying data request frequency**: comparing the performance of different values of T.
2) **Varying buffer space**: comparing the performance of different values of the buffer space of each node.



(a) Data access delay (b) Successful ratio (c) Overhead

Fig. 2. Comparison of the performance with different varying data request frequency.

3) **Varying sizes of each component in hierarchical cooperative caching scheme**: comparing the performance under different sizes of each component in the hierarchical cooperative caching scheme.

In the first two categories, we assume that the size of each component in the hierarchical cooperative caching scheme is equal, $\frac{1}{3}$ of the total buffer size for each.

C. Simulation results

1) Varying data request frequency: we first evaluate the different caching schemes in varying data request frequencies in the MIT reality trace. We set the buffer size of the mobile nodes (S) to 540MB. Then, we adjust the data request satisfaction deadline (T) from 12 hours to 3 months.

Fig. 2 shows the simulation results with different values of T. We find that the performance is mainly restrained by the data request satisfaction deadline. It is clearly showing that our proposed hierarchical cooperative cache scheme has much better performance than any other schemes. As shown in Fig. 2(a), our scheme has 33.5%, 38.2%, and 41.7% shorter delay than selfish cache, random cache, and unselfish cache schemes, respectively. By comparing the successful ratio in Fig. 2(b), we find that our scheme can increase to 104%, 147%, and 168% successful ratio, compared with selfish cache, random cache, and unselfish cache schemes, respectively.

Fig. 2(c) shows that hierarchical cooperative cache scheme only requires moderate overhead, which is much lower than random cache scheme. When T is 12 hours, our scheme only has 2.7 average copies of each data item, while the random cache scheme has 3.1 copies, which increases the cost by about 15%. When T increases to 3 months, the random cahe scheme has 6.1 average copies, which increases the cost even more, to about 22%, compared to our scheme. The major reason is that, in the random cache scheme, each node caches any data item it receives until its buffer space is full. The selfish cache and unselfish cache schemes can reduce about 12% overhead, compared with our scheme. Since our scheme has much shorter delay and much higher successful ratio, we claim that our scheme is more cost-effective.

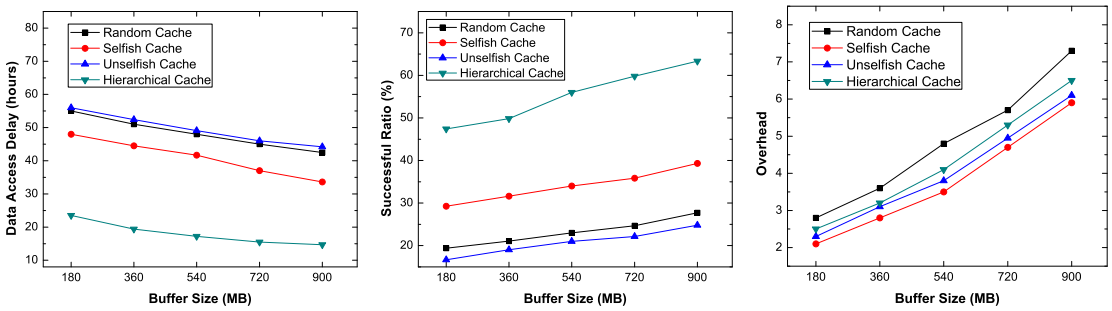
2) Varying buffer space: we evaluate the performance with different buffer size constraints in the MIT reality trace. We set the data request satisfaction deadline (T) to 1 week. Then, we adjust the buffer size (S) from 180MB to 900MB.

It shows that when the buffer size becomes larger, more data items can be cached, as shown in Fig. 3(c). Hence, the data access delay reduces and the successful ratio increases in Figs. 3(a) and 3(b). When S increases from 180MB to 900MB, the data access delay of our proposed hierarchical cooperative cache scheme decreases about 37.5%, and the successful ratio increases from 47% to 63%. From Fig. 3(a), we find that our scheme can have 32% less delay, compared with the selfish cache scheme, and 39% less delay than the random cache scheme. Fig. 3(b) shows that our scheme can increase the overall successful ratio by about 63%, 139%, and 168% compared with the slefish cache, random cache, and unselfish cache schemes, respectively. In Fig. 3(c), it shows that our scheme slightly increases the overhead compared with the selfish cache and unselfish cache schemes. Our scheme has less copies of data items than the random cache scheme.

3) Varying size of each component in hierarchical cooperative caching scheme: in this part, we will compare the performance of our proposed hierarchical cooperative caching scheme with different sizes of each component, which can indicate the importance of all components in designing the caching scheme, in Infocom2006 trace.

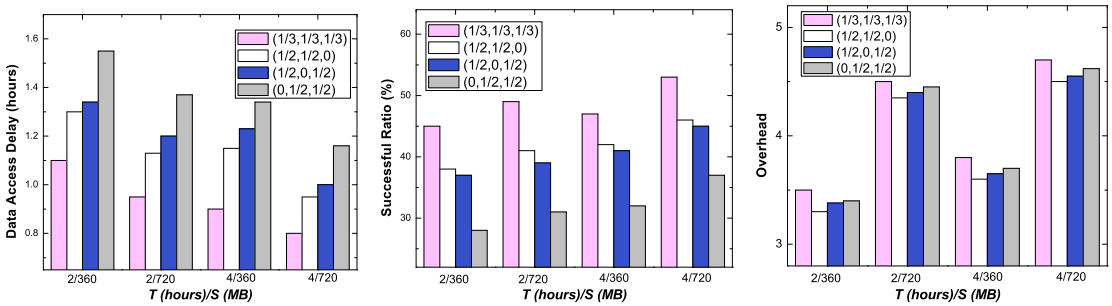
We compare four conditions of our scheme: the first one includes all components, and each one is $\frac{1}{3}$ of the whole buffer space – $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$; the second one only includes self and friends components, and each one is half of the whole buffer space – $(\frac{1}{2}, \frac{1}{2}, 0)$; the third one includes self and strangers components, and each one is half of the whole buffer space – $(\frac{1}{2}, 0, \frac{1}{2})$; the final one includes friends and strangers components, and each one is half of the whole buffer space – $(0, \frac{1}{2}, \frac{1}{2})$. We set T to 2 or 4 hours, and S to 360MB or 720MB.

The simulation results are shown in Fig. 4. It shows that our scheme, including all components, has the best performance among all schemes, which means that all components are very important in cooperative caching scheme. Fig. 4(a) shows that missing the strangers component, the delay will increase about 21%; missing the friends component, the delay will increase about 27.5%; and missing the self component, the delay will increase about 44.8%. In Fig. 4(b), we find that missing one of the three key components, the successful ratio decreases about 14%, 16.5% or 34%, resepectively. For the average number of copies, the four schemes perform similarly.



(a) Data access delay (b) Successful ratio (c) Overhead

Fig. 3. Comparison of the performance with different buffer sizes.



(a) Data access delay (b) Successful ratio (c) Overhead

Fig. 4. Comparison of the performance with different sizes of each key component.

D. Summary of Simulation

We evaluate our proposed scheme with other existing schemes in different network conditions. The performance of all schemes are restrained by the buffer size and data request frequency constraints. Our proposed hierarchical cooperative cache scheme performs best in all conditions, which reduces the data access delay and increases the successful ratio dramatically. To evaluate the importance of all components in our scheme, we compare the performance by deleting one of the key components. The simulation results show that the scheme, including all components, has the best performance.

VI. Conclusion

In this paper, we propose a hierarchical cooperative caching scheme in mobile opportunistic social networks (MOSNs). We divide the cache space into three key components: self, friends, and strangers. In the self component, each mobile node caches its most frequently requested data items. In the friends component, each node helps its friend nodes to cache the data items which are the most popular for its friends. In the strangers component, the remaining data items are randomly cached. We also investigate the cache replace policies for each component, respectively. Then, we formally analyze the data access delay by using the proposed hierarchical cooperative caching scheme. Trace-driven simulation results show that our proposed caching scheme performs better than other schemes. Our future work will focus on the adaptive adjustment of the size of each component, based on the popularity of the mobile nodes in the network.

References

[1] M. J. Pitkänen and J. Ott, "Redundancy and distributed caching in mobile DTNs," in *Proc. of ACM/IEEE MobiArch*, 2007.

[2] Y. Huang, Y. Gao, K. Nahrstedt, and W. He, "Optimizing file retrieval in delay-tolerant content distribution community," in *Proc. of IEEE ICDCS*, 2009.

[3] W. Gao, G. Cao, A. Iyengar, and M. Srivatsa, "Supporting cooperative caching in disruption tolerant networks," in *Proc. of IEEE ICDCS*, 2011.

[4] X. Zhuo, Q. Li, G. Cao, Y. Dai, B. Szymanski, and T. La Porta, "Social-based cooperative caching in dtns: A contact duration aware approach," in *Proc. of IEEE MASS*, 2011.

[5] Y. Zhang and J. Zhao, "Social network analysis on data diffusion in delay tolerant networks," in *Proc. of ACM MobiHoc*, 2009.

[6] Y. Wang, J. Wu, and W.-S. Yang, "Cloud-based multicasting with feedback in mobile social networks," *IEEE Transactions on Wireless Communications*, vol. 12, no. 12, 2013.

[7] J. J. Brown and P. H. Reingen, "Social ties and word-of-mouth referral behavior," *Journal of Consumer Research*, vol. 14, no. 3, 1987.

[8] X. Zhang, G. Neglia, J. Kurose, and D. Towsley, "Performance modeling of epidemic routing," *Comput. Netw.*, vol. 51, no. 10, 2007.

[9] A. Krifa, C. Baraka, and T. Spyropoulos, "Optimal buffer management policies for delay tolerant networks," in *Proc. of IEEE SECON*, 2008.

[10] M. Seligman, K. Fall, and P. Mundur, "Alternative custodians for congestion control in delay tolerant networks," in *Proc. of the SIGCOMM workshop on Challenged networks*, 2006.

[11] Y. Wang, J. Wu, Z. Jiang, and F. Li, "A joint replication-migration-based routing in delay tolerant networks," in *Proc. of IEEE ICC*, 2012.

[12] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *Proc. of IEEE INFOCOM*, 2010.

[13] C. Boldrini, M. Conti, and A. Passarella, "Design and performance evaluation of contentplace, a social-aware data dissemination system for opportunistic networks," *Comput. Netw.*, vol. 54, no. 4, 2010.

[14] D. S. Hochbaum, "Heuristics for the fixed cost median problem," *Mathematical Programming*, vol. 22, 1982.

[15] L. Breslau, P. Cue, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *Proc. of IEEE INFOCOM*, 1999.

[16] A. Balasubramanian, B. Levine, and A. Venkataramani, "DTN routing as a resource allocation problem," in *Proc. of ACM SIGCOMM*, 2007.

[17] L. Yin and G. Cao, "Supporting cooperative caching in ad hoc networks," *IEEE Trans. on Mobile Computing*, vol. 5, no. 1, Jan. 2006.

[18] J. Scott, R. Gass, J. Crowcroft, P. Hui, C. Diot, and A. Chaintreau, "CRAWDAD trace cambridge/haggle/imote/infocom2006 (v. 2009-05-29)," May 2009.

[19] N. Eagle, A. Pentland, and D. Lazer, "Inferring social network structure using mobile phone data," in *PNAS*, vol. 106(36), 2009.