CUBIST: High-Quality 360-Degree Video Streaming Services via Tile-based Edge Caching and FoV-Adaptive Prefetching

Dongbiao He[*][†][¶], Jinlei Jiang[*][‡‡], Teng Ma[*][‖], Guangwen Yang[*], Cedric Westphal[§],
JJ Garcia-Luna-Aceves[‡], Shu-Tao Xia[¶]
[*]Beijing National Research Center for Information Science and Technology
Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
[†]Sangfor Inc., Shenzhen, China [‖]Alibaba Inc., Beijing, China
[‡‡]Research Institute of Tsinghua University in Shenzhen, Shenzhen 518057, China
[¶]Tsinghua Shenzhen International Graduate School, Tsinghua University, Shenzhen 518055, China
[§]Futurewei Technologies, Inc., 2330 Central Expressway, Santa Clara, CA 95050, USA
[‡]Computer Science and Engineering Department, University of California, Santa Cruz, Santa Cruz, CA 95064, USA

***Abstract*—360-degree video streaming, which is becoming more and more popular as the fast development of VR/AR applications nowadays due to the immersive viewing experience it can offer, poses enormous challenges to the current network infrastructure in terms of high bandwidth and low latency requirements. To address this problem and to ensure the QoE (quality of experience) of end-users, this paper presents CUBIST, a method and system for high-quality 360-degree video streaming in networks with cache nodes at the edge. To the best of our knowledge, it is the first tile-based edge caching solution that incorporates proactive tile prefetching and hierarchical cache organization into reactive caching to maximize the caching benefit while reducing the cost of 360-degree video streaming. Experimental results show that CUBIST can achieve a cache hit ratio of 87% and improve the effective video bitrate by 12.9% with most rate transitions being small when compared with the latest FoV-aware edge caching scheme.**

***Index Terms*—360-degree video, edge caching, FoV-adaptive prefetching, QoE**

I. Introduction

360-degree videos (a.k.a. immersive videos, panoramic videos, and spherical videos) are becoming more and more popular due to the fast development of virtual/augmented reality (VR/AR) applications in recent years. It is believed that the use of 360-degree video will go beyond gaming and entertainment into our daily lives, enhancing education, business and medicine [3], [28]. Also, nowadays, more and more 360-degree videos are delivered over the Internet. For example, YouTube, Facebook, and Vimeo have already provided 360-degree video service and still more are coming. Because the size of a 360-degree video can be up to 4-5 times larger than that of a traditional video with the same resolution [45], streaming 360-degree videos imposes stringent requirements on the network in terms of bandwidth and latency.

Two approaches have been suggested that try to address the bandwidth and latency requirements of 360-degree videos. The first one is called FoV-adaptive (a.k.a FoV-aware) streaming that tries to reduce network bandwidth consumption by, instead of transferring the whole video, only transferring parts of the video within users' field of view (FoV). Examples can be seen in [7], [9], [13], [16], [18], [37], [40], [47], [46], to name but just a few. The second one tries to reduce network access latency by in-network caching, which has been deployed in content delivery networks (CDNs) for many years and is becoming more and more popular with the rapid development of mobile edge computing. For example, the work in [25] discussed the impact of caching on adaptive video streaming. The work in [8] provided a model for video content providers to make better caching decision so as to improve cache performance. The work in [12], [28], [29] suggested caching videos at the edge so as to improve user's quality of experience (QoE).

In spite of the above work, due to so many factors involved in streaming and caching, the goal is far from reached to stream 360-degree videos in high-quality while with low bandwidth consumption and access latency. The challenges facing are as follows.

First, caching space is limited and there is a trade-off between caching efficiency and the end user's QoE. For example, caching the whole video may lead to poor cache utilization because parts of the cached video may never be viewed by any user. On the other hand, caching only parts of the video might lead to cache misses, causing frequent video quality switches and rebuffering [14] that harm end-user QoE. Also, cache organization matters, for there are multiple storage media available, each with its own character (e.g., capacity, performance and price).

Second, 360-degree video streaming happens in a dynamic environment. Both the available network bandwidth and the end-user behavior change from time to time, making it hard to determine the rate for video streaming and to do effective FoV-adaptive prefetching. Given the limited cache space and the

Corresponding author: Jinlei Jiang (jjlei@tsinghua.edu.cn)

978-1-6654-1681-8/21/$31.00 ©2021 IEEE
DOI 10.1109/ICWS53863.2021.00039

diverse preferences of different users, it becomes even harder to design a caching solution to reach the goal of 360-degree video streaming.

Finally, application-specific factors must be taken into account. For example, 360-degree video playing is usually controlled by the client via such techniques as DASH (Dynamic Adaptive Streaming over HTTP). Since each video segment can be stored in various resolutions, the cache node must select the most proper resolution to cache and the end-users must be served with the right resolution so as to avoid bitrate oscillations and sudden rate changes [25]. Moreover, this must be done transparently.

To deal with the above challenges, this paper presents CUBIST, a 360-degree video streaming solution in edge-enabled networks. CUBIST achieves its goal through cost-effective edge cache design on the basis of FoV-adaptive streaming. By cost-effective, we mean that CUBIST (a) reduces the network bandwidth consumption and access latency facing 360-degree videos streaming, and (b) organizes the cache with high cost performance ratio.

Our contributions in this paper are as follows:

- We present a series of mechanisms (§III) to estimate video popularity, to predict tile requirement, and to measure the network bandwidth and select the right bitrate. They not only form a solid foundation for CUBIST but also set up some reference to other related work.
- We present CUBIST (§IV), a method and system for high-quality 360-degree video streaming with low bandwidth consumption and access latency. CUBIST fulfills the task via a cost-effective edge caching solution on the basis of FoV-adaptive streaming. To the best of our knowledge, it is the first caching solution that incorporates both proactive tile pre-fetching and reactive video caching while considering hierarchical cache organization to improve QoE of 360-degree video streaming.
- We evaluate CUBIST against a wide range of benchmarks (§V). The experimental results show that CUBIST can achieve a cache hit ratio of 87% and improve the video quality, or more precisely the effective video transmission bitrate by 12.9% compared with the latest tile-based caching [28].

The rest of this paper is organized as follows. Section II presents some background of 360-degree video streaming and provides an overview of CUBIST. Section III and IV describes the key mechanisms behind CUBIST and the cache system design and implementation respectively. Section V discusses the methodology for performance evaluation and the evaluation results. Section VI lists some related works. The paper ends in Section VII with some conclusions.

II. Background and CUBIST Overview

A. 360-Degree Video Streaming

A 360-degree video is usually encoded with tile partitions [11] in a time-space way. In the time domain, a 360-degree video consists of multiple $partitions$ (or $segments$) and each of them again consists of many $frames$. In the space domain, each $frame$ of a 360-degree video consists of many $tiles$, each of which records a scene at a certain direction.

A $viewport$ describes the $gaze$ direction of a user, which may change as the user navigates the contents over space and time. Corresponding to each $viewport$ there is an FoV containing all visible tiles in that direction. FoV-adaptive streaming [13] saves bandwidth by delivering only the tiles within an FoV (sometimes tiles outside of the FoV are delivered in low resolution). The working basis of FoV-adaptation is the predictability of the user head movement at least to a short period of time (e.g., 1 to 2 seconds). Please note, viewport-adaptive shares the same meaning as FoV-adaptive and the scope of FoV is jointly determined by the limit of eyes and the display device (e.g., headset and mobile phones).

B. Issues in Caching System Design

Caching improves system throughput or reduces access latency by prefetching the required content in advance so as to hide the overhead in searching and transmission. For 360-degree video caching, the purpose is to ease the network burden and guarantee low access latency, and it could be divided into the following two procedures.

Video popularity prediction: This is also called the caching admission policy. It predicts the video request pattern so as to identify items to be cached to get maximum reward. Since the concept tiling is introduced in the 360-degree video, most recent work [18], [28], [45], [47], [26], [34], [6], [17] made prediction directly on tiles.

Segment selection: It selects the best resolution for the video $segment$ to be cached within the network and storage constraints. In the real world, there are multiple resolutions available for the same $segment$ and users can make selection freely based on the network condition. For caching, this must be done automatically.

From another aspect, storage organization of the edge node is very important. If not dealt properly, it would become a bottleneck in WAN (wide area network) optimization, 5G and so on. However, it is often ignored by the video caching solutions.

C. Key Idea and Components of CUBIST

Fig. 1 shows the architecture of the CUBIST system. It mainly consists of three kinds of components, namely client, cache node at the edge, and the video server. The client performs video display and viewport capture. The cache node is responsible for content serving by efficient tiles caching and prefetching. The video server serves video contents, performs video popularity and static tile requirement estimation, and instructs the cache node according to the result for efficient caching.

The key idea behind CUBIST is straightforward: by combining reactive caching and proactive prefetching in the edge to reduce not only bandwidth consumption but also access latency in FoV-adaptive 360-degree video streaming. Though the idea is straightforward, the journey to deliver such a system is full of challenges as pointed out previously in Section I and also discussed in other work [38], [45], [36], [26], [28], [34]. We contribute to the literature with a cost-effective hierarchical cache design and the corresponding policy for reactive caching and proactive prefetching, including the period-by-period network bandwidth estimation method and the updated mechanisms for video popularity estimation and tile requirement prediction.

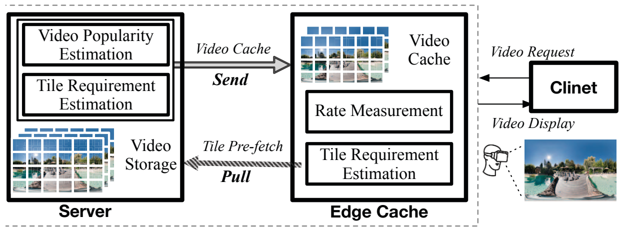


Fig. 1: The architecture of CUBIST system. Three roles are involved in the system, namely the client for video display and request, the edge cache for tiles caching and serving, and the server for video storage and serving.

III. Key Mechanisms behind CUBIST

A. Video Popularity Estimation

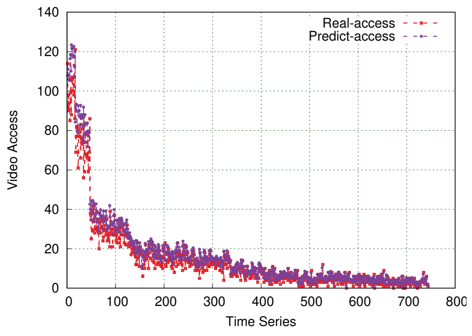
In CUBIST, popularity is calculated once at the beginning and updated as new requests occur. In detail, we treat every video segment as a basic unit for analysis and use the first segment of a video to represent the request of the whole video. The request is serialized into segments with a pre-allocated video name and increasing segment numbers. Formally, for a system with K videos, the incoming requests are denoted by a vector $\{R_1, R_2, ..., R_i, ..., R_K\}$, where R_i is the request for video i. R_i is defined as a tuple $R_i =< N_i, T_i >$, where $N_i = \{n_i(1), n_i(2), ..., n_i(k)\}$ is the historical information of the request for the video i, and $T_i = \{t_1, t_2, ..., t_k\}$ is the timestamp of the request.

Besides that past access information can provide some positive influence on future popularity, the interval between two successive accesses for a video is also an indicator for the future request. These two features are the widely-used and so-called "Frequency" and "Recency" in existing caching systems. Based on the principles discussed above, we adopt the self-exciting point process [15] for popularity estimation, where the event that occurred in the past is more likely to occur in the future. Below is the equation for estimating request of video i at time t:

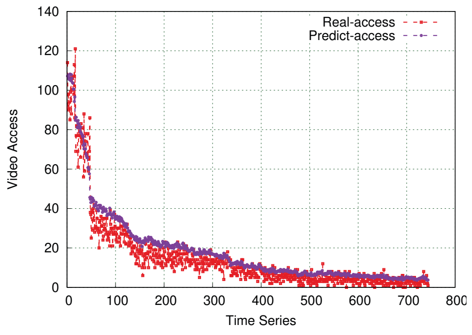
$$n_i(t) = \lambda \sum_{t'=1}^{t} n_i(t')\phi(t - t'). \tag{1}$$

where $\phi(t - t')$ is the kernel function describing the influence of "Recency", $\sum_{t'=1}^{t} n_i(t')$ represents the access frequency, and λ is an adjustment parameter. The kernel function $\phi(t-t')$ is non-increasing with the variable $t - t'$, indicating the video demand would decrease as it is getting "old" (large age).

The performance of the estimation varies with the deployed kernel functions. We test the exponential function ($\lambda = 2$, $\phi(t - t') = exp(t' - t)$) and power-law function ($\lambda = 0.25$, $\phi(t-t') = pow(t-t', -1)$) in predicting the future access with the workload generated with GlobeTraff [22]. As illustrated in Fig. 2, both functions could track the access change well. In detail, the exponential function is good at handling dynamic workloads, achieving 18% better accuracy than the power-law function. On the contrary, the power-law function can produce a smoother estimation curve. In the real world, users can select the right function according to the application scenario.



(a) exponential functions



(b) power-law functions

Fig. 2: The results of video popularity estimation with different kernel functions.

B. Tile Requirement Estimation

Tile requirement estimation is necessary for caching and prefetching. Corresponding to the two purposes, it is divided into two parts, namely static estimation for caching based on the historical behavior of many users and dynamic estimation for tile prefetching based on the current behavior of a user. The former is fulfilled on the video server, for it possesses full behavior information needed for analysis. The latter is done on the cache node, with the information collected from the client. The purpose of static estimation is to find out the most popular regions of interest (ROIs). Since it has been discussed thoroughly in the literature [23], [42], here we focus only on dynamic estimation.

Dynamic estimation follows the locality principle as most FoV-adaptive streaming methods do. That is, it prefetches video segments based on the predicted variation of the user's viewport as well as the network latency. As the cache node serves videos for multiple users, it is unfeasible to run a complex model online each time a tile miss occurs. To make the estimation efficient, a simple function describing the variation of viewport as the time is needed to determine the tiles to be fetched. Since 360-degree video is mapped to a unit sphere, we use the *distance* that the center of a user's field of view covers to measure the viewport variation. The distance here can be the orthodromic, haversine or great-circle distance. For any given period of time, the longest distance that a user's gaze can move is π.

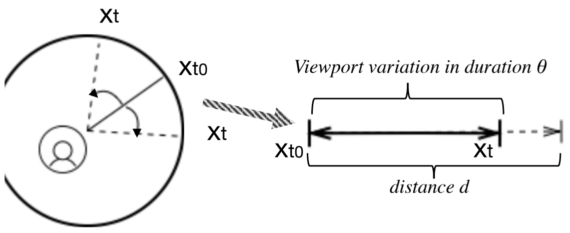


Fig. 3: An illustration of viewport variation.

Fig. 3 illustrates how to use the distance to predict viewport variation. Given a distance d ($0 \leq d \leq \pi$) and a period θ starting at t_0, the probability p that a user's gaze will stay within the circle of radius d at time t satisfies the following relationship:

$$P(\text{argmax}_{t_0<t<t_0+\theta}||x_t - x_{t_0}|| < d)) \geq p \tag{2}$$

where x_t is the center of FoV at time t.

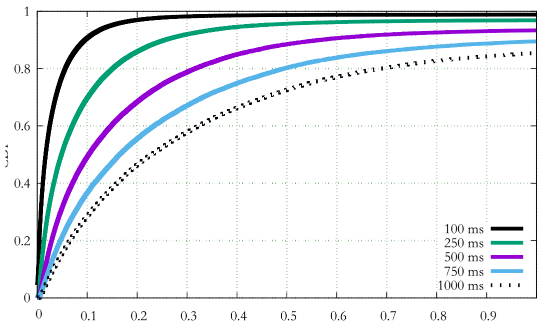


Fig. 4: An illustration of the relationship between distance d, the probability p that the user's gaze will stay within d, and the time t; generated according to the data in [16]

.

According to the previous work [7], [16], the user's viewport does not vary drastically in most cases, and a linear-regression method can predict user's future FoV for up to $2s$. This lays a good basis for prefetching relevant FoV tiles so as to smooth video playback. For the instance shown in Fig. 4, the probability is as high as 85% that the user's FoV will remain within a distance of less than π after 1000ms from the initial FoV. Therefore, the required video content can be transmitted without any complex predictions of the user head movement.

The above result indicates a locality property when the users watch 360-degree videos, which is leveraged in our design. This property also makes prefetching of future data based upon the current viewport very attractive. For simplicity, we compute the mapping to the viewport movement d as:

$$d \leftarrow f(p, \theta), \theta < 2s \tag{3}$$

C. Rate Measurement

Rate measurement is critical to select a proper resolution for the video segment to be cached. From the perspective of QoE (e.g., high bitrate, short stall duration, and fewer bitrate transitions), users want to watch a video in the highest resolution that could be supported by the available bandwidth. However, the available bandwidth may change from time to time, making it complicated to select the right resolution for the video segments to be cached.

Consider an example that a user requests a specific FoV of a 360-degree video, which is lucky in the cache node. To ensure QoE, the cache node should serve the content without adapting the resolution to the network bandwidth. In the real world, users' requests arrive at the cache node continuously, and CUBIST makes decisions for caching and prefetching accordingly with the requests. Therefore, we need to determine the video quality (i.e., bitrate) to be cached in both the caching phase and the prefetching phase.

1) Network Bandwidth Estimation: : Bandwidth estimation is critical in the video caching system. It is not economic for the edge cache node to store multiple resolutions of a single video at the same time. Either, caching videos with high resolution might be not effective when the bandwidth between the client and cache is very low, for the client will suffer great packet loss and cause many video stalls for rebuffering. In a word, the cache node must reduce the storage consumption while ensure better QoE by fully utilizing the bandwidth.

To fully utilize the bandwidth, first of all the cache node should have enough bandwidth to transfer the data smoothly to the user. In addition, given that the 360-degree video is large and the entire content may not be totally cached, the cache node needs to obtain the video in advance from the video server if the user request is predicted to cause a miss at the cache. This also consumes the bandwidth between the cache node and the server.

Calculating the end-to-end delay is still an open problem since the bandwidth is shared by many nodes and the traffic of these nodes might change dynamically. In CUBIST, the caching node triggers delay estimation process once there is a new group of video segments to be cached. CUBIST estimates the end-to-end bandwidth with the following two equations via the packet dispersion techniques.

$$C = median(C_i) = median(\frac{L}{t_{i,e} - t_{i,s}})_{i\in\{1,2,\ldots,n\}} \tag{4}$$

$$R = \frac{m \times L}{\sum_{i=1}^{m}(t_{i,e} - t_{i,s})} \tag{5}$$

where L is the packet size, $t_{i,e}, t_{i,s}$ are the arrival time of the packet pair i, n and m are the probe number set for estimating path capacity C and bandwidth R respectively.

The above equations give two steps to estimate the end-to-end bandwidth: (i) Use continuous packet pairs to detect the path capacity. To guarantee the efficiency and responsiveness, a small n ($n = 30$) is adopted by CUBIST. (ii) Calculate the available throughput by sending packets with the rate obtained in the first step and here the cross traffic is taken into considerations. For the sake of accuracy, the parameter m is set to 150.

2) Reactive Caching: For caching, because the quality of the link between the client and the edge is relatively stable, in order to save the cache space, the edge cache does not need to store multiple resolutions of the same video segment for network-adaptive streaming. However, the network bandwidth does vary. Therefore, it is crucial for the cache node to choose the most appropriate quality (or more precisely, bitrate) for the segment to be cached.

Unlike traditional ways that try to make the estimated bitrate closer to the real value, CUBIST determines the video quality for caching with *average* available bandwidth over a period. The process is triggered once a new video is inserted into the cache in the caching phase. CUBIST maintains a list of bandwidth levels $\boldsymbol{Bw}$ between the users and the cache node. The bandwidth change rate at time t is calculated by:

$$\delta(t) = \frac{|Bw_t - Bw_{t-1}|}{\min\{Bw_t, Bw_{t-1}\}} \tag{6}$$

At the same time, a network state indicator ϑ is given to adjust the length of the period. When $\delta(t) > \vartheta$, the period is reset to 0. Otherwise, the period keeps increasing. In other words, the period is adjusted according to the change rate of network bandwidth and the given threshold ϑ. With the period determined, the cache node then calculates the average bandwidth $\hat{\boldsymbol{Bw}}$ for choosing the right video quality for caching.

Fig. 5 illustrates how the rate determination procedure works with the real-world trace of network throughput [43], where $\vartheta = 1$. By following the bandwidth change rate, the network is divided into multiple periods of different lengths, each of which has a relatively stable change rate.

3) Proactive Prefetching: Prefetching is triggered after a cache miss. Given that the end-to-end delay between the cache node and the video server is known (for example, by some delay measurement method), CUBIST can predict and prefetch tiles (identified by id) to be accessed soon but not in the cache yet. As opposed to the caching phase, the resolution of the prefetched tiles is adapted to the real-time end-to-end delay. Of course, such an adaptation is only feasible when the allocated bandwidth between the video server and the cache node is larger than that between the server and the client. Details of efficient prefetching are as follows.

Given a tile τ_0 and suppose a request to it triggers a prefetch task. Let t_b be the timestamp when τ_0 is buffered at the client, and $\Delta t = \tau_0.t_e - t_b$ be the time taken to transfer τ_0 from the

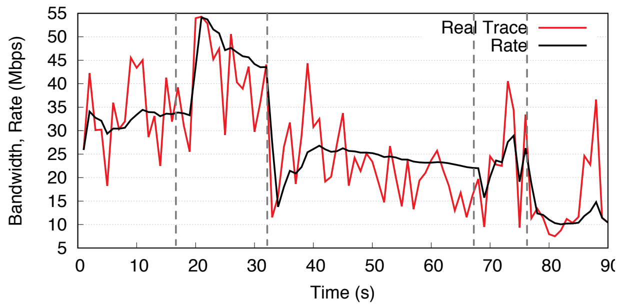


Fig. 5: The rate measurement procedure, where the network is divided into multiple periods of different lengths and each period has a bandwidth change rate less than a given threshold.

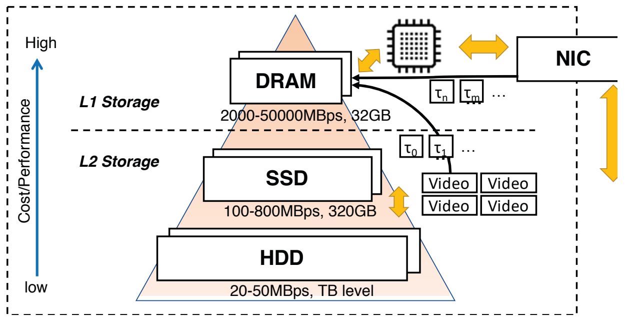


Fig. 6: A reference design of the edge cache system.

video server to the client. In order to make the prefetching efficient, the user viewport during Δt should remain within the scope of the prefetched tiles. The maximum amount of data that could be transmitted from the server to the cache node during the time is $Bw \times \Delta t$. According to Eq.(3), for a given probability p, the distance to the required view is $d = \epsilon f(p, \Delta t)$, where $\epsilon = 2$ if there is no prediction for the direction, and $\epsilon = 1$ otherwise. Therefore, the tiles within the area $|d| * |\pi|$ should be prefetched.

Since the playback length of each segment is fixed, the bitrates of the tiles to be prefetched are calculated as follows:

$$bitrates(\tau) = \frac{Bw \times \tau.t}{|d| * |\pi|} \tag{7}$$

Once a prefetched tile is no longer within the scope of the next request, it will be deleted or moved to L2 storage according to the caching policy, which will be detailed in the next section.

IV. CUBIST Design and Implementation

Given the vast volume of 360-degree videos and the diversity of users, it is not an easy task to design a cache system with a high throughput and low cost so as to provide users with a good QoE. This section details our solution.

A. Hierarchical Cache System Design

The cache system we design is illustrated in Fig. 6. It adopts a hierarchical design to balance the cost and performance, consisting of L1 storage of high speed but small volume

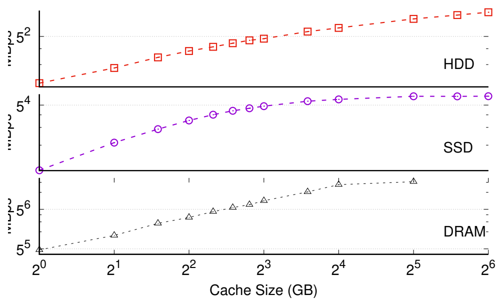


Fig. 7: The throughput of different storage media.

DRAM (dynamic random-access memory) and L2 storage of low speed but large volume SSD (solid-state drive) and/or HDD (hard-disk drive). Fig. 7 displays the throughput of HDD, SSD and DRAM on a server with the same configuration shown in Section V-A, where the size of each required packet is set to 64Kb and the requests are issued with full network speed. It can be seen from the figure that: 1) the gap between DRAM, SSD and HDD is large, and 2) the cache volume matters to throughput. On account of this and the prices of different storage media, obviously it is a good choice to adopt a hierarchical design. To the best of our knowledge, it is the first hierarchical caching solution for 360-degree video streaming. Given that the bandwidth between the edge node and end users would reach 10-20Gbps [1], [35] in the future and larger cache capacity would do help, it is especially necessary to do so.

In CUBIST, the L1 cache is used to store the most frequently accessed as well as prefetched tiles, which might be probably accessed soon after, and the L2 cache is used to store other popular but less frequently accessed tiles (e.g., those within ROIs). It is also the duty of the cache system to prefetch tiles from the source server on behalf of users when a cache miss happens. In other words, CUBIST not only does caching reactively, but also prefetches missing tiles proactively. In the following section, we will detail how CUBIST works as the caching policy.

To get the most from caching, CUBIST uses a pipelined thread model to handle users' requests. In detail, a dedicated thread is used to handle NIC (network interface controller) operations. When a request arrives, it checks the request identifier. If the required tile is not found in the cache, the prefetching procedure is triggered to forward the request to the source server. Otherwise, the request is dispatched to the corresponding I/O threads to read and return the desired tile. Meanwhile, the location of tiles may be changed to get better performance. Besides, some other optimizations such as batching, content indexing, and parallel read/write could also be applied to this process to improve performance further.

B. Caching Policy Driven by Video Reward

Caching policy is at the core of any cache system. Least Frequently Used (LFU), Least Recently Used (LRU), and First In First Out (FIFO) are all the well-known and widely-used caching policies. The caching policy of CUBIST is illustrated in Algorithm 1. It mainly consists of three procedures, namely cache initialization and update, request handling, and event handling. The details of them are as follows.

Algorithm 1: CUBIST Caching Policy All in One

```
(1) cache initialization and update:
begin
    Get the list of popular tiles from the Video Server;
    Fetch popular tiles from the Video Server accordingly;
end
(2) request handling:/* main loop of user serving */
foreach $\tau$ requested by a user do
    Search $\tau$ in the cache;
    if $\tau$ is found then  /* cache hit */
        Send $\tau$ to the user;
        Update the record of $\tau$;
    else  /* cache miss */
        Forward the request to the Video Server;
        Estimate future tiles requirement (refer to §III-B);
        Prefetch the missing tiles from the Video Server;
    end
end
(3) event handling: /* handle tile-related events */
switch $\tau$.event do
    case $\tau$ is accessed do  /* ignore prefetched tiles here */
        Re-rank $\tau$ according to Equation 9;
        Relocate $\tau$ if necessary;
    end
    case $\tau$ expires do  /* prefetched tile is replaceable now */
        Rank $\tau$ with Equation 9;
    end
    case $\tau$ is prefetched do
        if L1 has enough space then
            Assign $\tau$ a lifetime and put it into L1;
        else  /* do cache replacement */
            Move the last ranked tiles to L2;
            Put $\tau$ into L1 with a lifetime;
        end
    end
end
```

The cache initialization and update procedure is called when the cache node starts or the cache hit ratio drops. After the node starts and the cache is initialized with popular tiles, the node is ready to handle users' requests. At the same time, a daemon is run (not shown in the algorithm) to monitor the cache hit ratio. The caching policy of CUBIST works around the cache hit ratio Φ within a given period θ, which is defined below.

$$\Phi = \frac{\sum_t hit(t)}{\sum_t req(t)}, t \in (max\{t_1, t_l - \theta\}, t_l) \tag{8}$$

where $\sum_t req(t)$ and $\sum_t hit(t)$ are the number of requests and the number of cache hits respectively within the period $(max\{t_1, t_l - \theta\}, t_l)$ and t_1 and t_l are the start and end time of the given observation window, which is used to update the previously recorded cache hit ratio. The decrease of Φ (i.e., $\frac{d\Phi}{dt} < 0$) means the newly got cache hit ratio is less than the recorded one, so the procedure is re-executed once again to update the cache and to ensure high cache hit ratio. Of course, no request, no update.

The request handling procedure is used to serve users' request. If the cache hits, the requested tile is sent back directly. Otherwise, the request is forwarded to the source video server and at the same time, the prefetching function is triggered that estimates and fetches the missing tiles in advance.

CUBIST works in an event-driven manner. The event handling procedure is used to handle various events generated during user serving. Two key operations involved are prefetched tiles processing and tiles (re-)ranking. The prefetched tiles will be put into the L1 cache as aforementioned to maximize the benefit. In the case that there is no enough space in L1, tiles ranked last will be flushed to L2 for space, which may trigger further cache replacement there. To avoid the prefetched tiles being evicted before access, each of them is assigned a lifetime, during which no replacement is allowed. Tiles ranking are based on the potential caching gain shown below.

The potential gain of caching a tile τ is jointly determined by the estimated future access of each video, the size of the tile and the performance of the cache. Formally, it is defined as:

$$G_i(\tau) = r_i \sum_{\tau \in i} (size(\tau) \times \tau.f[\mathcal{T}(u, s_i) - \mathcal{T}(u, cache)]) \quad (9)$$

where r_i is the popularity (detailed in Section III) of video i, u, s_i and $cache$ are the client, the video server for video i and the cache node respectively, $\mathcal{T}(u, s_i)$ and $\mathcal{T}(u, cache)$ are the latency to get the content from the video server and the cache respectively, and $\tau.f$ ($\sum_{\tau \in i} \tau.f = 1$) is the ratio τ is accessed.

With $G_i(\tau)$ defined, CUBIST determines the optimal caching choice for each video i by calculating argmax$\{G_i(\tau)\}$. As for the whole system, it ranks the selected tiles of each video and gets a list of videos for caching according to the available cache space. During cache replacements, those tiles with least $G_i(\tau)$ will be evicted.

V. Performance Evaluation

We have implemented CUBIST and evaluated it against a wide range of benchmarks. This section reports the results.

A. Methodology and Experiment Setup

The testbed. Our testbed consists of two parts, namely the lab testbed and the simulated testbed. The lab testbed consists of two physical servers, each of which runs CentOS 7.2, with two eight-core Xeon E5-2640 v2 CPUs running at 2.0 GHz, 20 MB Intel Smart Cache, 32 GB DDR3 RAM, a set of 2 TB SATA hard disks and two 160 GB SATA Intel SSDs configured as RAID 0. Each server is equipped with a Gigabit Ethernet NIC connecting to a Gigabit Ethernet switch. We use the Linux Traffic Control (tc [20]) tool to adapt the bandwidth. The simulated testbed ships an HTTP server for 360-degree video streaming based on the DASH-enabled platform [24]. The GlobeTraff [22] is used to generate large-scale workloads with more requests.

By default, the average link rate between users and the video server is 10Mbps, and the link rate between users and the cache node is 50Mbps. In addition, the bandwidth of the cache node and the server is 25Mbps. The link rate of the lab testbed is fixed without change, whereas the link rate in the simulation testbed varies in the same way as the real-world 4G network [43].

Datasets and workloads. We use the head movement sequences of two datasets [5], [44] to emulate users' behavior in video watching. A Markov based tool [17] is used to generate more tile distributions and to enlarge each video to 10 minutes. Each video is divided into 6×4 tiles for efficient encoding and bandwidth saving, based upon the results from [13]. Therefore, each video contains 7,200 different tiles with a fixed segmentation of 2 seconds. The top-$[1, 4]$ viewed tiles along with the corresponding three nearest neighboring tiles are cached if the video is selected by the cache node. Additionally, the bitrate of each video segment on the content server is adaptable according to the network bandwidth. The Weibull distribution [39] is used to generate the video streaming requests, for the video content access does not follow Zipf's law [48].

Evaluation metrics. We use the following metrics to study the performance of CUBIST. *(1) Video Bitrate:* it is defined as the average bitrate (Mbps) during each video streaming session. It is the key indicator of the video quality perceived. The larger the bitrate, the higher the video quality. *(2) Cache Hit Ratio:* this refers to ratio that the user requests are served by the cache node. The higher the cache hit ratio, the more efficient the caching system and the more the saved bandwidth. *(3) Rate Transitions:* this refers to the number of resolution changes between two consecutive tiles when users watch a 360-degree video. It is also a major factor of QoE.

Benchmarks. To validate the efficiency of CUBIST, we select the following benchmarks for comparisons: *(1) No Cache:* The content server handles all the requests from the users. *(2) Video Cache:* The cache node places the entire video according to the video popularity estimation in Section III-A. *(3) Tile Cache:* We implement the tile-based cache scheme presented in [28], which is the latest and most related work to ours.

B. The Benefit of Hierarchical Cache

CUBIST adopts hierarchical cache design to balance the cost and performance. Fig. 8 plots the throughput of CUBIST (DRAM-SSD) and other configurations, where: 1) the cache space is 20% of the total video size, 2) the ratio of L1 to L2 cache is 3:2, and 3) the ratio of L1 to L2 hit varies between 9:1 and 7:3 randomly. As shown in the figure, the throughput of CUBIST is only a bit lower than the case that all packets are read directly from the DRAM. This is because cache miss is inevitable, resulting in interaction with remote video server, which reduces the marginal utility of DRAM. Compared with a single-layer SSD, CUBIST improves the throughput from 765MBps to 39GBps, but with a lower cost

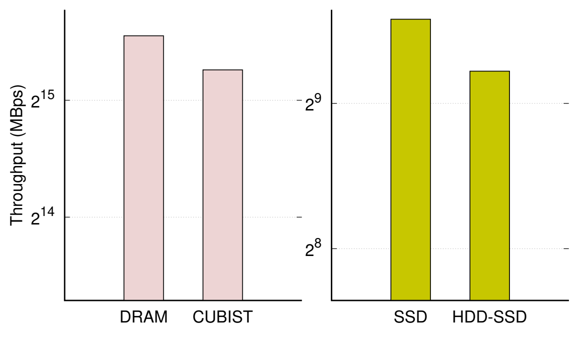
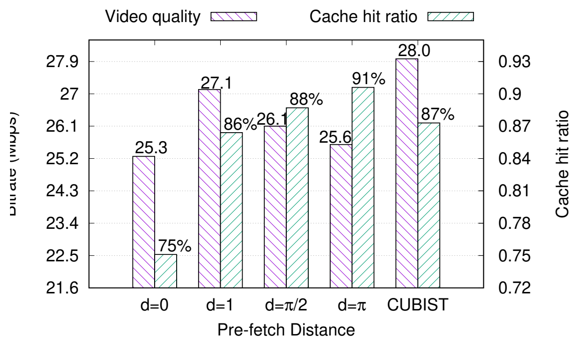
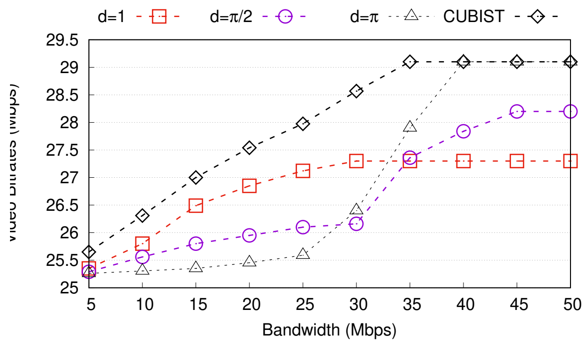


Fig. 8: Throughput of the cache node with different cache configurations.

than the pure DRAM scheme — the cost of CUBIST is reduced by about 38%. For the HDD-SSD case, the system behaves in a similar way. In a word, hierarchical cache design can meet the need of 360-degree video caching in a good cost-performance ratio. Indeed, this scheme has been widely supported by many caching systems. In the end, hierarchical cache design with fixed cache cost means larger cache space and higher cache hit ratio, which would bring in more benefit.



(a) video bitrate & cache hit ratio



(b) Prefetch performance vs. bandwidth

Fig. 9: The benefit of CUBIST prefetching.

C. The Benefit of Prefetching and Caching

1) The Benefit of Prefetching: We evaluate the benefit of prefetching in the lab testbed. Each test has 1,000 requests to 20 different videos, with the cache capacity set to 20% of the total video size. In order to show the adaptability of CUBIST prefetching strategy, 4 different settings are used, with fixed prefetching distance beginning at $d = 0$ (no prefetching) and ending at $d = \pi$ (full prefetching). The experimental results are shown in Fig. 9.

Fig. 9(a) shows the impact of prefetching on the video quality (i.e., bitrate) and cache hit ratio. We can see that the cache hit ratio for $d = 0$ and $d = \pi$ sets a lower and an upper bound of CUBIST, respectively. The 16% improvement in cache hit ratio indicates that prefetching is effective as a supplement to the cache system that only caches the most frequently accessed tiles. However, if we look at the effective bitrates of the delivered videos, full prefetching nearly does no help compared with the case of no prefetching. This is because full prefetching consumes too much network bandwidth. By adapting the prefetching distance, CUBIST can get the highest video bitrate while with a good cache hit ratio.

Fig. 9(b) shows the behavior of prefetching when the network bandwidth between the video server and the cache node increases. Interestingly, the bitrate delivered by full prefetching increases rapidly when the network bandwidth is larger than 25Mbps. Prefetching with distance $\pi/2$ exhibits a similar phenomenon. The reason is obvious: more bandwidth means more data can be transmitted from the video server to the cache within the same period. It can also be seen from the figure that the adaptable prefetching strategy presented by CUBIST can utilize network resources better by selecting and transferring tiles to be requested shortly.

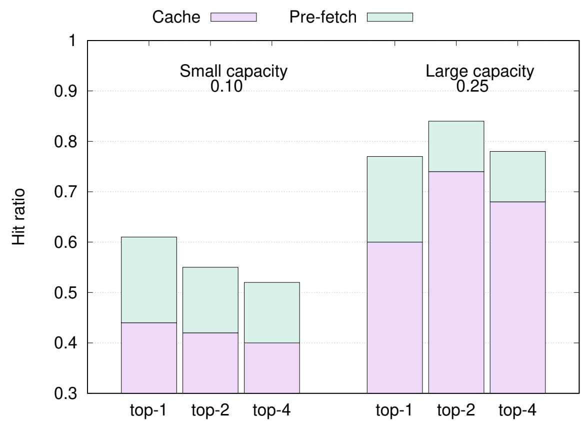


Fig. 10: The cache hit ratio of CUBIST with different cache capacity and tiles selection schemes. The benefit of tiles prefetching is also shown.

2) The Behavior of Caching: We study the cache hit ratio of CUBIST in a simulated testbed with various settings. Each test has 500 videos with 4.2×10^5 requests. The first quarter of the videos is used to warm-up and initialize the CUBIST system. The results are shown in Fig. 10 and Fig. 11.

Fig. 10 shows the cache hit ratio with regard to different cache capacities and tile selections. Here the impact of

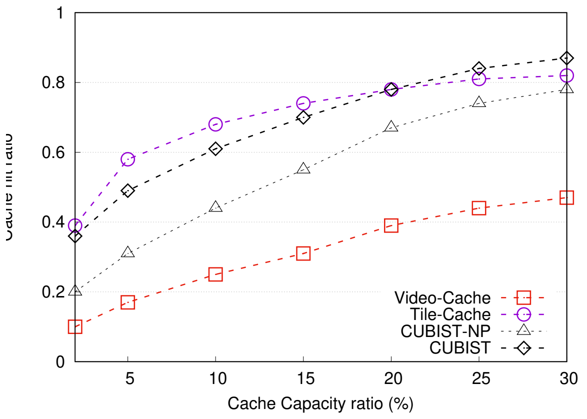


Fig. 11: The cache hit ratio of different caching policies along with the change of cache capacity.

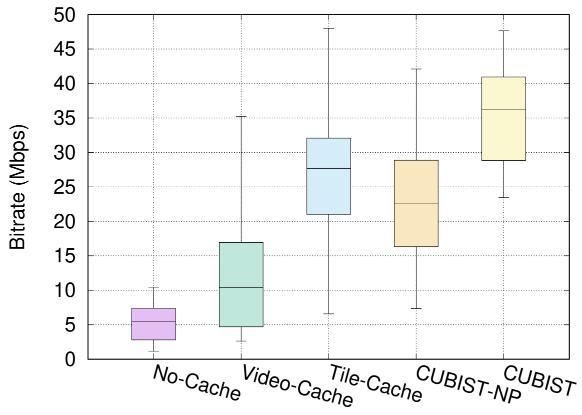
prefetching on the cache hit ratio is also shown. By top-n (n=1, 2, 4), we mean the mostly viewed n tiles along with their three nearest neighbors are cached. We can see from the figure that, when the cache is small (with a capacity ratio of 0.10), the top-1 case can get the highest cache hit ratio of 61%, with 17% contributed by prefetching. The reason is that selecting top-1 tiles can cover more videos, but for each video it only covers parts of the most frequently accessed tiles and thus leaves much more chances for prefetching to function. As the cache space grows from 0.10 to 0.25, selecting top-2 tiles would be a better choice. It can reach a hit ratio of 84% among which 10% is contributed by prefetching. Finally, caching top-4 tiles always leads to sub-optimal performance.

Figure 11 shows the cache hit ratio of 4 caching schemes when the cache capacity ratio grows from 2% to 30%. When the cache capacity ratio is less than 15%, the top-1 tiles are cached. Otherwise, the top-2 tiles are cached. It can be seen from the figure that the cache hit ratio of CUBIST and $TileCache$ increases from 0.36 to 0.87 and from 0.39 to 0.82 respectively as the cache capacity grows. $TileCache$ is more efficient in storage utilization due to finer granularity in video manipulation when the cache capacity ratio is less than 20%. However, the gap between CUBIST and $TileCache$ is not large because of the prefetching mechanism — as indicated by the curve CUBIST-NP, without prefetching the cache hit ratio would drop greatly below that of $TileCache$. The cache hit ratio of the $VideoCache$ scheme is always under 50% because some parts of the cached videos would never be accessed as pointed out previously.

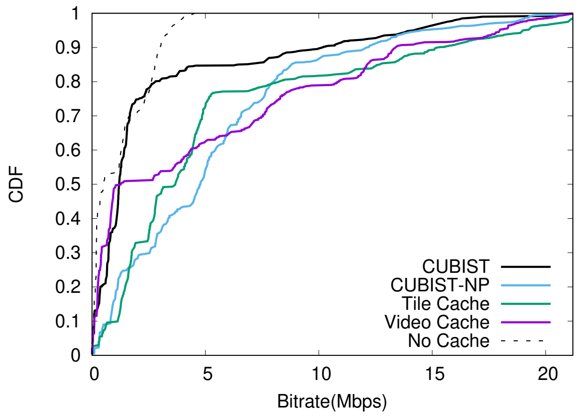
D. QoE Comparison with Related Work

We also evaluate the end-user QoE in terms of effective video bitrate and the frequency of bitrate transitions using the same settings in Section V-C2, with the results shown in Fig. 12.

Fig. 12(a) shows the effective video bitrate obtained by various caching schemes, where the case of no cache is also given. It can be seen that the gain made by caching



(a) video bitrates of 5 caching policies



(b) CDF of the video bitrate transitions

Fig. 12: QoE perceived by end users in various settings.

is significant. The median bitrate of CUBIST is 34.8Mbps, which is the highest among all caching schemes. CUBIST outperforms $TileCache$, whose median bitrate is 26.9Mbps, by 12.9% in video bitrate. The median bitrate of CUBIST without prefetching (denoted by CUBIST-NP) is 23.6Mbps, which is lower than that of the $TileCache$ scheme. This indicates the benefit of prefetching from another angle.

Fig. 12(b) shows the CDF of the video rate transitions of 5 caching schemes over 300 segments. The smaller the bitrate change times and amplitude, the higher the video quality. For the 5 schemes, the bitrate change frequencies are similar, but the change distributions are different. As can be seen in the figure, the $NoCache$ scheme has the lowest bitrate changes due to the limited bandwidth, but its bitrate is always low as shown in Fig. 12(a). The $VideoCache$ scheme has a relatively stable bitrate, with nearly half of the bitrate changes less than 1Mbps. CUBIST shows better control of video transitions, with about 75% of the bitrate changes less than 2Mbps, about 2.2× higher than that of the $TileCache$ scheme. The reason is that a cache miss in $TileCache$ incurs a larger bitrate change due to the lack of prefetching. On the contrary, CUBIST provides support for prefetching and makes it possible to issue tiles request in advance so as to hide some transmission time. Therefore, it outperforms all the other caching schemes.

VI. Related Work

FoV-adaptive streaming. Recently, a number of papers have used user behavior or head movement information to optimize 360-degree video delivery (e.g., [9], [13], [18], [37], [6], [38], [40], [47]). Technologies ranging from data analytics and statistics [10], [27] to machine learning algorithm [2], [32], [33] are applied to predict users' future behavior with historical information and video content properties. These FoV-adaptive video streaming mechanisms lay a good foundation for our work. Indeed, our method on tile requirement estimation is specially designed on the basis of [6], [17]. The work reported here differs from these work in that it presents different methods for estimating video popularity, predicting tile requirement, and selecting the right video bitrate. In addition, it supplements these work with edge caching.

Edge cloud and cache-enabled delivery. Edge cloud [19] and caching nowadays have also been adopted to enhance 360-degree video delivery [29], [50]. FoV-aware caching policy [28] uses a probabilistic model to cache the common FoV of each video. It works on tiles directly without considering tiles prefetching. JERTC [26] uses a genetic algorithm to optimize tile-based caching while the work [34] jointly considers tiles caching and encoding for 360-degree videos. Tile-based edge caching is also supported in Allies [41], but the work focuses on how to improve the cache utilization of edge clouds and minimize video service costs by jointly optimizing cache placement and request scheduling. Chakareski[4] designs a framework that, given specific caching and computational resources at each base station, allows the base stations to coordinate caching/rendering/streaming strategies to maximize the aggregate reward for users. It offers an optimization solution for edge caching. Maniotis and Thomos [30] presented a tile-based edge caching scheme for 360-degree video streaming, but with a focus on live-streaming in mobile networks, which is quite different from the scenario of CUBIST. Liu et al. [49] presented a 360-degree video streaming scheme based on multi-access edge computing (MEC), but with a focus on operation offloading and real-time video transcoding.

In-network cache organization. It is common and widely-accepted to use a small, fast cache to improve the throughput of various systems and applications. HCS [31] exploited both DRAM and SSD in routers for efficient contents storing and packet forwarding. Netcache [21] incorporated cache in the switch to improve the performance of in-memory storage. CUBIST shares the same idea, but does caching at the edge, which is more powerful in terms of cache space and processing capability. Though HCS and Netcache inspire our idea of hierarchical cache design in CUBIST, the methods they presented are not applicable to 360-degree video delivery because the cache space in a router or a switch is too limited and their caching strategies do not take into consideration the characteristics of 360-degree video delivery.

VII. Conclusion

We presented CUBIST, a method and system for high-quality 360-degree video streaming in networks with the capability of edge caching. At the core of CUBIST are the cost-effective edge cache design and FoV-adaptive prefetching and caching. The cache is cost-effective in the sense that DRAM, SSD and/or HDD are organized hierarchically and contents are placed at different levels to achieve high performance to cost ratio. Unlike most existing work, CUBIST synthesizes proactive (FoV-adaptive) tile prefetching and reactive caching so that it can deliver 360-degree video in high quality but with low bandwidth consumption and access latency. Experiments with real-world datasets and simulated workloads show that CUBIST can achieve its design goal well in terms of video quality, cache hit ratio, and rate transitions — compared with the latest work [28], CUBIST can improve the effective video bitrate by 12.9% with most rate transitions being small.

CUBIST can be further improved in two ways. First, video encoding and transcoding play a key role in 360-degree video delivery as shown in [34], [26], [41], [30]. Currently, CUBIST does not pay much attention to it. So, we will study how to improve the delivered video quality further by a more efficient tile-based network encoding scheme and the corresponding caching mechanisms. Second, the edge cache node in the real world may consist of multiple servers as in the case of multi-access edge computing paradigm [49], which introduces new dimensions to consider. Therefore, we will investigate cooperative caching mechanisms and request routing algorithms to make CUBIST more applicable in such environments.

Acknowledgment

This work is co-sponsored by Natural Science Foundation of China (U1701263, 61572280).

References

[1] M. Agiwal, A. Roy, and N. Saxena. Next generation 5G wireless networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 18(3):1617–1655, 2016.

[2] Y. Bao, H. Wu, T. Zhang, A. A. Ramli, and X. Liu. Shooting a moving target: Motion-prediction-based transmission for 360-degree videos. In *Big Data (Big Data), 2016 IEEE International Conference on*, pages 1161–1170. IEEE, 2016.

[3] E. Bastug, M. Bennis, M. Médard, and M. Debbah. Toward interconnected virtual reality: Opportunities, challenges, and enablers. *IEEE Communications Magazine*, 55(6):110–117, 2017.

[4] J. Chakareski. Vr/ar immersive communication: Caching, edge computing, and transmission trade-offs. In *Proceedings of the Workshop on VR/AR Network*, pages 36–41. ACM, 2017.

[5] X. Corbillon, F. De Simone, and G. Simon. 360-degree video head movement dataset. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, pages 199–204. ACM, 2017.

[6] X. Corbillon, A. Devlic, G. Simon, and J. Chakareski. Optimal set of 360-degree videos for viewport-adaptive streaming. *in Proc. of ACM Multimedia (MM)*, 2017.

[7] X. Corbillon, G. Simon, A. Devlic, and J. Chakareski. Viewport-adaptive navigable 360-degree video delivery. In *Communications (ICC), 2017 IEEE International Conference on*, pages 1–7. IEEE, 2017.

[8] S. Dernbach, N. Taft, J. Kurose, U. Weinsberg, C. Diot, and A. Ashkan. Cache content-selection policies for streaming video services. In *IEEE INFOCOM 2016*, pages 1–9. IEEE, 2016.

[9] F. Duanmu, E. Kurdoglu, S. A. Hosseini, Y. Liu, and Y. Wang. Prioritized buffer control in two-tier 360 video streaming. In *Proceedings of the Workshop on VR/AR Network*, pages 13–18. ACM, 2017.

[10] C.-L. Fan, J. Lee, W.-C. Lo, C.-Y. Huang, K.-T. Chen, and C.-H. Hsu. Fixation prediction for 360 video streaming in head-mounted virtual reality. In *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 67–72. ACM, 2017.

[11] V. R. Gaddam, M. Riegler, R. Eg, C. Griwodz, and P. Halvorsen. Tiling in interactive panoramic video: Approaches and evaluation. *IEEE Transactions on Multimedia*, 18(9):1819–1831, Sept. 2016.

[12] 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 Transactions on Multimedia*, 19(10):2222–2237, 2017.

[13] M. Graf, C. Timmerer, and C. Mueller. Towards bandwidth efficient adaptive streaming of omnidirectional video over http: Design, implementation, and evaluation. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, pages 261–271. ACM, 2017.

[14] R. Grandl, K. Su, and C. Westphal. On the interaction of adaptive video streaming with content-centric networking. In *IEEE Packet Video Workshop*, Dec. 2013.

[15] A. G. Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90, 1971.

[16] D. He, C. Westphal, and J. Garcia-Luna-Aceves. Joint rate and fov adaptation in immersive video streaming. In *ACM Sigcomm workshop on AR/VR Networks*, Aug. 2018.

[17] D. He, C. Westphal, J. Jiang, G. Yang, and J. Garcia-Luna-Aceves. Towards tile based distribution simulation in immersive video streaming. In *IFIP NETWORKING 2019*, 2019.

[18] M. Hosseini. View-aware tile-based adaptations in 360 virtual reality video streaming. In *2017 IEEE Virtual Reality (VR)*, pages 423–424. IEEE, 2017.

[19] X. Hou, Y. Lu, and S. Dey. Wireless VR/AR with edge/cloud computing. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–8. IEEE, 2017.

[20] B. Hubert et al. Linux advanced routing & traffic control howto. *setembro de*, 2002.

[21] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica. Netcache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th SOSP*, pages 121–136. ACM, 2017.

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

[23] C. Koch, G. Krupii, and D. Hausheer. Proactive caching of music videos based on audio features, mood, and genre. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, pages 100–111, 2017.

[24] C. Kreuzberger, B. Rainer, H. Hellwagner, L. Toni, and P. Frossard. A comparative study of DASH representation sets using real user characteristics. In *Proceedings of the 26th International Workshop on Network and Operating Systems Support for Digital Audio and Video, Klagenfurt, Austria, May 13, 2016*, pages 4:1–4:6, 2016.

[25] D. H. Lee, C. Dovrolis, and A. C. Begen. Caching in HTTP adaptive streaming: Friend or foe? In *Proceedings of the 24th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV 2014, Singapore, March 19-20, 2014*, page 31, 2014.

[26] K. Liu, Y. Liu, J. Liu, A. Argyriou, and Y. Ding. Joint EPC and RAN Caching of Tiled VR Videos for Mobile Networks. In *International Conference on Multimedia Modeling*, pages 92–105. Springer, 2019.

[27] X. Liu, Q. Xiao, V. Gopalakrishnan, B. Han, F. Qian, and M. Varvello. 360° innovations for panoramic video streaming. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pages 50–56, New York, NY, USA, 2017. ACM.

[28] A. Mahzari, A. T. Nasrabadi, A. Samiei, and R. Prakash. Fov-aware edge caching for adaptive 360° video streaming. In *2018 ACM Multimedia Conference*, pages 173–181, 2018.

[29] S. Mangiante, G. Klas, A. Navon, Z. GuanHua, J. Ran, and M. D. Silva. VR is on the edge: How to deliver 360° videos in mobile networks. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*, pages 30–35. ACM, 2017.

[30] P. Maniotis and N. Thomos. Tile-based edge caching for 360° live video streaming. *IEEE Transactions on Circuits and Systems for Video Technology*, 2021.

[31] R. B. Mansilha, L. Saino, M. P. Barcellos, M. Gallo, E. Leonardi, D. Perino, and D. Rossi. Hierarchical content stores in high-speed icn routers: Emulation and prototype implementation. In *Proceedings of the 2nd ACM Conference on ICN*, pages 59–68. ACM, 2015.

[32] A. Nguyen, Z. Yan, and K. Nahrstedt. Your attention is unique: Detecting 360-degree video saliency in head-mounted display for head movement prediction. In *Multimedia Conference*, pages 1190–1198. ACM, 2018.

[33] H. Pang, C. Zhang, F. Wang, J. Liu, and L. Sun. Towards Low Latency Multi-viewpoint 360° Interactive Video: A Multimodal Deep Reinforcement Learning Approach. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 991–999. IEEE, 2019.

[34] G. Papaioannou and I. Koutsopoulos. Tile-based caching optimization for 360 videos. In *MobiHoc*. ACM, 2019.

[35] I. Parvez, A. Rahmati, I. Guvenc, A. I. Sarwat, and H. Dai. A survey on low latency towards 5g: Ran, core network and caching solutions. *IEEE Communications Surveys & Tutorials*, 20(4):3098–3130, 2018.

[36] S. Petrangeli, V. Swaminathan, M. Hosseini, and F. D. Turck. An http/2-based adaptive streaming framework for 360° virtual reality videos. In *ACM Multimedia*, pages 306–314. ACM, 2017.

[37] F. Qian, B. Han, L. Ji, and V. Gopalakrishnan. Optimizing 360 video delivery over cellular networks. In *ACM MobiCom All Things Cellular Workshop*, Oct. 2016.

[38] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 99–114. ACM, 2018.

[39] H. Rinne. *The Weibull distribution: a handbook*. Chapman and Hall/CRC, 2008.

[40] P. Rondao-Alface, M. Aerts, D. Tytgat, S. Lievens, C. Stevens, N. Verzijp, and J. Macq. 16K cinematic VR streaming. In *ACM Multimedia*, pages 1105–1112. ACM, 2017.

[41] J. Shi, L. Pu, and J. Xu. Allies: Tile-based joint transcoding, delivery and caching of 360° videos in edge cloud networks. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, pages 337–344. IEEE, 2020.

[42] T. Trzciński and P. Rokita. Predicting popularity of online videos using support vector regression. *IEEE Transactions on Multimedia*, 19(11):2561–2570, 2017.

[43] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck. HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks. *IEEE Communications Letters*, 20(11):2177–2180, 2016.

[44] C. Wu, Z. Tan, Z. Wang, and S. Yang. A dataset for exploring user behaviors in VR spherical video streaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, pages 193–198. ACM, 2017.

[45] M. Xiao, C. Zhou, Y. Liu, and S. Chen. Optile: Toward optimal tiling in 360-degree video streaming. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 708–716. ACM, 2017.

[46] M. Xiao, C. Zhou, V. Swaminathan, Y. Liu, and S. Chen. Bas-360: Exploring spatial and temporal adaptability in 360-degree videos over http/2. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 953–961. IEEE, 2018.

[47] L. Xie, Z. Xu, Y. Ban, X. Zhang, and Z. Guo. 360probdash: Improving QoE of 360 video streaming using tile-based http adaptive streaming. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 315–323. ACM, 2017.

[48] C. Xu, C. Dale, and J. Liu. Statistics and social network of youtube videos. In *International Workshop on Quality of Service*, 2008.

[49] L. Yanwei, L. Jinxia, A. Antonios, and C. Song. Mec-assisted panoramic vr video streaming over millimeter wave mobile networks. *IEEE Transactions on Multimedia*, 21(5):1302–1316, 2019.

[50] L. Zhang, S. O. Amin, and C. Westphal. VR video conferencing over named data networks. In *Proceedings of the Workshop on VR/AR Network*, pages 7–12. ACM, 2017.