IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON MOBILE MULTIMEDIA: METHODOLOGY AND APPLICATIONS

Received October 22, 2019, accepted November 19, 2019, date of publication November 25, 2019,
date of current version December 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2955749

Short-Term Time-Varying Request Model Based Chunk Caching Scheme for Live Streaming in Mobile Edge-Cloud Environment

WOO-JOONG KIM, KYUNG-NO JOO, (Member, IEEE), AND CHAN-HYUN YOUN, (Senior Member, IEEE)
School of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon 34141, South Korea

Corresponding author: Chan-Hyun Youn (chyoun@kaist.ac.kr)

This work was supported by "The Cross-Ministry Giga KOREA Project" through the South Korea Government (MSIT) in part under Grant GK19P0600, Development and Demonstration of Smart City Service over 5G Network, and in part under Grant GK19P0400, Development of Mobile Edge Computing Platform Technology for URLLC Services.

ABSTRACT Mobile Edge Computing Caching System (MECCS) realizes low-latency and high-bandwidth content access and enables seamless 4K Ultra High Definition (UHD) video streaming by caching content in advance at edge-servers of a cellular network. The objective of MECCS is to maximize cache hit by caching highly popular video content while utilizing the storage capacity efficiently in edge-servers. Most of existing caching schemes estimate the popularity of each content based on content request history in off-line or on-line manners, considering the characteristics of Video-on-Demand (VoD) content which has long-term time-varying popularity. However, since live streaming follows Short-term Time-Varying (STV) characteristics, estimating popularity based on content request history do not guarantee acceptable performance on cache hit for live streaming. In this paper, we propose a request model to estimate the popularity distribution considering STV characteristics. Also, we propose a STV request model-based chunk caching scheme to cache highly popular content and enhance cache hit in multiple live channels, utilizing the storage capacity of collaborative edge-servers efficiently. Experimental results show that the proposed scheme outperforms existing schemes regarding cache hit and backhaul traffic.

INDEX TERMS Mobile edge computing, content caching, content popularity, live streaming.

I. INTRODUCTION

The mobile traffic is expected to reach 40 exabytes per month by 2021 [1]. Because of the centralized mobile network architecture, the growth of mobile traffic will exert more pressure on the capacity of backhaul links connecting the core and Radio Access Network (RAN) [2]. This is a bottleneck that challenges the efficiency of backhaul links. While most of the mobile traffic, expected to occur in the future, is caused by content delivery such as video streaming [1], the conventional Content Delivery Network (CDN), provided by CDN operators such as Akamai and Google Global Cache, cannot relieve the backhaul traffic load. The rationale is that the content delivery of CDN is performed from the Internet to mobile end-users through several core networks and backhaul links [3]. Since these issues cause network latency and throughput degradation for future mobile video streaming, a future architecture is required for CDN to relieve the backhaul traffic load and satisfy Quality of Experience (QoE) for future mobile video streaming. Meanwhile, with the advent of advanced mobile services such as Virtual and Augmented Reality (VR/AR), supporting the 4K Ultra High Definition (UHD) video streaming for mobile end-users is required for future CDN architectures [4]. Recently, social-media service providers such as Twitch, YouTube and Facebook are trying to provide live streaming as well as video-on-Demand (VoD) streaming service in 4K UHD video quality for mobile end-users [5]. The social-media service typically utilizes HTTP live streaming, which operates based on the TCP protocol over the end-to-end (E2E) content delivery path involving both RAN and backhaul links including the internet for mobile edge-users. Due to the characteristics of

The associate editor coordinating the review of this manuscript and approving it for publication was Honggang Wang.

the E2E content delivery path: fluctuating RAN latency combined with long-latency backhaul, typical 4K video stream scenarios do not satisfy 15Mbps bitrate, which is required to provide seamless QoE with high video quality and minimal interruption to playback [5]. Causing the slow increase of TCP congestion window in TCP slow-start phase, this characteristics deteriorates TCP throughput over the E2E content delivery path [5]. In order to provide seamless 4K video streaming, it is necessary to resolve these issues on the poor TCP performance caused by the long-latency E2E content delivery.

In this regard, Mobile Edge Computing (MEC) has been identified as one of the most promising solutions to address these issues. The MEC places cloud-computing capabilities on the RAN, which gets locally closer to end users [6]. This proximity of the MEC not only reduces the service latency but also alleviates the heavy load of the backhaul network. Hence, it can provide a good solution for the future CDN architecture enabling seamless 4K UHD video streaming by caching and/or prefetching content on edge-servers called Mobile Edge Computing Caching System (MECCS). Including this proximity, MECCS inherits characteristics of MEC: collaboration with other edge-servers, location-awareness, supporting mobility, etc. [7], [8]. These characteristics make MECCS provide more sophisticated caching scheme and guarantee the service QoE. Furthermore, since MEC providers, such as telecommunication companies [9], may charge content providers for storage usage, it is also necessary for MECCS to utilize the storage efficiently for caching [10].

Recently, there have been several previous works on how to design an efficient content caching scheme in MECCS. Most of these previous works tried to increase cache hit for VoD streaming, estimating the popularity of each VoD content based on content request history.

However, their works have limitation to apply to live streaming. In live streaming, estimating popularity based on content request history does not work properly and results in low accuracy since it quickly becomes invalid in a short-term period due to the characteristics of live streaming called as Short-term Time-Varying (STV) in this paper. Consequently, they cannot achieve the acceptable performance on cache hit and increase the cache management cost including the MEC storage cost and the backhaul network traffic cost. It is necessary to propose a new model for popularity estimation applicable to STV characteristics of live streaming. In order to resolve these problems, we propose a request model to estimate the popularity distribution considering STV characteristics. Based on this model, we propose a caching scheme for multiple live channels in live streaming, in order to maximize cache hit for satisfying the QoE of end-users and reduce the backhaul traffic load. Furthermore, the proposed scheme benefits from the potential of increasing cache hit more using collaboration between nearby edge-servers.

II. DESIGN OF CACHING SCHEME FOR LIVE STREAMING IN MEC ENVIRONMENT

In this section, we describe the cache management in MEC environment and the static popularity characteristics of its previous works for VoD streaming. We describe the characteristics of live streaming regarding content generation time and user request pattern compared to VoD streaming. Also, we address the limitation of static popularity model, utilized by previous works, when applied to live streaming of which popularity distribution follows STV characteristics.

A. CACHE MANAGEMENT SCHEMES IN MEC ENVIRONMENT

The current telco networks cannot satisfy the bandwidth requirements of future bandwidth-hungry mobile video streaming services that may require guaranteed 4K UHD quality at all time, due to backhaul network congestion caused by the future aggregated service demand which exceeds the capacity of the centralized mobile network [11]. It makes end-users objectionable due to poor QoE with low video quality and frequent interruption to playback. It discourages the use of emerging services and increases the QoE management cost. In order to resolve the congestion problem and satisfy the performance required by the emerging services, one approach is to locate the services closer to end-users in MEC environment. In this regard, MECCS has been introduced as a good solution for the explosive demand on video streaming service to save the congestion cost and provide assured QoE to end-users with low-latency and high-bandwidth by caching content at edge-servers. While the storage usage cost at a cloud environment is cheaper compared to a MEC environment considering the economy of scale, the incurred bandwidth usage cost is higher for services located at a cloud compared to a MEC [11]. Considering this trade-off, it is essential for MECCS to maximize cache hit by caching highly popular content while utilizing the MEC storage capacity efficiently for the cache management, as shown in Figure 1 [10]. We discuss with previous works proposing the caching schemes to increase cache hit by caching popular content in MECCS. Ahlehagh, Hasti, and Sujit Dey. [3] proposed RAN-aware reactive and proactive caching policies based on user preference profiles of active users in a small cell. Tran *et al.* [12] proposed a collaborative hierarchical caching scheme in multiple edge servers to alleviate backhaul usage and minimize average delay with the known popularity of each VoD content. They assumed the popularity distribution of content is stationary in Zipf-like distribution for a long-term period, which is commonly used for VoD content popularity with high accuracy and can be estimated based on request history [13], [14]. Lee *et al.* [15] and Guo *et al.* [16] focused on caching segments derived from VoD content. They proposed a caching scheme based on chunk popularity with VCR-like interactive operations (e.g. jump, pause, fast forward) for VoD content. They also assumed the popularity distribution of chunks in VoD content is stationary for a long-term period. In practice,

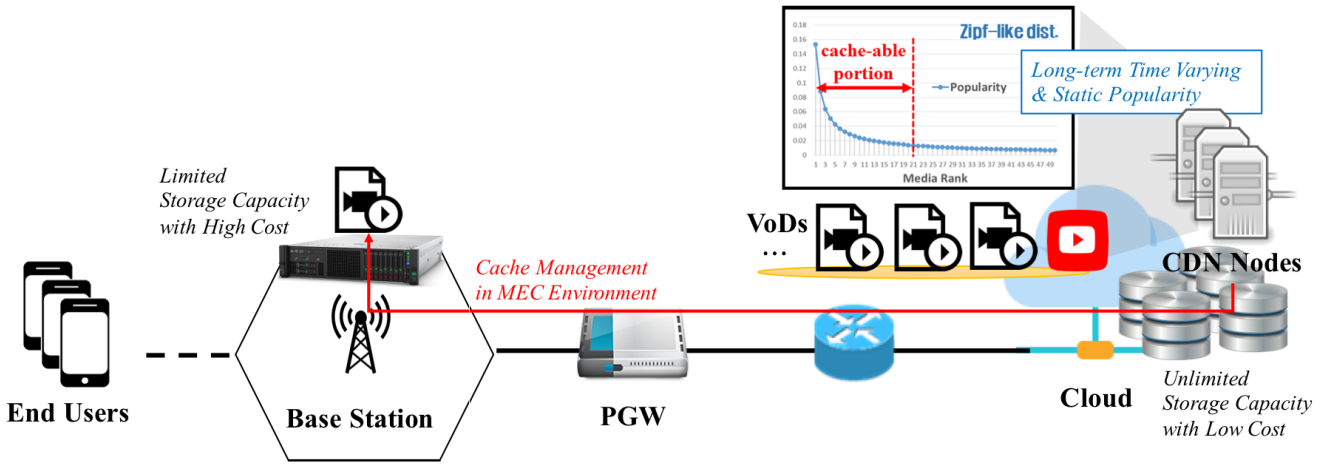


FIGURE 1. Problem of cache management in MEC environment: Maximizing cost-effectiveness on cache hit by caching highly popular content, utilizing MEC storage capacity efficiently.

the popularity distribution of chunks forms around the beginning chunks or the specific popular chunks statically in VoD content. Leconte *et al.* [17] proposed an age-based threshold scheme using learning-based popularity estimation in order to overcome the difficulty in estimating popularity with small population of small cells. Their scheme is based on a dynamic request model, the recently proposed Poisson shot noise model, which fits well real data of long-term time-varying VoD content requests in cellular networks [17]. Most of these previous works tried to estimate the popularity of each content based on content request history in off-line or on-line manners. Considering the characteristics of VoD content which has long-term time-varying popularity, they assumed that the requests on each content would continue to be generated and the estimated popularity based on content request history is fairly valid for each content in a long-term period. They showed the quite good performance on cache hit for VoD content.

B. LIMITATION OF CACHE MANAGEMENT SCHEMES IN LIVE STREAMING

However, these previous works have limitation to apply to live streaming, which has different characteristics with VoD streaming regarding content generation time and user request pattern [14]. In VoD streaming, a sequence of chunks has been generated in advance before streaming to end-users and the popularity distribution of chunks forms around the beginning chunks or the specific popular chunks statically in a long-term period [15], [16]. In live streaming, content is generated on-the-fly as a chunk, which is a short-term video, from a live channel and is requested mostly near its generation time [14]. In addition, with a shift of requests in a short time, the next generated chunk is requested. The reason is that end-users watching a live channel are more likely to request the recent chunks and then continually request the subsequent chunks. Therefore, the popularity distribution of chunks forms around the recent chunks being most likely to be requested by end-users and is shifted over time to the chunks generated recently [14], [16]. Each chunk popularity increases and decreases over time dynamically with the shift of the distribution. Each chunk becomes outdated in a short-term period, and its chunk popularity disappears rapidly. For that reason, the chunks that are already heavily requested in the previous period are hardly requested anymore. As described in Figure 2, if $\{H11\}$ is requested frequently in the last period, $\{H11\}$ is unlikely to be requested again from this time onward. In addition, the recently generated chunks, which have not been requested yet, such as $\{H16\}$ and $\{H17\}$ have a high request probability in a short-term period. These characteristics of live streaming is called Short-Term Time-Varying (STV) in this paper. Due to STV characteristics, the estimated popularity based on content request history, utilized by previous works, become invalid in a short-term period and shows low accuracy for each chunk in a live channel. Besides, the width of this popularity distribution is determined by live latency, which is the time difference between the actual live event and the video viewed by the user. It ranges from several seconds to several tens of seconds depending on various factors such as buffering time (including the initial delay) [18]. As described in Figure 2, the chunks requested in a short-term period is concentrated on several chunks limited by min/max live latency. In order to resolve the limitation of previous works and reflect STV characteristics of live streaming, we analyze a request model-based on chunk requests of end-users in each short-term period to estimate the chunk popularity distribution in the next period. We consider the shift of the distribution occurred by the fact that end-users watching a live channel have the high probability to request the chunks sequentially over time periods. We refer to a dynamic request model, Poisson shot noise model utilized by Leconte *et al.* [17], [33]. It formulates the popularity occurred by content l as a shot, which represents the request rate pattern characterized by a shape (λ_l), a duration (L), an arrival time (t_l), and

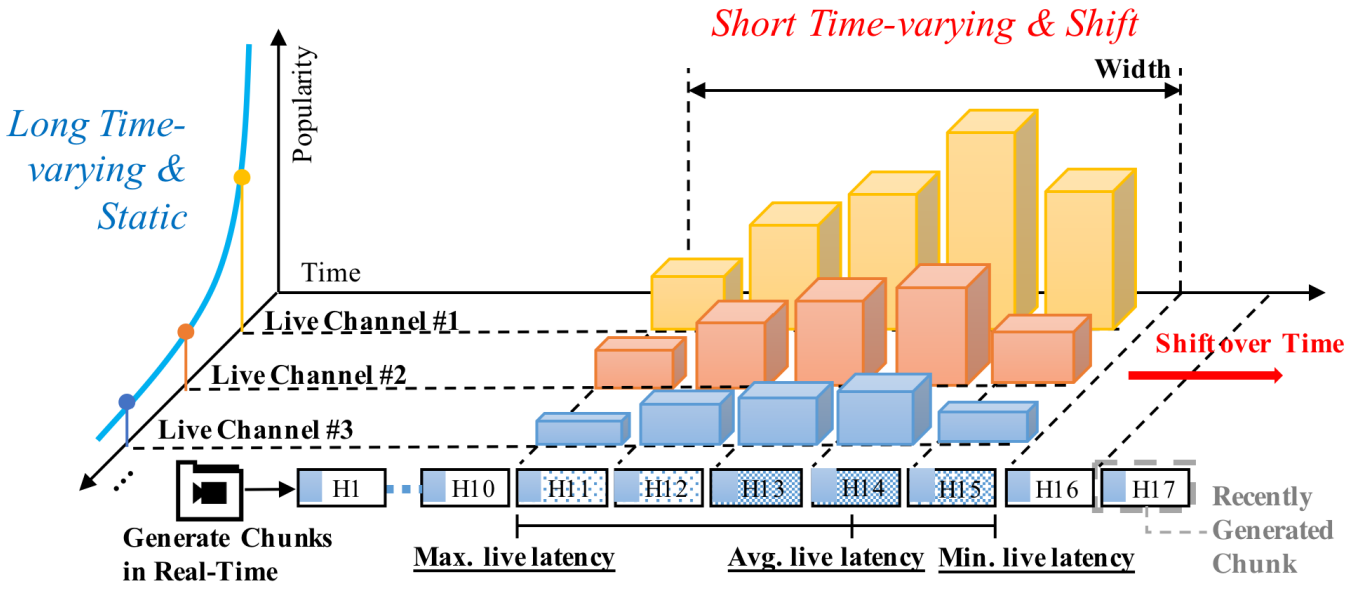


FIGURE 2. Sketch of popularity distribution in the unit of chunk for live streaming. The popularity distribution of chunks forms around the recent chunks being most likely to be requested by end-users and is shifted over time to the chunks generated recently.

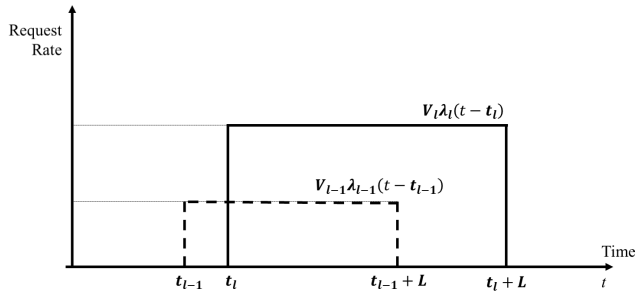


FIGURE 3. Time-varying request model for popularity distribution estimation regarding VoD Content [17].

a volume (V_l). The request rate is treated as the popularity of content l. Since it is reported that the shape and duration have a smaller impact to the hit probability for VoD content of which popularity is long-term time-varying relatively, previous works mainly use simple rectangular pulses of fixed long durations for mathematical simplicity as shown in Figure 3 [17], [33]. A time-inhomogeneous Poisson process describes the request process for a given content l.

$$r_l = V_l\lambda_l(t-t_l),$$
$$where\ \lambda_l(t) = \begin{cases} 1 & \text{if } t_l \le t \le t_l + L \\ 0 & \text{if } otherwise \end{cases} \quad (1)$$

However, the static shape and long duration are not suitable for live streaming. In the next section, we define a new dynamic request model adapt to STV characteristics in a unit of chunk discussed in this subsection and then, based on it, we propose the heuristic model, called chunk score, which is practical to be applied to a caching scheme for live streaming.

III. COLLABORATIVE CACHING SCHEME BASED ON STV REQUEST MODEL FOR LIVE STREAMING IN MEC ENVIRONMENT

In this section, we describe the target MEC environment and system with several assumptions and conditions and define the problem with an objective function to maximize cache hit for the multiple live channels. Then, in order to resolve the problem, we propose the STV request model to precisely estimate the popularity distribution in live streaming. Lastly, we propose STV request model-based collaborative chunk caching scheme to cache highly popular chunks for multiple live channels in order to maximize cache hit for satisfying the QoE of live streaming even in 4K UHD video quality and reduce the backhaul traffic load.

A. SYSTEM MODEL AND ENVIRONMENT

Figure 4 presents MECCS architecture for live streaming. We consider that a single edge server, called as MEC server interchangeably in the rest of this paper, is attached to a base station in a small cell area. This architecture consists of end-users, MEC server, Cloud Central Unit (CCU) and CDN. Live streams of live channels broadcasted by streamers are uploaded to CDN from live sources. CDN transcodes each live stream into a sequence of chunks with various bitrates, which are short-term videos. Chunks are represented as $A = \{a_{i,j}\}$ where i is channel index, and j is chunk index. In addition, we denote that $b(a_{i,j})$ is the file size of $a_{i,j}$. MEC server acts as a cache server based on a caching scheme for live streaming. It periodically determines which chunks to be evicted or cached from CDN in advance. We denote $C = \{a_{i,j}\}$ as a set of cached chunks in a MEC server. It has a constraint on cache size represented as C_{size}. End-users watching a live channel transfer their chunk requests to

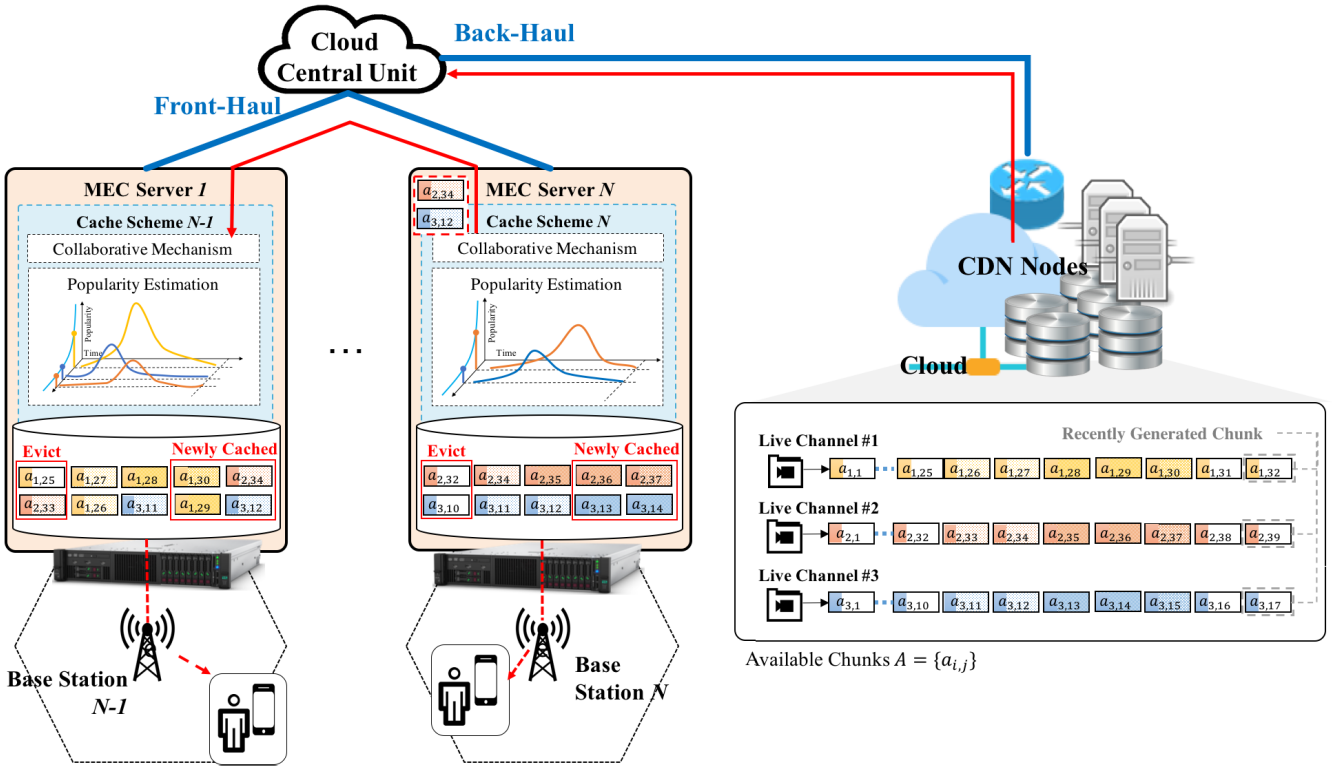


FIGURE 4. Collaborative hierarchical MECCS architecture for live streaming in MEC environment.

MEC server directly through a nearby base station in a cell area. If the requested chunks are not cached in the MEC server, CDN responds the chunks to end-users through backhaul links. Otherwise, they are responded directly from the MEC server with faster latency. We denote d as delay occurred by transferring a unit-size file from MEC server to end-users through RAN and denote d_b as delay occurred by transferring a unit-size file from CDN to end-users through backhaul links. We assume $d \ll d_b$. Furthermore, MEC server has a hierarchical topology in which nearby MEC servers can communicate and collaborate with each other in front-haul links through CCU[1] with faster latency than the latency between the MEC server and CDN [12]. If the requested chunks are not cached in the nearest MEC server but cached in nearby MEC servers, the chunks are responded from the nearby MEC servers in front-haul links through CCU with additional delay d_m occurred by transferring a unit-size file between nearby MEC servers. Otherwise, if the requested chunks are not cached in any MEC servers, the requests are responded from CDN through backhaul links with delay d_b. Hence, a MEC server can decide not to cache some of the chunks if other nearby MEC servers can share the corresponding chunks within an acceptable latency in order to save the storage of MEC server. The saved storage space can be used for storing other chunks to increase cache hit ratio more. We consider N MEC servers collaborate with each other. We assume they are located in close within k hops and $d_m \ll d_b - d$.

[1]CCU is a central collaboration controller hosted in a Cloud RAN, connected to all the BSs through low-latency and high capacity fronthaul links [12]. In this paper, CCU plays only a role in supporting the collaboration between BSs, although it is also able to cache some contents.

B. STV REQUEST MODEL AND CHUNK SCORE FOR ESTIMATING POPULARITY DISTRIBUTION IN LIVE STREAMING

In this section, we propose a new dynamic request model considering STV characteristics of live streaming based on Equation (1) discussed in Section II-B. The model estimates the request rate of a group of live streaming chunks for the next period, based on requests arrived beforehand. In this model, we define the time period as τ and the discrete time instance t mapped to continuous time interval $[t\tau, (t+1)\tau]$. In the end, τ is used as a chunk update period of our proposed algorithm. In order to estimate the request rate, we need to analyze the STV characteristics discussed in Section II. We observed that end-users watching chunk $a_{i,j}$ are more likely to continually request the subsequent chunks while the certain percentage of them are possible to leave live channel i and do not request the subsequent chunks [19]. In order to reflect this possibility, we use exponential distribution for $\lambda_j(t)$ with window size L representing the continuity of end-users and aging factor α representing the departure rate of end-users for the subsequent chunks. The exponential shape is determined based on the user behavior model on viewing duration in a mobile live streaming system [19]. We showed

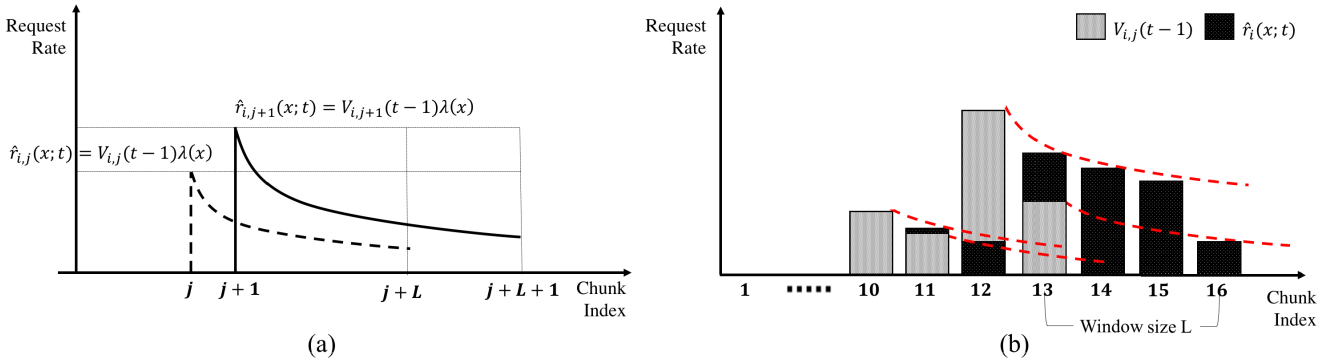


FIGURE 5. A model for popularity distribution estimation in live streaming : (a) A new operation model for STV characteristics, Equation (2), along to chunk index in a live channel, (b) Sketch of the procedure to estimate chunk popularity distribution for a live channel. The grey box represents the chunk popularity distribution based on requests arrived at time instance $t-1$ and the black box represents the chunk popularity distribution estimated Equation (5) for time instance t.

that our proposed STV request model based on this exponential shape estimate the popularity of chunks in a twitch.tv live channel with high accuracy in section IV-B. Thus, the STV request model can be formulated by Equation (2).

$$\hat{r}_{i,j}(x;t) = V_{i,j}(t-1)\lambda(x),$$
$$where\ \lambda(x) = \begin{cases} \alpha^{x-j} & \text{if } j \le x \le j+L \\ 0 & \text{if } otherwise \end{cases} \quad (2)$$

$\hat{r}_{i,j}(x;t)$ is the estimated request rate of a chunk $a_{i,x}$ considering only a single chunk $a_{i,j}$ at time instance t. $V_{i,j}(t-1)$ is the volume which means the observed request rate of chunk $a_{i,j}$ at time instance $t-1$, which is the most recent time instance. The window size L is the number of subsequent chunks to be predicted, defined as $[\tau/t_{ch}]$ where t_{ch} is the chunk duration. The shape $\lambda(x)$ is the function starting from 1 and decreasing exponentially by α over the window size L. Figure 5(a) represents the STV request model for the estimated request rates of chunk $a_{i,j}$ and $a_{i,j+1}$ at time instance t, where $V_{i,j}(t-1)$ and $V_{i,j+1}(t-1)$ are the observed request rates of chunk $a_{i,j}$ and $a_{i,j+1}$ at time instance $t-1$, respectively. In practice, since multiple chunks in multiple live channels are requested simultaneously at each time instance with live latency, their STV request models must be combined and organized to estimate the request rate for the next time instance. In order to resolve this issue, we propose a chunk score model heuristically to distinguish highly popular chunks in multiple live channels. The chunk score model estimates the popularity of chunks in multiple live channels for time instance t via the observation of requests at time instance $t-1$. The chunk score consists of the estimated channel popularity of live channel i, $\hat{p}(a_i;t)$, and the estimated chunk popularity of chunk $a_{i,j}$, $\hat{p}(a_{i,j}|a_i;t)$, at time instance t. Especially, the estimated chunk popularity is constructed based on the STV request model.

We assumed that the channel popularity $\hat{p}(a_i;t)$ follows the Zipf-like distribution. In fact, even in live streaming, the Zipf-like distribution is also used effectively for the popularity model of live channel i [20][2]. Let U_i denote the set of chunk requests arrived for live channel i during time instance $t-1$. Let $n(\cdot)$ denote the number of element in a set. Then, the estimated channel popularity of live channel i at time instance t can be defined as follows:

$$\hat{p}(a_i;t) \approx p(a_i;t-1) = \frac{n(U_i)}{\sum_k n(U_k)} \quad (3)$$

The chunk popularity $\hat{p}(a_{i,j}|a_i;t)$ is calculated based on the distribution of chunk requests arrived for live channel i during time instance $t-1$. We apply the STV request model to U_i and predicts the request rate of chunks at time instance t. Let $n(a_{i,j})$ be the number of requests arrived for chunk $a_{i,j}$ during instance $t-1$. Then, the request rate of chunk $a_{i,j}$ at time instance $t-1$ can be defined as follows:

$$V_{i,j}(t-1) = \frac{n(a_{i,j})}{n(U_i)}. \quad (4)$$

By applying the STV request model with the window size of $L = [\tau/t_{ch}]$, we can obtain the estimated request rate of chunk $a_{i,j+k}$ considering only a single chunk $a_{i,j}$ at time instance t, defined as follows:

$$\hat{r}_{i,j}(j+k;t) = \frac{n(a_{i,j})}{n(U_i)}\alpha^k,$$
$$where\ k \in \left\{1,\ldots,\left[\frac{\tau}{t_{ch}}\right]\right\}. \quad (5)$$

There are several estimated request rates of the chunks requested for live channel i at time instance $t-1$ simultaneously with live latency. The estimated request rates of the chunks may be overlapped each other. In order to combine and organize them, we take the maximum value of the estimated request rates on each chunk $a_{i,x}$, defined as follows:

$$\hat{r}_i(x;t) = \max(\alpha^k \cdot \frac{n(a_{i,x-k})}{n(U_i)}),$$
$$where\ k \in \left\{1,\ldots,\left[\frac{\tau}{t_{ch}}\right]\right\}. \quad (6)$$

[2]In live streaming service, Pires, K.,& Simon, G. concluded that the popularity of live channels follows a zipf-like distribution

Finally, by normalizing the estimated request rate of chunk $a_{i,x}$, we can obtain the estimated chunk popularity of chunk $a_{i,x}$ at time instance t, defined as follows.

$$\hat{p}(a_{i,x}|a_i;t) = \frac{\hat{r}_i(x;t)}{\sum_j \hat{r}_i(j;t)} \tag{7}$$

Figure 3 (b) depicts the procedure of estimating the request rate until just before normalization. The request rate is predicted by applying the STV request model to the observed request rate $V_{i,j}(t-1)$ at time instance $t-1$, represented as a grey box. We obtain the estimated request rate $\hat{r}_i(x;t)$ by taking the maximum value when overlapped, represented as a black box. Finally, we can figure out the chunk score for time instance t by multiplying the estimated channel popularity and the estimated chunk popularity.

Definition 1: Chunk Score of Chunk $a_{i,j}$ at time instance t We define it as a multiplication of the estimated channel popularity at time instance t, denoted by $\hat{p}(a_i;t)$ and the estimated chunk popularity at time instance t, denoted by $\hat{p}(a_{i,j}|a_i;t)$.

$$p(a_{i,j};t) = \hat{p}(a_i;t) \cdot \hat{p}(a_{i,j}|a_i;t) \tag{8}$$

In live streaming, each request makes a joint decision on live channel and chunk. The chunk popularity is defined as the conditional probability that chunk $a_{i,j}$ is requested by end-users watching live channel i. Therefore, by multiplying the channel popularity defined as the probability that live channel i is requested by end-users, we can calculate the probability that chunk $a_{i,j}$ is requested by end-users, which enables comparing the popularities of chunks in different channels and distinguishing highly popular chunks in multiple live channels.

C. COLLABORATIVE CHUNK CACHING SCHEME BASED ON STV REQUEST MODEL FOR MULTIPLE LIVE CHANNELS

In this section, we formulate the chunk caching problem for a single MEC server in an integer linear programming (ILP) form. Its objective is to determine an optimal caching strategy $C(t) \subset A$ at time instance t to maximize the instantaneous cache byte hit ratio, utilizing the cache of C_{size}. The problem is formulated as follows:

$$\begin{aligned} \text{maximize} \quad & \sum_{a_{i,j} \in C(t)} b(a_{i,j}) p(a_{i,j};t) \\ \text{subject to} \quad & \sum_{a_{i,j} \in C(t)} b(a_{i,j}) \le C_{size}. \end{aligned} \tag{9}$$

In order to resolve the problem, we propose a new chunk caching scheme to cache highly popular chunks based on chunk score until the cache of C_{size} become full. We prove the problem can be solved in near-optimal with the proposed scheme by Corollary 1.

Corollary 1: If $b(a_{i,j}) \le \epsilon \cdot C_{size}$ for all i, j, the proposed strategy gives $(1-\epsilon)$ approximation to the optimal solution

Proof: Equation (9) can be interpreted as a 0-1 Knapsack problem with item size $s_k = b(a_{i,j})$, and item price $pr_k = p(a_{i,j};t) \cdot b(a_{i,j})$. In other words, the size of item k is the size of the chunk, the price of item k is the response time, and knapsack size is C_{size}. Equation (10) represents the modified problem in the form of 0-1 knapsack problem.

$$\begin{aligned} \text{maximize} \quad & \sum_k pr_k \cdot x_k \\ \text{subject to} \quad & \sum_k s_k \cdot x_k \le C_{size} \\ & x_k \in \{0,1\} \quad \textit{for all } k \end{aligned} \tag{10}$$

The Equation (10) is a form of ILP which is proven to be NP-complete. There is no known algorithm to solve the problem by both correct and fast in polynomial-time. Thus, we approach to solve the 0-1 knapsack problem in a greedy way by sorting items in decreasing order of $\frac{pr_k}{s_k}$ and caches high valued chunks preferentially. $\frac{pr_k}{s_k}$ is simplified to equation (11)

$$\frac{pr_k}{s_k} = p(a_{i,j};t) \tag{11}$$

Thus, the greedy solution is caching chunks with high chunk score $p(a_{i,j};t)$ with boundary condition $\sum_{a_{i,j} \in C(t)} b(a_{i,j}) < C_{size}$. When $b(a_{i,j}) \ll C_{size}$ for all i, j, it is proven to be close to optimal solution as following: Let OPT be the optimal solution and $\frac{pr_k}{s_k} \ge \frac{pr_K}{s_K}$ for all $1 \le k \le K$. Then, following equation holds:

$$\begin{aligned} pr_1 + pr_2 + \cdots + pr_K &\ge (s_1 + s_2 + \cdots + s_K) \cdot \frac{pr_K}{s_K} \\ &\ge C_{size} \cdot \frac{pr_K}{s_K} \\ pr_K &\le (pr_1 + \cdots + pr_K) \cdot \frac{s_K}{C_{size}} \\ &\le \epsilon(pr_1 + \cdots + pr_K) \end{aligned}$$

Re-arrange the equation for pr_K

$$pr_K \le \frac{\epsilon}{1-\epsilon}(pr_1 + \cdots + pr_{K-1})$$

Since OPT is the optimal solution,

$$OPT \le pr_1 + \cdots + pr_K \le \frac{1}{1-\epsilon}(pr_1 + \cdots + pr_{K-1}) \tag{12}$$

Which yields

$$(1-\epsilon)OPT \le pr_1 + \cdots + pr_{K-1} \tag{13}$$

Therefore, the solution is $(1-\epsilon)$ approximation to the optimal solution. □

Through Corollary 1, we found that near-optimal solution can be obtained by preferentially caching the chunk with the high cache byte hit ratio first.

In this paper, we assumed that multiple nearby MEC servers are in a collaborative relationship. The MEC servers can compensate for each other's insufficient caching chunks. If the requested chunks are not cached in the nearest MEC server, the chunks can be received from neighboring MEC servers through front-haul links as mentioned in Section III-A. They are responded with relative fast latency since the delay of front-haul links is faster than

that of back-haul links. Even in this case, end-users can suffer from additional delay d_m, so some chunks (especially, mostly popular chunk) should be replicated on multiple MEC servers in order to guarantee the acceptable average delay on end-users. Hence, in order to reflect this issue, we formulate the collaborative chunk caching problem for N nearby MEC servers in an ILP form. Its objective is to determine an optimal caching strategy $\{C_n(t)|n = 1, \ldots, N\}$ at time instance t to maximize the instantaneous cache byte hit rate with $\delta(a_{i,j})$, utilizing their caches of $\{C^n_{size}|n = 1, \ldots, N\}$. The problem is formulated as follows:

$$\begin{aligned} &\text{maximize} \sum_{a_{i,j}\in\{\cup C_n(t)\}} b(a_{i,j})p(a_{i,j};t)\delta(a_{i,j}) \\ &\text{subject to} \sum_{a_{i,j}\in C_n(t)} b(a_{i,j}) \le C^n_{size}\ \forall n \in \{1, \cdots, N\} \end{aligned} \quad (14)$$

$\delta(a_{i,j})$ term, described in Equation (17), is added to compensate $b(a_{i,j})p(a_{i,j};t)$ depending on how many $a_{i,j}$ are cached in multiple MEC servers totally, considering the delay of $a_{i,j}$ on end-users. We derive $\delta(a_{i,j})$ from the average delay on end-users, which enables to maximize the cache byte hit ratio and minimize the average delay on end-users concurrently for arbitrary chunk requests.

$\bar{d}(t)$ is defined as an average service delay at time t when a random request arrives. $p(a_{i,j};t)$ is the chunk score defined in Definition 1 which means the probability of chunk $a_{i,j}$ is requested. We assumed that $d \ll d_b$. The average response time of a single MEC server can be represented as:

$$\begin{aligned} \bar{d}(t) &= \sum_{a_{i,j}\in C(t)} b(a_{i,j})p(a_{i,j};t)d \\ &\quad + \sum_{a_{i,j}\notin C(t)} b(a_{i,j})p(a_{i,j};t)d_b \\ &= D_b - \sum_{a_{i,j}\in C(t)} b(a_{i,j})p(a_{i,j};t)(d_b - d), \\ \text{where } D_b &= \sum_{a_{i,j}\in A} b(a_{i,j})p(a_{i,j};t)d_b \end{aligned} \quad (15)$$

The average response time of N MEC servers can be represented as:

$$\begin{aligned} \bar{d}(t) = \frac{1}{N}\sum_{n=1}^{N}\Bigg(&\sum_{a_{i,j}\in C(t)} b(a_{i,j})p_n(a_{i,j};t)d \\ &+ \sum_{a_{i,j}\notin C(t)} b(a_{i,j})p_n(a_{i,j};t)d_b \\ &+ \sum_{a_{i,j}\in (C(t)-C_n(t))} b(a_{i,j})p_n(a_{i,j};t)kd_m\Bigg) \end{aligned} \quad (16)$$

Theorem 1: When N MEC servers are in a collaborative relationship and have similar chunk score each other, the average response time is minimized if chunks with high $p_n(a_{i,j};t)\cdot\delta(a_{i,j})$ value are cached, where $\delta(a_{i,j})$ is defined as:

$$\delta(a_{i,j}) = \begin{cases} d + \left(1 - \frac{m(a_{i,j})}{N}\right)kd_m & \text{if } m(a_{i,j}) > 0\ \&N\ > 1 \\ d_b & \text{if } otherwise \end{cases} \quad (17)$$

Proof: When we consider only a single MEC server ($N = 1$), $\delta(a_{i,j})$ is fixed to d_b. Therefore, the net average response time is minimized by Corollary 1. When we consider multiple MEC servers ($N > 1$) in a collaborative relationship, chunks can be shared from nearby MEC servers so that delay $kd_m + d$ occurs instead of d_b. The objective function is to minimize the average response time described in equation (15). Let $m(a_{i,j})$ be the number of MEC servers which have the chunk $a_{i,j}$ in their cache at time instance t. If $m(a_{i,j}) = 0$, there is no MEC server to share the chunks so that requests should be delivered through backhaul. Thus, the average response time can be written as:

$$\begin{aligned} \bar{d}(t) &= \frac{1}{N}\sum_{n=1}^{N} b(a_{i,j})p_n(a_{i,j};t)d_b \\ &= b(a_{i,j})p(a_{i,j};t)d_b, \\ \text{where } p(a_{i,j};t) &= \sum_{n=1}^{N} p_n(a_{i,j};t) \end{aligned} \quad (18)$$

If $m(a_{i,j}) > 0$, $m(a_{i,j})$ MEC servers will directly respond to the request since they have $a_{i,j}$ in the cache. Also, $N - m(a_{i,j})$ MEC servers will deliver $a_{i,j}$ via neighbor MEC servers. The average response time can be formulated as:

$$\begin{aligned} \bar{d}(t) = \frac{1}{N}\sum_{n=1}^{N} b(a_{i,j})\Big(&\sum_{a_{i,j}\in C_n} p_n(a_{i,j};t)d \\ &+ \sum_{a_{i,j}\notin C_n} p_n(a_{i,j};t)(d + kd_m)\Big) \end{aligned} \quad (19)$$

Assuming that each MEC server has the similar chunk score, $p(a_{i,j};t) \approx \sum_{a_{i,j}\in C_n(t)} p_n(a_{i,j};t) \approx \sum_{a_{i,j}\notin C_n(t)} p_n(a_{i,j};t)$ holds and we can simplify the equation (18) by:

$$\begin{aligned} \bar{d}(t) &\approx \frac{1}{N}b(a_{i,j})\Big(N\cdot p(a_{i,j};t)d + \big(N - m(a_{i,j})\big) \\ &\quad \cdot p(a_{i,j};t)kd_m\Big) \\ &= b(a_{i,j})p(a_{i,j};t)\Big(d + \Big(1 - \frac{m(a_{i,j})}{N}\Big)kd_m\Big) \end{aligned} \quad (20)$$

Therefore, the average response time can be simplified as:

$$\bar{d}(t) = \sum_{a_{i,j}\in A} b(a_{i,j})\cdot p(a_{i,j};t)\cdot\delta(a_{i,j}) \quad (21)$$

Based on Corollary 1, Equation (21) is minimized when chunks with high $p_n(a_{i,j};t)\cdot\delta(a_{i,j})$ value are chunked. □

Based on Theorem 1, we propose STV request model-based collaborative chunk caching scheme for multiple live channels in collaborative MEC servers. The proposed scheme consists of two procedures: *Chunk Caching* and *Popularity Update*. *Chunk Caching* procedure decides the caching strategy periodically, and Popularity Update procedure updates channel and chunk popularity when requests arrive.

Chunk Caching procedure determines which chunks to be newly cached and which chunks to be removed from cache with time period τ. At each time instance, it caches

Algorithm 1 Collaborative Chunk Caching Scheme Based on STV Request Model for Multiple Live Channels in Each MEC Server

Input:
$m(a_{i,j})$: The number of MEC servers which have the chunk $a_{i,j}$ in their cache currently,
$a_{i,j}$: Chunk to calculate popularity,
U_i : Set of accumulated chunk requests for the last period in live channel i
Output:
$p(a_{i,j}; t)$: Chunk score of $a_{i,j}$ at current time t
Function `PopularityUpdate` $(a_{i,j}, U_i)$ **:**
  $p(a_i; t) \leftarrow n(U_i)/\sum_k n(U_k)$
  if $p(a_i; t) < \eta$ **then**
    return $p(a_{i,j}; t) \leftarrow 0$
  end
  for $k = 1$ *to* $[\tau/t_{ch}]$ **do**
    $\hat{r}_i(x; t) = max(\alpha^k \cdot n(a_{i,x-k})/n(U_i))$
  end
  $\hat{p}(a_{i,j}|a_i; t) = \hat{r}_i(x; t)/\sum_j \hat{r}_i(j; t)$
  return $p(a_{i,j}; t) \leftarrow \hat{p}(a_i; t) \cdot \hat{p}(a_{i,j}|a_i; t) \cdot \delta(a_{i,j})$
End Function
Input:
$p(a_{i,j}; t), \forall a_{i,j} \in A$: All chunk scores at current time t
Output:
$C_n(t)$: A set of chunks to be cached for the next period
Function `ChunkCaching` (N) **:**
  $M \leftarrow 0$
  $C_n(t) \leftarrow \emptyset$
  sort A in descending order of chunk score
  for $c \in A$ **do**
    if $M + b(c) < C^n_{size}$ **then**
      $M \leftarrow M + b(c)$
      $C_n(t) \leftarrow C_n(t) \cup c$
    else
      break
    end
  end
  return $C_n(t)$
End Function

highly popular chunks by sorting chunk score in a descending order. In this procedure, chunks with channel popularity less than threshold η are not considered in order to utilize the storage capacity efficiently and reduce the complexity of the scheme. *Popularity Update* procedure updates channel and chunk popularity at each time instance time of time period τ based on Equation (3) and Equation (7) respectively. Also, it calculates the collaborative relationship term $\delta(a_{i,j})$, Equation (17), by receiving the information on cached chunks from other collaborative MEC servers. Then, it updates chunk score of each chunk in multiple live channels. As a result, the proposed scheme is described as Algorithm 1 in detail and the entire process of the proposed scheme in MECCS is shown in Figure 6.

IV. PERFORMANCE EVALUATION

In this section, we evaluated the cache performance of STV request model-based collaborative chunk caching for multiple live channels in collaborative MEC servers. Firstly, we evaluated the estimation performance of STV request model for the popularity of chunks in a live channel (see section IV-B). Secondly, we implemented MECCS for live DASH streaming and conducted real live streaming experiments in a single MEC server scenario (see section IV-C). Lastly, we also performed trace-driven experiments with real trace data of Twitch.tv [21] in multiple MEC servers scenario (see section IV-D). In these experiments, STV request model-based collaborative chunk caching scheme was compared with existing well-known caching schemes such as LRU[3] and MPV, which have been applied to MEC environment by previous works [3], for several metrics such as join time, buffering ratio, cache (bytes) hit ratio and back-haul bytes ratio.

A. EXPERIMENTAL ENVIRONMENT FOR PERFORMANCE EVALUATION

We implemented MECCS for live DASH streaming illustrated in Figure 6. In MECCS, MEC servers and CDN nodes are implemented as live DASH streaming servers and end-users are implemented as live DASH streaming clients. The implemented CDN node caches chunks of various bitrates transcoded from live streams of live channels in public network and send the requested chunks to MEC servers. The implemented MEC server analyzes chunk requests of end-users at each time instance t of time period τ while exchanging the information on chunks cached in other MEC servers through CCU to get $m(a_{i,j})$ and calculate $\delta(a_{i,j})$. Also, it calculates chunk scores of chunks in each live channel by estimating the channel popularity and the chunk popularity for the next time instance. Finally, it caches popular chunks of live channels from the CDN nodes through backhaul links or from other MEC servers through CCU and fronthaul links and discards chunks with low chunk score based on the updated chunk score for the next time instance. The chunk requests of end-users watching a live channel are satisfied by MEC servers through radio access network and fronthaul links or by CDN nodes through backhaul links. In order to evaluate the estimation performance of STV request model, Equation (2), for the popularity of chunks in a live channel, we measure its accuracy by calculating Hellinger distance between the estimated chunk popularity distribution and the real.

Hellinger distance [22]: represents the similarity between two probability distributions in probability. It is defined for two discrete probability distributions $P = (p_1, p_2, , p_k)$ and $Q = (q_1, q_2, , q_k))$ as:

$$H(P, Q) = \frac{1}{\sqrt{2}}\sqrt{\sum_{i=1}^{k}(\sqrt{p_i} - \sqrt{q_i})^2}$$
$$\text{subject to } 0 \leq H(P, Q) \leq 1 \quad (22)$$

[3]LRU (Least Recently Used) caches the contents requested recently

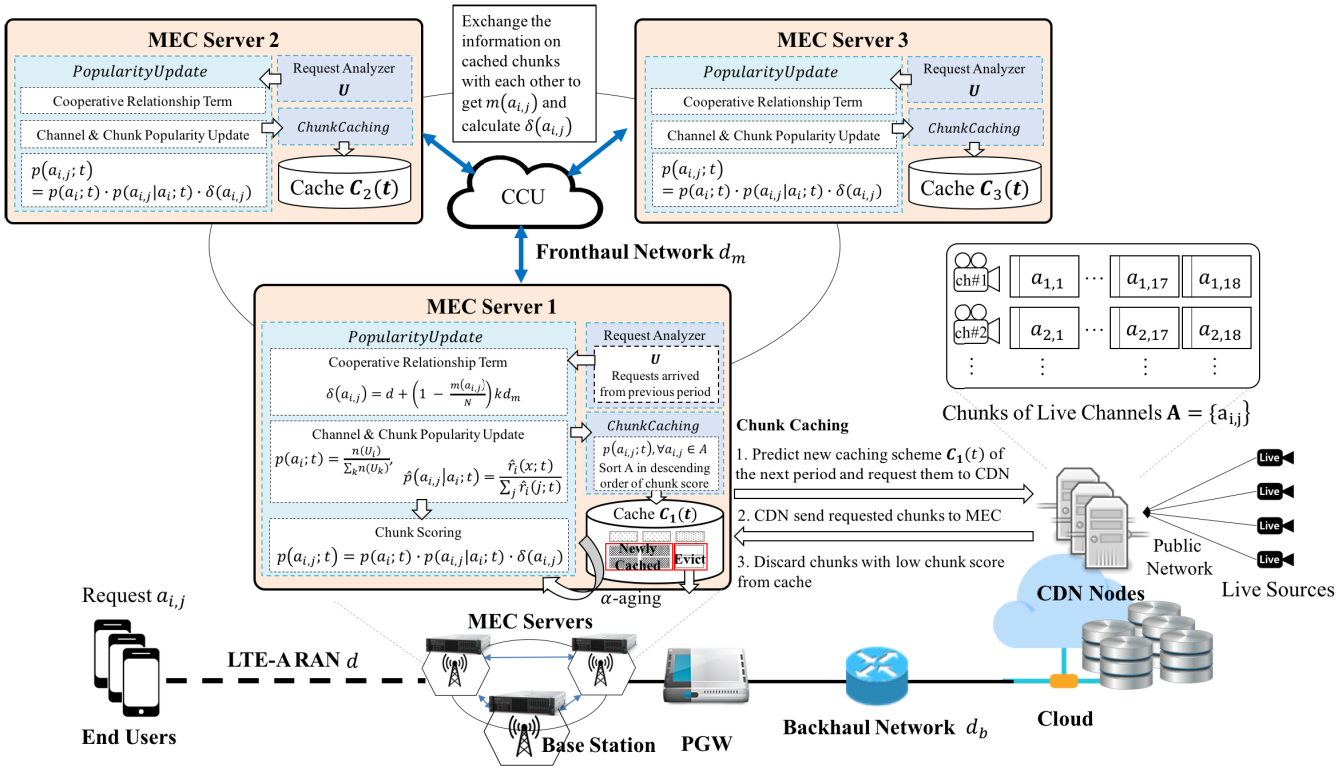


FIGURE 6. Implemented mobile edge computing caching system for live DASH streaming.

In order to evaluate the cache performance of the collaborative chunk caching scheme for maximizing cache hit and reducing the backhaul traffic, we measure cache (bytes) hit ratio and backhaul bytes ratio. We also measure the QoS of end-users on seamless playback without interruption by calculating joint time and buffering ratio.

Join time [23]: represents the elapsed time until the requested video actually starts to play. In DASH streaming, it means the time spent to satisfy play-start conditions (i.e. downloading MPD and several chunks to fill a video buffer).

Buffering ratio [23]: represents the percentage of the delayed time in video streaming, which is the time spent by re-buffering in DASH streaming.

Cache (bytes) hit ratio [3]: represents the percentage of requests (request bytes) satisfied by the cache in MEC servers.

Backhaul bytes ratio [24]: represents the percentage of request bytes generated from CDN nodes through backhaul links due to caching operations and cache miss.

For each time instance t, the number and bytes of chunk requests satisfied by the cache are measured for cache (bytes) hit ratio and the bytes of chunk requests generated from CDN nodes are measured for backhaul bytes ratio.

B. EXPERIMENTAL ANALYSIS OF STV REQUEST MODEL

In order to evaluate the estimation performance of STV request model for the popularity of chunks in a live channel, We conducted the experiment based on the trace data from Twitch.tv for the most popular live channel (i.e. hanryang1125) in Korea [21]. The trace data provides the number of viewers from 20:45 on August 26 2018 to 00:30 on Aug 27 2018 in the live channel. For simplicity, we assume the chunk duration is $t_{ch} = 5s$ and there are no variants (i.e. only the original video quality and bitrate is available) for video quality. We also assume chunk requests for the live channel are generated with the live latency randomly assigned following a uniform distribution in the ranges [20, 30] (s). The STV request model estimates the chunk popularity distribution for the next time instance at each time instance t of time period τ. For each time instance t, the Hellinger distance between the real chunk popularity distribution $p(t)$ and the chunk popularity distribution $\hat{p}(t)$ estimated by the STV request model is measured. Since the STV request model focuses only on the behavior of chunk requests in a given live channel, the channel popularity is not considered in this experiment. Figure 7 shows that the STV request model can guarantee the small distance (i.e. $H(p, \hat{p}) < 0.1$) in the small window size (i.e. $[\tau/t_{ch}] = 1$ or 2). Even in the bigger window size (i.e. $[\tau/t_{ch}] = 3$ or 4), it can guarantee the quite acceptable distance (i.e. $H(p, \hat{p}) < 0.2$). As the window size is bigger, the distance is bigger due to the short-term time-varying characteristics of chunk popularity. Since small window sizes are usual for MECCS to satisfy sequential chunk requests seamlessly in live streaming, the STV request model is fully applicable.

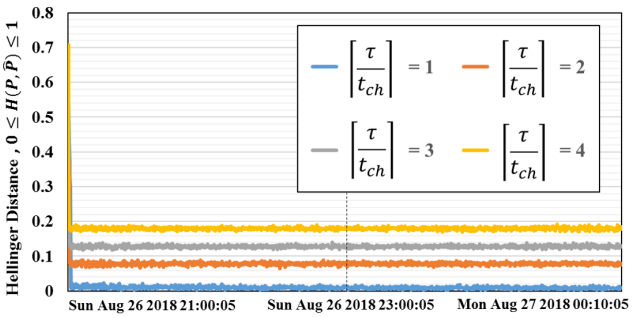


FIGURE 7. Characteristics analysis of Hellinger distance in STV request model: Hellinger distance between $p(t)$ and $\hat{p}(t)$.

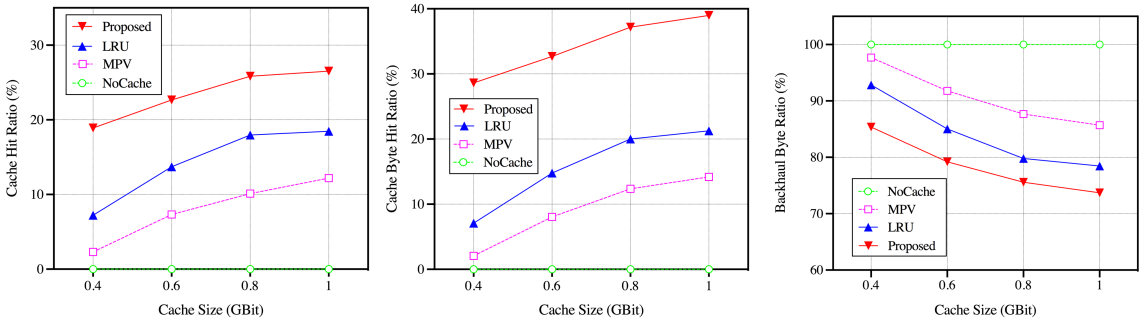
TABLE 1. Hardware specifications in experiment environment.

Machine	Specification
MEC Server	One Intel Xeon W3565@3.33GHz processor, 12GB memory, 1000GB disk
CDN node	Two Intel Xeon E5520 @ 2.39Ghz processor, 18GB memory, and 640GB disk
End-user	2VCPU, 2GB memory, and 40GB Disk (Openstack VM Instance)

C. EXPERIMENTAL ANALYSIS OF COLLABORATIVE CHUNK CACHING SCHEME WITH IMPLEMENTED MECCS FOR LIVE DASH STREAMING

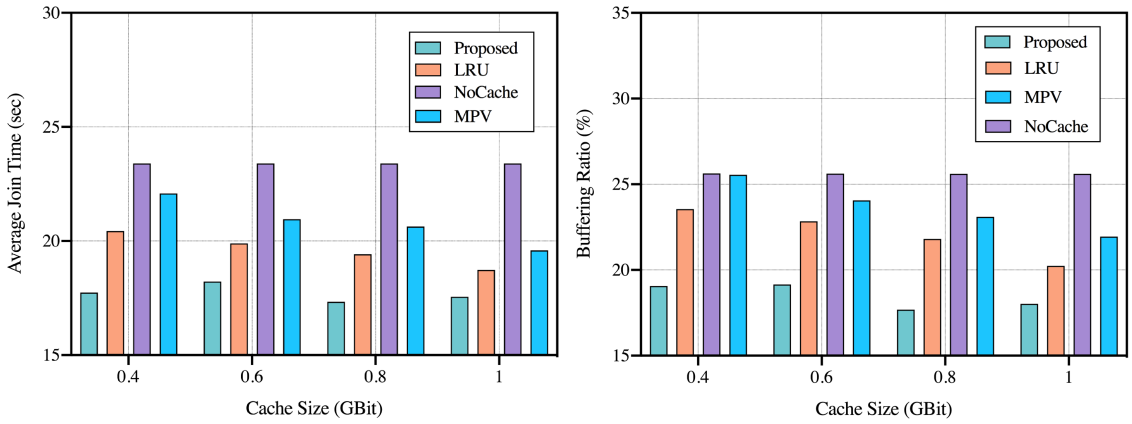
In this experiment, we evaluate the cache performance of STV request model-based collaborative chunk caching scheme for maximizing cache hit for satisfying the QoS of end-users on seamless playback without interruption and reducing the backhaul traffic in the implemented MECCS for live DASH streaming system consisting of a MEC server, a CDN node and end-users. We deploy them on the physical or virtual machines with specifications described in Table 1.

The live channels for the MEC server and CDN node are implemented by 4K videos of H.264 format by ffmpeg, each of which is encoded in 4K(17Mbps), 2K(8.5Mbps), 1080p(4.5Mbps), referring to the recommended settings in YouTube [25]. MP4Box generates Media Presentation Description (MPD) and chunks of each live channel for live DASH streaming [26]. We assume live channels follows a Zipf-like distribution (zipf parameter 1.2) [27], referring to the work of Pires, K. et al [20] concluding that the popularity of live channels follows a zipf-like distribution, and the number of them is set to 50. A Linux-based network tool, Traffic Control (TC), is used to emulate each link in the MECCS. The latency between the CDN node and the MEC server is set to $40ms \pm 4ms$ (uniform distribution). The latency and peak data rate between the MEC server and the end-users are set to $8ms \pm 2ms$ (uniform distribution) and 300Mbps (LTE-A spectral efficiency 3.75bps / Hz, Carrier Bandwidth 20Mhz, 4×4 MIMO Antenna), respectively. We assume all end-users share the corresponding radio link in a cell of the MEC server, so the actual data rate of each end-user differs depending on the number of connected end-users. Each end-user is implemented by a virtual machine instance, created by Openstack [28], including Google Shaka DASH player [29], which download/read MPD and play a sequence of chunks in a live DASH channel [30]. We assume that end-users arrive in a Poisson process with an average inter-arrival time of 5 sec and watch the requesting live channel for 120 sec. Besides, for the proposed scheme, the chunk duration t_{ch} and the window size $\lceil \tau / t_{ch} \rceil$ is set to 5 sec and 2, respectively (i.e. the chunk update period τ is set to 10s). The aging constant α and the threshold value for popularity η is set to 0.5 and 0.05, respectively. Figure 8 shows the results of cache performance in terms of various cache sizes for NoCache, LRU, MPV, and the proposed scheme. The cache sizes are set as [0.4, 0.6, 0.8, 1] (Gbit), which can store about 7.5%, 12%, 15% and 19% of chunks generated by all live channels. In Figure 8. (a) and (b), the proposed scheme shows more than 15% performance improvement on cache bytes hit ratio and more than 7% performance improvement on cache hit ratio in comparison of existing schemes for all cache sizes. In Figure 8. (c), the proposed scheme shows better performance on backhaul bytes ratio in comparison of existing schemes for all cache sizes. MPV shows the worst cache performance because it caches highly popular chunks using the estimated popularity based on chunk request history in multiple live channels. Since watching end-users for a live channel are more likely to request the recent chunks and then continually request the subsequent chunks according to STV characteristics of live streaming as mentioned in Section II, the subsequent chunks rarely requested yet is more likely to be requested by end-user than the outdated chunks already heavily requested in the previous period. However, MPV recognizes the outdated chunks, which are hardly requested anymore, as more popular chunks than the subsequent chunks since it caches preferentially the chunks heavily requested in history. This is the reason why MPV shows the worst cache performance. LRU shows the comparable performance because LRU caches preferentially the chunk recently requested reactively even if it works reactively after a cache miss. LRU can satisfy the chunk popularity distribution of each live channel in part indirectly according to STV characteristics of live streaming in which the recent chunks are being most likely to be requested by end-users. However, its simple heuristic policy does not support the optimized performance to maximize cache hit. The proposed scheme shows the best performance because it caches highly popular chunks of live channels optimally considering STV characteristics of live streaming. As a result, it increases cache hit and reduces backhaul traffic more in comparison of existing schemes by maximizing the reuse of chunks in the cache. In addition, for the same reason, Figure 9 shows the results of QoS performance in terms of various cache sizes for NoCache, LRU, MPV, and the proposed scheme. All caching schemes improve the QoS performance of NoCache considerably on average join time and buffering ratio by caching and serving in a MEC server hosted close to end-users. The proposed scheme shows more than 2.3s performance improvement on average join time and more



(a) cache hit ratio versus cache size (b) cache byte hit ratio versus cache size (c) backhaul bytes ratio versus cache size

FIGURE 8. Cache performance of proposed chunk caching scheme in the implemented MECCS for live streaming.



(a) average join time of all users versus cache size (b) buffering ratio versus cache size

FIGURE 9. QoS performance of proposed chunk caching scheme in the implemented MECCS for live streaming.

than 2% performance improvement on buffering ratio in comparison of existing schemes for all cache sizes. Besides, the proposed scheme has the high complexity to execute in comparison of LRU. We measure the execution time of the proposed scheme. The result of average execution time is 1.18 ms, which is quite long in comparison of LRU whose execution time is 0.18 ms. However, it is short enough in comparison of a predetermined time interval τ which is in a unit of seconds. Therefore, this overhead is negligible.

D. EXPERIMENTAL ANALYSIS OF COLLABORATIVE CHUNK CACHING SCHEME WITH REAL TRACE DATA OF TWITCH.TV

In this experiment, we use the trace data from Twitch.tv for our large-scale experiment [21]. The data provides the duration of sessions[4] in each channel and the number of viewers in each session from 12:00 to 16:00 on October 7th. 2017 for 1280 sessions of 500 live channels in KOREA.[5] However, it does not provide detail information of live channels, such as available variants, and viewers, such as network condition, video quality, and location. Therefore, the deficient information is complemented by some assumptions. For live channels, we assume five variants for video quality are available in all live channels: 4K, 2K, 1080p, 720p, and 480p class whose bitrates are 17Mbps, 8.5Mbps, 4.6Mbps, 3.2Mbps, and 2.2Mbps respectively, referring to the recommended settings in YouTube and Twitch.tv [25], [31]. For viewers, we assume the viewers of each live channel choose one of the variants uniformly at random for their chunk requests. This assumption is the most difficult case to predict the popularity of available variants in a given chunk. The case that the specific variant is requested heavily in comparison of other variants in a given chunk can be resolved more easily. In addition, the chunk requests of viewers for each live channel are generated with the live latency randomly assigned following a uniform distribution in the ranges $[0, \zeta]$ (s) and are transferred to one of MEC servers uniformly at random in a small local area. We also assume there are only the two available chunks, which have not been requested yet, at any time in CDN for each live channel, referring to Akamai systems [32].

In this experiment, the MECCS for live DASH streaming consists of 3 MEC servers hierarchically connected by CCU within one hop ($k = 1$). We assume that the backhaul/fronthaul links and radio access capacity are sufficient to handle all the generated chunk requests. The latency transferring a unit-size file (i.e. 1 Gbit) from the MEC server to

[4]A session is defined as a live channel in online
[5]Top-500 ranked channels in KOREA are used

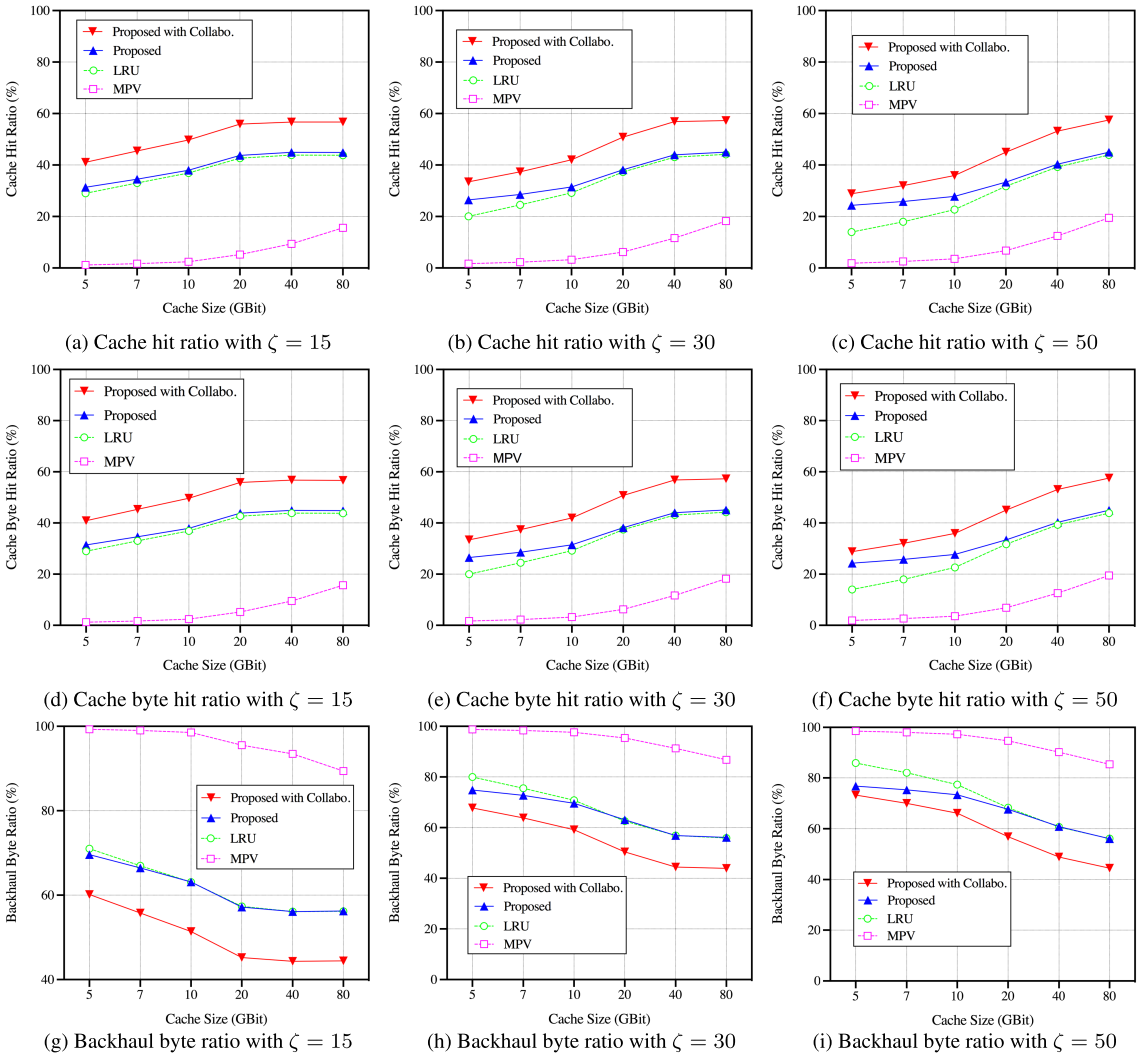
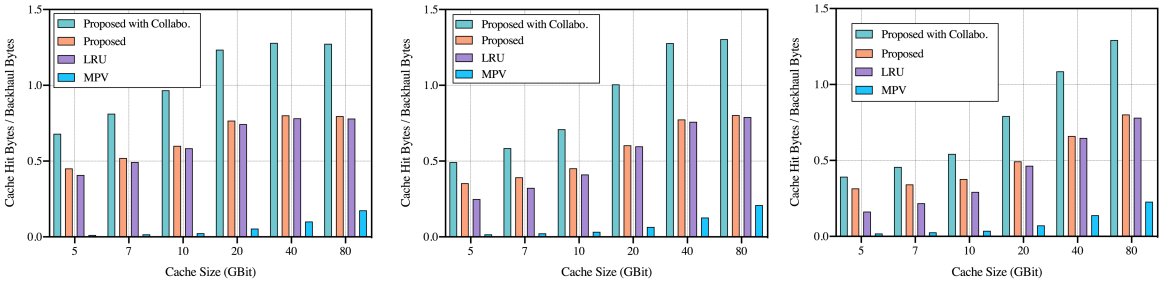


FIGURE 10. **Comparision results on cache performance in terms of cache hit ratio (%), cache byte hit ratio (%) and backhaul byte ratio (%) with $\zeta = 15, 30, 50$ over cache sizes.**

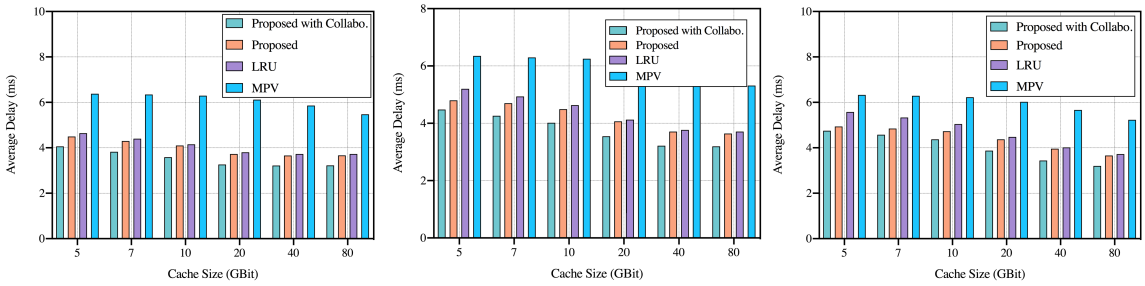
the end-user, between the MEC Servers, and from the CDN node to the MEC server are randomly assigned following a uniform distribution in the ranges [5, 10](ms), [20, 50](ms), and [150, 200](ms), respectively [34]. We use one hundredth of scale for the used trace data because it is too large for this MECCS. Parameter settings of the proposed scheme are same with Section IV-C. The average delay, which represents the average time spent to transfer the chunk from the MEC servers or the CDN to the requesting user, is measured instead of joint time and buffering ratio in Section IV-C.

Figure 10 shows the results of cache performance on cache (bytes) hit ratio and backhaul byte ratio with $\zeta = 15, 30, 50$ over various cache sizes for LRU, MPV, the proposed scheme and the proposed scheme with collaboration. The cache sizes are set as [5, 7, 10, 20, 40, 80] (Gbit). As shown in Figure 10(a)-10(f), the proposed scheme achieves better performance on cache (bytes) hit ratio over cache sizes compared to LRU entirely, as follows: $\{+2.4\%, +1.5\%, +1.1\%, +1.1\%, +1.1\%, 1\%\}$ with $\zeta = 15$, $\{+6.5\%, +4.1\%, +2.3\%, +0.8\%, +0.9\%, +0.9\%\}$ with $\zeta = 30$ and $\{+10.3\%, +7.8\%, +5.1\%, +1.6\%, +1\%, +1.1\%\}$ with $\zeta = 50$. Note that the performance of the proposed scheme get better as the live latency ζ increases and the cache size decreases. This indicates that the proposed scheme is more robust than LRU for the variations of live latency. In the small live latency $\zeta = 15$ or in the large cache sizes [20, 40, 80] (Gbit), the proposed scheme achieve very slight better performance than LRU. The reason is that, in this scale-down experiment environment, the cache sizes [20, 40, 80] are very sufficient for LRU to cache all of available highly popular chunks and the small live latency significantly reduces the range of highly popular chunks to be cached. For that reason, the performance of LRU get worse with decreasing the cache size or increasing the live latency. As shown in Figure 10(g)-10(i), the proposed scheme also achieve better performance in most cases but slightly worse performance in some cases on backhaul byte



(a) $\zeta = 15$ (b) $\zeta = 30$ (c) $\zeta = 50$

FIGURE 11. Comparision results on cache hit bytes per backhaul bytes with $\zeta = 15, 30, 50$ over cache sizes.



(a) $\zeta = 15$ (b) $\zeta = 30$ (c) $\zeta = 50$

FIGURE 12. Comparision results on average delay with $\zeta = 15, 30, 50$ over cache sizes.

ratio over cache sizes compared to LRU, as follows: $\{-1.4\%, -0.5\%, +0.02\%, -0.2\%, -0.01\%, +0.08\%\}$ with $\zeta = 15$, $\{-5.1\%, -2.8\%, -1.2\%, +0.5\%, +0.04\%, +0.2\}$ with $\zeta = 30$ and $\{-9.2\%, -6.8\%, -4\%, -0.6\%, +0.2\%, -0.1\%\}$ with $\zeta = 50$. However, as shown in Figure 11, the proposed scheme outperforms LRU on cache hit bytes per backhaul bytes for every case regardless of the live latency ζ and the cache size, even though LRU shows slightly better backhaul bytes ratio than the proposed scheme in some cases. Basically, the proposed scheme generates the additional backhaul bytes due to caching operations to download chunks in advance. Therefore, if most of the chunks cached in advance are not used enough, the proposed scheme can generate much bigger backhaul bytes than LRU. The results show the proposed scheme resolving these problems efficiently by maximizing the reuse of chunks in the cache. Meanwhile, the proposed scheme with collaboration achieve in average the higher performance on cache (bytes) hit ratio about 12.7%, 13.2%, 13.8% over $\zeta = 15, 30, 50$ in comparison of LRU. The proposed scheme with collaboration also achieve in average the lower performance on backhaul bytes ratio about 11.5%, 12%, 11.8% over $\zeta = 15, 30, 50$ in comparison of LRU. Note that the proposed scheme with collaboration outperforms LRU on cache (bytes) hit ratio and backhaul byte ratio for every case regardless of the live latency ζ and the cache size. Since LRU and MPV are not aware of the collaboration of MEC servers, they cannot benefit from the potential of increasing cache hit and utilizing the storage capacity of MEC servers more efficiently. Finally, Figure 12 shows the results of the average delay with $\zeta = 15, 30, 50$ over various cache sizes for LRU, MPV, the proposed scheme and the proposed scheme with collaboration. The proposed scheme reduce the average delay over cache sizes compared to LRU, as follows in percentage: {1% 3.2%, 2.3%, 1.2%, 2%, 1.8%, 1.6%} with $\zeta = 15$, {7.7%,4.8%, 3%, 1.5%, 1.5%, 1.8%} with $\zeta = 30$ and {11.5%, 9.1%, 6.3%, 2.4%, 1.4%, 1.7%} with $\zeta = 50$. The proposed scheme with collaboration also achieves in average the reduction of average delay in percentage about 13.4%, 13.8%, 14% over $\zeta = 15, 30, 50$ in comparison of LRU. Note that the proposed scheme and the proposed scheme with collaboration outperforms LRU on the average delay for every case regardless of the live latency ζ and the cache size.

V. CONCLUSION

In this paper, we addressed the limitation of static popularity estimation model when applied to live streaming following STV characteristics. In live streaming, they cannot achieve the acceptable performance on cache hit and increase the cache management cost including MEC storage cost and backhaul network traffic cost. In order to resolve these problems, we proposed STV request model to estimate the popularity distribution of live channels. We also proposed STV request model-based collaborative caching scheme to cache highly popular content in multiple live channels, utilizing the storage capacity of collaborative edge-servers efficiently. In order to evaluate the performance of STV request model-based collaborative caching scheme,

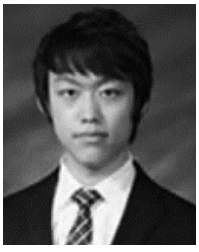
we implemented MECCS for live DASH streaming and conducted real live streaming experiments in laboratory scale. In addition, we also conducted trace-driven experiments with real trace data of Twitch.tv in large scale. In these experiments, the results showed that the proposed scheme outperforms the existing schemes on cache (bytes) hit ratio and average delay (or join time and buffering ratio) for every case regardless of the live latency ζ and the cache size, while guaranteeing a reasonable backhaul bytes ratio. Especially, the proposed scheme with collaboration improve the cache (bytes) hit ratio about $12 \sim 13\%$ while reducing the backhaul bytes ratio about $11 \sim 12\%$, compared to the existing schemes. It also reduces the average delay about $13 \sim 14\%$. As a result, we prove that the proposed scheme is able to maximize cache hit for satisfying the QoE of end-users and reduce the backhaul traffic for live streaming.

ACKNOWLEDGMENT

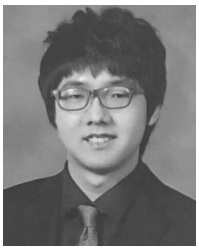
(Woo-Joong Kim and Kyung-No Joo contributed equally to this work.)

REFERENCES

[1] Cisco, "Visual networking index: Global mobile data traffic forecast update, 2015–2020," White Paper, Feb. 2016.
[2] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.
[3] H. Ahlehagh and S. Dey, "Video-aware scheduling and caching in the radio access network," *IEEE/ACM Trans. Netw.*, vol. 22, no. 5, pp. 1444–1462, Oct. 2014.
[4] C. Ge, N. Wang, G. Foster, and M. Wilson, "Toward QoE-assured 4K video-on-demand delivery through mobile edge virtualization with adaptive prefetching," *IEEE Trans. Multimedia*, vol. 19, no. 10, pp. 2222–2237, Oct. 2017.
[5] C. Ge, N. Wang, W. K. Chai, and H. Hellwagner, "QoE-assured 4K HTTP live streaming via transient segment holding at mobile edge," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 8, pp. 1816–1830, Aug. 2018.
[6] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—A key technology towards 5G," *ETSI White Paper*, vol. 11, no. 11, pp. 1–16, 2015.
[7] T. X. Tran and D. Pompili, "Octopus: A cooperative hierarchical caching strategy for radio access networks," in *Proc. IEEE Int. Conf. Mobile Ad Hoc Sensor Syst. (MASS)*, Brasilia, Brazil, Oct. 2016, pp. 154–162.
[8] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.
[9] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges," *Future Gener. Comput. Syst.*, vol. 78, pp. 680–698, Jan. 2018.
[10] A. Gharaibeh, A. Khreishah, B. Ji, and M. Ayyash, "A provably efficient online collaborative caching algorithm for multicell-coordinated systems," *IEEE Trans. Mobile Comput.*, vol. 15, no. 10, pp. 1863–1876, Aug. 2016.
[11] A. Mehta, W. Tärneberg, C. Klein, J. Tordsson, M. Kihl, and E. Elmroth, "How beneficial are intermediate layer data centers in mobile edge networks?" in *Proc. IEEE 1st Int. Workshops Found. Appl. Syst. (FAS* W)*, Sep. 2016, pp. 222–229.
[12] T. X. Tran, A. Hajisami, and D. Pompili, "Cooperative hierarchical caching in 5G cloud radio access networks," *IEEE Netw.*, vol. 31, no. 4, pp. 35–41, Jul./Aug. 2017.
[13] T. Kupka, P. Halvorsen, and C. Griwodz, "An evaluation of live adaptive HTTP segment streaming request strategies," in *Proc. IEEE 36th Conf. Local Comput. Netw.*, Oct. 2011, pp. 604–612.
[14] J. Kua, G. Armitage, and P. Branch, "A survey of rate adaptation techniques for dynamic adaptive streaming over HTTP," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1842–1866, 3rd Quart., 2017.
[15] C. Lee, E. Hwang, and D. Pyeon, "A popularity-aware prefetching scheme to support interactive P2P streaming," *IEEE Trans. Consum. Electron.*, vol. 58, no. 2, pp. 382–388, May 2012.
[16] L. Guo, S. Chen, Z. Xiao, and X. Zhang, "DISC: Dynamic interleaved segment caching for interactive streaming," in *Proc. 25th IEEE Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2005, pp. 763–772.
[17] M. Leconte, G. Paschos, L. Gkatzikis, M. Draief, S. Vassilaras, and S. Chouvardas, "Placing dynamic content in caches with small population," in *Proc. IEEE INFOCOM 35th Annu. Int. Conf. Comput. Commun.*, Apr. 2016, pp. 1–9.
[18] S. Wei and V. Swaminathan, "Low latency live video streaming over HTTP 2.0," in *Proc. ACM Netw. Oper. Syst. Support Digit. Audio Video Workshop*, 2014, p. 37.
[19] Z. Li, M. A. Kaafar, K. Salamatian, and G. Xie, "Characterizing and modeling user behavior in a large-scale mobile live streaming system," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 27, no. 12, pp. 2675–2686, Dec. 2017.
[20] K. Pires and G. Simon, "YouTube live and Twitch: A tour of user-generated live streaming systems," in *Proc. 6th ACM Multimedia Syst. Conf.*, 2015, pp. 225–230.
[21] *Twitch.tv*. Accessed: 2011. [Online]. Available: https://www.twitch.tv
[22] R. Beran, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang, "Minimum Hellinger distance estimates for parametric models," *Ann. Statist.*, vol. 5, no. 3, pp. 445–463, May 1977.
[23] F. Dobrian, Florin, "Understanding the impact of video quality on user engagement," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 362–373, 2011.
[24] P. Blasco and D. Gündüz, "Learning-based optimization of cache content in a small cell base station," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2014, pp. 1897–1903.
[25] *YouTube Live*. Accessed: 2008. [Online]. Available: https://www.youtube.com
[26] *MP4Box*. Accessed: 2000. [Online]. Available: https://gpac.wp.imt.fr/
[27] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Characteristics of YouTube network traffic at a campus network—Measurements, models, and implications," *Comput. Netw.*, vol. 53, no. 4, pp. 501–514, 2009.
[28] *OpenStack*. Accessed: 2010. [Online]. Available: https://www.openstack.org/
[29] *Shaka-Player*. Accessed: 2014. [Online]. Available: https://github.com/google/shaka-player
[30] *Delivering Live YouTube Content via DASH*. Accessed: 2008. [Online]. Available: https://developers.google.com/youtube/v3/live/guides/encoding-with-dash
[31] *Twitch TV Live Encoder Settings*. Accessed: 2011. [Online]. Available: https://stream.twitch.tv/encoding/
[32] *Akamai Developer*. Accessed: 1998. [Online]. Available: https://developer.akamai.com/learn/Optimization/Pre-fetching.html
[33] S. Traverso, M. Ahmed, M. Garetto, P. Giaccone, E. Leonardi, and S. Niccolini, "Temporal locality in today's content caching: Why it matters and how to model it," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 5, pp. 5–12, 2013.
[34] X. Li, X. Wang, S. Xiao, and V. C. M. Leung, "Delay performance analysis of cooperative cell caching in future mobile networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2015, pp. 5652–5657.



WOO-JOONG KIM received the M.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2014, where he is currently pursuing the Ph.D. degree in electrical engineering. His research interests include mobile cloud computing, and mobile edge computing and networks.



KYUNG-NO JOO received the M.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2014, where he is currently pursuing the Ph.D. degree in electrical engineering. His research interests include graph scheduling and mobile edge computing.

CHAN-HYUN YOUN (S'84–M'87–SM'19) received the B.Sc. and M.Sc. degrees in electronics engineering from Kyungpook National University, Daegu, South Korea, in 1981 and 1985, respectively, and the Ph.D. degree in electrical and communications engineering from Tohoku University, Japan, in 1994. Before joining the University, from 1986 to 1997, he was the Leader of the High-Speed Networking Team with KT Telecommunications Network Research Laboratories, where he had been involved in the research and developments of centralized switching maintenance systems, high-speed networking, and ATM networks. Since 2009, he has been a Professor with the School of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea. He was an Associate Vice-President of office of planning and budgets with KAIST, from 2013 to 2017. He is currently the Director of the Grid Middleware Research Center and the XAI Acceleration Technology Research Center with KAIST, where he is also developing core technologies that are in the areas of high performance computing, edge-cloud computing, AI acceleration systems, and others. He wrote a book on *Cloud Broker and Cloudlet for Workflow Scheduling* (Springer, 2017). He has served as a TPC Member for many international conferences. He was the General Chair of the 6th EAI International Conference on Cloud Computing (Cloud Comp 2015), KAIST, in 2015. He was also a Guest Editor of the IEEE WIRELESS COMMUNICATIONS, in 2016.