Multimed Tools Appl (2018) 77:1453–1474
DOI 10.1007/s11042-016-4308-z

Dependency- and similarity-aware caching for HTTP adaptive streaming

Cong Zhang[1] · Jiangchuan Liu[1,5] · Fei Chen[2] · Yong Cui[3] · Edith C.-H. Ngai[4] · Yuemin Hu[5]

Received: 6 May 2016 / Revised: 24 October 2016 / Accepted: 26 December 2016/
Published online: 13 January 2017

Abstract There has been significant interest in the use of HTTP adaptive streaming for live or on-demand video over the Internet in recent years. To mitigate the streaming transmission delay and reduce the networking overhead, an effective and critical approach is to utilize cache services between the origin servers and the heterogeneous clients. As the underlying protocol for web transactions, HTTP has great potentials to explore the resources within state-of-the-art CDNs for caching; yet distinct challenges arise in the HTTP adaptive streaming context. After examining a long-term and large-scale adaptive streaming

✉ Jiangchuan Liu
csljc@ieee.org

Cong Zhang
congz@cs.sfu.ca

Fei Chen
chenf@jiangnan.edu.cn

Yong Cui
cuiyong@tsinghua.edu.cn

Edith C.-H. Ngai
edith.ngai@it.uu.se

Yuemin Hu
ymhu@scau.edu.cn

1 School of Computing Science, Simon Fraser University, Burnaby, BC, Canada

2 School of Digital Media, Jiangnan University, Wuxi, People's Republic of China

3 Department of Computer Science, Tsinghua University, Beijing, People's Republic of China

4 Department of Information Technology, Uppsala University, Uppsala, Sweden

5 College of Natural Resources and Environment, South China Agricultural University, Guangzhou, People's Republic of China

dataset as well as statistical analysis, we demonstrate that the switching requests among the different qualities frequently emerge and constitute a significant portion in a per-day view. Consequently, they have substantially affected the performance of cache servers and Quality-of-Experience (QoE) of viewers. In this paper, we propose a novel cache model that captures the dependency among the segments in the cache server for adaptive HTTP streaming. Our work does not assume any specific selection algorithm on the client's side and hence can be easily incorporated into existing streaming cache systems. Its centralized nature is also well accommodated by the latest DASH specification. Moreover, we extend our work to the multi-server caching context and present a similarity-aware allocation mechanism to enhance the caching efficiency. The performance evaluation shows our dependency- and similarity-aware strategy can significantly improve the cache hit-ratio and QoE of HTTP streaming as compared to previous approaches.

Keywords Caching strategy · Segment dependency · Request similarity · Dynamic adaptive streaming over HTTP

1 Introduction

There has been significant interest in the use of HTTP adaptive streaming for live or on-demand video over the Internet in recent years. The Dynamic Adaptive Streaming over HTTP (DASH) has been adopted as an international standard, and the second version was released in July 2013. It allows a single video content to be encoded into multiple-resolution versions in advance and simultaneously adapts different types of devices with different screen sizes, networking conditions, and QoE requirements. There are three impressive novelties continuously making such adaptive HTTP streaming attractive: (1) the adaptability to dynamic networking fluctuation, (2) the flexibility in penetrating firewalls, and (3) the compatibility for the existing Internet resources that are generally optimized for web services. As such, it has been widely used in the real-world commercial systems. As a notable example and industrial flagship, Netflix delivers millions of movies, TV-series, and other shows to over 50 million global subscribers [25]. CBC also adopted HTTP adaptive streaming to broadcast the 2014 FIFA World Cup to millions of viewers during the month-long tournament in Brazil [3]. According to the recent report from YouTube [33], the second presidential debate of the United States attracted over 1.5 million peak viewers and over 124 million total views.

To mitigate the streaming transmission delay and reduce the networking overhead, an effective and critical approach is to utilize cache servers between the origin servers and the heterogeneous clients. For example, Netflix employs multiple commercial CDNs (Content Delivery Networks) to deliver video contents to end users. These CDNs, including $Akamai$, $LimeLight$, and $Level-3$ [1], cache video segments of multiple versions, making them closer to millions of clients from various geographical locations. From the perspectives of service availability and flexibility, caching mechanism can meet the Quality-of-Experience requirements of massive heterogeneous viewers in the practical large-scale HTTP adaptive streaming systems. As the underlying protocol for web transactions and adaptive streaming services, HTTP has great potentials to explore the resources within state-of-the-art CDNs for caching; yet distinct challenges arise in the HTTP adaptive streaming context.

First, the selection logic in DASH clients substantially impacts the utilization of cached segments, particularity with heterogeneous devices and the various quality levels [14]. In DASH, clients have to adaptively choose different bit-rate streaming segments to accommodate frequent networking changes. Our measurement result shows that the proportion of switching activities accounts for more than 55 % of a session, and there can be more than 10 different levels to be chosen for each video segment. Moreover, there exists strong dependency among the segments. As such, conventional caching strategies for a single quality level and full object do not work well.

Second, due to the dynamic networks, it is hard to identify user's downloading capacity consistently, classify them into appropriate quality group accurately and explore an optimal caching strategy to improve the performance of whole streaming system [12]. As such, previous works that focus on the traditional streaming system fail in handling current more dynamic networking condition of heterogeneous devices. Moreover, designed for conventional web objects, these solutions need to be substantially revised to accommodate the multi-quality nature of HTTP adaptive streaming.

In this paper, we closely examine the caching for HTTP adaptive streaming with strong segment dependency. Our contributions can be summarized as follows. First, we propose a novel cache model that captures the dependency among the segments in the cache server. Second, we extend the solution to the multi-server caching context and present a similarity-aware allocation approach to precisely capture the inter-relationship among the requests in HTTP adaptive streaming. Third, our work does not assume any specific selection algorithm on the client's side and hence can be easily incorporated into existing streaming cache services (e.g., CDNs). Its centralized nature is also well accommodated by the latest DASH specification [28]. We have evaluated our solution based on the real-world DASH dataset. The experimental results show that, comparing with the previous works, the hit-ratio and QoS of dependency- and similarity-aware cache can be significantly improved.

The rest of the paper is structured as follows. Section 2 introduces the background and motivation. Section 3 illustrates the formulation and solution of our dependency-aware cache model in HTTP adaptive streaming system. Section 4 formulates the improved strategy to optimize the caching performance in the multi-server caching context. The results of caching performance evaluation are offered in Section 5. We briefly review related work in Section 6 and finally conclude the paper in Section 7.

2 Background and motivation

Due to the rapid growth of social networks (e.g., Facebook) and the widespread deployment of high-speed communication networks (e.g., LTE), video streaming has become one of the most popular Internet services and attracted an increasing number of users as the content providers and consumers. To overcome the growing workloads, the architecture of video streaming is evolved from typical Client-Server model, Peer-to-Peer (P2P) network to current Cloud-based HTTP live streaming system [15]. The corresponding streaming protocols are also adopted from Realtime Transport Protocol (RTP), P2P Streaming Protocol to Hyper-Text Transport Protocol (HTTP). Particularly, in recent twenty years, there are a substantial amount of researches that focus on the optimization of streaming systems [5, 24] and the improvements of Quality-of-Service (QoS) [13, 20, 21] and Quality-of-Experience (QoE) [18, 34] from service providers' and video audiences' perspectives, respectively. Currently, lots of studies focus on the HTTP-based streaming system due to its widespread implementation in commercial video services, including Apple HTTP Live Streaming, Adobe HTTP Dynamic Streaming, and Microsoft Smooth Streaming, to name but a few. The recent dynamic adaptive HTTP streaming standard, DASH, encodes each video into continual segments of different alternatives with lower or higher bit-rates and resolutions.

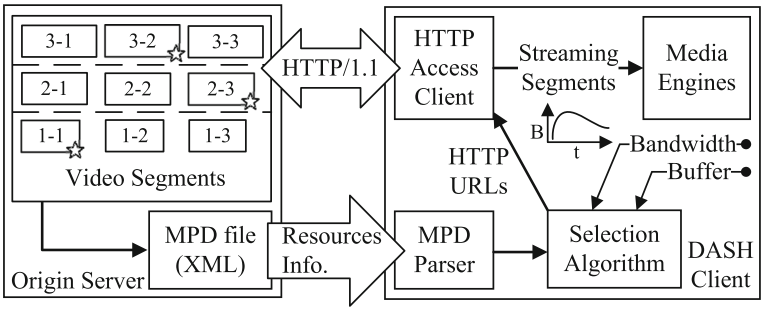


Fig. 1 A generic DASH architecture

Figure 1 describes a generic DASH architecture.[1] The basic data transfer and information exchange rely on HTTP protocol, including MPD (*Media Presentation Description*) file that is to be parsed by clients. In this example, the streaming content includes three video levels, which are divided into three segments, respectively. A DASH client first requests and analyzes MPD file, and then utilizes a selection algorithm to choose an appropriate video layer. After fully downloading a segment, the media engines decode and playback this video clip on the viewer's device. Through monitoring the dynamic networking and buffer status, the selection algorithm can adaptively choose proper streaming quality among the various layers and instruct the HTTP access component to fetch the corresponding segments. More details about the DASH standard and MPD files can be found in [6, 27].

Intermediate cache server is an indispensable part of the modern networked data-intensive systems [19], including HTTP streaming. Through well-designed caching strategies, it effectively accelerates the response time, reduces the long-haul bandwidth consumption from the original servers, and improves data availability, even with limited cache space. DASH naturally promotes that each streaming segment can be cached as a traditional web object. Meanwhile, the intrinsic characteristic of media streaming has to be considered in a meticulous manner. However, the proxy caching strategy in existing DASH system could overlook the streaming features and encounter with the potential issues, e.g., the bit-rate oscillation [14, 23].

To understand the characteristic of current DASH streaming system, we have closely examined the recent Neubot DASH dataset [2] from the segment-level granularity. This dataset is collected from more than a thousand Internet clients, using a DASH module built on top of Neubot,[2] an open source tool for the collection of network measurements. As shown in Fig. 2, we observe that the switching requests account for more than 55 % of the daily requests in January, 2014. This indicates that most of the DASH clients generally suffer unstable bandwidth, and the QoE thus fluctuates over time, which is undesirable for smooth and high-quality streaming [12]. Moreover, this type of request pattern also influences on the caching strategy in mobile streaming scenario [9]. As such, we consider the switching request as the dependency among the neighboring segments in different quality,

[1]For simplicity, we only show the video layers and segments. The general streaming components consist of video layers, audio tracks, subtitles of different languages, multiple DRM (Digital Rights Management) information and common encryption.

[2]http://www.neubot.org/

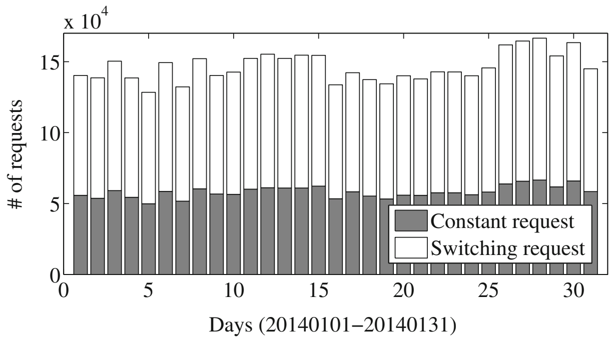


Fig. 2 Switching requests vs. constant requests

which is not be operated by most existing caching strategies in such a distributed streaming system. To enhance the user's experience, as well as improve the performance of cache management, we develop a novel dependency-aware caching strategy in next section.

3 Dependency-aware caching

In this section, we develop a dependency-aware cache model, which considers the dependency among the segments through our utility function.

3.1 Utility function

Since we concern about the switching requests in the presence of requests from heterogeneous clients, we start from a simple streaming scenario in which the origin server directly serves video streaming to clients. Table 1 gives important notation used in this paper. Each client session is given by i. We assume that the number of sessions is M and each session has N_i requests, so $i \in \{1, 2, \ldots, M\}$. Let $r_j^{(i)}$ denote the preferred bandwidth requirement

Table 1 Summary of major notations

Notation	Description
$r_j^{(i)}$	preferred bandwidth requirement of segment that is downloaded by the request j of session i
$d_j^{(i)}$	average downloading bandwidth of request j of session i
$w_j^{(i)}$	utility of request j of session i without any cache server
$u(r_j^{(i)})$	dependency of the request j of session i
$f(\theta_j^{(i)})$	loss function
S_{kl}	storage size of the segment l of level k
R_{kl}	set of requests of the segment l of level k
λ	a factor that reflects the scale of dependency
α	a factor that reflects the loss scale in loss function
μ	a factor that reflects the size of available cache storage

of the segment that is pulled by request j of session i, $j \in \{1, 2, \ldots, N_i\}$. Let $d_j^{(i)}$ be the average download bandwidth for request j of session i. With the assistance of local buffer, video clients can fetch the high-quality segments, even if the average bandwidth is lower than the requirement of higher bit-rate streaming. As such, $r_j^{(i)} > d_j^{(i)}$ means that the client utilizes the local buffer to acquire more appropriate quality level, while $r_j^{(i)} \leq d_j^{(i)}$ means that the client selects the proper quality level to adapt view speed for current video session. For the former, if the clients cannot fetch successive same quality segments as soon as possible, the user's QoE will be influenced largely. As such, we define $\omega_j^{(i)} = d_j^{(i)}/r_j^{(i)}$ and use it to represent the utility of a request between the server and a client. Considering the dependency of continual requests, we define a parameter λ to determine that how many neighboring requests should be exploited, $\lambda \in \mathbb{N}$. In current scenario, the dependency of request j of session i is denoted as $u\left(r_j^{(i)}\right)$, defined in the following equation.

$$u(r_j^{(i)}) = \begin{cases} \frac{1}{j}\sum_{k=1}^{j} \omega_k^{(i)} & \text{if } j \leq \lambda \\ \frac{1}{\lambda}\sum_{k=j-\lambda}^{j} \omega_k^{(i)} & \text{otherwise} \end{cases} \tag{1}$$

Considering the more general case, that is, an HTTP cache server schedules the cached segments and relays uncached segments from the origin server to the heterogeneous clients. Each streaming session starts from video users who request the MPD file of their favorite media by a DASH player, such as VLC,[3] and the player receives and interprets MPD file to parse the details of pulling video segments from the HTTP cache servers. Under this scenario, the cache server consumes extra time to fetch uncached contents from the origin server. However, the extra fetching time will impact the user's QoE and generate oscillations [23]. To reflect its influence on the utility value in (1), we use $f\left(\theta_j^{(i)}\right) = \left(\theta_j^{(i)}\right)^{\alpha}$ denote the loss function, where $\theta_j^{(i)} = t_j^{(i)}/\left(t_j^{(i)} + \hat{t}_j^{(i)}\right)$, $t_j^{(i)}$ means the downloading time from the cache server to clients, $\hat{t}_j^{(i)}$ means the downloading time from the origin server to the cache server, and α is a tunable parameter. $f(\theta)$ is a non-decreasing function,[4] $\theta \in (0, 1]$, and $f \in (0, 1]$. For the certain request, the dependency of request j of session i is defined in (2).

$$u(r_j^{(i)}) = \begin{cases} \frac{1}{j}\sum_{k=1}^{j} \omega_k^{(i)} f\left(\theta_k^{(i)}\right) & \text{if } j \leq \lambda \\ \frac{1}{\lambda}\sum_{k=j-\lambda}^{j} \omega_k^{(i)} f\left(\theta_k^{(i)}\right) & \text{otherwise} \end{cases} \tag{2}$$

There are three distinct features of the loss function $f(\theta)$. First, it represents the status of the requested segments in a cache server. For instance, if the requested segment has been cached, this request can obtain the whole utility ($\theta = 1, f(\theta) = 1$), otherwise, the uncached segment has to be fetched from the origin server and relayed to clients, which increases the transmission delay. Moreover, the utility will experience a certain decrement ($\theta < 1, f(\theta) < 1$). Second, it implies a potential replacement strategy that maximizes the total utility. For example, if $\hat{t} \ll t$, θ is very close to 1, which means that even if the requested segment is not cached, the total utility only encounters a quite small scaled decrease; thus, this segment can be evicted and replaced. Third, the parameter α reflects loss scale when θ decrease or increase largely. Intuitively, α is related to the video length, this is

[3]A popular, free and open source cross-platform multimedia player and framework. http://www.videolan.org/vlc/index.html

[4]In this paper, we omit the subscript and superscript when there is no confusion.

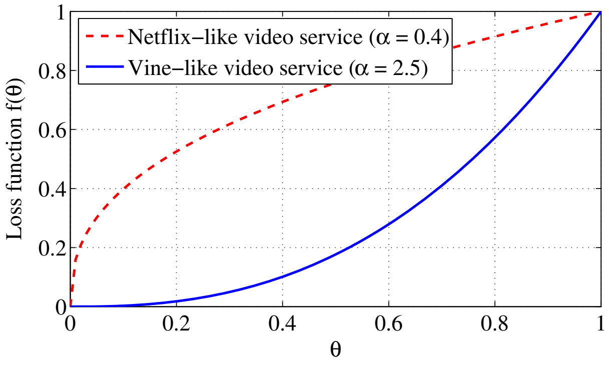


Fig. 3 Loss function f(θ)

because a small α means that the streaming service can endure a relative high relayed delay with the assistance of client's local buffer; while a large α means that a streaming service has higher requirement on minimizing the delay, e.g., for short video service, Vine.[5] We display the loss functions of these two video types with different parameter α in Fig. 3.

3.2 Problem formulation

We now formulate the objective function in our dependency-aware caching. Let K and L be the number of quality level and the number of segment sequence, respectively. We define the segment sets of video streaming $\mathbb{S}$ contains K rows and L columns segments. Let S_{kl} be the storage size of the k-th level l-th segment (k, l) in video streaming and R_{kl} be the requests set of segment (k, l), $k \in \{1, 2, \ldots, K\}$ and $l \in \{1, 2, \ldots, L\}$. Then, consider $[T_s, T_e]$, the streaming service available period and Δt, the update period of cache status, our aim is to maximize the total utility of all sessions during one time slot $[T - \Delta t, T)$ and select the proper segments to cache at time T, $\forall T \in [T_s + \Delta t, T_e]$. The objective function is defined as follows.

$$\max \sum_{k\in[1,K]} \sum_{l\in[1,L]} \sum_{r_j^{(i)}\in R_{kl}} u\left(r_j^{(i)}\right) \tag{3}$$

subjecting to the following storage availability constraint:

$$\sum_{k\in[1,K]} \sum_{l\in[1,L]} I_{kl} S_{kl} \leq \mu H \tag{4}$$

$$I_{kl} = \begin{cases} 1 & \text{if the } l\text{-th segment of the } k\text{-th level is cached} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

$$H = \sum_{k=1}^{K} \sum_{l=1}^{L} S_{kl} \tag{6}$$

where I_{kl} is an indicator function of whether a segment is cached in (5), H is the total data size of a video, and μ is a tunable parameter that controls the ratio between the cached segments and H.

[5]https://vine.co

(2) Bandwidth Availability Constraint:

$$\sum_{i \in [1,M]} B_i \leq \sigma B \tag{7}$$

where B_i is the bandwidth requirement of session s_i during current time slot, B is the total bandwidth of a cache server and the parameter σ can be controlled to adjust the available bandwidth usage in a time unit within the cache server must provide enough bandwidth usage to meet the demands of all video sessions. Therefore, for any session s_i in time slot $[T, T + \Delta t]$, the bandwidth availability constraint guarantees that the total maximum downloading bandwidth is lower than the ratio of the total bandwidth.

3.3 Equivalent problem

Given that the certain storage and bandwidth constraints will impact the value of utility function, we first discuss the following four situations.

(1) $b_j^{(i)} < r_j^{(i)}$, $f(\theta) = 1$: The requested segment is cached, and the downloading bandwidth is lower than the requirement. Thus, the client's local buffer fully supports the streaming playback. The selection algorithm has two choices for the next step: the first is to maintain current quality and wait until the better networking condition; the second is to switch the streaming layer to download the lower quality segments.

(2) $b_j^{(i)} < r_j^{(i)}$, $f(\theta) < 1$: The requested segment is uncached, and the downloading bandwidth is lower than the requirement. Comparing with the first situation, $f(\theta) < 1$ means that the request will generate a lower utility than the last situation, which is reasonable. Meantime, the cache server tends to cache this segment and acquire a higher utility if the storage size is enough.

(3) $b_j^{(i)} \geq r_j^{(i)}$, $f(\theta) = 1$: The requested segment is cached, and the downloading bandwidth is higher than the requirement, which means that the client has enough bandwidth to fetch this streaming segment, but it does not choose the higher quality layer. There are two reasons: (1) the selection algorithm needs more time to make sure that the higher bandwidth is consistent; (2) the client's device does not need or support higher quality streaming, e.g. due to resolution and hardware limitation.

(4) $b_j^{(i)} \geq r_j^{(i)}$, $f(\theta) < 1$: The requested segment is uncached, and the downloading bandwidth is higher than the requirement. The alternation between the third and the fourth situation likely produce oscillations phenomenon [14], Let $\hat{b}_j^{(i)}$ be the bandwidth between the origin server and the cache server. We define $\hat{r}^*$ as the requirement of the maximum support quality level at the bandwidth $\max\{r_j^{(i)}, \min\{b_j^{(i)}, \hat{b}_j^{(i)}\}\}$. The cache system control the client's bandwidth at a center level after a long-term consistent downloading, e.g. half of video length. For the cache server, it can get higher utility by caching this segment.

Based on the above discussions, we take care of the storage availability constraint. The bandwidth availability constraint will be explored as a load-balance condition in Section 4. We assume that the total bandwidth of the single cache server is enough to support all video sessions. Given that each segment has a total utility from a series of requests, each request will generate two distinct utilities for different cache states. Let $\hat{U}(r)$ and $\Delta U(r)$ denote the utility baseline and increment, respectively. That is, each request has a utility baseline $\hat{U}(r)$

when the requested segment is uncached, and has a utility improvement $\Delta U(r)$ when the requested segment is cached, as shown in (8).

$$U\left(r_j^{(i)}\right) = \hat{U}\left(r_j^{(i)}\right) + I_{kl}\Delta U\left(r_j^{(i)}\right) \tag{8}$$

For a certain request, $\hat{u}(r)$ can be calculated in advance. Our aim is to maximize the following (9). As such, the segment cache selection problem can be naturally related to the 0-1 knapsack problem: selecting a segment to cache corresponds to pick an item into a knapsack, and vice verse.

$$\max \sum_{k\in[1,K]} \sum_{l\in[1,L]} I_{kl} \underbrace{\sum_{r_j^{(i)}\in R_{kl}} \Delta U(r_j^{(i)})}_{\text{Profit } P_{kl}} \tag{9}$$

We define that each item has weight w and a profit p. For each segment, the storage requirement and utility increment can be denoted as (w, p) of each item, respectively. The aim is now to find a subset $\hat{\mathbb{S}} \in \mathbb{S}$ of maximum profit $\sum_{(k,l)\in\hat{S}} p_{(k,l)}$, subject to one constraint in which the total weight of the set should not exceed μH: $\sum_{(k,l)\in\hat{S}} w_{(k,l)} \le \mu H$. This segment cache selection problem is equivalent to the 0-1 knapsack problem, defined in the (10).

$$\max \sum_{(k,l)\in S} x_{(k,l)} p_{(k,l)} \tag{10}$$

subject to:

$$\sum_{(k,l)\in S} x_{(k,l)} w_{(k,l)} \le \mu H$$
$$x_{(k,l)} \in 0, 1, \forall (k,l) \in \mathbb{S} \tag{11}$$

Many works contribute to solve the 0-1 knapsack problem, such as dynamic programming, which adopts a recursive function and can solve our problem with time complexity $O(KL\mu H)$. But this is still time-consuming solution in a practical system. As such, we design an effective heuristic algorithm to address our caching problem. The corresponding algorithm is shown in Algorithm 1.

Algorithm 1 Dependency-aware caching algorithm

```
Input:
     The set of sessions $\{1, 2, \cdots, M\}$ in $[T - \Delta t, T]$,
1:  Set $S' \leftarrow 0$;
2:  for each session in $\{1, 2, \cdots, M\}$ do
3:      Calculate $\Delta u$ of each request in session $i$ by the (8);
4:  end for
5:  for each segment in $\mathbb{S}$ do
6:      Calculate the Profit $P_{kl}$ of segment $(k, l)$ by the (9);
7:  end for
8:  Sort $(k, l)$ by descendant order of $P_{kl}/S_{kl}$;
9:  for each segment $(k, l)$ in $\mathbb{S}$ do
10:     if $S' + S_{kl} \le \mu H$
11:         cache segment $(k, l)$;
12:         $S' \leftarrow S' + S_{kl}$
13:     end if
14: end for
```

4 Similarity-aware allocation

In HTTP adaptive streaming, clients retrieve the location of video segments from the streaming manifest file (e.g. MPD files in DASH). Selection algorithm then examines the responds time and chooses the proper cache server to start streaming playback. There are two potential issues: (1) inaccurate segment selection due to the decrease of local networking bandwidth [14], and (2) unpredictable QoE fluctuation and playback delay during the initial process of establishing the connection. These issues trigger the cache replacement in cache server frequently. Consequently, cache servers consume extra bandwidth to request uncached segments and confront with load-balance problem. Hence, it is necessary to design an optimal allocation strategy for cache servers and clients. A good allocation scheme should assign the proper cache server to a bunch of clients. In Section 3, our aim is to optimize the caching strategy in the single cache server. In this section, we extend our caching scheme to the multi-server caching scenario. The key problem is that the allocation of cache servers should guarantee the load-balance and benefit the caching performance. As such, we present a similarity-aware allocation strategy that not only promotes the caching performance, but also balances the traffic load between cache servers.

4.1 Similarity calculation

The earlier proposals classified video sessions by the download link bandwidth in RTP/RTCP/RTSP context, which was effective and reasonable due to the centralized streaming control and lower bit-rate requirement. While this classification scheme cannot sufficiently adapt current distributed clients, because of the decreasing bandwidth needs and unstable networking condition in high-speed movement and wireless circumstances. Hence, recent studies classify video sessions through the geographical information [11]. This method accommodates the mobile devices, but lacks the recognition for private domains. To overcome these issues, we propose a similarity-aware classification approach. Our aim is to explore the similar video sessions and assign them to the same cache servers with the load-balance constraint.

Thus, we first introduce that how to measure the similarity between two video sessions. We keep the previous definitions about the streaming session in Section 3.1. In addition, each session s_i has a $K \times L$ matrix A_i, in which the element $A_i(k,l)$ is equal to 1 if the l-th segment of the k-th quality level is requested. To measure the similarity, we introduce a new index, weighted overlap coefficient, $woc(s_i, s_j)$ to denote the similarity between s_i and s_j in (12).

$$woc(s_i, s_j) = \frac{W(B)}{\min\{W(A_i), W(A_j)\}} \tag{12}$$

where B is the $1 \times L$ matrix in which the $B(1,l) = 1$ if the l-th column segments in two sessions have the same quality, otherwise $B(1,l) = 1$. Thus, $B(1,l) = \sum_k (A_i(k,l) \wedge A_j(k,l))$, so $k \in \{1, 2, \cdots, K\}, l \in \{1, 2, \cdots, L\}$; the weight function W considers the continual requests in session s_i, thus, $W(A_i) = \sum_l w_l I_l(\sum_k A_i(k,l))$, where indicator function I_l and the weight value w_l are defined in (13) and (14), respectively.

$$I_l\left(\sum_k A_i(k,l)\right) = \begin{cases} 1 & \text{if } \sum_k A_i(k,l) \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

$$w_l = \begin{cases} I_l & \text{if } l = 1 \text{ or } w_{l-1} = 0 \\ I_l(w_{l-1} + 1) & \text{otherwise} \end{cases} \tag{14}$$

There are three specific features in our similarity calculation for the video sessions. First, $0 \leq woc(s_i, s_j) \leq 1$. The more identical requests, the higher similarity. Second, it carefully handles the fully fetching and partial fetching. As we know, 60 % of streaming viewers do not watch the whole video content [8], which means that some video sessions only request partial segments. For instance, if the set of all segments in the session s_2 is a subset of the set of all segments in the session s_1, $woc(s_1, s_2) = 1$. Third, the weight function enhances the effects of continual request matching. Considering four sessions $\{s_1, s_2, s_3, s_4\}$ in Fig. 4, the session s_1 has three same requests with other three sessions. Thereinto, the same requests in sessions s_2 and s_3 are continual. Different with Jaccard index (J) [31] and Overlap coefficient (O) [32], our weighted overlap coefficient will get the different similarity among (s_1, s_2), (s_1, s_3), and (s_1, s_4). This is closer to our streaming caching context.

4.2 Problem formulation

We now present the formulation of our similarity-aware allocation strategy. To improve the cache utilization, the sessions that have the closer similarity should be placed into a cache server. Let P denote the number of cache servers, the target is to assign all sessions into these cache servers and maintain the load-balance among them. This problem can be transformed into a clustering problem in which the similar sessions must be grouped into a cache server. At the meantime, the difference of the similarity in a session set is minimized. Let $d_w(s_i, s_j) = 1 - woc(s_i, s_j)$ denote the difference between two sessions. The problem can be formulated in (15).

$$\min \sum_{p=1}^{P} \sum_{\substack{s_i, s_j \in \mathbb{S}_p \\ s_i \neq s_j}} d_w(s_i, s_j) \tag{15}$$

subject to the following constraints:

(1) Bandwidth Availability Constraint:

$$\mathbb{B}_p \leq \sigma B \tag{16}$$

where $\mathbb{B}_p$ is the bandwidth usage of session set $\mathbb{S}_p$. Similar to the (7), $\mathbb{B}_p = \sum_{s_i \in \mathbb{S}_p} B_i$.

(2) Load-balance Constraint:

$$\mathbb{B}_p - \mathbb{B}_q < \beta P, \forall p, q \in \{1, 2, \cdots, P\}, p \neq q \tag{17}$$

where β is the load-balance threshold, βP is the bandwidth difference between two cache servers. This constraint achieves the load-balance among the cache servers.

Our problem can be solved by the $k - medoids$ algorithm. Instead of calculating the mean value of the objects in a cluster as a reference point in the general cluster algorithm $k - means$, $k - medoids$ algorithm randomly chooses the representative objects as the cluster centers and diminishes the sensitivity for outliers largely [10]. Thus, $k - medoids$ algorithm shows higher robustness and reliability. To accommodate $k - medoids$ algorithm,

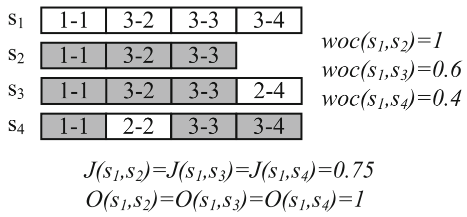


Fig. 4 The example of weighted overlap coefficient

some sessions are first selected as the cluster centers c_p randomly, and the new formulation is shown in (18). To fit our problem, we modify the $k - medoids$ algorithm in [4] and propose *Similarity-aware Allocation Algorithm* in Algorithm 2.

$$\min \sum_{p=1}^{P} \sum_{s_i \in \mathbb{S}_p} d_w(s_i, c_p) \tag{18}$$

Algorithm 2 Similarity-aware allocation algorithm

```
Input:
    The set of sessions $\{s_1, s_2, \cdots, s_M\}$ in $[T, T + \Delta t]$,
    The new session $s_{new}$
output:
    The session set of each cluster or the cache server location.
 1: Arbitrarily choose $P$ sessions in $S$ as the initial representative session $c_p$;
 2: Calculate the distances between each unrepresentative session and centers;
 3: Assign every session to its closest $c_p$ under the bandwidth availability and load-
    balance constraints;
 4: repeat
 5:     Randomly select a nonrepresentative session, $s_{ran}$;
 6:     Compute the total cost, $S = \sum_{s_i \in \mathbb{S}_p} d_w(s_i, s_{ran}) - \sum_{s_i \in \mathbb{S}_p} d_w(s_i, c_p)$, of
        swapping representative session, $c_p$, with $s_{ran}$;
 7: if $S < 0$ and $\mathbb{B}_p \leq \sigma B$ then
 8:     Swap $s_{ran}$ with $c_p$;
 9: end if
10: until no change;
11: if there is a new session $s_{new}$ then
12:     Find the most similar representative session $c_p$ with new session $s_{new}$ under the
        bandwidth availability and load-balance constraints;
13:     return the cache server location of $c_p$;
14: else
15:     return the session set of each cluster;
16: end if
```

5 Performance evaluation

In this section, we validate our analysis and examine the performance of the proposed algorithms for cache management.

5.1 Simulation setup

Based on the same dataset as we measured in Section 2, we simulate the request activities of user's video session. We randomly select 10 samples from the dataset in January 2014 and each sample consists of 30,000 requests. All simulation results are the mean value of these samples.

Current dataset is based on the basic streaming scenario, namely, several streaming servers and distributed clients. To accommodate our context, we assume that these streaming

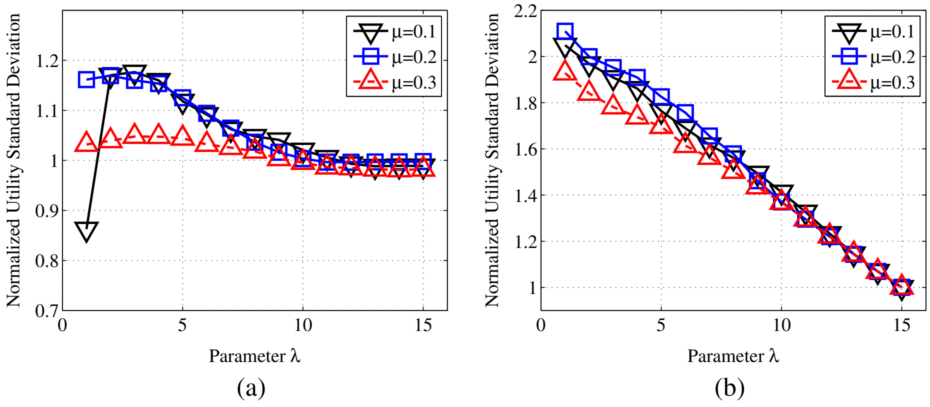


Fig. 5 The impact of λ

servers are cache servers that communicate with one origin server in the cloud-based environment. The throughput for origin server is shared by each cache server. Followed by the study in [29], we assume that the origin server is a small instance in Amazon EC2[6] and the throughput is set to $500Mbps$. We adopt the configuration of other parameters as follows: (1) Because the segment duration in the dataset is $2sec$, we set the service available basis and update period Δt to $2sec$. (2) Another tunable parameter α is related to the video length. The simulation dataset only contains one video, thus the α can be selected freely. Due to the short duration ($30sec$) of video, we set $\alpha = 2$. To evaluate the dependency-aware caching, we keep above settings and assume the single cache server scenario. The caching system collects session information and updates cache state of each segment by Algorithm 1 periodically.

In our caching strategy, parameter λ is critical for our model design. To select the proper λ, we first simulate the basic dependency-aware caching system to investigate the impact of λ. Let D be the total data transferring from the origin server to the cache server. The perfect concept is to utilize lower traffic cost and receive better hit-ratio. As such, we consider the hit-ratio per unit D as the criterion. Figure 5a shows how the criterion performs with different λ values at the certain cache size. For ease of comparison, the input data are first normalized by the corresponding minimum values. Note that, the number of video segments in simulation dataset restricts the range of λ is $\{1, 2, \cdots, 15\}$. The results show that λ can promote the best expectation when we set it to 3. Unless otherwise specified, $\lambda = 3$ is used in our following simulations. We also investigate why a higher λ does not mean the better caching performance. Figure 5b plots the normalized standard deviation of utility for each request in the video sessions. The higher λ means that the caching system considers more continual requests, which reduces the difference of utilities among the requests in a session. However, from the results in Fig. 5a, we can observe the proper choice for λ.

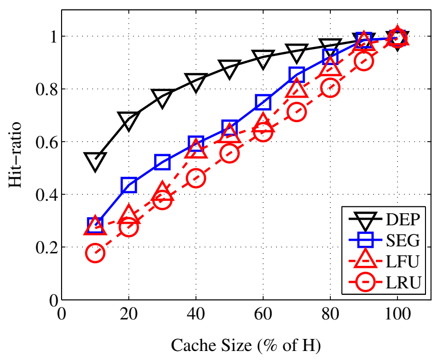
5.2 Efficiency in dependency-aware cache model

In our experiments, we investigate the caching performance in single cache server scenario. We compare our dependency-aware caching strategy (**DEP**) with the following three

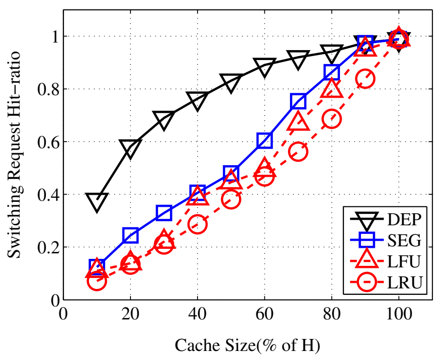
[6]http://aws.amazon.com/ec2/

cache algorithms that are widely used in practical web cache servers or streaming cache systems. (1) Least Recently Used (**LRU**), where the algorithm records the recent requested time of a segment, and discards the least recently requested segment first to release the cache space. (2) Least Frequently Used (**LFU**) where the system keeps track of the number of times a segment is requested in the cache server. If the cache space hardly stores the new segment, the system will purge the segment with the lowest request frequency. (3) Segment-based replacement (**SEG**) [26], where the replacement of cached segments follows the popularity-based and priority-based rules. When the cache space is full, the victim quality level first is recognized by the lowest popularity, and then the segments in this level will be flushed from back to front.

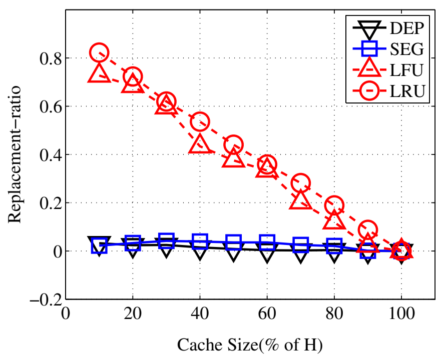
Figure 6a shows that the cache hit-ratio at different cache sizes. Comparing with other three approaches, when available cache space is lower than 50 % of the total streaming size, our DEP approach can guarantee 20 % improvement. The utilization of storage also can be increased substantially. Similar results are observed for the switching request hit-ratio as shown in Fig. 6b. We have mentioned that the switching request impacts on the stability of QoE, implying that the higher switching request hit-ratio can effectively achieve the smoothed quality alteration in client's end.



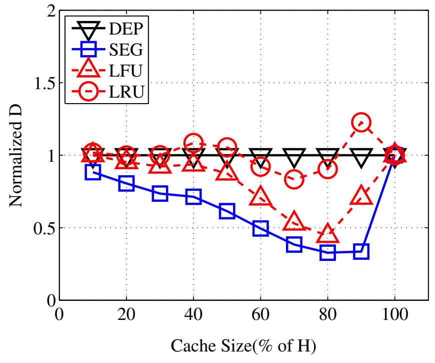
(a) Hit-ratio



(b) Switching request hit-ratio



(c) Replacement ratio



(d) Traffic

Fig. 6 Four criteria among LFU, LRU, Segment-based and Dependency-aware approaches

Figure 6c illustrates that the replacement-ratio versus cache size. Because SEG and DEP consider the specific characteristics and update cache state periodically, the number of cache replacement maintains a consistent level. In the meantime, all of them are far below LRU and LFU. Note that DEP persists the lowest replacement ratio.

Let D be the amount of data transferring from the origin HTTP adaptive streaming server to the cache server. Figure 6d performs the extra traffic consumption in the cache server. D is decreased by the growth of available cache size. But in this figure, to illustrate the distinction clearly, all values are normalized by the D of DEP strategy. Interestingly, SEG always utilizes the smallest D, which is determined by the pre-defined segment-based priority. From the results in this figure, we can observe that, to achieve the optimal cache storage strategy, our DEP strategy utilizes more traffic to download the uncached segments from the origin server and relay to the clients. This is because the cache server discards several rare and bulky segments. From Fig. 6c, we can know that DEP does not increase the number of connection between the origin server and the cache servers.

5.3 Efficiency of similarity-aware allocation model

We now consider the multi-server caching context and evaluate the efficiency of similarity-aware allocation strategy (SIM) in multi-servers HTTP cache system. For the sake of comparison, we also simulate the random allocation (RAN) in which the system randomly assign a cache server to the clients under the bandwidth and load-balance constraints. All cache servers apply DEP as the default caching strategy. We keep the same criteria in Section 5.2. The results are shown in Fig. 7. The Fig. 7a and b show the effectiveness in the cache hit-ratio and switching request hit-ratio. The maximum improvements in these two aspects are 4 % and 7 %, respectively, which largely increase the available storage size in the multi-server cache context. Figure 7c represents the different replacement-ratio. Comparing with DEP+RAN, our DEP+SIM witnesses a significant decrease when cache servers suffer lower available storage size. Figure 7d illustrates the effective of DEP+SIM in the practical caching context. Comparing Figs. 6d and 7d, DEP+SIM decrease the data transferring consumption, which indicates that our joint approach gains better performance in the multi-server caching system.

5.4 Performance of viewers' QoE

To measure the performance of viewers' QoE, we also conduct the practical deployment in the HTTP adaptive streaming scenario. We choose two metrics to identify viewers' QoE: startup latency and playback discontinuity. Startup latency reflects that how long the viewers have to wait after selecting a video until its initial playback, Playback discontinuity is the duration of discontinued playback, which is mainly caused by that the streaming segments fail to arrive at a user before its playback deadline. As such, the lower playback discontinuity represents that the users experience the better playback smoothness. The video stream is selected from a public DASH streaming dataset,[7] which is located at Klagenfurt. We implement four cache services with different caching algorithms using Python and deploy them

[7]Without loss of generality, we choose the first ten segments from 20-quality versions in the dataset of Big Buck Bunny: http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014/BigBuckBunny/4sec/

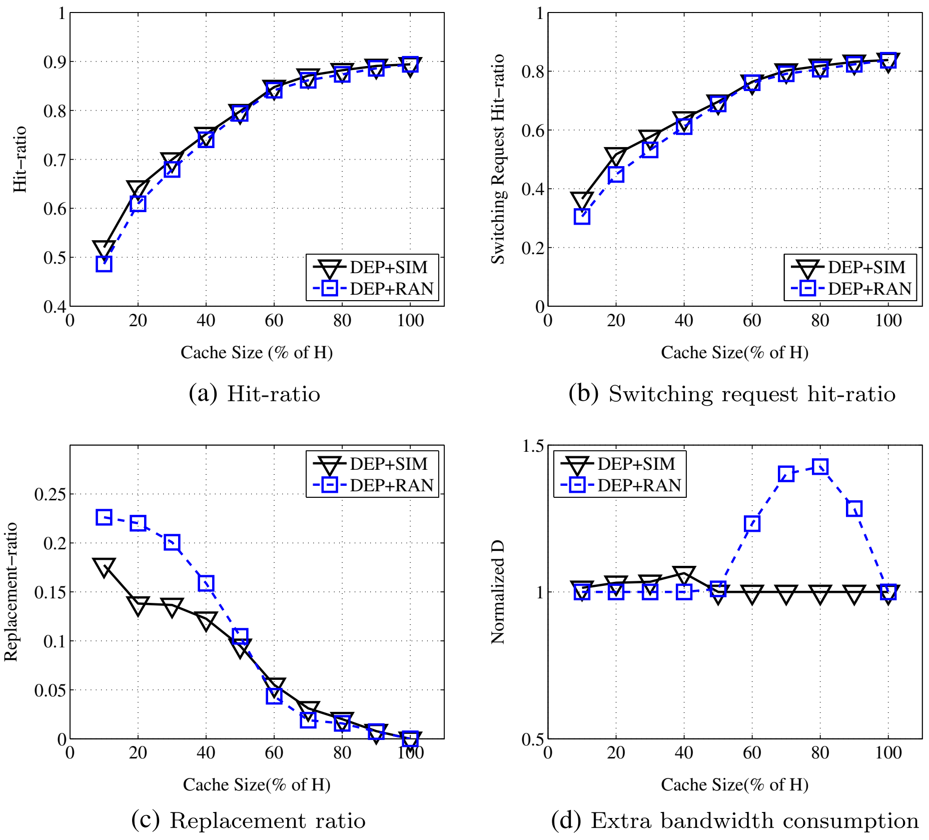


Fig. 7 Four criteria between similarity-aware and random approaches

on the m4.large instances[8] at Amazon EC2 Oregon area. These instances can receive the clients' requests from and pull/cache/forward corresponding streaming segments from the original server to clients. To investigate the dynamics of two metrics conveniently, we also implement a local client program that employs the basic DASH adaptive algorithm to pull the streaming segments in real networking condition. The client's device is DELL 7010 with Intel Core i7-3770 3.4GHz CPU, 16GB memory, and campus networks.

For closely examining startup latency and playback discontinuity, we employ ten clients to request streaming segments from the cloud-based server under different caching strategies and repeat this experiment five times for each caching algorithm. We therefore get the comparisons of viewers' QoE in Fig. 8. According to the adaptive algorithm in the client which prefers to select the initial segment with the lowest quality, LFU and LRU keep the highest priority for the initial segment with the lowest quality, but SEG could flush all segments in lowest quality. We therefore observe that the startup latency has a slight increase in DEP strategy compared with LFU and LRU in Fig. 8a, while SEG suffers the longest delay

[8]The instances are launched on Intel Xeon E5-2676 v3 2.4GHz Processor. Configuration: 2 vCPU; 8GB memory; 8G storage size; Enhanced Networking setting.

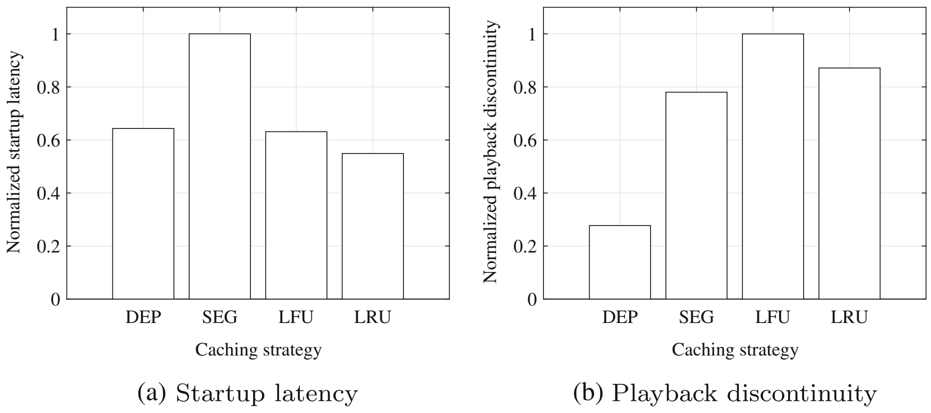


Fig. 8 Performance of viewers' QoE

at the initial step. Figure 8b shows that performance of playback discontinuity. We find that DEP has the lowest playback discontinuity due to the highest hit-ratio. That is, our method can provide a better playback smoothness during streaming playback.

6 Related work

Both caching and HTTP streaming have been extensively studied in the literature [30]. We now briefly list the existing works that are closely related to our study in this paper.

Comparing with the conventional web objects, media streaming objects demands significantly more storage size and higher bandwidth requirement. It is known that traditional web caching algorithms, such as Least Recently Used (LRU) and Least Frequently Used (LFU), do not work well for media streaming applications, especially in the era of explosive user-generated content (UGC). There have been extensive studies on media streaming caching, focusing on the cache replacement and management in proxy-assisted cache servers; examples include *sliding-interval caching*, *segment caching*, and *video staging* [16], which were effective and reasonable due to the centralized streaming control and lower bit-rate requirement. But the lack of a good adaptation for heterogamous devices impacts that they cannot sufficiently support high-definition videos and likely suffer the unstable networking condition in high-speed movement and wireless circumstances. Different from aforementioned studies, our work considers a promising HTTP adaptive streaming scenario, in which previous works may fail or suffer poor caching performance due to the networking fluctuation. Our work is also motivated by the mobile streaming and caching analysis in [9]. We, however, consider the fundamental feature among continual requests, as well as utilize the dependency of segment-level granularity to optimize the cache utilization and reduce the extra traffic overhead in HTTP adaptive streaming scenario. Besides, due to the nature of HTTP adaptive streaming, cache in CDNs is a general approach to handle the requests from massive viewers [7]. As such, our proposed method, as a caching module in the streaming delivery, also can be considered as the improvement of the caching part in existing CDNs.

In HTTP adaptive streaming, the selection of segments at the clients' end is another key issue in the literature. There have been a series of works seeking to offer better QoE for video receivers in the presence of dynamic network conditions and limited cache sizes. Liu et al. [22] demonstrated that switching back-and-forth between different video bit-rates leads to inferior QoS and proposed a better bandwidth prediction and stabilization algorithm to address this problem. STRA (Smoothed Throughput based Rate Adaptation) [17] seeks to a receiver-driven rate adaptation method, which is further extended in Zhou et al. [35] through a buffer-aware bit-rate switching framework based on control theory. Our work complements them by considering the impacts at the cache server side, which calls for a novel solution to the smoothed QoE for end-users.

7 Conclusion and further work

In this paper, we incrementally addressed the cache problem of HTTP adaptive streaming. Through the trace analysis of a long-term large-scale DASH dataset, we found that the switching requests constitute a significant portion in a per-day view. As a consequence, the cache servers have to request the uncached segment from origin server frequently, which not only decreases the performance of cache servers, but also interrupts the smoothed QoE of viewers. To solve this problem, we proposed a novel utility function to consider the dependency of continual requests, and explored the efficient dependency- and similarity-aware caching scheme in the cache server. Simulation results illustrated that the proposed DEP not only achieves impressive caching performance improvement, but also smooth the switching requests for heterogeneous devices.

We are currently implementing the private cloud-based streaming cache and delivery system to further evaluate proposed caching scheme in the practical environment. There are many unaddressed research issues worth further exploration. For example, what is the relationship between caching performance and power consumption? We are also interested in designing an energy-aware caching system with the DEP-assisted caching strategy and exploring the tradeoff between performance improvement and energy consumption in HTTP adaptive streaming context.

References

1. Adhikari V, Guo Y, Hao F, Varvello M, Hilt V, Steiner M, Zhang Z-L (2012) Unreeling Netflix: Understanding and improving multi-cdn movie delivery. In: Proceedings of the IEEE INFOCOM'12, Orlando, FL, USA
2. Basso S, Servetti A, Masala E, De Martin JC (2014) Measuring DASH streaming performance from the end users perspective using neubot. In: Proceedings of the ACM MMSys'14, New York, NY, USA
3. CBC Sports (2014) CBC FIFA World Cup watched by Canadians in record numbers. http://www.cbc.ca/1.2706905
4. Cheng X, Liu J (2011) Load-balanced migration of social media to content clouds. In: Proceedings of the ACM NOSSDAV'11, New York, NY, USA
5. Cheuk WK, Lun Daniel PK (2007) Throughput optimization for video streaming proxy servers based on video staging. Springer Multimedia Tools and Applications 35(3):311-333
6. DASH Industry Forum. Resource library, http://dashif.org/white-papers/
7. Dreier T (2016) Rio 2016: Online Olympic Viewing Is Skyrocketing, Reports Akamai. http://www.sportsvideo.org/2016/08/15/rio-2016-online-olympic-viewing-is-skyrocketing-reports-akamai, Auguest 2016

8. Erman J, Gerber A, Ramadrishnan KK, Sen S, Spatscheck O (2011) Over the top video: The gorilla in cellular networks. In: Proceedings of the ACM IMC'11, New York, NY, USA
9. Gouta A, Hong D, Kermarrec A-M, Lelouedec Y (2013) HTTP adaptive streaming in mobile networks: Characteristics and caching opportunities. In: Proceedings of the IEEE MASCOTS'13, San Francisco, CA, USA
10. Han J, Kamber M, Pei J (2011) Data Mining: Concepts and Techniques, 3rd edn. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA
11. Hao J, Zimmermann R, Ma H (2014) GTube: Geo-predictive video streaming over HTTP in mobile environments. In: Proceedings of the ACM MMSys'14, New York, NY, USA
12. Huang T-Y, Johari R, McKeown N, Trunnell M, Watson M (2014) A buffer-based approach to rate adaptation: Evidence from a large video streaming service. ACM SIGCOMM Comput Commun Rev 44(4):187-198
13. Kim HJ, Choi SG (2014) QoE assessment model for multimedia streaming services using QoS parameters. Springer Multimedia Tools and Applications 72(3):2163-2175
14. Lee DH, Dovrolis C, Begen AC (2014) Caching in HTTP adaptive streaming: Friend or foe? In: Proceedings of the ACM NOSSDAV'14, New York, NY, USA
15. Li B, Wang Z, Liu J, Zhu W (2013) Two Decades of Internet Video Streaming: A Retrospective View. ACM Trans Multimedia Comput Commun Appl 9(1s):33:1-20
16. Liu J, Xu J (2004) Proxy caching for media streaming over the internet. IEEE Commun Mag 42(8):88–94
17. Liu C, Bouazizi I, Gabbouj M (2012) Rate adaptation for adaptive HTTP streaming. In: Proceedings of the ACM MMSys'12, New York, NY, USA
18. Liu P-C, Leu J-S, Lee T-C, Chen T-H, Yee Y-S, Shih W-K (2012) WuKong: a practical video streaming service based on native BitTorrent and scalable video coding. Springer Multimedia Tools and Applications 60(1):47-68
19. Lu Y, Abdelzaher TF, Saxena A (2004) Design, Implementation, and Evaluation of Differentiated Caching Services. IEEE Transaction on Parallel Distributed Systems 15(5):440-452
20. Malamos AG, Varvarigou TA, Malamas EN (1999) Quality of service admission control for multimedia end-systems. In: Proceedings of the IMACS/IEEE CSCC'99, Athens, Greece
21. Malamos AG, Malamas EN, Varvarigou TA, Ahuja SR (2002) A model for availability of quality of service in distributed multimedia systems. Springer Multimedia Tools and Applications 16(3):207-230
22. Mok RKP, Luo X, Chan EWW, Chang RKC (2012) QDASH: a QoE-aware DASH system. In: Proceedings of the ACM MMSys'12, New York, NY, USA
23. Mueller C, Lederer S, Timmerer C (2012) A proxy effect analyis and fair adatpation algorithm for multiple competing Dynamic Adaptive Streaming over HTTP clients. In: Proceedings of the IEEE VCIP'12, San Diego, CA, USA
24. Nafaa A, Gourdin B, Murphy L (2012) A dependable multisource streaming system for peer-to-peer -based video on demand services provisioning. Springer Multimedia Tools and Applications 59(1):169-220
25. Netflix. Overview. http://ir.netflix.com/
26. Rejaie R, Yu H, Handley M, Estrin D (2000) Multimedia proxy caching mechanism for quality adaptive streaming applications in the Internet. In: Proceedings of the IEEE INFOCOM'10, Tel Aviv, Israel
27. Sodagar I (2011) The MPEG-DASH standard for multimedia streaming over the Internet. IEEE MultiMedia 18(4):62-67
28. Timmerer C (2013) Dynamic Adaptive Streaming over HTTP (DASH): Past, present, and future. http://www.streamingmediaglobal.com/Articles/ReadArticle.aspx?ArticleID=93275, November 2013
29. Wang G, Ng T (2010) The impact of virtualization on network performance of Amazon EC2 data center. In: Proceedings of the IEEE INFOCOM'10, San Diego, CA, USA
30. Wang Z, Sun L, Wu C, Zhu W, Yang S (2014) Joint online transcoding and geo-distributed delivery for dynamic adaptive streaming. In: Proceedings of the IEEE INFOCOM'14, Toronto, ON, Canada
31. Wikipedia. Jaccard index. http://en.wikipedia.org/wiki/Jaccard_index
32. Wikipedia. Overlap coefficient. http://en.wikipedia.org/wiki/Overlap_coefficient
33. YouTube Official Blog (2016) Second presidential debate-related videos rack up 40 percent more views than the first. https://youtube.googleblog.com/2016/10/second-presidential-debate-related.html
34. Yu F, Zhang Q, Zhu W, Zhang Y-Q (2003) QoS-adaptive proxy caching for multimedia streaming over the Internet. IEEE Transactions on Circuits and Systems for Video Technology 13(3):257-269
35. Zhou C, Zhang X, Huo L, Guo Z (2012) A control-theoretic approach to rate adaptation for dynamic HTTP streaming. In: Proceedings IEEE VCIP'12, San Diego, CA, Canada

Cong Zhang received the M.S. degree in information engineering from Zhengzhou University, Zhengzhou, China, in 2012, and is currently working toward the Ph.D. degree in computing science at Simon Fraser University, British Columbia, Canada. He is currently working with the Network Modeling Research Group, Simon Fraser University, Burnaby, BC, Canada. His research interests include multimedia communications, cloud computing, and crowdsourced live streaming.

Jiangchuan Liu received the B.Eng. degree (cum laude) from Tsinghua University, Beijing, China, in 1999, and the Ph.D. degree from The Hong Kong University of Science and Technology in 2003, both in computer science. He is a University Professor in the School of Computing Science, Simon Fraser University, British Columbia, Canada, and an NSERC E.W.R. Steacie Memorial Fellow. He is an EMC- Endowed Visiting Chair Professor of Tsinghua University, Beijing, China (2013-2016). From 2003 to 2004, he was an Assistant Professor at The Chinese University of Hong Kong. His research interests include multimedia systems and networks, cloud computing, social networking, online gaming, big data computing, wireless sensor networks, and peer-to-peer networks. He is a co-recipient of the inaugural Test of Time Paper Award of IEEE INFOCOM (2015), ACM SIGMM TOMCCAP Nicolas D. Georganas Best Paper Award (2013), ACM Multimedia Best Paper Award (2012), IEEE Globecom Best Paper Award (2011), and IEEE Communications Society Best Paper Award on Multimedia Communications (2009). His students received the Best Student Paper Award of IEEE/ACM IWQoS twice (2008 and 2012). He has served on the editorial boards of IEEE Transactions on Big Data, IEEE Transactions on Multimedia, IEEE Communications Surveys and Tutorials, IEEE Access, IEEE Internet of Things Journal, Computer Communications, and Wiley Wireless Communications and Mobile Computing. He is Steering Committee Chair of IEEE/ACM IWQoS (2015-2017) and TPC co-chair of IEEE IC2E'2017 and IEEE/ACM IWQoS'2014. He serves as Area Chair of IEEE INFOCOM, ACM Multimedia, IEEE ICME, and etc.

Fei Chen received the M.S. degree from the School of Electronics Engineering and Computer Science, Northeastern University, Shenyang, China, in 2009, and the Ph.D degree from the School of Computing Science, Simon Fraser University, Burnaby, BC, Canada, in 2014. He is currently an Assistant Professor with the School of Digital Media, Jiangnan University, Wuxi, China. His research interests include cloud computing, peer-to-peer networks, crowdsensing systems, and multimedia communications.

Yong Cui received the BE degree and the Ph.D. degree both on Computer Science and Engineering from Tsinghua University, China, respectively in 1999 and 2004. He is currently a full professor at the Computer Science Department in Tsinghua University. He published over 100 papers in the refereed conferences and journals with several Best Paper Awards. He coauthored 7 Internet standard documents (RFC) for his proposal on IPv6 technologies. His major research interests include mobile cloud computing and network architecture. He served or serves at the editorial boards on IEEE TPDS, IEEE TCC and IEEE Internet Computing. He is currently a co-chair of IETF IPv4/IPv6 Transition WG Softwire.

Edith C.-H. Ngai is currently an Associate Professor in Department of Information Technology, Uppsala University, Sweden. She received her PhD from The Chinese University of Hong Kong in 2007. She was a post-doc in Imperial College London, United Kingdom in 2007-2008. Her research interests include Internet-of-Things, mobile crowdsourcing, network security and privacy, smart cities and healthcare. She is a visiting researcher at Ericsson Research in 2015-2016. Edith is a VINNMER Fellow (2009) awarded by VINNOVA, Sweden. Her co-authored papers have received best paper runner-up awards in IEEE IWQoS 2010 and ACM/IEEE IPSN 2013. Edith has served as TPC members in leading networking conferences, including IEEE Infocom, IEEE ICDCS, IEEE ICC, IEEE Globecom, IEEE/ACM IWQoS, etc. She was a TPC cochair of Swedish National Computer Networking Workshop (SNCNW'12), QShine'14, SmartCity'15, ISSNIP'15, and iThings'16. She is a project leader of the Vinnova GreenIoT project (2014-2017) in Sweden. She has served as a guest editor in special issue of IEEE Internet-of-Things Journal, IEEE Transactions of Industrial Informatics, and Springer Mobile Networks and Applications (MONET). Edith is a senior member of IEEE and a member of ACM.

Yuemin Hu is a professor of land information technology at South China Agricultural University. He's also the director of the Guangdong Province Key Laboratory of Land use and consolidation, and the chairman of the Guangdong Mapping and Geoinformation Industry Technology Innovation Alliance. Yueming has a PhD in soil geography from Zhejiang University.