Steady Status Study of Distributed Data Caching in Ad Hoc Networks

Julinda Taylor[1], Bin Tang[2], Mehmet Bayram Yildirim[3]
[1]Department of Electrical Engineering and Computer Science, Wichita State University, Wichita, KS, USA
[2]Department of Computer Science, Azusa Pacific University, Azusa, CA, USA
[3]Department of Industrial and Manufacturing Engineering, Wichita State University, Wichita, KS, USA
Email: {jltaylor, bayram.yildirim}@wichita.edu, btang@apu.edu

Abstract— **There has been extensive research on distributed data caching in ad hoc networks. However, most of them focus on how to reduce the average delay of requests and improve the packet delivery ratio, not much work has been done to study the steady-state status achieved by the distributed caching algorithms. Information related to steady-state status includes the convergence time of the distributed caching algorithms, the final data replica placement in the network, and the performance comparison of distributed caching algorithms with optimal centralized caching solution. Previous theoretical results show that to minimize the average access cost (or average search cost) in the network, the optimal number of replica of each data object is proportional to the square root (or two-third) of the data's access frequency. In this work, we empirically show that the optimal number of replicas of each data depends on not only the access frequencies of the data, but also the storage capacity of each node. We evaluate the steady-states of both cooperative distributed data caching technique and a selfish caching in ad hoc networks, and compare them to that of the optimal caching solution obtained using integer linear programming (ILP). Via ns2 simulations, we gain some insights about the steady-states of distributed data caching.**

Keywords – Performance Evaluation, Distributed Data Caching, Ad Hoc Networks

I. Introduction

Ad hoc networks are multi-hop wireless networks consisting of small wireless computing devices such as conventional computers (e.g., PDA, laptop, or PC), or embedded processors such as tiny, low-cost, and low-power sensor motes. Ad hoc networks are constructed mainly for the information sharing and task coordination among a group of people, without the support of any communication infrastructure. For example, in an ad hoc network established for spontaneous meeting, several authors can meet and coordinate to modify the same document (e.g., an article or a powerpoint slides) in a distributed fashion. Similarly, in interconnected distributed information systems, an object (a web page, an image, a video clip, or a file) may be accessed from multiple distributed locations (network nodes) simultaneously.

Caching has been proposed to be an effective technique to facilitate information access in ad hoc networks. Besides the traditional advantages brought by caching such as less data access delay, improved data reliability and fault tolerance, utilizing caching to optimize network performance of ad hoc networks is further motivated by the following two aspects. First, the ad hoc networks are multi-hop networks. Thus, remote access of information typically occurs via multi-hop routing, wherein access latency can be particularly improved by data caching. Second, ad hoc networks are generally resource constrained in terms of wireless bandwidth, storage capacity and battery energy of nodes. Data caching can help reduce communication cost among nodes, which results in conserving battery energy and minimizing bandwidth usage in ad hoc networks.

The data caching problem in ad hoc networks has its theoretical root in the multi-facility location problem [2], [1], which is NP-hard. The problem asks: given an ad hoc network graph, the storage capacity of each node, the initial data distribution in the network and the data sizes, the data access pattern (the probability with which each node accesses each data), how to find the optimal data placement in the ad hoc network such that the average access cost of each node is minimized?

There has been extensive research on designing cooperative distributed data caching algorithms in ad hoc and sensor networks (please refer to Section II for a comprehensive literature review). Even though some of the centralized caching algorithms (e.g. [19]) have endeavored to achieve performance bounds, the distributed caching algorithms (e.g. [19], [21]) are all heuristic based without any performance guarantee. More specifically, none of the existing distributed data caching algorithms studies the final cache/replica placements produced in these algorithms and compares to the optimal replica placement. Thus it is unclear how they perform compare to the optimal data caching algorithm.

Therefore, there is a need to study the "stable" cache placement yielded by distributed caching algorithms and to define the corresponding steady-state of data caching. Steady-state is an important stage at which the quality of the distributed data caching techniques is evaluated – by defining and quantifying the steady-state of distributed data caching, we can gain insight on how it performs compared to the optimal caching algorithm. Specifically, we are asking the following questions: a) Does the steady-state of distributed caching always exist in ad hoc networks? How to define and characterize steady-state of distributed data caching? b) How fast does each caching algorithm achieve such steady-state if it exists (i.e., how fast is the converge time)? c) How do the distributed caching algorithms compare to centralized optimal caching algorithm in terms of average data access cost based on the cache placement in steady-state?

Optimality of cache placement has been studied in the context of mesh networks and peer-to-peer networks [14], [4], [15]. Jin and Wang [14] find that to minimize average access cost in multi-hop wireless mess networks, the number of replicas of each object in optimal solution is proportional to $p^{2/3}$, where p is the access probability of the object. The work by Cohen and Shenkar [4] and by Lv et al. [15] are one of the first theoretical studies of replication in unstructured peer-to-peer networks. They show that to minimize search cost in unstructured peer-to-peer networks, the optimal replication strategy is to replicate objects in proportional to the square root of their popularity (i.e. access probability). We refer to these two caching strategies as $p^{2/3}caching$ and $p^{1/2}caching$ throughout the paper. However, both work do not show how storage capacity of each node affects the optimal number of replicas. In this paper, we empirically show that the number of each data's replica not only depends on the access frequencies of data, but also depends on the storage capacity of each node.

On the other hand, the steady-state behaviors of cache replacement schemes (such as LRU) have been evaluated using either exponential time complexity or approximated closed form expression [18], [12], [5]. In this paper we focus on experimental study of the steady-states of a co-operative data caching algorithm and a selfish data caching technique. Specifically, we obtain the cache placements in distributed caching algorithms and use them to calculate the average data access cost, and compare to the optimal cache placement and optimal average data access cost using integer linear programming (ILP) approach. We have two main observations. First, we empirically show that the optimal number of replicas of each data in a network not only depends on the access frequencies of data, but also depends on the storage capacity of each node. Second, we show that selfish caching, which was shown to perform much worse than cooperative caching, perform comparable with cooperative caching in terms of the average data access cost based on the cache placement of the steady-states. We give some analysis of this phenomenon. To the best of our knowledge, our work is the first one to formally formulate the data caching problem in ad hoc networks using ILP and solve it optimally.

Paper Organization. The rest of the paper is organized as follows. We discuss the related work in Section II. In Section III, we present our network model and formulate the data caching problem using ILP. Section IV presents the cooperative and selfish caching schemes studied in this paper. Section V proposes the model and defines the steady-states for the distributed data caching. We present and analyze the simulation results in Section VI. Section VII concludes the paper and points out some future work.

II. **Related Work**

Data caching in ad hoc and sensor networks has been an active research area. Below are a few work published recently. Du et al. [7] propose COOP, a novel cooperative caching scheme for on-demand data access applications in MANETs. The objective is to improve data availability and access efficiency by collaborating local resources of mobile nodes. Montanari et al. [16] use probabilistic failure models to adaptively create and maintain a number of replicas of the data, to provide data availability in sensor networks. Dimokas et al. [6] propose a new cache consistency and replacement policy in a wireless multimedia sensor networks, with the goal of latency minimization. Wu et al. [20] utilize the overhearing property of wireless communications for performance improvement of data caching in ad hoc networks. Fan et al. [9] design distributed caching heuristics, via which better performance can be achieved by detecting the variation of contentions to evaluate the benefit of selecting a node as cache node.

Hara and Madria [13] are among the first to propose replica allocation methods in ad hoc networks, by taking into account the access frequency from mobile hosts to each data item and the status of the network connection. Yin and Cao [21] design and evaluate three distributed caching techniques, viz., *CacheData* which caches the passing-by data item, *CachePath* which caches the path to the nearest cache of the passing-by data item, and *HybridCache* which caches the data item if its size is small enough, else caches the path to the data. Fiore et al. [11] design a cooperative caching scheme to create a content diversity in ad hoc networks, so that a requesting user likely finds the desired information nearby. Zhao et al. [21] propose a novel asymmetric cooperative cache approach, where the data requests are transmitted to the cache layer on every node, but the data replies are only transmitted to the cache layer at the intermediate nodes that need to cache the data. Tang et al. [19] present a polynomial-time centralized approximation algorithm to replicate data, which reduces the total data access delay at least half of that obtained from the optimal solution. They also present a distributed caching technique, called benefit-based data caching, derived from the centralized approximation algorithm.

However, all the above work does not study the steady-state of distributed data caching, which is the topic of this work. Specifically, we experimentally study the steady-state of the benefit-based caching [19] and a selfish caching scheme, and compare their cache placements with that of the optimal solution.

III. **ILP Problem Formulation of the Data Caching Problem**

A multi-hop ad hoc network can be represented as an undirected graph $G(V, E)$ where $V = \{1, 2, ..., i, ...|V|\}$ represents the nodes in the network, and E is the set of weighted edges in the graph. Two network nodes that can communicate directly with each other are connected by an edge in the graph. There are multiple data items in the network, and each is served by its *source node*. Each network node has limited storage space and can cache multiple data items subject to the storage constraint. The objective of the data caching problem is to minimize the total (or average) access cost. Below, we give a formal definition of the cache placement problem addressed.

Network Model. The set of data items in the network is $D = \{1, 2, ..., p, ..., |D|\}$, where data item $p \in D$ is originally stored at its source node $S_p \in V$ (it is possible that a source node can have multiple data items). Source nodes always keep their original data. The size of data item p is s_p units. Node $i \in V$ has a storage capacity of m_i units (for source node i, m_i is the available storage space after storing its original data). Two nodes can communicate with each other if they are within the transmission range of each other. The edge weight may represent a link metric such as loss rate, distance, delay, or transmission bandwidth. In this paper, the edge weight represents the bandwidth and we assume that all the edges have the same bandwidth B.

We use a_{ip} to denote the access frequency with which node i requests data item p, i.e., the number of times node i requesting the data item p within unit time. The transmission time of sending data item p along any network edge is s_p/B. Let d_{ij} denote the number of transmissions to transmit a data item from i to j, which equals to the number of edges of the shortest path between these two nodes. The *total data access cost* in ad hoc network before caching is thus the total transmission time spent by all the nodes to access all the initial copies of the data:

$$\sum_{i=1}^{|V|}\sum_{p=1}^{|D|} a_{ip} \times d_{iS_p} \times s_p/B.$$

We omit B for the rest of the paper. The objective of the data caching problem is to minimize the total data access cost by caching data items in the ad hoc network, under the storage constraints of nodes. For ease of presentation, we assume that a source node is also a cache node. Below, we give a formal definition of the caching problem using integer linear program (ILP).

ILP Problem Formulation. Let $x_{ip} \in \{0,1\}$ denote whether data item p is cached in node i, and let $y_{ijp} \in \{0,1\}$ denote whether node i accesses data item p from node j. Then the objective function is to minimize:

$$\sum_{i\in V}\sum_{j\in V}\sum_{p\in D} a_{ip} \times d_{ij} \times y_{ijp} \times s_p, \quad (1)$$

where

$$\sum_{j\in V} y_{ijp} = 1, \qquad \forall i \in V, p \in D \quad (2)$$

$$x_{jp} - y_{ijp} \geq 0, \qquad \forall i, j \in V, p \in D \quad (3)$$

$$\sum_{p\in D} s_p \times x_{jp} \leq m_j, \qquad \forall j \in V \quad (4)$$

Condition (2) guarantees that node i accesses data item p from a node in the network. Condition (3) ensures that each data item can only be accessed from a node which stores that data. Condition (4) states that the total size of data cached at node j can not exceed j's storage capacity m_j. The data caching problem is NP-hard since essentially, it is multi-facility location problem wherein each constructed facility is a cache node. Multi-facility location problem is NP-hard [2], [1].

An Example. Fig. 1 shows a simple example of a small ad hoc network of four nodes (0, 1, 2, 3). There are two data items (A and B) in the network – node 0 stores A, node 1 stores B. Every node has one storage capacity, i.e., it can store one data item. For all the nodes, their access frequency to data A is 10, access frequency to data B is 1. The optimal data placement is that node 0 caches a copy of data B, while node 1, 2, and 3 each caches a copy of data A. The optimal total access cost is 2.

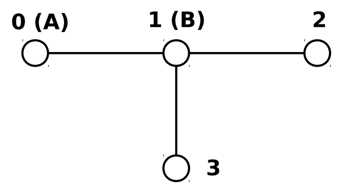

Fig. 1. An illustrative example of optimal data caching.

IV. Cooperative and Selfish Distributed Data Caching

In this section, we first present the data access model of distributed caching. We then discuss two representative algorithms of cooperative data caching and selfish data caching, viz benefit-based caching and LRU-based caching.

Data Access Model of Distributed Caching. Data access model includes data access pattern, which indicates the popularity of the data, and data access interval, which determines the query traffic in the ad hoc network.

- Data Access Pattern. We consider Zipf data access pattern. In this pattern, the data is ranked by their access popularity following *Zipf-like* distribution [22], [3], where the access frequency of i^{th} popular data item is $\frac{1/i^\theta}{\sum_{h=1}^{|D|} 1/h^\theta}$, where $0 \leq \theta \leq 1$. When $\theta = 1$, the above distribution follows the strict Zipf distribution; while for $\theta = 0$, it follows the uniform distribution, in which all $|D|$ data items are equally popular and randomly accessed by the $|V|$ nodes.
- Data Access Interval. Each node in the network sends out a single stream of read-only queries, each of them requesting for a data item following above data access pattern. The *data access interval (or the query generate time)* is the time interval between two consecutive queries and follows exponential distribution with some mean value T_q.

Before each node sends out a data request, it checks if the data is already cached locally. If so, this request is then satisfied locally; if not, it sends out the request. After receiving the data, whether it caches the data locally depends on whether the distributed caching scheme is cooperative or selfish, as explained below.

Cooperative Data Caching [19]. Benefit-based caching is a cooperative data caching algorithm that works as follows. Each node maintains a *nearest cache table*, which records for each data item, the closest cache node (including the source node) that has a copy of the data. If the node itself is a cache node of the data item (i.e., it is the closest cache node for this data item), it records the second-nearest cache node that has a copy of the data. By maintaining the accurate nearest cache table (please refer to [19] for its maintenance mechanism), each node can not only directly access the closest copy of the data item without searching it, but also make intelligent local caching decision with the knowledge of data placement information in its neighborhood. On the other hand, each node observes all the data request messages passing through it. These data request messages include the node's own data requests being a data requestor, the data request messages that the node forwards being a relay node, and data requests it receives being a cache (or source) node of the requested data.

With above nearest cache table and message observation, each node can constantly compute the *local benefit* of each data item. For each data item p not cached at node i, i calculates the local benefit gained by caching p, while for each data item p cached at node i, i computes the local benefit lost by removing it. In particular, the local benefit B_{ip} of caching (or removing) p at node i is the reduction (or increase) in access cost given by

$$B_{ip} = t_{ip}\delta_p,$$

where t_{ip} is the number of request messages observed by node i for data item p, and δ_p is the distance from i to the nearest-node other than i that has the copy of the data item p (obtained from nearest cache table). Using its nearest-cache table, each node can compute the local benefits of data items using traffic information observed locally.

Cooperative Cache Replacement Policy. When there is free storage space, a node caches the passing-by or requested data item. Otherwise, the caching decision is based on the local benefit calculation – if the benefit of caching the newly passing-by or requested data item is larger than that of the cached data item with the smallest benefit, cache replacement takes place. Note that cooperative cache replacement scheme differs significantly from most existing cache schemes (such as LRU below), since data item most beneficial to the network (not just the data item most beneficial to the node itself) is cached.

Selfish Data Caching. In selfish caching, each node accesses the source node for each data item. It caches any passing-by or requested data item when it has free storage space. When its storage space is full, it uses Least Recently Used (LRU) policy [10] to replace cached data. Each node's request can also be satisfied on the way to the source node if one of the intermediate nodes caches the requested data. We call this caching scheme selfish because each node caches its requested data item and passing-by data items to benefit its own data access, not others.

V. Heuristics Model of Distributed Data Caching

Let M_p^t be the set of network nodes that store a copy of data p at time t. We assume that a source node is also a cache node, i.e., $S_p \in M_p^t$. Given the cache placement of the network at time t, the total access cost of the network is the sum of the access costs of all the nodes, each of which goes to the nearest cache node to access each data item. Denote the total access cost of the network at time t as $\tau(t)$, then

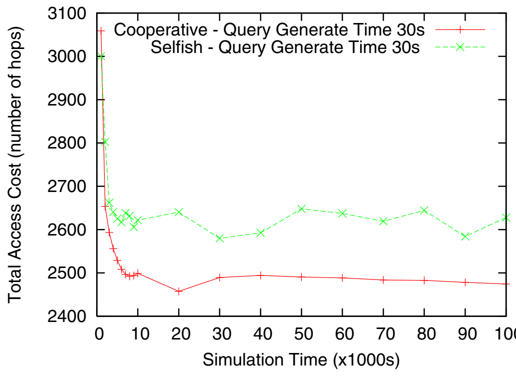

Fig. 2. Convergence time of cooperative and selfish caching.

$$\tau(t) = \sum_{i=1}^{|V|}\sum_{p=1}^{|D|} a_{ip} \times \min_{l \in M_p^t} d_{il} \times s_p. \qquad (5)$$

Equation 5 is similar to Equation 1, with one difference. Equation 1 is used to find a *centralized* optimal caching strategy to minimize the total (or average) access cost, while Equation 5 is used to calculate the total access cost (which is not optimal) based upon a specific cache placement resulted from a *distributed* caching strategy at a specific time. Below we formally define steady-state and convergence time of a distributed data caching algorithm.

Definition 1: (Steady-state and convergence time of a distributed data caching.) The distributed caching is in its steady-state at time t_s if and only if for any time $t'_s > t_s$, $|\tau(t_s) - \tau(t'_s)| \leq \tau_{th}$, where τ_{th} is a prefixed threshold value. Convergence time is the smallest time elapse for a caching scheme to reach its steady-state. □

Therefore, whether a distributed caching scheme reaches its steady-state depends on the choice of τ_{th} (in the simulation, we choose τ_{th} to be small enough in order to quantify steady-state). Below we empirically study the convergence time of above two distributed caching algorithms.

Empirical Study of the Convergence Time. We take the cache placement at some different times of the caching process and calculate the total access cost of the network using Equation 5. Figure 2 shows that the cooperative algorithm stabilizes very well (it reaches its steady-state at around $30,000$ seconds), while selfish caching does not. This can the explained by the different cache replacement policies adopted in each algorithm. In cooperative caching, nodes only cache data which are beneficial (i.e., reducing the access cost in the network). In selfish caching, nodes adopts LRU cache replacement policy, which always caches the new data even it is not a beneficial one to the network. In an extreme case where each node's storage can only store one data, in selfish caching, node always stores the latest requested data, which is changing dynamically all the time. However, the cooperative one will stick with the beneficial data and do not replace it if the newly requested data is less beneficial. For the rest of the paper, we adopt the simulation run time of 100,000 seconds to study the steady-states of both algorithms.

Comparison Between Cooperative Caching and Selfish

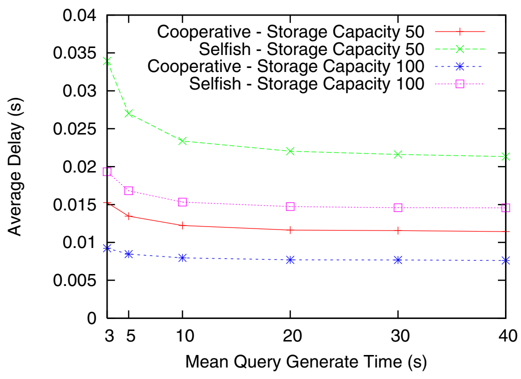

Fig. 3. Average delay comparison between cooperative and selfish caching.

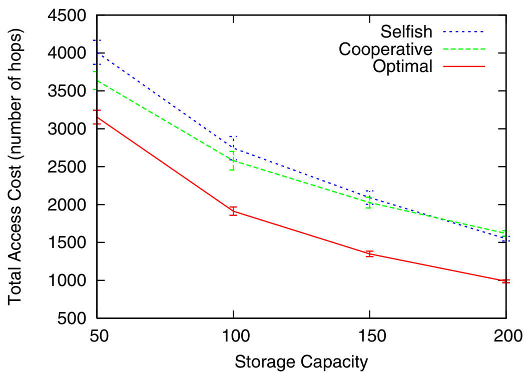

Fig. 4. Total access cost comparison based on the cache placement obtained at steady-states (simulation time is $100,000$ seconds). Here, the query generate time is 30s.

Caching at Steady-States. In cooperative caching, nodes cache data based on whether it is beneficial to the whole network. Meanwhile, since the selfish caching uses LRU as its cache replacement policy, it eventually keeps its popular data in its local memory, even though such data could be available in its close neighborhood. Therefore, cooperative caching achieves smaller average access cost than selfish caching does. Figure 3 shows the comparison of the average query delay of both caching schemes by varying the mean query generate time from 3 to 40 seconds, under difference storage capacity (50 and 100 data items). It shows that when node storage capacity is 100 data items, cooperative caching performs about twice as better as the selfish caching does. The performance differential gets larger in more challenging scenarios, such as higher query traffic (when query generate time is 3 seconds) and smaller node storage (50 data items).

VI. Performance Evaluation

We demonstrate through simulations the performance of our designed cache placement algorithms over randomly generated network topologies. We first compare the relative quality of the cooperative and selfish caching schemes using the ns-2 simulator [8].

Simulation Setup. We simulated our algorithms on a network of randomly placed 40 nodes in an area of $1000 \times 1000m^2$. We adopt DSDV [17] as the underlying ad hoc routing protocol. The transmission range for two directly communicating nodes in the *ns2* simulator is 250 meters. The wireless bandwidth is 2 Mb/s. In our network, there are 500 data items, each of 750 bytes. There are two randomly placed source nodes S_0 and S_1 where S_0 stores the data items with even IDs and S_1 stores the data items with odd IDs. Each ad hoc node sends out a single stream of read-only queries following the Zipf distribution. We choose θ to be 0.8 based on real web trace studies [3]. The access frequency of data item 0 is set as 10, while other data's access frequency is calculated accordingly using Zipf function. In the cooperative caching scheme, the query to a data item is forwarded to the nearest cache of that data item (based on the nearest-cache table). In the Selfish scheme, the query is forwarded to the source nodes. In both schemes, if the query encounters a node with the requested data item cached, then the query is answered by the encountered node itself.

Total Access Cost in Steady-States. Figure 4 shows the total access cost comparison between ILP optimal, cooperative, and selfish caching, by varying storage capacity of nodes. The total access cost is calculated based on the cache placement at the steady states using Equation 5. In this figure, each data point represents an average of five runs on different network topologies, and the error bars indicate the 95% confidence interval (we only show confidence intervals in Figure 4 since other figures are clear to our claim).

We have the following observations. First, it shows that both cooperative and selfish perform worse than optimal ILP solution, with 25% of difference; however, the selfish caching performs comparably with the cooperative one. This seems contradicting with Figure 3, which shows that cooperative caching performs about twice as better as the selfish caching. This is because for distributed selfish caching scheme, the data access is usually not from the closest cache since each node is not aware other close-by cache nodes, while in total access cost calculation, both selfish and cooperative caching assumes that data accesses from the closest cache following Equation 5. This demonstrates one important difference between the theoretical root (multi-facility location problem) and the distributed implementation of the caching schemes: even though a distributed caching scheme yields a good cache placement, it could still perform poorly in terms of the user-perceived access delay. Second, when storage capacity is small at 50, cooperative does perform better than selfish, showing its superiority to selfish caching in challenging scenarios. However, their performance difference is only around 15%, which is much smaller than the two-time difference if we use the average delay shown in Figure 3.

Number of Copies of Data Items in Steady-States. Next we study the number of copies of data items in steady-states of distributed caching. We compare with the previous results, viz. $p^{2/3}caching$ and $p^{1/2}caching$ [14], [4], [15]. Figure 5 and 6 show the varying of number of copies of data items with respect to their popularity rankings, under storage capacity of 100 data items and 200 data items respectively. We only show the number of

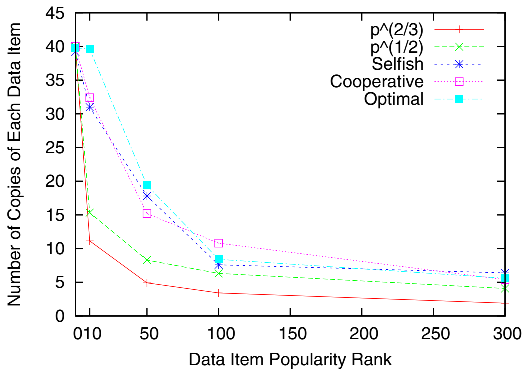

Fig. 5. Number of replicas for data with different access frequencies in steady-states (simulation time is 100, 000 second). Here, storage capacity is 100 data items.

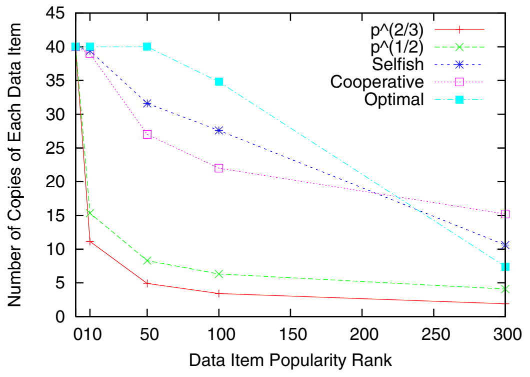

Fig. 6. Number of replicas for data with different access frequencies in steady-states (simulation time is 100, 000 seconds). Here, storage capacity is 200 data items.

copies for data items 0, 10, 50, 100, and 300. We have the following observations. First, when node storage capacity is small (100 data items), both cooperative and selfish schemes correspond with the ILP optimal well. However, when the storage capacity increases to 200 data items (Figure 6), their performance difference becomes obvious – comparing to cooperative and selfish, ILP optimal caches more copies for the more popular data and less copies for less popular data. In both cases, seems selfish caching performs a little better than cooperative caching in terms of number of copies of more popular data cached. Second, as shown in Figure 5 and 6, both $p^{2/3}$ and $p^{1/2}$ caching perform worse compared to ILP optimal, and in large storage capacity case, the performance difference is more significant. This shows that the two existing analytical results do not take into account the effect of storage capacity upon the optimal replica numbers.

VII. **Conclusions and Future Work**

In this paper we study the steady-states of distributed data caching in wireless ad hoc networks. Unlike previous results showing the superiority of cooperative caching to the selfish caching, this work demonstrates that the average data access cost based on the cache placement at steady-states are comparable for the selfish and cooperative caching. Besides, we formulate and solve the data caching problem optimally using integer linear programming, and compare the optimal solution with the distributed caching techniques. We empirically show that the optimal replica number not only depends on the access frequencies of data, but also depends on the storage capacity of each node. We plan to augment our finding and the existing results (viz., $p^{2/3}$ and $p^{1/2}$) by deriving an analytical model for the steady-states with the storage capacity incorporated.

References

[1] I. Baev and R. Rajaraman. Approximation algorithms for data placement in arbitrary networks. In *Proc. of ACM-SIAM SODA*, 2001.

[2] I. Baev, Rajmohan Rajaraman, and Chaitanya Swamy. Approximation algorithms for data placement problems. *SIAM J. Comput.*, 38(4):1411–1429, 2008.

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

[4] E. Cohen and S. Shenkar. Replication strategies in unstructured peer-to-peer networks. In *Proc. of ACM SIGCOMM*, 2002.

[5] Asit Dan and Dan Towsley. An approximate analysis of the lru and fifo buffer replacement schemes. In *Proc. of ACM SIGMETRICS, 1990*.

[6] N. Dimokasa, D. Katsarosb, and Y. Manolopoulosa. Cache consistency in wireless multimedia sensor networks. *Ad Hoc Networks*, 8(2):214–240, 2010.

[7] Yu Du, Sandeep K. S. Gupta, and Georgios Varsamopoulos. Improving on-demand data access efficiency in manets with cooperative caching. *Ad Hoc Netwetworks*, 7(3):579–598, 2009.

[8] K. Fall and K. Varadhan (Eds.). The *ns* manual. available from http://www-mash.cs.berkeley.edu/ns/.

[9] Xiaopeng Fan, Jiannong Cao, and Weigang Wu;. Contention-aware data caching in wireless multihop ad hoc networks. In *Proc. of IEEE 6th International Conference on Mobile Adhoc and Sensor Systems (MASS '09)*, pages 1–9, 2009.

[10] Ribel Fares, Brian Romoser, Ziliang Zong, Mais Nijim, and Xiao Qin. Performance evaluation of traditional caching policies on a large system with petabytes of data. In *Proc. of the IEEE Seventh International Conference on Networking, Architecture, and Storage (NAS 2012)*, pages 227–234.

[11] Marco Fiore, Francesco Mininni, Claudio Casetti, and Carla-Fabiana Chiasserini. To cache or not to cache. In *Proc. of IEEE INFOCOM 2009*.

[12] Philippe Flajolet, Daniele Gardy, and Loys Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Applied Mathematics*, 39:207–229, 1992.

[13] Takahiro Hara and Sanjay K. Madria. Data replication for improving data accessibility in ad hoc networks. *IEEE Transactions on Mobile Computing*, 5(11):1515–1532, 2006.

[14] Shudong Jin and Limin Wang. Content and service replication strategies in multi-hop wireless mesh networks. In *Proc. of the ACM MSWiM 2005*, pages 79–86.

[15] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *Proc. of the 16th international conference on Supercomputing (ICS '02)*, pages 84–95, 2002.

[16] Mirko Montanari, Riccardo Crepaldi, Indranil Gupta, and Robin Kravets. Using failure models for controlling data availability in wireless sensor networks. In *Proc. of IEEE Infocom Minisymposium, 2009*.

[17] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Proc. of ACM SIGCOMM*, 1994.

[18] David Starobinski and David N. C. Tse. Probabilistic methods for web caching. *Performance Evaluation*, 46(2-3):125–137, 2001.

[19] Bin Tang, Samir Das, and Himanshu Gupta. Benefit-based data caching in ad hoc networks. *IEEE Transactions on Mobile Computing*, 7(3):289–304, 2008.

[20] Weigang Wu, Jiannong Cao, and Xiaopeng Fan. Overhearing-aided data caching in wireless ad hoc networks. In *Proc. of the 2009 29th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW '09)*, pages 137–144, 2009.

[21] Liangzhong Yin and Guohong Cao. Supporting cooperative caching in ad hoc networks. *IEEE Transactions on Mobile Computing*, 5(1):77–89, 2006.

[22] G. K. Zipf. *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*. Addison-Wesley, 1949.