Content size-aware edge caching

Content Size-Aware Edge Caching: A Size-Weighted Popularity-Based Approach

Qiang Li*, Wennian Shi*, Yong Xiao†, Xiaohu Ge*, Ashish Pandharipande‡
*Huazhong University of Science and Technology, Wuhan 430074, Hubei, P. R. China
†University of Arizona, Tucson, AZ 85721, USA. ‡Philips Lighting, 5656 AE Eindhoven, Netherlands
Emails: *{qli_patrick, souvenir, xhge}@hust.edu.cn, †yongxiao@email.arizona.edu, ‡ashish.p@philips.com

***Abstract*—In this paper, content caching is considered at the edge of the network with an objective of offloading recurrent traffic on the capacity-stringent backhaul links to the vicinity of end users. A radio access network equipped with edge servers is considered for caching contents of various sizes, based on which problems of maximizing the edge cache-hit-ratio and minimizing the average content-provisioning cost are respectively formulated. To solve the underlying 0-1 Knapsack problem, a size-weighted popularity (SWP)-based caching framework is proposed, where both content popularity and content size are taken into account when determining the contents to be cached. Depending on the available knowledge and the manner in which the contents are pre-fetched and cached at the edge servers, two algorithms: proactive and reactive, are proposed for the implementation of SWP-based caching. Simulation results are presented to evaluate the performance of our proposed algorithms. We observe a fundamental tradeoff between the average content-provisioning cost and the cache-hit-ratio, and the proactive algorithm outperforms the reactive algorithm.**

***Index Terms*—Edge caching, content popularity, content size, cache-hit-ratio, proactive and reactive.**

I. Introduction

With the rapid proliferation of smart devices and mobile data services, we have witnessed an unprecedented data explosion in the past decade. The main contributors are video and related multimedia applications, which are expected to take up 75% of the mobile data traffic by 2020 [1]. 5G network will support newly emerging multimedia applications such as augmented reality/virtual reality that require stringent demand on high data rate (e.g., $>$ 1 Gbps) coupled with extremely low service latency (e.g., $<$ 10 ms). The existing connection-centric communication architecture [2] is now believed to be inefficient for information sharing and content dissemination, and difficult to meet the stringent demand for 5G.

Recent studies discover that a majority of mobile data traffic is generated by the repeated views and downloads of a limited number of popular contents [3]. Motivated by this observation, content caching at the network edge has been proposed as a key enabling technique for content-centric 5G systems [4]-[6]. By allowing popular contents that are frequently requested by users to be pre-fetched from the remote content provider (RCP) and stored at the network edge, e.g., servers or storage devices co-located with the base stations (BSs), the network congestion can be alleviated while reducing the content delivery delay experienced by end users [5].

Due to the limited storage availability at the edge of the network, determining which contents should be cached at each BS is of critical importance [4]. While most existing works adopt the popularity of content as a key criterion for determining whether a content should be cached or not, they further assume the "same size" for each content irrespective of its popularity [4]-[11]. This can be impractical under many practical scenarios [8], [12]-[14]. In [12], the authors argue that it is important to consider the content size when making caching decisions. A least recently used (LRU) replacement scheme with variable content sizes is considered in [13], [14], where the probability that a requested content is moved to the front of the cache, i.e., of longer expiration period, is determined completely by its size. In [8], the new arriving content object is compared with a set of cached content objects in terms of popularity and size for possible cache replacement, instead of comparing with the least popular object that is cached. Currently, the theoretical foundations for understanding and evaluating the relationship between the storage capacity, content size and popularity are still lacking.

Motivated by the above observation, in this paper we study content caching at the edge of the network taking into consideration of the various sizes of contents. We observe that it is non-trivial to develop simple and efficient caching strategies where different contents can have different sizes. For example, between a popular content requiring large storage space and several less popular contents with smaller sizes, there is no straightforward way to determine which ones should be cached over the others, subject to a limited storage capacity. To provide insights into this problem, we propose a size-weighted popularity (SWP)-based caching framework to balance the impact of content size and content popularity, such that the critical performance metrics of the edge-caching-enabled systems, e.g., content-provisioning cost and delivery latency, can be enhanced. The main contributions of this work are summarized as follows:

- An edge-caching-enabled cloud radio access network (C-RAN) is considered to offload traffic from the capacity-scarce backhaul links to the edge servers. For a given storage capacity constraint at the edge server, we formulate two optimization problems: the cache-hit-ratio maximiza-

The authors would like to acknowledge the support from National Key R&D Program of China (2016YFE0133000): EU-China study on IoT and 5G (EXICITING-723227).

978-1-5386-4727-1/18/$31.00 ©2018 IEEE

tion problem and the average content-provisioning cost minimization problem. We show that these two problems correspond to a typical 0-1 Knapsack problem under various settings [15].

- To address the formulated problems, a SWP-based caching framework has been proposed to optimize the caching decision by jointly considering the content popularity and content size. Based on the proposed SWP framework, proactive and reactive algorithms have been respectively proposed for the cache content placement and update at the edge servers.
- From the simulation results we observe a fundamental tradeoff existing between the edge cache-hit-ratio and the average content-provisioning cost, for each given size-weight factor. A near-optimal performance can be achieved by our proposed SWP-based caching framework with a carefully chosen size weight factor. Furthermore, with more prior knowledge available when making decisions, proactive caching outperforms reactive caching.

The remainder of this paper is organized as follows: Section II presents the system model of edge-caching-enabled C-RAN, where problems of maximizing the cache-hit-ratio and minimizing the average content-provisioning cost have been formulated respectively. Then a SWP-based caching framework is established in Section III, where proactive and reactive algorithms are proposed, respectively. Simulation results are presented to demonstrate the system performance in section IV. Finally, Section V concludes this paper.

II. System Model and Problem Formulation

A. An Edge-Caching-Enabled C-RAN

We consider a C-RAN [16] architecture shown in Fig. 1, in which distributed remote radio heads, i.e., small-cell base stations (SBSs), are deployed to support short-range data transmissions to the associated users. On the other hand, major control logic is abstracted and centralized at the Base Band Unit (BBU) pool, which controls the SBSs through high-speed fronthaul networks and is responsible for resource management, transmission scheduling, etc. This control-data separation architecture [17] enables flexible network configurations and innovations, and enhances the network performance while curtailing the CapEx and OpEx of network operators.

To facilitate content caching at the network edge, an edge server is deployed at each SBS [6] for caching popular contents at the network edge. In the rest of this paper, we use terms "edge server" and "SBS", interchangeably. Then by exploiting the global network view, e.g., network topology, traffic status, content popularity distribution etc., smart caching policy can be derived and implemented at the BBU pool, which will then request the edge servers to execute the corresponding caching operations [10], e.g., cache content placement and update.

Provided that it is generally impossible to cache all contents of interest at the edge of the network, we consider the scenario that an edge server with limited storage capacity S_c is installed at each SBS for storing popular contents that are rapidly accessible to the users within the service coverage. To focus on the impact of content size on caching strategy designs, we consider caching at a single SBS. Our model, however, can be directly extended to the cases with cooperative caching among multiple SBSs where additional performance gains can be accomplished [7]-[11]. We assume that each user can only associate with a single SBS [4], [7], [8].

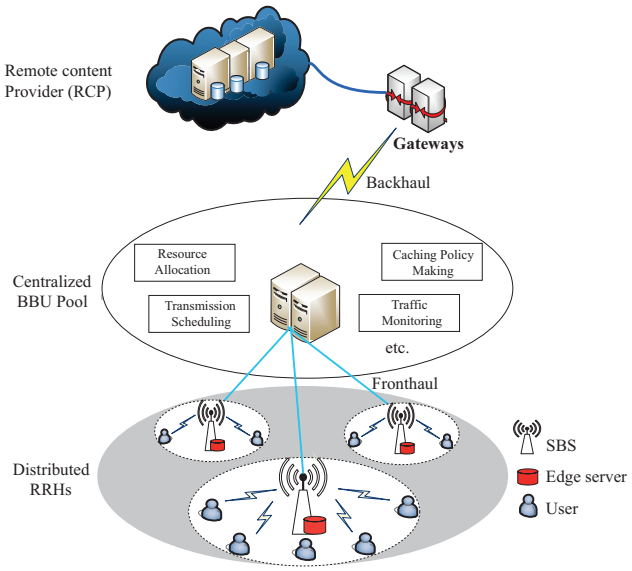

Fig. 1. An illustration of the edge-caching-enabled C-RAN.

To simplify our description, we consider a content library consists of a limited number of M distinct content files. We follow a widely adopted setting and assume the frequency for users to request each of these content files, i.e., popularity of content, follows a Zipf distribution [18]. Without loss of generality, we rank these content files in a descending order according to their popularities, we can then write the popularity of the mth most popular content as

$$p_m = \frac{\frac{1}{m^\alpha}}{\sum_{m=1}^{M} \frac{1}{m^\alpha}}. \quad (1)$$

Here we focus on a homogeneous setting and assume that the content popularity distribution of different small cells are the same [4]. The Zipf exponent α in (1) determines the skewness in the users' group preference. To be specific, a greater α means that the users' requests concentrate more on the first few most popular contents. Otherwise, the users' requests tend to spread more evenly among all the contents.

The popularity distribution of the content files is assumed to remain static within a certain duration [7], [8] and thus can be estimated by the system by using big data analytics [12]. Inspired by the named contents in information-centric networking [19], it is assumed that a popularity tag specifying the rank of the popularity of the requested file is inserted into the frame header of each content by the RCP or the BBU pool [8]. This unique tag can then be recognized by the SBSs to perform cache content placement and update.

B. Problem Formulation

For ease of illustration, we introduce an indicator function $\mathbf{1}_m$, where $\mathbf{1}_m \in \{1, 0\}$ and $m \in \{1, ..., M\}$, as a content placement function specifying whether the mth most popular content is cached by the SBS or not. Then for an arbitrary user request generated within a small cell, the corresponding cache-hit-ratio at the associated SBS can be defined as

$$h_{\text{sbs}} = \sum_{m=1}^{M} \mathbf{1}_m p_m. \tag{2}$$

If the requested content is not cached by the associated SBS, then the request is directed to the BBU pool and forwarded to the RCP to fetch the requested content. This occurs with a probability

$$h_{\text{rcp}} = 1 - h_{\text{sbs}}. \tag{3}$$

To maximize the utilization of the contents cached at the edge server, we formulate the cache-hit-ratio maximization problem as follows:

$$\max_{\{\mathbf{1}_m | m \in \{1,...,M\}\}} h_{\text{sbs}}, \quad s.t. \sum_{m=1}^{M} \mathbf{1}_m s_m \leq S_{\text{c}} \tag{4}$$

where s_m is the size of the mth most popular content. The constraint in (4) guarantees that the size of all cached contents cannot exceed the storage capacity S_{c}.

Cache-hit-ratio characterizes the probability that an arbitrary requested content of a user is available at its associated SBS. It is however difficult to capture the cost of fetching and delivering a requested content between end users and remote/edge servers. To address this issue, we also introduce an important performance metric of the average content-provisioning cost [8].

Formally, we define two cost functions, denoted as, c_{sbs} and c_{rcp}, specifying the per-content-unit delivery costs for each user to provision content stored at the network edge (e.g., the associated SBS) and the RCP, respectively. c_{sbs} can be the price charged by the cellular network operator for transferring each unit of data to its subscribers. c_{rcp} can correspond to the combined price charged by the RCP, Internet service provider as well as the wireless service provider for transferring each unit of data across the backbone Internet all the way to the end user. The values of c_{sbs} and c_{rcp} depend on the logic distance, wireless access cost, energy consumption, etc. Without loss of generality, we assume $c_{\text{sbs}} < c_{\text{rcp}}$. Applying a linear weighted sum of these two cases, the average content-provisioning cost per user's request can be defined as

$$\begin{aligned} C_{\text{avg}} &= c_{\text{sbs}} \sum_{m=1}^{M} \mathbf{1}_m p_m s_m + c_{\text{rcp}} \sum_{m=1}^{M} (1 - \mathbf{1}_m) p_m s_m \\ &= c_{\text{rcp}} \sum_{m=1}^{M} p_m s_m - (c_{\text{rcp}} - c_{\text{sbs}}) \sum_{m=1}^{M} \mathbf{1}_m p_m s_m. \end{aligned} \tag{5}$$

We can then formulate the average content-provisioning cost minimization problem as follows:

$$\min_{\{\mathbf{1}_m | m \in \{1,...,M\}\}} C_{\text{avg}} \Rightarrow \max_{\{\mathbf{1}_m | m \in \{1,...,M\}\}} \sum_{m=1}^{M} \mathbf{1}_m p_m s_m, \quad s.t. \sum_{m=1}^{M} \mathbf{1}_m s_m \leq S_{\text{c}}. \tag{6}$$

We can observe that (6) is equivalent to maximizing the traffic offloaded from the backhaul networks.

Both problems (4) and (6) can be considered as special cases of the classic 0-1 Knapsack problem [15]. To be specific, from a set of items (i.e., M distinct content files) each with different weights (i.e., sizes) and values (i.e., popularities), it needs to determine a collection of contents to be cached at an edge server for maximizing the edge cache-hit-ratio as well as the volume of traffic offloaded from the backhaul, subject to a limited storage capacity S_{c}.

For the case where all content have the same size as assumed in most existing work [4]-[11], it is obvious that the edge server should always store the most popular contents. In contrast, for the general scenarios where contents can have different sizes, it has already been proved that the equivalent 0-1 Knapsack problem is in general NP-hard [7], albeit solutions can be obtained by dynamic programming [15].

III. A Size-Weighted Popularity (SWP)-based Caching Framework

To investigate the impact of different content sizes on the caching decision making process, we define a SWP $p'_m = s_m^{\theta} \cdot p_m$, where θ denotes the size weight factor. If $\theta = 0$, we have $p'_m = p_m$, which corresponds to the case considered in most existing work where the content with a higher popularity should be cached with a higher priority. If $\theta = -1$, we have $p'_m = \frac{p_m}{s_m}$, which means that the content with a higher normalized popularity per size unit should be cached with a higher priority. By adjusting the value of θ, the impact of content size and content popularity can be balanced and different tradeoffs between the edge cache-hit-ratio and average content-provisioning cost can be achieved. We will provide more detailed discussions in Section IV.

Next, within the framework of SWP, we propose two caching algorithms, referred to as the proactive and reactive algorithms.

A. A Proactive Caching Algorithm

Consider the scenario that prior knowledge on the popularity and size of contents is available at the BBU pool, the SWP p'_m of each content m, where $m \in \{1, \cdots, M\}$, can be pre-calculated. Then the contents can be pre-fetched and cached by the SBS from the highest to the lowest p'_m, until the storage capacity limit S_{c} is reached or no more contents can be accommodated. The detailed proactive caching algorithm is illustrated in Algorithm 1.

By running Algorithm 1, the caching policy on what contents should be cached at the SBS can be determined immediately. Then the SBS is instructed to pre-fetch the corresponding

Algorithm 1 Proactive Caching Algorithm

Input: Content popularity p_m, content size s_m, weight factor θ

for $m = 1$ to M **do**
  Calculate $p'_m = s_m^{\theta} \cdot p_m$ and sort p'_m in a descending order, each assigned with a new index m', where $m' \in \{1, \cdots, M\}$
end for
for $m' = 1$ to M **do**
  if $S_{\text{c}} \geq s_{m'}$ **then**
    record the index m'
    $S_{\text{c}} = S_{\text{c}} - s_{m'}$
  end if
end for

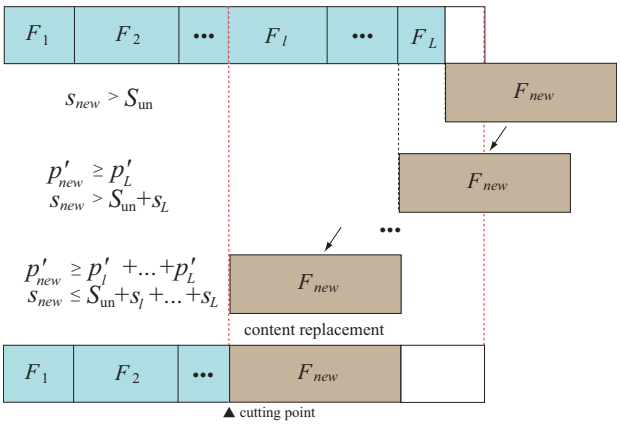

Fig. 2. An illustration of forward-sliding window-based content comparison and replacement in the reactive algorithm where a cutting point is found.

contents during off-peak hours through the backhaul links, so the contents that are mostly likely to be requested by the users can be pre-stored at the edge server. In other words, popular contents can be pre-fetched and stored at the edge server before they are actually requested.

B. A Reactive Caching Algorithm

For reactive caching, each time a content is requested and fetched from the RCP, this content will be kept at the edge server if there is still room left. Otherwise, the cache update will take place in which the SWP and size of the incoming content are compared with those of the already cached contents. As shown in Fig. 2, a forward-sliding window-based comparison is conducted to determine which content should be kept or discarded. The detailed reactive caching algorithm is given in Algorithm 2.

As shown in Fig. 2, in Algorithm 2 we consider the scenario where there are already L contents cached by the SBS, with the remaining storage S_{un} left unoccupied. The cached contents are sorted in a SWP descending order $\{F_1, \cdots, F_L\}$, each with SWP p'_l and size s_l, where $l \in \{1, \cdots, L\}$. Then when a new content F_{new} arrives with SWP p'_{new} and size s_{new}, it will be cached according to the following criteria:

- If $s_{new} \leq S_{\text{un}}$, then F_{new} is directly cached without content replacement;
- Otherwise, if $s_{new} > S_{\text{un}}$ and $p'_{new} \geq p'_L$, check if $s_{new} \leq S_{\text{un}} + s_L$. If so, replace F_L by F_{new};

Algorithm 2 Reactive Caching Algorithm

Input: A new content F_{new} is requested and fetched, with SWP p'_{new} and size s_{new}; L contents $F_1, \cdots, F_L$ are already stored in the cache space, each with SWP p'_l and size s_l, where $l \in \{1, \cdots, L\}$; S_{un} denotes the remaining storage left unoccupied.

if $S_{\text{un}} \geq s_{new}$ **then**
  cache F_{new}
else
  $l = L$, $p' = 0$, $s = 0$
  while $l \geq 0$ **do**
    $p' = p' + p'_l$ and $s = s + s_l$
    if $p'_{new} \leq p'$ **then**
      break
    else if $s_{new} \leq s + S_{\text{un}}$ **then**
      Remove contents $F_l, \cdots, F_L$ and cache F_{new} instead and break
    else
      $l = l - 1$
    end if
  end while
end if

- Otherwise, if $s_{new} > S_{\text{un}} + s_L$ and $p'_{new} \geq p'_L + p'_{L-1}$, check if $s_{new} \leq S_{\text{un}} + s_L + s_{L-1}$. If so, replace the F_{L-1} and F_L by F_{new};
- The above forward-sliding window-based comparisons are conducted until a cutting point l is found, where $p'_{new} \geq p'_l + \cdots + p'_L$ and $s_{new} \leq S_{\text{un}} + s_l + \cdots + s_L$, then the already cached contents $F_l, \cdots, F_L$ are removed and replaced by F_{new}. If however no such a cutting point can be found, then F_{new} will not be cached.

By running Algorithm 2, frequent content comparisons and content replacements may occur depending on how frequent the users initiate requests. When the remaining storage S_{un} is insufficient for accommodating new incoming contents and no cached contents can be replaced, Algorithm 2 converges, as will be shown later in Fig. 6.

C. Proactive Caching vs. Reactive Caching

For proactive caching, it is assumed that both the popularity and size of each content in the library are known to the BBU pool for deciding which contents, of what sizes, should be cached. Then once a caching policy is derived, the SBSs are instructed to pre-fetch the corresponding contents from the RCP through backhaul links during the off-peak hour automatically. Thus the complexity of the proposed proactive algorithm is $O(M)$ where M is the number of contents in the library.

On the other hand, for reactive caching, a content could be cached at the SBS only when it is requested by a user [11]. Specifically, without requiring prior knowledge on the popularity and size of all contents, each time a new content arrives, the SBS reads its popularity tag and size information to determine whether this content should be cached or not [11]. Thus the complexity involved in running Algorithm 2 is $O(L)$ where L is the number of cached contents at the edge cloud server. However, compared to proactive caching where all cached contents can be determined immediately,

TABLE I
A LIST OF PARAMETERS ADOPTED IN THE SIMULATIONS.

Parameter	Definition	Value
M	Number of contents in the library	1000
S_c	Storage constraint at SBS	400
c_{sbs}	Per-content-unit cost from SBS	1
c_{rcp}	Per-content-unit cost from RCP	10
θ	size weight factor	$[-2, +2]$

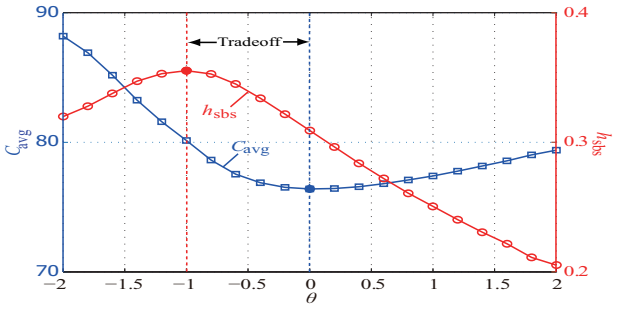

Fig. 3. cache-hit-ratio h_{sbs} and average content-provisioning cost C_{avg} of the proactive caching algorithm with respect to size weight factor θ.

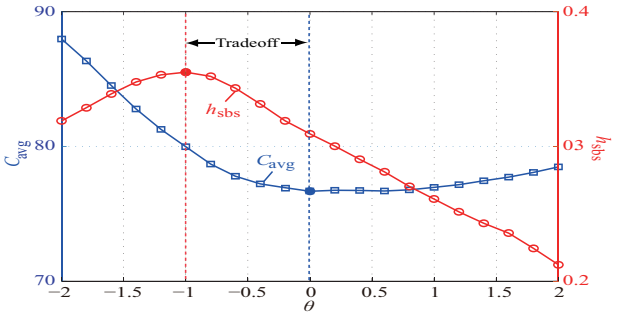

Fig. 4. cache-hit-ratio h_{sbs} and average content-provisioning cost C_{avg} of the reactive caching algorithm with respect to size weight factor θ.

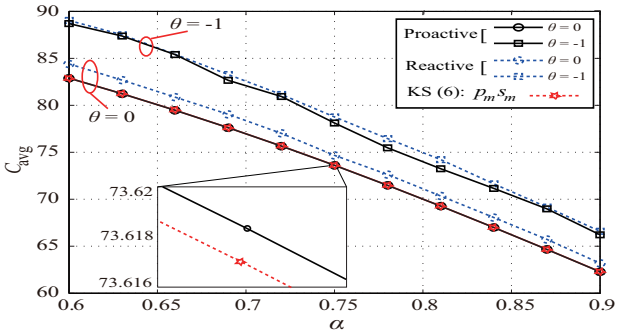

Fig. 5. Average content-provisioning cost C_{avg} of the proposed proactive and reactive caching algorithms with respect to α, where $\theta = -1, 0$, respectively.

it takes longer time for the proposed reactive algorithm to converge, before which frequent popularity comparisons and cache content replacement will take place.

Although no prior knowledge of the popularity distribution and size of contents is required by reactive caching, extra overhead is required for the frequent content comparisons and replacements, before no cached contents can be replaced. Both the proposed schemes have cons and pros, and a selection between proactive and reactive caching depends on the specific application scenario and availability of prior knowledge.

IV. Simulation Results

In this section, simulations are conducted to demonstrate the performance of the considered edge-caching-enabled C-RAN. The cache-hit-ratio h_{sbs} at the SBS and the average content-provisioning cost C_{avg} per request are used as performance metrics. We follow a commonly adopted setting and assume the size of each content in the library follows a discrete uniform distribution Unif$[1, 20]$. A Zipf distribution with $\alpha = 0.7$ is adopted to generate the random requests originated by end users. To be specific, the simulation results averaged over 100 realizations, each consisting of 10000 random user requests consecutively, are demonstrated. Detailed parameters of the settings for our simulations are listed in Table I.

In Fig. 3, both h_{sbs} and C_{avg} that can be achieved by the proactive caching algorithm are compared under different values of the size weight factor θ. It is observed that with the increase of θ, h_{sbs} firstly increases, and reaches a maximum at $\theta = -1$, before it decreases. This is because when $\theta = -1$, the content with a higher normalized popularity per unit size will be given a higher priority to be cached, which significantly enhances the cache-hit-ratio (4) for a given storage constraint S_c. On the other hand, when increasing θ, C_{avg} firstly decreases, and reaches a minimum at $\theta = 0$, before it increases. This is because C_{avg} given in (5) is defined per user request. The content with a higher popularity is more likely to be requested and therefore should be cached with a higher priority.

Furthermore, it is worth noting that when $-1 < \theta < 0$, a performance tradeoff exists between h_{sbs} and C_{avg}. Thus the size weight factor θ needs to be prudentially designed for reaching a balance between the cache-hit-ratio and average content-provisioning cost, depending on specific requirements. We can have similar observations for the proposed reactive caching as shown in Fig. 4, when Algorithm 2 converges and no cached contents at the SBS can be replaced.

C_{avg} is demonstrated in Fig. 5 under proactive and reactive algorithms, where the result achieved by solving a Knapsack problem (6) with value $p_m s_m$ for each content by using dynamic programming [15], referred to as KS, is also illustrated. It is observed that with a higher α, since the users' requests concentrate more on a limited number of the most popular contents, the overall C_{avg} decreases. In addition, it is observed that a significant performance gain can be achieved when setting $\theta = 0$ compared to $\theta = -1$, as is shown in Fig. 3 and Fig. 4. Furthermore, it is observed that the proposed proactive caching algorithm achieves a performance gain over the proposed reactive caching algorithm. This is reasonable as with more prior knowledge on the popularity distribution and size of contents, smart caching decisions can be made