# MCPC: Improving In-Network Caching with Network Partitions

Dongbiao He*, Jinlei Jiang*‡, Guangwen Yang*, Cedric Westphal†
*Beijing National Research Center for Information Science and Technology
Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
†Department of Computer Engineering, University of California, Santa Cruz, Santa Cruz, CA 95064, USA
‡Corresponding author: jjlei@tsinghua.edu.cn
hdb13@mails.tsinghua.edu.cn, ygw@tsinghua.edu.cn, cedric@soe.ucsc.edu

***Abstract*—In-network caching is considered to be an important solution to efficiently using network resources to achieve a high overall content delivery performance in both information-centric networks (ICNs) and 5G wireless networks. Content placement plays a key role in achieving this goal. Unfortunately, most content placement strategies today rely on opportunistic caching due to the problem complexity. We analyze the content placement problem in detail and present MCPC, a new content placement strategy for architectures that support in-network caching. Unlike existing content placement approaches that try to increase cache hit ratio, MCPC leverages the information recorded in each network node to reduce content access latency. MCPC proposes two new mechanisms: 1) a content load allocation estimation method based on local requests aggregation information; and 2) a content placement algorithm that avoids long-distance signaling messages by partitioning the network into smaller domains. We evaluate MCPC on a variety of network topologies, cache sizes and content popularity distributions. The experimental results show that MPCP can reduce by up to 56% the content access latency while providing comparable cache hit ratios as traditional benchmarks.**

***Index Terms*—In-Network Caching, Network Partition, Access Latency, Content Placement, Requests Aggregation**

## I. Introduction

In-network caching tries to reduce data access latency and increase network throughput by putting data near users. In the past, this is mostly achieved by CDN (Content Delivery Network) overlay at the application layer. In order to accommodate the exponential growth of the Internet traffic [9], the concept of information-centric networks (ICN) has been put forward recently. It provides support for data caching directly at the network layer. Nowadays, there are many ICN implementations [2], [6], [30], [34] available. Moreover, some caches are being installed in a number of small-cell base stations for offloading macro-cell base station traffic as well as improving the services capability of base stations in 5G network [10], [33].

In-network caching fundamentally differs from previous caching methods. First, the cache can appear everywhere in the network so that network operators have to consider where to place the caches and which content should be cached. These issues are usually termed cache and content placement problem. Secondly, the topology of the cache network varies from hierarchical trees to arbitrary graphs [36], making contents placement even harder. Finally, content is accessed without any location specification, making the caching policies designed for the fixed overlay and servers [13] unsuitable. For the sake of simplicity, most ICNs architectures use caching everything everywhere as the default caching policy, and LRU (Least Recently Used) as the default cache replacement algorithm. Such an approach has a low cache hit ratio and often results in redundant data transmission over the network [24].

In order to improve in-network caching, namely the cache hit ratio, two categories of caching schemes have been suggested, that is, on-path caching and off-path caching. On-path caching [15], [17], [19], [21], [23] puts caches just along the delivery path. It is closely related with packet forwarding and routing [35], with the advantages of ease of implementation and supporting arbitrary topologies. The LCE (Leave Copies Everywhere) policy [15]) is a most typical on-path caching scheme. Off-path caching [3], [20], [25], [26], a.k.a. content replication or content storing, replicates content within a network regardless of the forwarding path. Off-path caching usually adopts a centralized architecture, and the content placement procedure involves a great amount of globally-collected network information. Hash routing [26] is one typical off-path caching example, which fulfills contents placement according to the hash value of content identifiers.

In this paper, we present MCPC (Multiple Clients Partition Caching), a novel approach to improve in-network caching. The key question MCPC tries to answer is that, *given the cache space, how much it contributes to reducing the access latency of the placed content item?* To solve this problem, we need to know the parameters (e.g., the frequency of requests and the delay cost) first so as to account for the access latency. Then, efficient methods can be devised to place contents in the cache so as to minimize the whole network latency.

MCPC is mainly based upon two key observations. First, there exist simple and practical greedy algorithms to place contents in the cache in the case that a network has only a *single client*. Second, the routing information of each node has the capability to estimate the loads generated by the corresponding clients. Accordingly, MCPC consists of two steps: it first estimates the content load allocation according to the count of requests and demands generated by each client, and then partitions the nodes according to the content

demands so that each partition can apply the *single-client* cache placement algorithm.

Based on the architecture of ICN, we demonstrate how MCPC can be implemented. We show how to use the information of client requests to partition the network, as it directly reflects the requirements of a specific subset of clients in each node during a given time interval. We show how to aggregate the demands for each content in each partition and how to estimate the caching cost of different nodes. We present a greedy method to place cached items at each partition according to the predicted load. The caching mechanism presented here takes content request popularity and the clients distribution into account and can be applied to highly dynamic large-scale networks, for it relies mostly on local information and local coordination.

Our contributions in this paper are as follows:

- We present a mathematic model of the content placement problem, and point out that the problem can be solved via a greedy algorithm in the single-client case — there is only one client in the network.
- We propose a partitioning method that reduces the complex multi-client case into a set of single-client networks by assigning each caching node to a unique cluster of clients based upon cache capacity, relative delay cost and content request demand.
- We devise an efficient and practical in-network caching policy called MCPC that takes advantage of the distributed information at each nodes as well as the multiple clients partition method above.
- A thorough evaluation has been conducted to demonstrate the significant performance gains of our caching policy against a wide range of benchmarks.

In the end, we would like to point out that though our evaluation is geared towards ICN, the theoretical underpinnings would also apply in the content management layer of a CDN overlay and 5G wireless edge caching networks.

The rest of the paper is organized as follows. Section II illustrates our network model, and offers a solution to the case when there is only one client in the network. Section III displays the details of the MCPC design with a comprehensive analysis and a practical scheme based on routing information. After that, we show the results of the evaluation for MCPC in section IV. The related work is discussed in section V and we offer some concluding remarks in section VI.

## II. Problem Formulation and Analysis

### A. A Mathematic Model of the Problem

Most of the contents placement literature, for example, FairCache [31], CL4M [7], and Hash routing [26], focuses on maximizing cache utilization. However, improving the utilization of caching nodes with a higher hit ratio is not the final goal of the network. In this paper, we directly focus on latency, with the purpose to minimize the latency of in-network caching for a given request set and a given network topology with multiple clients.

The network is modelled as a connected graph $G = (V, E)$ in which nodes in $V$ are the routers, connected by a set of bidirectional edges in $E$. $O$ is the set of content, indexed by $k \in O$. We use $U \subset V$ to represent the set of client nodes. Node $i \in V$, where $i = 1, 2, ..., |V|$ is equipped with content storage (or cache) with capacity $s_i$. Hence, our model consists of the following major input parameters: (1) a demand matrix $R_j^k$ describes the required data for each node $j \in U$ for each content $k$; (2) a caching capacity vector $s_i$: the storage capacity for each node $i \in V$. The non-caching nodes can be classified via setting the corresponding capacity as 0; (3) a delay cost matrix $c_{ij}$: the latency to retrieve the data chunks from node $j$ to node $i$. Here $j \in U$ and $i \in V$; (4) an extra cost $\lambda$: it mainly refers to the caching cost to store the item in a caching node, such as retrieving the item $i$ from another node (say, the content server) to the node $j$. This cost shows that the utility of retrieving remote content decreases as the distance increases.

In order to gain a satisfactory understanding of the network, we build a caching model which targets minimizing the latency cost for data transmission in the network. Thus we can formulate the content placement problem as an integer program, denoted as the Multi-Client Problem (MC-P). We use $i$ to index the caching nodes in $V$, and $j$ to index the clients in $U$.

$$\begin{aligned}
\text{minimize} \quad & \sum_{k \in O} [\sum_i \lambda y_i^k + \sum_j \sum_i c_{ij} x_{ij}^k] \quad \text{(MC-P)} \\
\text{subject to} \quad & \sum_k y_i^k \leq s_i, \ \forall i, k, \\
& x_{ij}^k \leq R_j^k y_i^k, \ \forall i, j, k \\
& \sum_i x_{ij}^k \geq R_j^k. \ \forall i, j, k \\
& y_i^k \in \{0, 1\} \ \forall i, k
\end{aligned}$$

Variable $y_i^k$, namely, the content placement solution, indicates if node $i$ stores item $k$. $x_{ij}^k$, namely, the load allocation, indicates the fraction of demand of client $j$ that is assigned to node $i$ (the node that caches the item or the content server) for the item $k$.

The first constraint states that each node could not cache more items than its capacity. The second constraint says that if client $j$ acquires the item $k$ from node $i$, $i$ must have the ability to offer this item. The third constraint illustrates that the total offered requests of item $k$ in client $j$ should be larger than client $j$'s demand. Finally, $y_i^k$ can take value in $\{0, 1\}$, so that the above constraints map a solution to our problem.

### B. The Single-Client Case Solution

In this section, we offer a solution to the special case of the problem *MC-P* that there is only one client in the network. Such a case is also known as the *single-client content placement* problem where the whole network serves only one client. One thing should be mentioned is that the single-client here is a *virtual* single client entity, which might consist a

number of clients in a same zone. The problem simplifies to the following:

$$\text{minimize} \quad \sum_{k\in O}[\sum_i \lambda v_i^k + \sum_i c_i w_i^k] \quad \text{(SC-P)}$$

where $w_i^k$ is the total demand assigned to node $i$ for item $k$, and $v_i^k$ indicates if the node $i$ caches item $k$.

We can use a greedy algorithm to return the optimal solution to *SC-P*. Given any feasible solution $(w, v)$, we can set $\hat{w}_i^k = R^k v_i^k$ (according to the second constraint) and obtain a feasible solution $(\hat{w}, \hat{v})$ of no greater cost. So we can eliminate the $w_i^k$ from *SC-P*, changing the objective function to $\min \sum_{k\in O, i\in V}[\lambda + c_i R^k]v_i^k$, and keeping the constraints $\sum_k v_i^k \le s_i$. It is easy to see now that the following greedy algorithm delivers an optimal solution: Consider two sorted arrays, the caching nodes in increasing order of $\lambda + c_i$ and the content request demands in the decreasing order of $R^k$. We assign the items to the caching node $i$ with a decreasing value of $R^k$ until the caching size reaches $s_i$. Then, if all the demands have been assigned, we finish this greedy algorithm. The following lemma is intuitively clear.

**Lemma 2.1.** The greedy algorithm that assigns caching items to each node in increasing order of $\lambda + c_i$ with decreasing request demands $R^k$ delivers an optimal solution to *SC-P*.

## III. A Solution for the Multiple Clients Case

The traditional solution to the multiple client problem might be to allocate different $x_{ij}^k$ and obtain the corresponding $y_i^k$. Then, the pair that accounts for the minimum of the latency would be the solution to the problem. However, the great amount of iterations for getting the optimal $(x_{ij}^k, y_i^k)$ is not acceptable in large networks. Hence, in this section, we describe the MCPC approach to obtaining an approximate solution (represented as $(\hat{x}_{ij}^k, \hat{y}_i^k)$ of the problem defined in section II. Some major parameters are listed in the notations table below.

TABLE: NOTATIONS

| **Notations** | Meaning |
|---|---|
| $V$ | The nodes set of the network |
| $O$ | The set of requested content |
| $U$ | The Clients $U \subset V$ |
| $R_j^k$ | The request of client $j$ for item $k$ |
| $\mathcal{C}$ | The caching center set $\mathcal{C} \subseteq U$ |
| $(x_{ij}^k, y_i^k)$ | the linear solution of *MC-LP* |
| $\mathcal{V}_j$ | the nodes set serve for j with $x_{ij}^k > 0$ |
| $\mathcal{B}_j$ | unclassified nodes serve for $j$ |
| $\xi$ | the caching center $\xi \in \mathcal{C}$ |
| $\mathcal{N}_\xi$ | The partition for caching center $\xi$ |
| $(\hat{x}_{ij}^k, \hat{y}_i^k)$ | the estimated solution of *MC-P* |

First of all, we assume that we have obtained $\hat{x}_{ij}^k$[1], to illustrate the procedures of partition. Based upon this estimate, we partition the network so that each sub-partition can be handled using the single client greedy algorithm. We then use this to find the solution.

[1]A simple estimation could be obtained by calculating the content packets transmitted directly from content servers

### A. Network Partitioning

**(1) Procedure 1: Partition**

At first, we offer a partitioning mechanism to classify the network nodes for each client $j \in U$.

**P1:** Let $\mathcal{V}_j = \{i : \sum_k \hat{x}_{ij}^k > 0\}$ be the nodes in $V$ that serve client $j$. In this procedure, we iteratively select some clients as *caching centers* which are surrounded by some *caching nodes*. Let $\mathcal{C}$ be the set of the current caching centers, which is initially empty $\mathcal{C} = \varnothing$. We use $\mathcal{N}_\xi$ to denote a set of caching nodes assigned to client $\xi \in \mathcal{C}$. For each client $j \notin \mathcal{C}$, we maintain a set $\mathcal{B}_j$ of unclassified nodes that are closer to it than any caching center, that is:

$$\mathcal{B}_j = \{i \in \mathcal{V}_j : i \notin \bigcup_{\xi\in\mathcal{C}} \mathcal{N}_\xi \quad \text{and} \quad c_{ij} \le \min_{\xi\in\mathcal{C}} c_{i\xi}\}. \quad (1)$$

We pick $j \in U$ one by one with decreasing value of $\sum_k R_j^k$ and form the partition $\mathcal{N}_j = \mathcal{B}_j$ around it until we go over all the clients in $U$. Please note that the set of $\mathcal{B}$ keeps being updated once a new caching center is created. In order to compute the total amount of requests, we have the following requirement for any node in $\mathcal{N}_\xi$:

$$\hat{x}_{i\xi}^k \ge \frac{1}{2} R_\xi^k, \qquad \exists k \in O \quad (2)$$

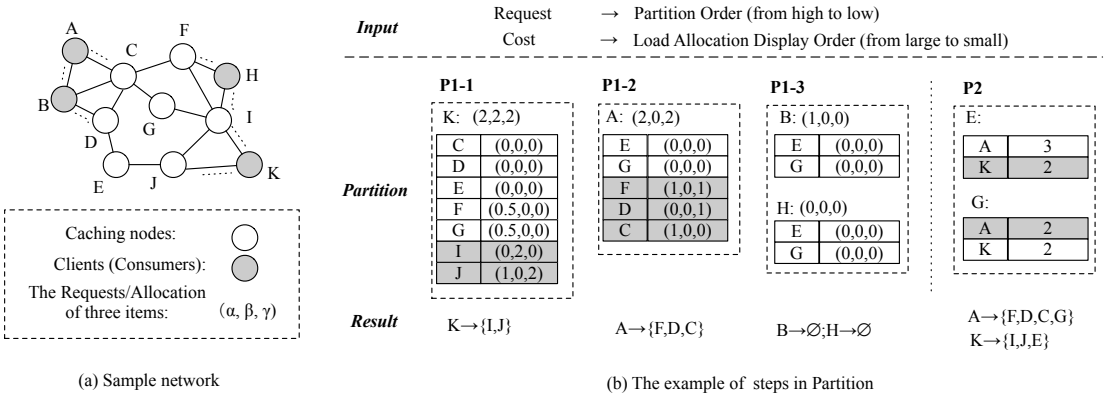
**P2:** There could still be some nodes that are not selected by any caching center $\xi \in \mathcal{C}$ after the previous step. We assign these nodes in $V \setminus \bigcup_{\xi\in\mathcal{C}} \mathcal{N}_\xi$ to partitions by cost $c_{ij}$. That is to say, we assign $i \in V \setminus \bigcup_{\xi\in\mathcal{C}} \mathcal{N}_\xi$ to the partition whose center has the minimum cost towards $i$.

An example partitioning process is illustrated in Fig. 1.

**Theorem 3.1.** Let $i$ be a caching node assigned to the partition $\mathcal{N}_\xi$ in procedure P1 or P2. Let $\mathcal{C}'$ be the set of caching centers just after the assignment. Then, $\xi$ is the caching center closest to $i$ among all the caching centers in $\mathcal{C}'$; that is, $c_{i\xi} = \min_{\xi'\in\mathcal{C}'} c_{i\xi'}$.

*Proof.* Obviously, $c_{i\xi} \ge \min_{\xi'\in\mathcal{C}'} c_{i\xi'}$, since $\xi$ is an element in $\mathcal{C}'$. If $i$ is included in partition in P1, then it must satisfy $i \in \mathcal{B}_\xi$. With the definition of set $\mathcal{B}$, then, $c_{i\xi} \le \min_{\xi'\in\mathcal{C}'} c_{i\xi'}$. Otherwise, if $\xi$ is assigned in P2, node $i$ has smallest cost to $\xi$ among all the caching centers. Combine each case with the initial equation, the theorem is true. □

For any client $j \in U$, it could either be a caching center which has some nodes around, or become a single client without caching nodes when the weight of the nodes in $\mathcal{B}_j$ decreased below $\frac{1}{2}R_j^k$ due to some other caching centers created with the selection of the nodes in $\mathcal{B}_j$. Hence, not all the nodes would get assigned to a partition followed by these two procedures. For the remaining unassigned nodes, we can use the cost parameter to complete the partitioning process. For a example, node $E$ is one of the unassigned nodes in the topology shown in Fig. 1 with two caching centers, it will




(a) Sample network

(b) The example of steps in Partition

Fig. 1: An example of network partitioning (for the sake of easy illustration, we use the distance between nodes to represent the cost): P1, each client selects the caching nodes which could serve at least one item with half of its requests; P2, the unselected caching nodes choose one of the nearest caching center to finish the partitions

select node $K$ as its caching center since the cost to $K$ is smaller compared with the other caching center ($A$).

After the process of partitioning terminates, all the nodes are assigned to one of the caching centers.

**(2) Procedure 2: The Greedy Method**

In this case, we need to obtain a solution of $y_i^k$ of *MC-P* from the first procedure. At this time, we can start to apply an instance of *SC-P* to assign the requests to the nodes within each caching partition. We use the greedy algorithm illustrated in Section II.B to find an estimated optimal solution $(\hat{x}_{ij}^k, \hat{y}_i^k)$ to this integer programming problem.

From the analysis above, we can find that the procedures for solving $MC-P$ (both partition and the greedy method) are highly dependent on the $x_{ij}^k$, namely the load allocation. The procedure displayed above is an ideal situation with the assumption that a global load allocation is got. In the real world, it is hard to do. So, we will present a method later to estimate the load allocation from the requests.

### *B. Contents Placement*

Now, we see that the following two factors are the keys to obtain a satisfactory content placement solution when using multiple clients partition caching method: 1) An estimator of $\hat{x}_{ij}^k$, which represents the load allocation to each node; 2) Each node should maintain the request demand $R_j^k$ and cost for caching $c_{ij}$.

***(1) Approximating load allocation with requests:*** Before the partition procedure, each node $i \in S$ should obtain an estimated value of the fraction value of $\hat{x}_{ij}^k$, and the estimation method should be suitable and applicable in real time network service system with dynamic data changes.

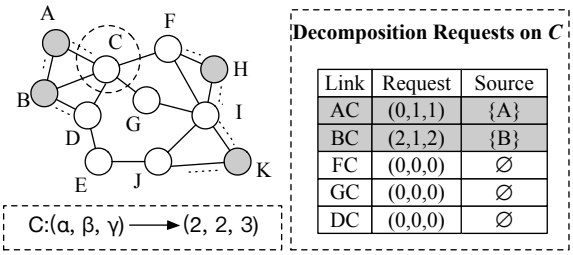
Apparently, each node would contain the requests generated by the corresponding clients, involving all the demand the node would probably serve. Thus, we can implement the estimator for obtaining the estimated value of $\hat{x}_{ij}^k$ via *Requests decomposition* for each client.

*Requests decomposition:* In order to measure the load allocation, the requests record the incoming interfaces with the corresponding counters. Thus, the estimated load $\hat{x}_i$ on node $i$ could be divided to different parts with different interfaces ($interface(\hat{x}_i)$). Besides, each interface would be attached with its serving clients $\{j|j \in interface(U)\}$ in the period of spreading request packets. For simplicity, the interface regards all the serving clients equally, namely, all the clients to be served would get the same load allocated to the interface. Finally, these steps yield the approximated load allocation of $\hat{x}_{ij}^k$.

***(2) Partition:*** Now we start to introduce MCPC partition procedures, which runs in two phases in accordance with *P1* and *P2* as showed in section III-A. In this stage, we divide the network nodes $i$ with $\hat{x}_{ij}^k > 0$ into different sets each of which will be centered around a client, for running the greedy method of *SC-P*. According to the procedures displayed before, MCPC should maintain one of the following two properties: 1) each partition node $i$ should contain the weight of at least $\frac{1}{2}R_j^k, i.e., \exists k, \hat{x}_{ij}^k \geq \frac{1}{2}R_j^k$ for caching center $j$; and 2) if $i$ belongs to the caching center $j$, the cost $c_{ij}$ is smaller than other caching centers. Since we need to pick $j$ with the decreasing order of $\sum_k R_j^k$, each node should maintain the request matrix of the clients. Combining the result from the first step (estimated $\hat{x}_{ij}^k$), we can apply the partition method with each client as presented previously.

In addition, the partition could also be solved in a distributed way, which means that the estimated load with request information in each node could obtain the result of the assignment by using only its local information. As it is displayed in Fig. 2, each node could obtain the load allocation corresponding to each client via *Request decomposition*. Then we check the value with the decreasing order of the request generated from each client. We choose the first one that matches the Equation 2 and assign this node to the corresponding client. The communication cost could be further reduced if we just choose the contributor to the highest load as its caching center.

***(3) Caching Decision:*** In the last phase of the algorithm, we use *SC-P* as the allocation policies for data content within each partition. We consider each partition separately to set up




**Decomposition Requests on *C***

| Link | Request | Source |
|---|---|---|
| AC | (0,1,1) | {A} |
| BC | (2,1,2) | {B} |
| FC | (0,0,0) | ∅ |
| GC | (0,0,0) | ∅ |
| DC | (0,0,0) | ∅ |

Fig. 2: Generating $\hat{x}_{ij}^k$ from the Requests: (1) According to *Requests decomposition*, the current requests record of node $C$ is $(2, 2, 3)$ and it consists of two parts: one is from link $BC$ $(2, 1, 2)$ and the other one comes from link AC $(0, 1, 1)$; (2) The link $BC$ is only used for client B and the link $AC$ operates for client $A$; (2) Thus we get the result of node $C$: $\{(0, 1, 1), (2, 1, 2), (0, 0, 0), (0, 0, 0)\}$

**Algorithm 1:** The Algorithm of Content Placement

**INPUT**: The request load of each partition: $R$;
Capacity Vector $s$;
Cost $c$ and $\lambda$;
**OUTPUT**: Content placement $\hat{y}_i^k$;
Set all $\hat{y}_i^k = 0$;
`/* two sorted lists according to lemma 2.1 */`
sort the nodes with increasing order of the cost, so that the sorted order of nodes is $< n_1, n_2, ..., n_t >$;
sort the Request $R$ with decreasing order of the amount, so that the sorted of item is $< R^1, R^2, ...., R^k >$;
**for** $\tau = 1$ *to* $k$ **do**
  `/* place each items */`
  **while** $R^\tau > 0$ **do**
    **for** $i = 1$ *to* $t$ **do**
      `/* find an appropriate node */`
      **if** $\sum_k \hat{y}_i^k < s_i$ **then**
        $\hat{y}_i^\tau = 1$;
        Serve $R^\tau$ with the value $R$;
        $R^\tau = R^\tau - R$;
      **end**
      **else**
        $\hat{y}_i^k = 0$;
      **end**
    **end**
  **end**
**end**
Return $\hat{y}_i^k$;

an instance of *SC-P* to avoid the whole network signaling. From our design, the instance could acquire the remaining client demand and the capacity of each node in the partition. Then, we can use the greedy algorithm above to obtain an optimal solution to the instance and we will get an assignment of content placement $\hat{y}_i^k$ for a given demand in a network. The procedures of the design are showed in Algorithm 1.

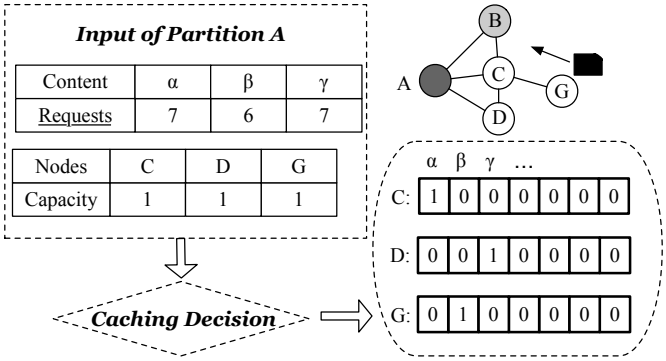



Fig. 3: An example of content placement method

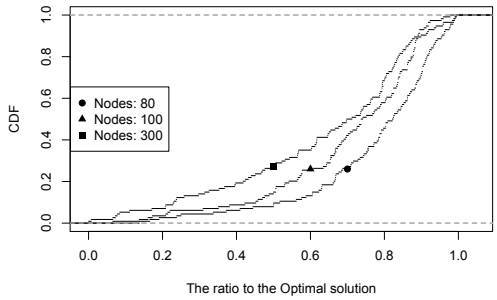
The content placement decisions are based on the results generated by Algorithm 1. The following is an example when we apply this caching decision. Suppose there is a scenario that node $i$ receives the Data Packet containing the corresponding chunk of the content $k_{new}$ which has not been cached at node $i$. If there is sufficient unused space in the node $i$ to place all chunks of the content $k_{new}$, then node $i$ proceeds to cache the Data Packet containing the content $k_{new}$. That is, the entire content $k$ is cached at node $i$. Otherwise, the node searches the caching decision in node $i$. If the corresponding value equals to 1, an eviction operation would be carried out. Let $C_{old}$ represent the set of currently cached items with value 0. If there exists $c' \in C_{old}$ that has a lowest cache hitting number, then content $c'$ is evicted and replaced with content $k_{new}$. Otherwise, the cache is unchanged. Fig. 3 shows an example for the content placement. Nodes $\{C, D, G\}$ are the members of the partition for the caching center $A$. Each of them stores the bitmap of the caching scheme generated by caching center and the items with value 1 are added to cache when it arrives.

## IV. Performance Evaluation

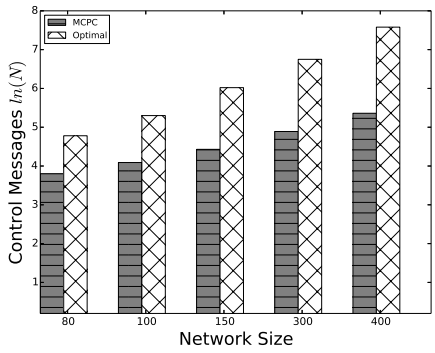
In this section, we evaluate the accuracy and efficiency of MCPC with a series of benchmarks. This evaluation demonstrates that: 1) MCPC achieves higher accuracy compared with the optimal solution, while it has very small overhead in spreading the partition information, and 2) MCPC has the minimum cost in latency among the alternative caching strategies, which is up to $56\%$ reduction. Meanwhile, MCPC uses less caching nodes to achieve the latency reduction.

### A. Accuracy, Cost and Scalability

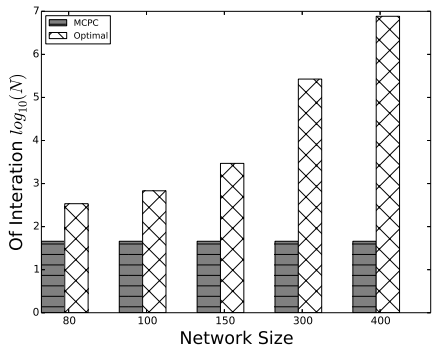
MCPC uses the information (Requests, hop count) in a single node to significantly reduce the computing time and the amount of message signaling overhead in the network. These advantages potentially create some issues about the accuracy of the algorithm. The optimal solution of *MC-P* for getting $y_i^k$ and $x_{ij}^k$ might suffer a higher network latency when compared to *Requests decomposition*. First, we use a numeric simulation to test the performance against the optimal solution. In this test, we employ different network topologies [28] with size varying from $80$ to $400$ nodes and up to $R = 8,000$ requests




(a) The accuracy of MCPC




(b) The communication cost




(c) Convergence on different networks

Fig. 4: MCPC performance compared with optimal solution

are issued for 100 content objects in the network. The caching capacity of each node is randomly generated between 0 to 5.

**Accuracy:** To illustrate the performance of accuracy of MCPC, we run the optimal algorithm and get the value of $\phi_{OPT}$ as a function of the size of the network. Meanwhile, we would also run MCPC with the same parameters as the optimal one to obtain the value of $\phi_{MCPC}$. We then calculate the accuracy of MCPC as its ratio to the optimal solution, $i.e. \frac{\phi_{OPT}}{\phi_{MCPC}}$. The calculated values should fall between $[0, 1]$. Fig. 4(a) plots the CDF of the ratio of each size of the network. We can discover that MCPC achieves a satisfactory accuracy. For small network with 80 nodes, more than $75\%$ of the cases would get $80\%$ more accuracy compared to the optimal one. For large size networks with 300 nodes, more than $60\%$ of the cases achieve the accuracy of $60\%$, and only a very small percentage of the cases perform below $20\%$. The accuracy degrades slightly when the size of the network grows. However, we cannot perform the comparison of these two methods on networks with thousand nodes, as finding the optimal solution takes too much time. Hence, the performance of MCPC is efficient and practical when driven by routing information. This means that MCPC can attain high accuracy without requiring to share information about all the objects (unlike in the optimal scenario).

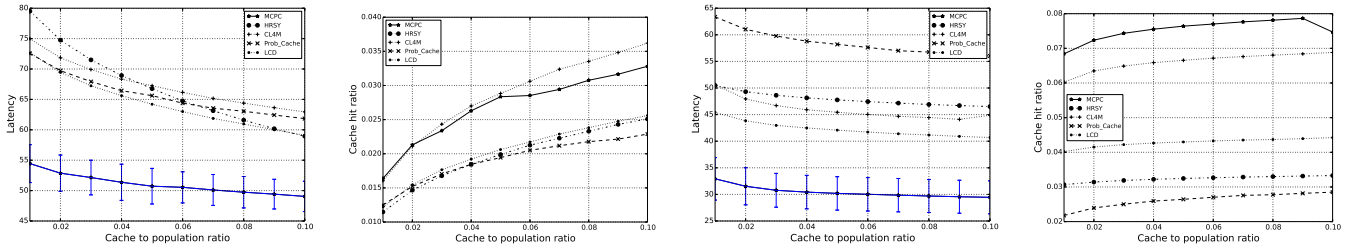
**Communication Cost:** In our strategy, the extra communication cost appears at these points: (1) the partition result to each caching center; (2) the result of caching decision $\hat{y}_i^k$ in each partition. To illustrate the cost of the extra communication cost, we calculate the metadata/control message of the two algorithms via subtracting the data packets from the total packets transmitted. Fig. 4(b) displays the comparison of the amount of control messages when we run MCPC and the optimal solution. We see that the increasing rate of MCPC is much more stable than the optimal solution. The major gaps exist in that MCPC uses the local information for partition, which reduces the data collections cost for partition when the requests change.

**Scalability:** We start to explore scalability, measured by the convergence rate, i.e. how many rounds it takes for the algorithm to initialize. From the illustration in the previous sections, mostly dynamic scenarios are addressed using periodic low cost updates with the request counts in the network nodes, and the re-computation of the partition for content placement happens only when the request patterns changes greatly. Fig. 4(c) compares the convergence rates of the optimal and MCPC on the different topology sizes. The optimal policy needs significantly more iterations to bootstrap than MCPC and it grows sharply. The iteration amount of MCPC remains stable for the number of rounds since MCPC directly uses the routing information to estimate the load without extra calculation. This small amount of iterations is generated by running *SC-P* in each partition.

### *B. Caching Performance*

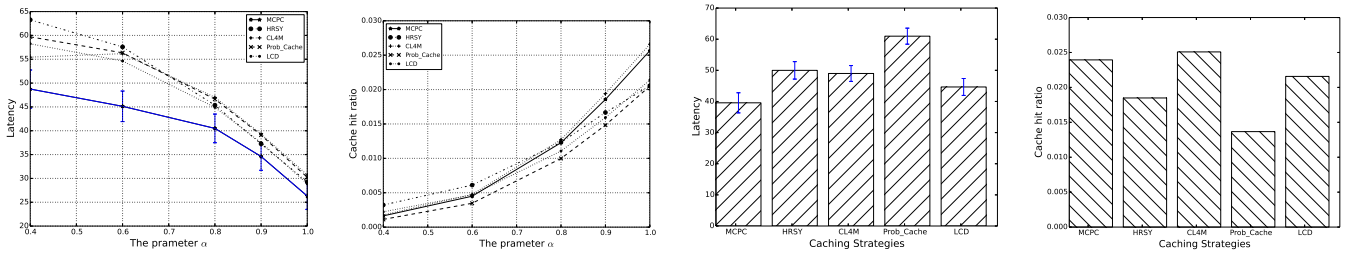
We have implemented our proposal in Icarus [27], a popular network simulator with in-network caching enabled. We compare MCPC with the following caching strategies: Symmetric Hash Routing (HRSY) [26], Caching less for more (CL4M) [7], Probability cache (ProbCache) [21] and Leave Copy Down (LCD) [14]. We use two different kinds of workloads: stationary and GlobeTraff [12]. Essentially, the stationary workload follows the Independent Reference Model (IRM) to generate the different popularity distributions for the content via setting different values to the parameter $\alpha$. GlobeTraff builds on ProWGen's functionality [5], providing additional traffic types such as P2P and Video streaming.

For each run, we use the default configurations of Icarus for the caching nodes, whose cache capacity was distributed uniformly and the caches were initially empty. We randomly allocate the loads to the leaf nodes to perform as the clients. A



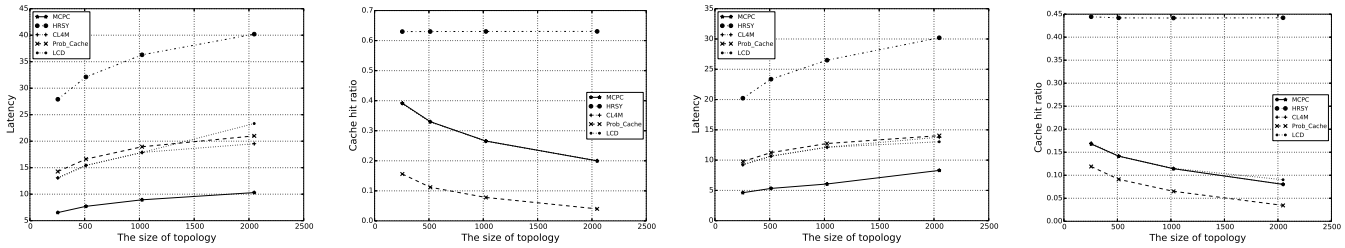

(a) Latency: Stationay (b) Cache hit ratio: Stationay (c) Latency: GlobeTraff (d) Latency: GlobeTraff

Fig. 5: Latency and cache hit ratio performance as a function of the cache to population ratios




(a) Latency: Stationay (b) Cache hit ratio: Stationary (c) Latency: GlobeTraff (d) Cache hit ratio: GlobeTraff

Fig. 6: Latency and cache hit performance under different workloads




(a) Latency: Stationay (b) Cache hit ratio: Stationary (c) Latency: GlobeTraff (d) Cache hit ratio: GlobeTraff

Fig. 7: Performance in different network sizes

total of $9\times10^5$ requests were generated in these scenarios with a total $3\times10^5$ content objects. The first $3\times10^5$ requests were used to build the content placement via MCPC, which would not be used for gathering statistics; the remaining $6\times10^5$ requests were recorded for the performance comparison.

A total of eight topologies were used in our evaluations. Four were real-world ISP-like topologies (nodes, edges): WIDE (30, 33), GEANT (53, 61), TISCALI (240, 810), and SPRINT (604, 2268). The results below displayed the average performance of these four topologies. In particular, the error bars in MCPC display the upper-bound and lower-bound of the results. The other were scale-free tree topologies: Topo1 (255, 254), Topo2 (511, 510), Topo3 (1023, 1022), Topo4 (2047, 2046). Each link latency was 2 ms. In all scenarios, each result was averaged over eight distinct runs.

**Behavior with regard to the cache to population ratio:** Fig. 5(a) and Fig. 5(b) display the result with the stationary workload while Fig. 5(c) and Fig. 5(d) plot the lines from the GlobeTraff workload. For latency, MCPC achieves the best performance among the five strategies, which costs only half the latency as HRSY in the case of using GlobeTraff workload. In terms of cache hit ratio, we plot the average cache hit ratio of each caching node in Fig. 5(b) and Fig. 5(d). LCD does a better job in the stationary workload, but the gap with MCPC is not very large. However, MCPC outperformed LCD with the GlobeTraff workload. On average, our results show MCPC could reduce by $17\%$ to $48\%$ latency against the benchmarks with varying cache to population ratio.

**Behavior under different workloads:** In this scenario, we test the stationary workloads with different parameter of $\alpha$: from $0.4$ to $1.0$. In addition, we also simulate five different temporal workloads obtained from GlobelTraff with the average result shown in Fig. 6(c) and Fig. 6(d). In Fig. 6(a), we can see that the latency decrease sharply with the increasing of $\alpha$. Though MCPC has the smallest values of latency, the gap with the other strategies is narrow. Since MCPC aims to place more items in each partition, it becomes less sensitive to Zipf parameter changes (MCPC achieves $15\%$ latency reduction in

average). As for the cache hit ratio, the result of each caching strategy is very close in stationary workload. In Fig. 6(d), HRSY and ProbCache have the lowest average cache hit ratio, since they use all the caching nodes in the network for caching. Actually, HRSY has the best total cache hit ratio, but it takes more time to retrieve the object and the utilization of caching space is not efficient.

**Behavior with regard to network size:** Fig. 7 shows the result, where the network is of a tree topology and the number of nodes varies from 255 to 2,047. In this simulation, all the nodes could catch objects except the clients and content servers. From the result, we can find that HRSY has the largest latency and can achieve the highest cache hit ratio for both workloads. MCPC costs less than half the latency compared with LCD and CL4M, while the caching hit ratio is almost the same. Actually, about $56\%$ latency is saved when using MCPC in comparison with others in this ubiquitous simulation scenario. It implies that MCPC could cache the requested objects much closer to the clients.

**The cost of caching nodes:** Fig. 8 draws the results of the caching nodes cost, which represents the amount of caching nodes contributing to the cache hit among four different topologies. We can observe from it that HRSY always has the highest ratio of caching nodes cost, accounting for the highest total caching hit ratio but lowest caching hit ratio per node. MCPC and CL4M always have the lowest caching node cost compared with other strategies, only taking $40\%$ to $65\%$ of the cost when compared with HRSY.

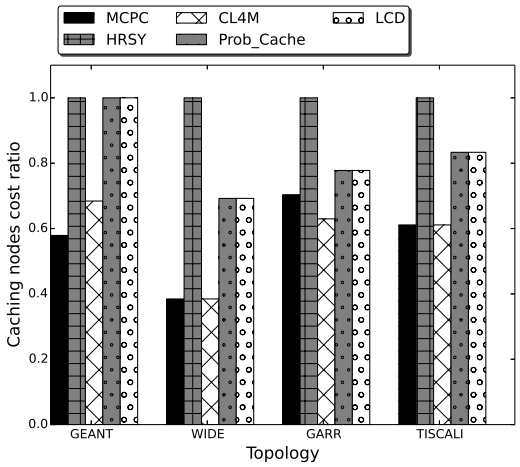



Fig. 8: The ratio of nodes used for caching

## V. Related Work

Over the past few years, researchers have put a lot of effort into the design and implementation of efficient in-network caching for both Web caching and ICNs.

WAVE [8] adjusts the number of chunks to be cached in depending on the content popularity (i.e., access count). As the access count increases, WAVE exponentially increases the number of chunks to be cached and disseminates them more widely. In MPC [4], every node counts the number of local requests for each content name and stores a pair (Content Name; Popularity Count) into a Popularity Table. Once a popularity count reaches a popularity threshold, the content name is tagged as popular and the neighbor nodes is suggested to cache the content through some suggestion primitive. [22] uses the hit rates in content store to calculate the popularity while in [29] uses the content access counts to set the caching threshold. MAGIC [23] aims at reducing the operations of caching with jointly considering content popularity and hop counts. From the above previous works, we can conclude that content popularity is relying on the request rates and hit rates.

Due to the inefficiency with a single factor of popularity, coordinated caching has been studied extensively. Li *et al.* [17] developed a holistic model to quantify the overall network performance of routing contents to clients and the overall provisioning cost incurred by coordinating the in-network storage capacity. J Li *et al.* [16] focused on minimizing the inter-ISP traffic at the administrative boundaries in NDN by considering popularity-driven coordination. They designed a process of information aggregation from end to nodes upwards to the root in their Top-down algorithm for coordination. CRCache [32] utilizes a cross-layer design to cache content in a few selected routing paths based on the correlation of content popularity and the network topology. FairCache [31] considers the caching problem as a Nash bargaining game. It is a heuristic solution that ensures the caches could achieve high performance without weakening others issues such as scalability and accuracy. MuNCC [18] is also a coordinated caching strategy using Bloom filters to aggregate cache states with the neighborhoods of each router.

Recently, some works have targeted caching in the nodes with better connectivity. Chai *et al.* [7] cache content at the nodes having the highest probability of getting a cache hit along the content delivery path. They use the concept *Betweenness Centrality* to find the caching nodes. [1] focuses on mobility prediction to find the best location for pre-fetching and caching in the network. Yeh *et al.* [11], [35] put forward a framework for optimizing forwarding and caching jointly and produced a distributed algorithms for arbitrary topologies. Our strategy use the routing information in a much simpler way with a distributed and adaptive algorithm.

## VI. Conclusion

We have developed a caching scheme MCPC to improve the overall performance of in-network caching. We try to minimize the content access latency mostly depending on the local forwarding information in the router. The design of MCPC also bears in mind the goal that dynamic caching decisions could be made online and in a distributed manner. To achieve these goals, we partition a network with multiple clients into several networks with a single (logical) client first, and solve the problem in a simplified way. We leverage the characteristics of request information in a router and take into account popularity of the content and the network delay from the clients. Our algorithm decreases the content access latency by over $50\%$ for the given benchmarks. It also shows

better caching nodes utilization over a wide range of varying network topologies, cache capacities, objects request patterns and popularity variations covered by the routers.

According to the illustrations in this paper, the caching strategy MCPC has the following advantages: (1) it estimates the distributed content load allocation from request information in each node; (2) it reduces the multiple clients problem to a simple single-client problem using network partition; (3) it works in an uncoordinated environment and offers a solution with less signaling overhead.

## ACKNOWLEDGMENT

This work is sponsored by Natural Science Foundation of China (61572280, 61672312, 61433008).

## REFERENCES

[1] N. Abani, T. Braun, and M. Gerla. Proactive caching with mobility prediction under uncertainty in information-centric networks. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*, pages 88–97. ACM, 2017.

[2] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman. A survey of information-centric networking. *Communications Magazine, IEEE*, 50(7):26–36, 2012.

[3] S. Bayhan, L. Wang, J. Ott, J. Kangasharju, A. Sathiaseelan, and J. Crowcroft. On content indexing for off-path caching in information-centric networks. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, pages 102–111. ACM, 2016.

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

[5] M. Busari and C. Williamson. Prowgen: a synthetic workload generation tool for simulation evaluation of web proxy caches. *Computer Networks*, 38(6):779–794, 2002.

[6] G. Carofiglio, G. Morabito, L. Muscariello, I. Solis, and M. Varvello. From content delivery today to information centric networking. *Computer Networks*, 57(16):3116–3127, 2013.

[7] W. K. Chai, D. He, I. Psaras, and G. Pavlou. Cache "less for more" in information-centric networks (extended version). *Computer Communications*, 36(7):758–770, 2013.

[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*, pages 316–321. IEEE, 2012.

[9] V. Cisco. Cisco visual networking index: Forecast and methodology 2016–2021.(2017), 2017.

[10] X. Huang, Z. Zhao, and H. Zhang. Latency analysis of cooperative caching with multicast for 5g wireless networks. In *Proceedings of the 9th International Conference on Utility and Cloud Computing*, pages 316–320. ACM, 2016.

[11] S. Ioannidis and E. Yeh. Jointly optimal routing and caching for arbitrary network topologies. *arXiv preprint arXiv:1708.05999*, 2017.

[12] K. V. Katsaros, G. Xylomenos, and G. C. Polyzos. Globetraff: a traffic workload generator for the performance evaluation of future internet architectures. In *New Technologies, Mobility and Security (NTMS), 2012 5th International Conference on*, pages 1–5. IEEE, 2012.

[13] P. Krishnan, D. Raz, and Y. Shavitt. The cache location problem. *IEEE/ACM Transactions on Networking (TON)*, 8(5):568–582, 2000.

[14] N. Laoutaris, H. Che, and I. Stavrakakis. The lcd interconnection of lru caches and its analysis. *Performance Evaluation*, 63(7):609–634, 2006.

[15] N. Laoutaris, S. Syntila, and I. Stavrakakis. Meta algorithms for hierarchical web caches. In *Performance, Computing, and Communications, 2004 IEEE International Conference on*, pages 445–452. IEEE, 2004.

[16] 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*, pages 15–26. ACM, 2012.

[17] Y. Li, H. Xie, Y. Wen, and Z.-L. Zhang. Coordinating in-network caching in content-centric networks: Model and analysis. In *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*, pages 62–72. IEEE, 2013.

[18] T. Mick, R. Tourani, and S. Misra. Muncc: Multi-hop neighborhood collaborative caching in information centric networks. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, pages 93–101. ACM, 2016.

[19] Z. Ming, M. Xu, and D. Wang. Age-based cooperative caching in information-centric networking. In *Computer Communication and Networks (ICCCN), 2014 23rd International Conference on*, pages 1–8. IEEE, 2014.

[20] X. N. Nguyen, D. Saucez, and T. Turletti. Efficient caching in content-centric networks using openflow. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 67–68. IEEE, 2013.

[21] I. Psaras, W. K. Chai, and G. Pavlou. Probabilistic in-network caching for information-centric networks. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*, pages 55–60. ACM, 2012.

[22] J. Ran, N. Lv, D. Zhang, Y. Ma, and Z. Xie. On performance of cache policies in named data networking. In *International Conference on Advanced Computer Science and Electronics Information*, pages 668–671, 2013.

[23] J. Ren, W. Qi, C. Westphal, J. Wang, K. Lu, S. Liu, and S. Wang. Magic: A distributed max-gain in-network caching strategy in information-centric networks. In *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, pages 470–475. IEEE, 2014.

[24] G. Rossini and D. Rossi. Coupling caching and forwarding: Benefits, analysis, and implementation. In *Proceedings of the 1st ACM Conference on Information-Centric Networking*, ACM-ICN '14, pages 127–136, 2014.

[25] S. Saha, A. Lukyanenko, and A. Ylä-Jääski. Cooperative caching through routing control in information-centric networks. In *INFOCOM, 2013 Proceedings IEEE*, pages 100–104. IEEE, 2013.

[26] L. Saino, I. Psaras, and G. Pavlou. Hash-routing schemes for information centric networking. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*, pages 27–32. ACM, 2013.

[27] L. Saino, I. Psaras, and G. Pavlou. Icarus: a caching simulator for information centric networking (icn). In *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques*, SIMUTOOLS '14, ICST, Brussels, Belgium, Belgium, 2014. ICST.

[28] N. Spring, R. Mahajan, and D. Wetherall. Measuring isp topologies with rocketfuel. *ACM SIGCOMM Computer Communication Review*, 32(4):133–145, 2002.

[29] K. Thar, T. Z. Oo, C. Pham, S. Ullah, D. H. Lee, and C. S. Hong. Efficient forwarding and popularity based caching for content centric network. In *2015 International Conference on Information Networking (ICOIN)*, pages 330–335. IEEE, 2015.

[30] A. V. Vasilakos, Z. Li, G. Simon, and W. You. Information centric network: Research challenges and opportunities. *Journal of Network and Computer Applications*, 52:1–10, 2015.

[31] L. Wang, G. Tyson, J. Kangasharju, and J. Crowcroft. Faircache: Introducing fairness to icn caching. In *Network Protocols (ICNP), 2016 IEEE 24th International Conference on*, pages 1–10. IEEE, 2016.

[32] W. Wang, Y. Sun, Y. Guo, D. Kaafar, J. Jin, J. Li, and Z. Li. Crcache: Exploiting the correlation between content popularity and network topology information for icn caching. In *2014 IEEE International Conference on Communications (ICC)*, pages 3191–3196. IEEE, 2014.

[33] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. M. Leung. Cache in the air: exploiting content caching and delivery techniques for 5g systems. *IEEE Communications Magazine*, 52(2):131–139, 2014.

[34] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos. A survey of information-centric networking research. *Communications Surveys & Tutorials, IEEE*, 16(2):1024–1049, 2014.

[35] E. Yeh, T. Ho, Y. Cui, M. Burd, R. Liu, and D. Leong. Vip: A framework for joint dynamic forwarding and caching in named data networks. In *Proceedings of the 1st ACM Conference on Information-Centric Networking*, pages 117–126. ACM, 2014.

[36] G. Zhang, Y. Li, and T. Lin. Caching in information centric networking: A survey. *Computer Networks*, 57(16):3128–3141, 2013.