Fair Caching Algorithms for Peer Data Sharing in Pervasive Edge Computing Environments

Yaodong Huang*, Xintong Song†‡, Fan Ye*, Yuanyuan Yang*, and Xiaoming Li†
*Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11794, USA
{yaodong.huang, fan.ye, yuanyuan.yang}@stonybook.edu
†School of Electronics Engineering and Computer Science, Peking University, Beijing, China, 100871
{songxintong, lxm}@pku.edu.cn

***Abstract*—Edge devices (e.g., smartphones, tablets, connected vehicles, IoT nodes) with sensing, storage and communication resources are increasingly penetrating our environments. Many novel applications can be created when nearby peer edge devices share data. Caching can greatly improve the data availability, retrieval robustness and latency. In this paper, we study the unique issue of caching fairness in edge environment. Due to distinct ownership of peer devices, caching load balance is critical. We consider fairness metrics and formulate an integer linear programming problem, which is shown as summation of multiple Connected Facility Location (ConFL) problems. We propose an approximation algorithm leveraging an existing ConFL approximation algorithm, and prove that it preserves a 6.55 approximation ratio. We further develop a distributed algorithm where devices exchange data reachability and identify popular candidates as caching nodes. Extensive evaluation shows that compared with existing wireless network caching algorithms, our algorithms significantly improve data caching fairness while keeping the contention induced latency similar to the best existing algorithms.**

I. Introduction

Our surrounding environments are increasingly penetrated with various kinds of *edge devices*, including mobile phones, tablets, connected vehicles, road-side cameras, and diverse Internet-of-Things (IOT). These devices possess sensing, computing, storage and communication capabilities. They produce pervasive sensing data about physical phenomena in the environment. By sharing sensing data among such peer edge devices, numerous novel applications can be created.

Consider a large outdoor public event (e.g., music festival, university commencement). Smartphones carried by people can capture diverse data, including human activities, their locations, and image/video clips. When shared among peer devices, such data can help people avoid food stands of long lines, discover interesting souvenirs and artifacts, or enjoy images, video clips of special, memorable moments.

Caching is a critical mechanism to enable such peer data sharing among edge devices. The movements of people thus devices, the varying availabilities of data and resources (e.g., battery, storage), constitute a highly fluid environment full of uncertainty and dynamics. By caching the data at willing and capable devices, the availability of data, the robustness and latency in their retrieval, can all be greatly improved. Recent content centric networks [1] even integrate caching as a fundamental component in their design.

‡Work done while visiting Stony Brook University.

Despite some earlier works [2], [3] on caching in wireless networks (MANET), the critical issue of *fairness* has not been addressed. These works focus on reducing the contention, thus latency in data retrieval. They can cause extremely unbalanced caching load, e.g., a few fixed devices are always chosen as cache [2], [4]. Although this may not be an issue in MANET if all devices listen to one authority, it is simply infeasible in edge environments: each device may belong to a different owner, and caching decisions can only be voluntarily accepted, not forcefully mandated.

In this paper, we study how to ensure caching fairness among peer devices in pervasive edge computing environments. We formulate the problem in integer linear programming form, and show that it is the summation of multiple Connected Facility Location (ConFL) problems. We propose an approximation algorithm and prove that it preserves a 6.55 approximation ratio to the optimal solution. We can also achieve very good caching fairness when 75% of data are cached on 71.4% of nodes. We further develop a distributed algorithm where nearby devices exchange data availability to make collaborative caching decisions.

We make the following contributions in this paper:

- We consider caching fairness in data sharing among peer edge devices. On top of contention costs thus retrieval latency, the fairness costs are incorporated into an integer linear programming problem, which is the summation of multiple traditional ConFL problems.
- We design an algorithm by leveraging one of the existing ConFL approximation algorithms. We prove that our algorithm preserves the same approximation ratio as the original ConFL algorithm.
- We further design a distributed algorithm where devices exchange data retrieval costs among 2-hop neighbors to identify candidates with the smallest costs, and popular candidates volunteer to cache data.
- We implement our algorithms and compare against other algorithms for distributed wireless network caching. The results show that our algorithms significantly improve data caching fairness while keeping the contention induced latency similar to the best existing algorithms with $O(N^3)$ complexity, where N is the number of nodes in the network.

To the best of our knowledge, *this is the first work to consider caching fairness for peer data sharing among edge*

devices. The rest of the paper is organized as follows: In Section II we discuss some related work on mobile caching and facility location problem. In Section III we give the system model and the formulation of the problem. We provide algorithms in Section IV to solve the problem. We evaluate our design and compare with other previous works in Section V. Finally, we conclude our work in Section VI.

II. Related Work

Caching is one classical mechanism to improve data access robustness and performance. It has been applied in various scenarios. Cooperative caching shares and coordinates data caching decisions among nodes has been applied in ad hoc networks. Yin et al. [5] propose two caching schemes and a method to obtain data in mobile networks. Hara et al. [6] propose a strategy to remove redundancy in neighborhood, and Hamlet [7] minimizes access costs by leveraging content diversity of different data in the neighborhood.

There are also some existing works that focus on improving data access rates by caching in wireless networks. One basic idea is to place cache based on content popularity such that the cached content can be used frequently. WAVE [8] decides popularity based on recommendation from upstream node request counts. Li et al. [9] dynamically place caching replicas on the en-route path in named data networks (NDN) [10], and MPC [11] caches only popular content adapted in Content-Centric Networks (CCN) [12]. Another approach is to find caching locations that minimize data access latency. Nuggehalli et al. [13] use the hop-count as the delay model to find the best places to caching data. Later, Fan et al [3] propose a contention aware caching algorithm, which is more accurate than the hop-count based algorithm since the packet contention incurs the most latency in MANET. Similarly, Sung et al. use contention as a key factor in determining the delay. They introduce a contention based solution on flat wireless networks [14], extend to two-tier wireless content delivery networks [4]. Caching also helps dealing with mobility in edge computing scenarios. Proactively caching data near where nodes need data can improve the data accessing rate despite the high node mobility [15], [16].

The problem of determining caching locations is closely related to the classical Facility Location (FL) problem. Most caching studies map their problems into different FL problems or modify FL problems to solve caching placement. Usually they use either Uncapacitated Facility Location (UFL) problem [17] or rent-or-buy problem [18]. The more general case for these two problems are Connected Facility Location (ConFL) problem [19]. However, UFL does not consider the content dissemination cost in ConFL problem, while rent-or-buy problem does not consider the facility building costs in ConFL.

In this paper, we design our algorithms by leveraging the ConFL problem. Since ConFL is a NP-hard problem, previous work has generated many approximation algorithms. The best deterministic constant approximation ratio is achieved by Jung et al in 2009 [20] with a 6.55-approximation primal-dual algorithm. The best approximation algorithm is a 4.00-approximation randomized algorithm [21], in which the authors argue that through a derandomization process, it could reach a factor of 4.32-approximation ratio. However, the derandomization process is a non-polynomial process, thus limiting the ability to solve ConFL problem. Heuristic [22] and greedy [23] solutions are also proposed. Though such algorithms may not have solid approximation bounds, they may still achieve good performance in practice. We focus on the algorithms with bounded approximation ratios and leverage them in our algorithms.

III. Problem Formulation

In this section, we introduce our system model and discuss how we quantify fairness and contention of the network. Then we provide an integer linear programming (ILP) formulation for the problem and explain its relation to the ConFL problem.

A. System Model

Let graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a connected undirected graph representing the network topology for the multi-hop wireless network. Vertexes ($\mathcal{V}$) represent nodes in the network. Edges ($\mathcal{E}$) represent connection links between nodes. In pervasive edge environments, nodes (e.g., customers in cafes) exhibit low to moderate mobility, and the amount of exchanged data is limited (e.g., a few MBs). Thus we assume that the network topology remains relatively stable before caching decisions are made.

Assuming some data should be shared among all nodes, the goal is to find an optimal way to distribute the data so as to achieve fairness while ensuring low data access latency. There are two phases. First, we divide these data into multiple equal size data chunks $\mathcal{N}$ and proactively disseminate them, letting certain node $i \in \mathcal{V}$ cache certain chunk $n \in \mathcal{N}$. We call this process the dissemination phase. Then, the node who desires data chunk n will acquire the data from a nearby node who has already cached it. We call this process accessing phase.

As the first step towards caching in edge environments, we will not discuss more advanced issues such as caching replacement or caching coherency. We do not assume any specific data fetching or transmission techniques. To simplify data requesting, we also assume that every node wants to acquire all the cached data.

B. Fairness Degree Cost

Data caching fairness is a critical issue in edge computing environments where many nodes are owned by different users. Both storage and battery are important resources, and the user can decide how much resource to contribute to caching. Any fixed selection (e.g., the same group of nodes) for caching will consume excessive resources on chosen nodes, thus their users may stop participation.

The key to achieve fair caching is to cache less data on nodes with less resources. We quantify the caching fairness of a node by defining a *Fairness Degree Cost* based on current node resource consumption conditions. For a node, the higher *Fairness Degree Cost* is, the fewer resources available, and the less likely to cache data on it. Intuitively, the ratio between used and remaining storage for caching can represent the usage

of storage of a node. [1] Those who cache more data and have less storage left will have higher *Fairness Degree Cost*. The *Fairness Degree Cost* for node i is defined as

$$f_i = \frac{S(i)}{S_{\text{tol}}(i) - S(i)} \tag{1}$$

where $S_{\text{tol}}(i)$ is the total caching storage of the node, and $S(i)$ is the storage used. Thus, $S_{\text{tol}}(i) - S(i)$ is the storage still available. Intuitively, it represents a "penalty": the less resources a node has, the more "cost" the network must pay to cache data on it. A "cost" of 0 indicates that the node has not cached anything, and ∞ means the storage of node is full and no further caching is possible. To reduce the cost, nodes with little storage thus high costs should not be chosen. Since all chunks have the same size, we define $S_{\text{tol}}(i)$ as the total number of chunks the node can cache, and $S(i)$ as the number of chunks the node has cached.

Fairness also unifies the solution to cache multiple data items. Previous wireless caching works, e.g., [13], only consider caching one piece of data. Although some of them argue that they can be extended to multiple data items, the exact process is unclear and may involve changes in the network topology, which may change the underlying problem into a different instance. When fairness is considered, the reduced available resources on nodes already caching some data make them less likely to be chosen for caching future data.

C. Contention Cost

Contention is one of the most important factors affecting per hop latency in multi-hop wireless networks. To minimize contention caused packet loss, back-off time and retransmission are adopted, both increasing the data access latency.

To decrease data access latency, we want to minimize the overall contention. The contention delay model that describes the delay due to contention has been considered in [3], [14], [4]. We call it *Contention-induced Delay Cost*, or simply *Contention Cost*. We define *Node Contention Cost* w_k as the *Contention Cost* on a specific node $k \in \mathcal{V}$. It is affected by the number of its contending neighbors, the number of cached data chunks and the amount of transmissions. In this scenario, we define the *Node Contention Cost* w_k as the number of data packet transmissions through node k, both receiving and sending. The accurate mathematical representation of this contention makes the problem very difficult to solve, and almost impossible for distributed algorithms. Thus, inspired by [4], we propose an estimation solution adopting a similar approach. For a node k, all the neighbors will send requests to it and the node will return all the data chunks it receives to direct neighbors. Thus, w_k can be regarded as its degree, which equals the number of data chunks the node will send to its neighbors (i.e., one chunk per neighbor).

The *Path Contention Cost* between two nodes i and j is based on the *Node Contention Cost* alongside the path. We formulate it as

$$c_{ij} = \sum_{k \in \text{PATH}(i,j)} w_k[1 + S(k)] \tag{2}$$

where all the *Node Contention Costs* are summed along the shortest path which the data packet will go through. Note that previously cached data chunks also affect the contention. Each of these data chunks (cached or new) increases the contention by the value of the node degree, since each chunk should be transmitted to all neighbors throughout the process.

The *Contention Cost* defined above focuses on the delay of the network caused by the contention in sending and receiving messages. Yang et al. give a delay estimation of contention in [24]. Such *Contention Cost* is roughly a linear transformation of the *Contention Delay* model for evaluating the delay proposed in [24]. Basically, it considers the processing (DCF Inter-Frame Space in 802.11) delay, back-off delay, transmission delay and collision delay for a hop, represented as

$$d(k, c) = DIFS + m_k\epsilon + w_k T_d + m_k^c T_c$$

where for node k, DIFS is DCF Inter-Frame Space in 802.11, m_k is the number of back-off slots, ϵ is the length of back-off slot, w_k is the number of chunks transmitted in neighboring nodes, T_d is the transmission duration of a data chunk. m_i^c is the number of collisions and T_c is the duration of a collision. Since the back-off slot time and collision duration are much smaller relatively, we can assume $T_d \approx T_c \approx \epsilon$. $m_k = S(k)$ is the number of stored data chunks of nodes. $m_i^c = (w_k - 1)S(k)$ is the maximum number of collisions when all neighbors except one send their stored chunks. [2] Thus, $d(k, c) \approx DIFS + T_d(w_k + S(k) + (w_k - 1)(S(k))) \approx DIFS + T_d(w_k + w_k S(k))$, which means that the one-hop *Contention Delay* is roughly a linear transformation of the one-hop *Contention Cost*. We will use contention cost to represent access latency in the following.

D. Problem Formulation

We now provide an Integer Linear Programming formulation for the problem discussed above. Note that c_{ij} is used for *Contention Cost* for both accessing and dissemination phases where the network topology remains the same.

The basic idea is to add the *Fairness Degree Cost* and *Contention Cost* in a weighted form, and expand each decision variable for different chunks. For simplicity, we consider them of the same weight in the following ILP formulation

$$\min \sum_i \sum_n f_i y_{in} + \sum_i \sum_j \sum_n c_{ij} x_{ijn} + \sum_{e \in \mathcal{E}} \sum_n c_e z_{en} \tag{3}$$

$$\text{s.t.} \quad \sum_i \sum_j x_{ijn} = 1, (\forall n \in \mathcal{N}) \tag{4}$$

$$y_{in} - x_{ijn} \geq 0, (\forall i, j \in \mathcal{V}, \forall n \in \mathcal{N}) \tag{5}$$

$$\sum_{i \in Y_n} x_{ijn} \leq \sum_{e \in \delta(Y_n)} z_{en}, (\forall j \in \mathcal{V}, \forall n \in \mathcal{N}, \forall Y \subseteq \mathcal{E}) \tag{6}$$

$$x_{ijn}, y_{in}, z_{en} \in \{0, 1\}. (\forall i, j \in \mathcal{V}, \forall e \in \mathcal{E}, \forall n \in \mathcal{N}) \tag{7}$$

[1] For simplicity, we only consider storage fairness. A *Fairness Degree Cost* on the battery can be defined similarly and considered together in weighted summation form of the two costs.

[2] Only one node sending causes no collision.

where c_e = c_{ij} if node i and node j are the two end points of edge e. x_{ijn}, y_{in} and z_{en} are assignment variables. x_{ijn} is the accessing variable. If $x_{ijn} = 1$, node j will access data chunk n from node i. y_{in} is caching indicator variable. $y_{in} = 1$ means data chunk n will be stored in node i. z_{en} is the dissemination variable. $z_{en} = 1$ means the data chunk n will disseminate through edge e in dissemination phase.

There are three terms in (3), the objective function: the *Fairness Degree Cost* of the whole network, the *Contention Cost* for the accessing phase and the dissemination phase of the whole network. Here, f_i is defined in (1) and c_{ij} is defined in (2). Constraint (4) ensures that every node j should get a specific data chunk n from exactly one node i. It is clearly not optimal if a node gets the same data chunk multiple times. Constraint (5) ensures that if node j gets a data chunk n from node i, node i must store that chunk. Constraint (6) is the connectivity constraint. Here, Y_n is any subset of nodes, and $\delta(Y_n)$ indicates all the edges that connect to Y_n. This is to ensure that for any subset of chosen nodes to cache data chunk n, they are connected in a Steiner tree [25]. We need these nodes to be connected to disseminate the data chunks along the Steiner tree. $z_{en} = 1$ means edge e is chosen in the Steiner tree so that data chunk n will be disseminated through this connection link.

Our problem is an extended case for the Connected Facility Location (ConFL) problem. It can be transformed as the summation of multiple ConFL problems. f_i can be regarded as the construction cost for a facility in ConFL problem, which in this case represents the cost that the network is willing to pay to select nodes caching a data chunk. c_{ij} and c_e can be regarded as modified distance cost, adding the factor of contention of the network. The y_i, x_{ij} and z_e have similar meanings in each problem. They represent node i as facility or caching nodes in each problem and node j wants to access to it through edge e. As shown in [19], the original ConFL problem is NP hard. The summation of all the different chunks is a polynomial time mapping, which maps our problem to the ConFL problem. Thus, our problem is also NP hard.

It is very difficult to directly solve the problem. Fortunately, for the ConFL problem, there are many existing approximation algorithms, among which the best algorithm has a 4.23 approximation ratio. To take advantage of approximation algorithms for ConFL, we make another transform of our problem as

$$\sum_{n}(\min\sum_{i} f_i y_{in} + \sum_{i}\sum_{j} c_{ij}x_{ijn} + M\sum_{e\in\mathcal{E}} c_e z_{en}) \quad (8)$$

We transform from one optimization goal of getting the minimization from variables with the chunks into the summation of multiple minimization problems for each chunk individually. We can apply the approximation algorithm to problem (8) by using it multiple times for different chunks. Although (8) is different from (3), we will later show that under certain conditions, we can use this iterative solution to (8) to solve (3), and the approximation ratio in the original approximation algorithm is preserved.

IV. Algorithms

A. Approximation Algorithm

In our approximation algorithm, we obtain the input information from the network and leverage an existing approximation algorithm to solve the problem. To show that our design indeed can solve the problem, we implement the algorithm [20], which has an approximation ratio of 6.55. This algorithm approximates the largest possible value of the dual problem. Basically, nodes with sufficient resources lying in the path between multiple nodes and an existing caching node or producer will be picked as caching nodes. The algorithm we propose is described in Algorithm 1.

Algorithm 1 Approximation algorithm

Input: $\mathcal{G} \leftarrow (\mathcal{V}, \mathcal{E}), P_v$
Output: $L(n)$

$L(n) \leftarrow \emptyset$
for all Data chunks n **do**
  $F \leftarrow \emptyset, T \leftarrow \emptyset, C_d \leftarrow \emptyset, C_i \leftarrow \emptyset, A \leftarrow \emptyset, B \leftarrow \emptyset$
  $\alpha_j \leftarrow 0, \beta_{ij} \leftarrow 0, \gamma_{ij} \leftarrow -1, \theta_{Sj} \leftarrow 0.\ \ \forall i, j, S$
  for all nodes i **do**
    Update $f_{in} \leftarrow S(i)/[S_{\text{tol}}(i) - S(i)]$
  end for
  for all nodes i **do**
    for all nodes j **do**
      Get shortest path PATH(i, j)
      Get $c_{ij} \leftarrow \sum_{a\in\text{PATH}(i,j)} w_a[1 + S(a)]$
    end for
  end for
  for all edges e **do**
    Update $c_{en} \leftarrow c_{ij}$
  end for
  while $F \neq \mathcal{V} - P_v$ **do**
    $\alpha_j += U_\alpha.\ \ \forall j \in \mathcal{V}, j \notin F$
    $\beta_{ij} += U_\beta.\ \forall i, j \in \mathcal{V}, j \notin F, \sum_i \beta_{ij} < f_{in}$
    $\gamma_{ij} += U_\gamma.\ \forall j \in \mathcal{V}, j \notin F, T[j] = i, \sum_i \beta_{ij} \geq f_{in}$
    for all nodes i, nodes j **do**
      if $\alpha_j \geq c_{ij}$ **then**
        $T[j] \leftarrow i, F \leftarrow F \cup \{j\}$
        $C_i[j] \leftarrow i (i \in A)$ or $B[i] (i \in B)$
      end if
    end for
    for all nodes i, nodes j, locations l **do**
      if $\gamma_{ij} \geq c_{ij}$ **then**
        if $i \in A$ **then**
          $C_i[j] \leftarrow i, F \leftarrow F \cup \{j\}$
        else if $i \in B$ **then**
          $C_i[j] \leftarrow B[i], F \leftarrow F \cup \{j\}$
        else
          $A \leftarrow A \cup \{i\}, B[i] \leftarrow i$
          $R \leftarrow \emptyset$
          if $\gamma_{ij} \geq c_{iln}$ **or** $\beta_{ij} > 0$ **then**
            $C_d[j] \leftarrow i, F \leftarrow F \cup \{j\}$
            $R \leftarrow R \cup \{j\}$
          end if
          $B[k] \leftarrow i.\ \ \forall k \in T[R]$
          $C_i[k] \leftarrow B[i].\ \ \forall k \in T[i] \notin R$
          $F \leftarrow F \cup \{k\}.\ \ \forall k \in T[i] \notin R$
        end if
      end if
    end for
  end while
  Construct Steiner tree between $i \in A$
  $L(n) \leftarrow A$
end for

For each chunk, we conduct one iteration (line 2). After some initialization steps, we update the *Fairness Degree Cost* for all nodes in the network (lines 5-7) and then estimate the *Contention Cost* of all links based on Equation (2) (lines 8-16). Lines 17-46 are phase 1 of the approximation algorithm. It iterates until every node finds at which place they can obtain the data chunk, (aka the state FROZEN in the original approximation algorithm [20]). First, the algorithm increases the *price* it is willing to pay for establishing a connection to a caching node (lines 18-20). Note that here the increasing step units U_α, U_β and U_γ can be different, since the increasing steps of the three parameters represent different meanings of cost in the dual problem.

As mentioned earlier, choosing the parameter wisely can make the solution better. If this *price* is larger than the cost of accessing one existing caching node, it can establish a connection to that node (lines 21-26), which is the first and second condition in phase 1 of the original approximation algorithm in [20]. If not, it goes to the third condition. Lines 29-30 and 31-32 are conditions in 3(a) and 3(b) respectively in [20]. Lines 33-42 deal with condition 3(c) in [20] where all direct connections, inactive nodes set and ADMIN set are created. This ends phase 1. Phase 2 is to connect the locations of corresponding nodes together. A lot of algorithms can be used to address the problem, from which we choose [25]. We will not discuss in detail here. Finally, for chunk n, the ADMIN set A are the nodes that will cache the chunk. We save this into $L(n)$ and start the next round until we find all the caching nodes for all chunks.

The best approximation ratio so far is 4.23 [21]. However, this algorithm can only obtain a deterministic approximation ratio with a derandomization process which needs to solve an exponential size linear programming relaxation. Thus this algorithm is practically inefficient and very hard to implement. The 6.55-approximation algorithm has the lowest deterministic approximation ratio of polynomial time to ConFL problem for now.

B. Approximation Algorithm Analysis

We now discuss the time complexity of the algorithm. We assume there are N nodes, Q chunks, and number of iterations of line 17 is C. Then the complexity of the algorithm in a grid network is $O(N^3)$. Apparently, the bottleneck of the algorithm is line 27. If we simplify the locations as the location of nodes in line 27, which is practical in real implementation, there will be $O(N^3)$ complexity by the number of loops. Note that lines 8-13 in Algorithm 1 requires the shortest path between two nodes, which costs at most $O(N^3)$ using Floyd-Warshall algorithm; if the network topology is simple, such as a grid network, it might drop the complexity to $O(N^2)$. Meanwhile, Steiner tree problem has polynomial time approximation algorithms. The work like [25] can achieve 1.55 approximation ratio at the time complexity of $O(N^3)$. The number of iterations is related to the chosen unit step U_α. If the unit step is large, it might quickly finish but may select fewer nodes and increasing the *Contention Cost* of accessing phase; or if the unit is small, it might take a long time. However, it will not exceed $\max\{c_{ij}\}/U_\alpha$. If we increase α_j to be larger than the maximum of all c_{ij}, the data chunks can be placed at anywhere. In this case, the iteration will end. So the iteration time is no more than $C = \max\{c_{ij}\}/U$. After all, in the worst case for each chunk the complexity is $O(CN^3)$. For the algorithm in total, its complexity is no more than $O(QCN^3)$. For $\max\{c_{ij}\}$, it depends on the data chunks of the network as well as the connection complexity. Some networks may have less complexity. For example, for a grid network, the number of maximum hops of the network is fixed. $O(C) = O(Q)$ since the size of $\{c_{ij}\}$ is fixed. We argue that in practical edge computing environments, the number of data chunks is a constant not relevant to the size of network. Thus, the overall complexity in a grid network is $O(N^3)$. The original approximation algorithm can be executed in polynomial time, which can still offer 6.55 approximation ratio to the ConFL problem, and our own problem as well.

Now we show that this iterative solution under proper increment unit settings will achieve the same approximation ratio as (3).

Theorem 1. *The above iterative algorithm can achieve 6.55 approximation ratio to the optimization problem (8).*

Proof. Equation (8) is the summation of a series of minimization problems. To address the problem, first, we must obtain the dual problem of each chunk represented in one of minimization problems in (8). For a linear programming problem, the maximum value dual problem is the same as the original problem. In this problem, it is easier to solve the dual problem than the original one. The dual problem formulation can be induced from [20]. For each minimization problem, we have

$$\max \quad \sum_j \alpha_{jn} - \sum_j \beta_{vjn} \tag{9}$$

$$\text{s.t.} \quad \alpha_{jn} \le c_{ijn} + \beta_{ijn} + \sum_{Y_n} \theta_{Y_n,jn}, (\forall i \ne v, j) \tag{10}$$

$$\alpha_{jn} \le c_{vjn} + \beta_{vjn}, (\forall j) \tag{11}$$

$$\sum_j \beta_{ijn} \le f_{ij}, (\forall i) \tag{12}$$

$$\sum_j \sum_{e \in \delta(Y_n)} \theta_{Y_n,jn} < c_{en}, (\forall e) \tag{13}$$

$$\alpha_{jn}, \beta_{ijn}, \theta_{Y_n,jn} \ge 0. \tag{14}$$

In (9), α, β and θ are three dual variables of x_n, y_n and z_n in (3). We need to find the values of dual variables

TABLE I
NOTATIONS USED IN OUT APPROXIMATION ALGORITHM

$L(n)$	The caching nodes set for data chunk n	P_v	The producer of this data chunk
F	FROZEN nodes set	T	TIGHT node pair set
A	ADMIN nodes set	B	INACTIVE node pairs set
C_d	Direct connection pairs set	C_i	Indirect connection pairs set
U_x	The unit increase value of var x	R	Regional village

to get the optimal value of the original problem. We let $Dual_n = \max(\sum_j \alpha_{jn} - \sum_j \beta_{vjn})$ be the approximate optimal value (not the real optimal value) of (9). Opt_n is the optimal value for each chunk n of (8). Similarly, we let $Dual(n) = n(\sum_j \alpha_{jn} - \sum_j \beta_{vjn})$ be the approximate optimal value for the dual problem of (3) and $Opt(n)$ be the optimum value of (3). The approximation algorithm is based on improving the dual value to approximate the objective value. We can conclude that if there was only one chunk in the network, $Dual(1) = Dual_1$. We assume that the cost is always lower if we use caching rather than directly getting all data from the producer. Thus, according to (9)-(11), $Dual_n < \sum_j c_{vjn}$. The result is the same to $Dual(n) < \sum_n \sum_j c_{vjn}$. We then define ϵ_{1n} and ϵ_{2n}, where $\forall n, \epsilon_{1n}, \epsilon_{2n} > 0$. Let $Dual(n) = \sum_j c_{vjn} - \epsilon_{1n}$ and $Dual_n = n\sum_j c_{vjn} - n\epsilon_{2n}$. For the first chunk, $\epsilon_{1,1} = \epsilon_{2,1}$. We assume that adding a cache should lower the total cost of facility construction. We want to choose U_α and U_β properly, so that after the first chunk, $\epsilon_{1n} \geq \epsilon_{2n}$. When all the chunks are cached into the network, $\sum_n Dual_n = \sum_n \sum_j c_{vjn} - \sum_n \epsilon_{1n} = n\sum_j c_{vjn} - \sum n\epsilon_{1n} \leq n\sum_j c_{vjn} - \sum n\epsilon_{2n} = Dual(n)$. Since there is a 6.55-approximation algorithm for the primal-dual problem, we set k as the approximation ratio. Then $Dual_n = k \times Opt_n$ and $Dual(n) = 6.55 \times Opt(n)$. Thus, we have $\sum_n Dual_n = nDual_n = nk \times Opt_n \leq Dual(n) = 6.55 \times Opt(n)$. Since $nOpt_n = Opt(n)$, We get $k \leq 6.55$. We conclude that it will achieve the same approximation ratio by using the algorithm with fixed approximation ratio multiple times. □

C. Distributed Algorithm

Sometimes nodes may not have the connection topology of the whole network. To address this issue, we make some extension from the approximation algorithm and propose a distributed algorithm. The basic idea is 1) keeping the variables associated with a node and let the node itself maintain them, 2) and sending control messages in k-hop range to get the contention and inform the state change of a node.

In the distributed algorithm, the initial states of nodes are the same as the approximation algorithm. Let's assume there is a new data chunk waiting to be cached. First there will be an NPI (New Packet Info) packet informing the network of this new data chunk waiting to be cached. Then, nodes will exchange information about contention. The exchange will be limited in k-hop range to avoid flooding. Then node i will increase a bid denoted as α_j. If the bid can cover the estimated contention cost between two nodes i and j, node i will send a TIGHT request to node j, meaning "Can I get data from you?", and start a bid for relay cost γ_j and resource cost β_j. If the bid for relay cost covers the contention cost, the node will send a SPAN request, "Can you fetch data for me from other nodes?" A node that has received enough SPAN requests will make itself an ADMIN node and send response back to the nodes who sent these SPAN requests and whose bit for recourse cost was large enough. Those served nodes will stop biding. Algorithm 2 shows the detailed process for a particular data chunk. It is basically event driven. Every node that receives a request or response message will call one of the receive processes listed below. Whenever a

TABLE II
MESSAGES IN THE DISTRIBUTED ALGORITHM

Packets	Content	Range
NPI	Inform there is a new data chunk to be cached	Broadcast
CC	Contention collection request	Local
TIGHT	Inform a node for tight (bid larger than contention cost)	Local
SPAN	Inform a node for span (bid larger than relay cost)	Local
FREEZE	Response message to freeze a certain node	Local
NADMIN	Inform self admin for those nodes who tights with this node	Local
BADMIN	Inform self admin for those nodes who has adequate resources	Broadcast

Algorithm 2 Distributed algorithm

Receive NEW PACKET INFO(NPI)
  Send Contention Collection (CC(i)) Request in k hops
  while True **do**
    Gradually increase α_j Until α_j larger than one of the Con_j received.
    Send TIGHT(i) Request to j.
    Gradually increase β_j, γ_j
    if γ_j larger than Con_j **then**
      Send SPAN(i) Request to j.
    end if
  end while
end Receive
Receive CC(i)
  $Con+ = 1$
  Send Con back
end Receive
Receive TIGHT/SPAN(i)
  $T = T \cup \{i\}$
  if Node is INACTIVE **then**
    Send FREEZE(a) to $\forall j \in T$
  else if Node is ADMIN **then**
    Send FREEZE(i) to $\forall j \in T$
  else if Node is ACTIVE and Message is SPAN **then**
    $c+ = 1$
  end if
  if Node is ACTIVE and $c >= M$ **then**
    Make myself ADMIN
    Send NADMIN(i) to $\forall j \in T$
    Broadcast BADMIN(i)
    Proactively request Data chunk from Producer
  end if
end Receive
Receive FREEZE(i)
  $a = i$
  Stop increasing α_j, β_j, γ_j
end Receive
Receive NADMIN(i)
  $a = i$
  Stop increasing α_j, β_j, γ_j
  Send FREEZE(a) to $\forall j \in T$
end Receive
Receive BADMIN(i)
  if Node is ACTIVE and $\beta_j > Con_j$ **then**
    $a = i$
    Stop increasing α_j, β_j, γ_j
    Send FREEZE(a) to $\forall j \in T$
  end if
end Receive

node determines itself to be an ADMIN node, it will proactively request the data chunk from the producers. Note that all types of messages, except NPI messages and BADMIN messages, are only limited in k-hop range.

D. Distributed Algorithm Analysis

We now discuss the number of messages of the distributed algorithm. All messages that will be transmitted in the network are listed in TABLE II. We assume there are N nodes and Q chunks. The number of messages is of $O(QN + N^2)$. The number of NPI messages equal to the number of chunks. Every node will send out CC packets for each chunk, so the total number is $O(QN)$ which is the number of chunks times the number of nodes. The number of TIGHT and SPAN messages are both $O(N^2)$, whose worst case is sending every other node a TIGHT message. This is unlikely to happen since there will always be nodes that have data copies and send responses. FREEZE message is $O(N)$, since every node will send it at most once when it finds a node who can provide cache. NADMIN and BADMIN are all messages sent from a node that decided to become a caching node. Each caching node will only send these two messages once. Thus, the worst case is the number of caching node, at most $O(N)$. We can conclude that the total number of messages is $O(QN + N^2)$, where TIGHT, SPAN and CC are the most dominating messages in the network.

V. Evaluation

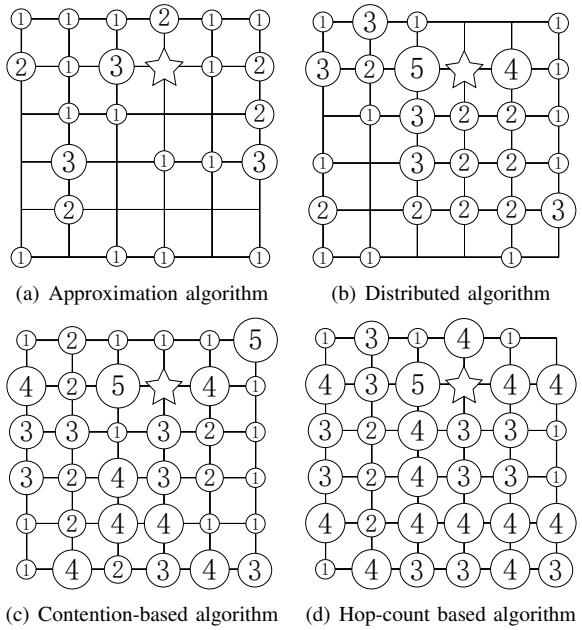
We evaluate the performance of the proposed approximation algorithm and distributed algorithm, and compared them with the most relevant work on caching in wireless networks.

A. Simulation Scenarios

We test our algorithms on grid network topologies and random network topologies. To compare the performance of our algorithms with the optimal solution, we first use PuLP Linear Programming modeler [26] to get the optimal solution by brute-force. We implement our two algorithms in Python. Meanwhile, to compare with existing work, we also implement the algorithms of Nuggehalli et al. [13] and Sung et al. [4]. The delay cost of [13] is based on the hop count between nodes, and [4] considers the contention of the network as delay cost, which is one major concern in edge computing environments. In the following, we denote these two schemes as *Hop Count-based* algorithm (Hopc) and *Contention-based* algorithm (Cont) respectively. For simplicity, we set the λ in both their algorithms to 1.

We conduct our simulations using Python 2.7 on a computer equipped with an Intel Core i7-5820K and 16 GB RAM. In simulations, we assume that all data chunks are of the same size. All the nodes will have the same caching storage capacity, and in simulations we set it to 5 chunks. The algorithm may also be applied on hundreds of chunks and thousands of nodes, but that are not representative for mobile network edge computing environments. We also assume that the caching storages are empty for all the nodes, including the data provider node, at the beginning. We assume that the producer node will not store data on its caching storage, and therefore, the calculation of costs will not include the producer node. Unless specified, node 9 is the data producer. Finally, as mentioned earlier, we consider all nodes requesting all data chunks. A node will find the nearest copy of a chunk and go through the shortest hop path. For the distributed algorithm, we limit all local messages exchanged within 2 hops.

In grid network topologies, all nodes can connect to other four neighbors except those on the network boundary. In random networks, we assume that the nodes within a certain range are connected, and make sure the random network is a connected graph.



(a) Approximation algorithm (b) Distributed algorithm

(c) Contention-based algorithm (d) Hop-count based algorithm

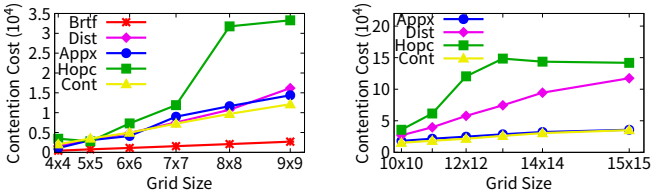
Fig. 1. The distribution of data chunks in a 6×6 grid network. The area size of and number in each circle show the **difference in the number of stored chunks** from the optimal solution on respective node. The star indicates the producer.

B. Performance Evaluation

We first compare the *Contention Cost* among the approximation algorithm (denoted in figures as Appx), distributed algorithm (Dist), brute-force algorithm (Brtf) and other algorithms mentioned earlier. In the network, we have 5 distinct data chunks needed by all nodes, each with a maximum storage capacity of 5. We illustrate the caching results of these four algorithms in Fig. 1 in a grid network of 6×6. The area of and number in each circle are the *difference* in the number of chunks stored on respective node, for the four algorithms compared to the brute-force optimal algorithm. Ideally, they should all be 0.

Other two algorithms choose nodes based on the cost (hop-count or contention) obtained from the current network topology, without considering already cached data. They will always choose the same group of nodes for each chunk. Thus, in these two algorithms, all the 5 chunks are cached at the same group of nodes. We can see that in our algorithms, more nodes are selected and data chunks are much more evenly distributed.

The difference between the results of our algorithms and the optimal solution is relatively small.



(a) Small Scale (b) Large Scale

Fig. 2. Contention cost of 5 different algorithms. Grid size from 4×4 to 15×15

As mentioned earlier, *Contention Cost* can be translated into data access latency. Fig. 2 shows the result on *Contention Cost*, the summation of the cost from *Accessing* and *Dissemination* phases, for small and large networks. We first test them in small networks and compare with optimal results from the brute-force algorithm. The proposed approximation algorithm preserves the approximation ratio, which at maximum is 5.6. Our algorithms achieve similar *Contention Costs* to the contention based algorithm, since the contention is the cost calculation of both algorithms. In most cases, the proposed approximation algorithm and the contention based algorithm are about the same on total *Contention Cost*, where on average the approximation algorithm is no more than 9% worse than the Contention-based algorithm on total *Contention Cost*. However, our algorithms are much better than Hop Count-based algorithm, with average 52.1% lower on total *Contention Cost*. We also obtain results for larger networks (from 100 to 255 nodes) where brute-force fails to obtain results within meaningful time (e.g., days). The approximation algorithm is still much better (62%) than the Hop Count-based algorithm and slightly worse (8%) than the Contention-based algorithm. It shows that our algorithms can also achieve a comparable result in larger scale.

As mentioned earlier, in the distributed algorithm, we limit the request message within certain hops to avoid excessive overhead. To study how this hop limitation affects the performance, we test the *Contention Cost* under different hop limitations (Fig. 3). When it is limited in 1 hop, the information exchange range is too small. Thus, nodes get too little information about the surrounding network. Finally, very few caching nodes are selected. This will cause high *Contention Cost* in *Accessing* phase because traffic is concentrated on these nodes. When the limitation is 2 or more hops, the difference between the total *Contention Cost* on different hop limitations is relatively small, since the nodes know more about the network and more nodes are selected as caching or relay nodes. To balance between message overhead and caching performance, we choose 2-hop limitations for the distributed algorithm.

We also conduct simulation on random networks to address more general cases. We tested the networks containing 20 to 180 nodes, using both our algorithms and the other two algorithms. Fig. 4 shows the result averaged over 5 runs. The approximation algorithm and distributed algorithm achieve 4.54% and lower delay costs than the Contention-based algorithm and are much better (62.0%) than the Hop Count-based algorithm. The results show that our algorithms can achieve comparable performance in random networks, especially under large network size.

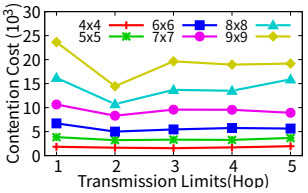


Fig. 3. Effect of message hop limitation on contention cost of distributed algorithm tested on different sizes of networks.

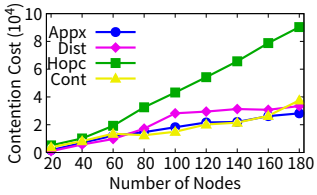


Fig. 4. Contention cost of four algorithms on random networks in varies scale. Number of nodes from 20 to 180.

The running time of algorithms is also a crucial factor. For the Hop Count-based algorithm, in grid networks, the complexity is $O(|\mathcal{V}||\mathcal{E}|^3)$, where $|\mathcal{V}|$ and $|\mathcal{E}|$ are the numbers of nodes and edges in the network, respectively. For the Contention-based algorithm, the complexity is $O(|\mathcal{V}|^3)$ in grid networks. In our approximation algorithm, we also achieve $O(|\mathcal{V}|^3)$ when the number of chunks is constant. In Fig. 5 we can see that to compute the caching locations of one data chunk in grid networks, our algorithm is much faster than other two algorithms, with average 21.6% and 85.1% less in running time. We do not include the running time of distributed algorithm, because it is based on message transmissions and totally different from other three algorithms.

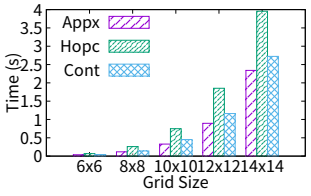


Fig. 5. Running time for three different times for grid networks from 6×6 to 14×14.

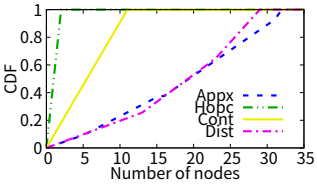


Fig. 6. Cumulative chunk distribution on nodes of a grid network of 6×6.

C. Fairness Evaluation

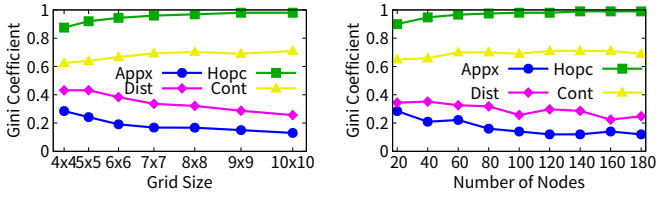
We evaluate fairness in different aspects between our algorithms and other algorithms. First, we evaluate how the data chunks in the network are distributed on nodes. Fig. 6 shows the number of nodes needed to store a certain ratio of all data. We can see that 50% of the total data chunks are distributed in one node in the hop-count based algorithm, 5 nodes in the Contention-based algorithm and about 20 nodes in both approximation and distributed algorithms. By including more nodes for caching, we avoid overload and achieve more robust data access. We define p-percentile fairness as the fraction of nodes needed to cache p% of the total data. Ideally, when all nodes have the same caching load, p-percentile fairness is strictly $p\%$. The smaller it is, the more uneven the load, thus less fair. We find that 75-percentile fairness in this 6×6 grid network is 71.4%, 68.6%, 4.28%, 22.8% for the approximation, distributed, Hop Count-based and Contention-based algorithms

respectively. The higher the number, the fairer it is. The approximation algorithm and the distributed algorithm achieve on average 3 times higher than Contention-based algorithm, as well as 16 times higher than Hop Count-based algorithm. We can see that the proposed approximation algorithm and distributed algorithm can improve the fairness for caching.

We also evaluate the fairness by Gini coefficient, which is widely used to depict incoming disparity [27]. The definition is as follows:

$$G = \frac{\sum_i \sum_j (t_i - t_j)}{2 \sum_i \sum_j t_j}$$

where t_i and t_j are the numbers of chunks stored in nodes i and j respectively. Note that in the denominator, t_j and t_i are commutable. It measures the inequality of cached chunks among different nodes. The higher the p-percentile fairness, the lower the Gini coefficients, and the fairer the network is. We calculate the Gini coefficients of all five algorithms in both grid and random networks. Fig. 7 shows the Gini coefficients of three algorithms in the grid network and random network, respectively. Our algorithms have Gini coefficient less than 40%, thus, the chunk distribution is fairer among all nodes. In addition, when the network size grows, the Gini coefficient of our algorithms drops while others remain roughly the same or even increasing. This shows that our algorithms can leverage larger network size and spread chunks more evenly, thus decreasing the Gini coefficient.



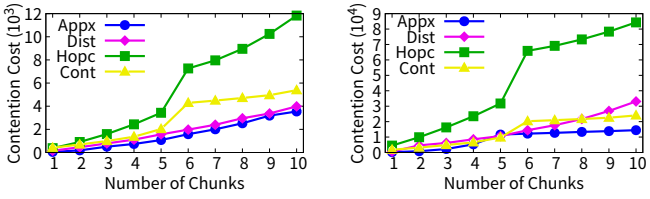
(a) Grid network (b) Random network

Fig. 7. Gini coefficient of the grid network (a) and an arbitrary network (b).

When many chunks exist, we want to avoid caching data on those nodes that have already cached a lot. Selecting only a few nodes will exhaust their caching storage. Other two algorithms select nodes based only on the network topology. They are not designed for multiple data items. Thus, in the simulation, all chunks are put on a few selected nodes until they are full. After that, no more data can be cached. Although one of the authors argued that it is easy to modify the algorithm to accommodate multiple data items, exact details are not specified. We extend both algorithms for multiple data items based on our understanding. We then compare ours with the extended versions of these algorithms for a fair comparison.

We modify these algorithms as follows. If a set of nodes is chosen, we will put all data chunks in these nodes until none of them has vacancy for caching. Then we construct a new subgraph consisting of other nodes, i.e., nodes not in the first set, which are connected to form the new subgraph. We perform the same operations on these nodes: we pick a set of nodes as caching nodes using the same algorithm. These nodes are selected continuously until their storage is exhausted. This process is repeated, until all chunks are cached, or if a subgraph becomes disconnected, we will perform the operations on the largest connected component. For each round, an accessing strategy is produced for a node to get one chunk from a certain caching node. After all the dissemination is done, we calculated the contention by putting all the chunks to the original connected graph based on which nodes access which chunks in all rounds to calculate the *Contention Cost*.

We tested the number of distinct chunks in the network and the total contention costs on the grid network size of 4×4 and 8×8. We gradually increase the number of distinct data chunks from 1 to 10. Each distinct chunk may have multiple copies stored on different nodes. The result in Fig. 8 shows that both of our algorithms can handle more distinct chunks. With the increasing of chunks, the total costs grow slower than other two algorithms, for about 25% less than Hop Count-based and 4% less than Contention-based algorithm. Other two algorithms have a large increase when the number of data chunks goes from 5 to 6. This is because that they start to put the data on the next set of nodes. These operations will increase the cost for both the contention of old ones and new ones. Meanwhile, our algorithms offer better performance if the network has multiple data items.



(a) 4×4 grid (b) 8×8 grid

Fig. 8. Accumulate contention cost on grid networks of 4×4 (a) and 8×8 (b).

Dividing large data items into multiple chunks is easy to maintain and can be cached separately to enhance robustness. However, all chunks belonging to one data item must be obtained for completeness. The Contention Cost of these chunks should be roughly even, such that they are obtained at about the same time. Otherwise one chunk with long latency can delay the completion of the whole data item. We tested such per chunk fairness by putting 10 distinct chunks into the network and calculating the total *Contention Cost* of each chunk, which may have multiple copies stored on different nodes. We tested on grid networks of the size of 4×4 and 6×6. Fig. 9 shows the result of all the four algorithms. We can see that other two algorithms always choose the same nodes for the first five chunks, and the same nodes for the next five chunks. The *Contention Cost* is evener in our algorithms and lower than other two algorithms for most chunks. Evener *Contention Cost* means shorter and more consistent latency for data access, thus better user experience.

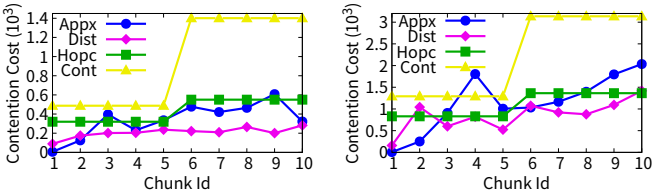


Fig. 9. Accessing cost of each chunk in grid networks of size 4×4 (a) and 6×6 (b).

VI. Conclusion and Discussion

In this paper, we propose two caching algorithms to achieve fair workload among selected caching nodes for data sharing in pervasive edge environments. We consider fairness in caching multiple data items while keeping contention cost low for data access. We propose an approximation algorithm and a distributed algorithm. Comparison with two existing algorithms on wireless network caching shows that our algorithms can achieve comparable or even lower latency while greatly improving fairness, thus data access robustness and performance.

We would like to stress that the accurate representation of contention cost or actual delay is very difficult. There are many factors that can affect them. The accurate formulation of contention cost is not the primary goal of our work in this paper. Thus, we adopt the contention-induced delay, a mature, well-studied model used in recent works to represent the contention cost.

Fair caching naturally produces solutions to multiple chunks because nodes caching more chunks will be less likely selected. Currently our algorithms only consider a constant number of data chunks. We use iterations to calculate the caching placement for each chunk. It is easy and efficient to apply when the number of chunks is small. However, when the number of chunks increases, it might become inefficient. Over long time periods, some chunks may become out-dated, necessitating cache replacement. We plan to further address these two issues and develop online distributed solutions to the problem in our future work, making it more applicable for the edge computing environment.

Acknowledgments

This work is supported in part by US National Science Foundation under grant number CSR-1513719.

References

[1] D. Rossi and G. Rossini, "Caching performance of content centric networks under multi-path routing (and more)," *Relatório técnico, Telecom ParisTech*, 2011.

[2] P. Nuggehalli, V. Srinivasan, and C.-F. Chiasserini, "Energy-efficient caching strategies in ad hoc wireless networks," in *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*. ACM, 2003, pp. 25–34.

[3] X. Fan, J. Cao, and W. Wu, "Contention-aware data caching in wireless multi-hop ad hoc networks," *Journal of Parallel and Distributed Computing*, vol. 71, no. 4, pp. 603–614, 2011.

[4] J. Sung, M. Kim, K. Lim, and J.-K. K. Rhee, "Efficient cache placement strategy in two-tier wireless content delivery network," *IEEE Transactions on Multimedia*, vol. 18, no. 6, pp. 1163–1174, 2016.

[5] L. Yin and G. Cao, "Supporting cooperative caching in ad hoc networks," *IEEE transactions on mobile computing*, vol. 5, no. 1, pp. 77–89, 2006.

[6] T. Hara, "Effective replica allocation in ad hoc networks for improving data accessibility," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3. IEEE, 2001, pp. 1568–1576.

[7] M. Fiore, F. Mininni, C. Casetti, and C.-F. Chiasserini, "To cache or not to cache?" in *INFOCOM 2009, IEEE*. IEEE, 2009, pp. 235–243.

[8] K. Cho, M. Lee, K. Park, T. T. Kwon, Y. Choi, and S. Pack, "Wave: Popularity-based and collaborative in-network caching for content-oriented networks," in *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*. IEEE, 2012, pp. 316–321.

[9] J. Li, H. Wu, B. Liu, J. Lu, Y. Wang, X. Wang, Y. Zhang, and L. Dong, "Popularity-driven coordinated caching in named data networking," in *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems*. ACM, 2012, pp. 15–26.

[10] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, D. Massey, C. Papadopoulos *et al.*, "Named data networking (ndn) project," *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, 2010.

[11] C. Bernardini, T. Silverston, and O. Festor, "Mpc: Popularity-based caching strategy for content centric networks," in *2013 IEEE International Conference on Communications (ICC)*. IEEE, 2013, pp. 3619–3623.

[12] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 1–12.

[13] P. Nuggehalli, V. Srinivasan, C.-F. Chiasserini, and R. R. Rao, "Efficient cache placement in multi-hop wireless networks," *IEEE/ACM Transactions on Networking*, vol. 14, no. 5, pp. 1045–1055, 2006.

[14] J. Sung, M. Kim, K. Lim, and J.-K. K. Rhee, "Efficient cache placement strategy for wireless content delivery networks," in *2013 International Conference on ICT Convergence (ICTC)*, 2013.

[15] H. Farahat and H. Hassanein, "Optimal caching for producer mobility support in named data networks," in *Communications (ICC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[16] N. Kumar and J.-H. Lee, "Peer-to-peer cooperative caching for data dissemination in urban vehicular communications," *IEEE Systems Journal*, vol. 8, no. 4, pp. 1136–1144, 2014.

[17] G. Cornuéjols, G. L. Nemhauser, and L. A. Wolsey, "The uncapacitated facility location problem," DTIC Document, Tech. Rep., 1983.

[18] A. Gupta, A. Kumar, and T. Roughgarden, "Simpler and better approximation algorithms for network design," in *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*. ACM, 2003, pp. 365–372.

[19] C. Swamy and A. Kumar, "Primal-dual algorithms for connected facility location problems," in *International Workshop on Approximation Algorithms for Combinatorial Optimization*. Springer, 2002, pp. 256–270.

[20] H. Jung, M. K. Hasan, and K.-Y. Chwa, "A 6.55 factor primal-dual approximation algorithm for the connected facility location problem," *Journal of combinatorial optimization*, vol. 18, no. 3, pp. 258–271, 2009.

[21] F. Eisenbrand, F. Grandoni, T. Rothvoß, and G. Schäfer, "Connected facility location via random facility sampling and core detouring," *Journal of Computer and System Sciences*, vol. 76, no. 8, pp. 709–726, 2010.

[22] A. Tomazic and I. Ljubic, "A grasp algorithm for the connected facility location problem," in *Applications and the Internet, 2008. SAINT 2008. International Symposium on*. IEEE, 2008, pp. 257–260.

[23] S. Guha and S. Khuller, "Greedy strikes back: Improved facility location algorithms," *Journal of algorithms*, vol. 31, no. 1, pp. 228–248, 1999.

[24] Y. Yang and R. Kravets, "Achieving delay guarantees in ad hoc networks by adapting ieee 802.11 contention windows," in *Proceedings of IEEE INFOCOM*, vol. 2006, pp. 1–8.

[25] G. Robins and A. Zelikovsky, "Improved steiner tree approximation in graphs." in *SODA*. Citeseer, 2000, pp. 770–779.

[26] S. Mitchell, M. OSullivan, and I. Dunning, "Pulp: a linear programming toolkit for python," *The University of Auckland, Auckland, New Zealand, http://www. optimization-online. org/DB_FILE/2011/09/3178. pdf*, 2011.

[27] D. Wei, K. Zhu, and X. Wang, "Fairness-aware cooperative caching scheme for mobile social networks," in *2014 IEEE international conference on communications (ICC)*. IEEE, 2014, pp. 2484–2489.