PLVER: Joint Stable Allocation and Content Replication for Edge-assisted Live Video Delivery

Huan Wang, Guoming Tang, *Member, IEEE,* Kui Wu, *Senior Member, IEEE,* and Jianping Wang *Member, IEEE*

***Abstract*—Live streaming services have gained extreme popularity in recent years. Due to the spiky traffic patterns of live videos, utilizing distributed edge servers to improve viewers' quality of experience (QoE) has become a common practice nowadays. Nevertheless, the current client-driven content caching mechanism does not support pre-caching from the cloud to the edge, resulting in a considerable amount of cache misses in live video delivery. By jointly considering the features of live videos and edge servers, we propose *PLVER*, a proactive live video push scheme to address the cache miss problem in live video delivery. Specifically, *PLVER* first conducts a one-to-multiple stable allocation between edge clusters and user groups to balance the load of live traffic over the edge servers. It then adopts proactive video replication algorithms to speed up video replication among the edge servers. We conduct extensive trace-driven evaluation, covering 0.3 million Twitch viewers and more than 300 Twitch channels. The results demonstrate that with *PLVER*, edge servers can carry 28% and 82% more traffic than the auction-based replication (ABR) method and the caching on requested time (CORT) method, respectively.**

I. Introduction

The last few years have witnessed the drastic proliferation of live videos over streaming platforms (such as Twitch, Facebook Live, and Youtube Live), which have generated billions of dollars in revenue [1]. According to Twitch, in 2019 over 660 billion minutes of live streams were watched by customers, and 3.64 million streamers (monthly average) broadcast their channels via Twitch [2].

Nevertheless, the delivery of live videos is quite different from the conventional video-on-demand (VoD) service. First, live videos have spiky traffic, which means that the viewer popularity of live streams grows and drops rapidly [3]. In particular, live streams often encounter the "thundering herd" problem [4], [5]– a large number of users, sometimes on the scale of millions, may join the same live video simultaneously when popular events or online celebrities start a live broadcast. Second, live video delivery has stringent latency requirements due to the new breed of live video services that support interactive live video streaming. These services allow the broadcasters to interact with their stream viewers in real-time during the streaming process. High interactivity requires low-latency end-to-end delivery with guaranteed Quality of Experience (QoE) for live viewers [6], [7], [8].

H. Wang and K. Wu are with the Department of Computer Science, University of Victoria, Victoria, BC V8W 3P6, Canada (e-mail: {huanwang, wkui}@uvic.ca).

G. Tang is with Peng Cheng Laboratory, Shenzhen, Guangdong 518000, China (e-mail: tanggm@pcl.ac.cn). Corresponding author.

J. Wang is with the Department of Computer Science, City University of Hong Kong, Hong Kong (e-mail: jianwang@cityu.edu.hk).

The "thundering herd" problem can overload the system, causing lags and disconnections from the server. One effective way to solve the "thundering herd" problem, while maintaining low latency in live videos, is to utilize edge caches. For example, Facebook uses edge points of presence (PoPs) distributed worldwide to deliver live traffic [4]. Providing content via the edge (e.g., CDN edge servers, crowdsourced edge devices [9]) makes content much closer to the end users and alleviates the traffic burden of backbone networks to the cloud.

Nevertheless, a new problem arises in edge-assisted live video delivery: when a large number of end users simultaneously request a newly-generated video segment, this segment may not have enough time to be cached at the edge due to the real-time property of live streaming [10], [11]. As shown in Fig. 1, the edge server would return a cache miss for the first group of requests that arrive at the edge before the segment is fully cached. These requests would pass the edge cache and go all the way to the original content server. As a result, live viewers would experience deteriorated QoE (e.g., increased startup latency and playback stall rates). According to Facebook [4], around 1.8% of Facebook Live requests encountered cache misses at the edge layer. This is a significant number of cache misses due to the large number of total live viewers. To make matters worse, high resolution videos, e.g., virtual reality (VR) streams, need more time to be replicated to the edge and would create an even higher cache miss rate. Note that the above problem only exists in live video streaming because people typically watch non-live videos at different times. Therefore, there is sufficient time for the non-live video chunks to be cached at the edge.

The state-of-the-art research solves the above problem by holding back the availability of some newly-encoded live segments from the playback clients so that client requests arrive at the edge after the caching process has finished [4], [11]. While this strategy solves the cache miss problem, it may incur undesired extra delays to the live streams [12].

The cache miss problem mainly occurs because the current client-driven caching strategy was not designed for live videos. Since the caching process in current content delivery networks (CDNs) is usually triggered by client requests, caching (replication) will only commence when the cloud responds to the first request for a video segment. While this strategy makes sense when delivering non-live videos, it slows down the caching process in the context of live videos: there exists a time *gap* (shown as T_1 in Fig. 1) between the time when a segment is generated from the cloud and when the caching process starts. This gap mainly consists of two parts: 1) the time it takes for playback clients to obtain the availability

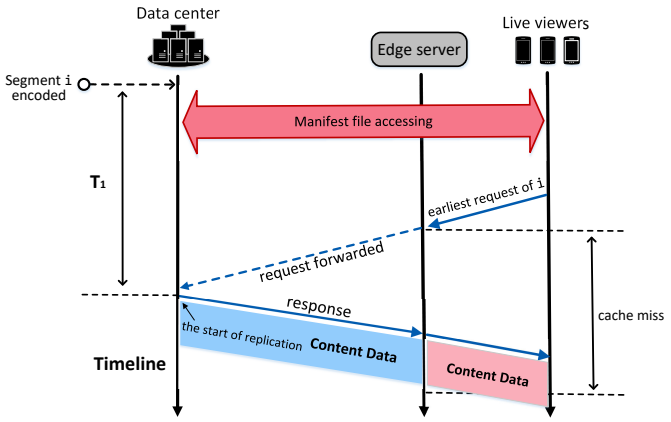


Fig. 1. Client-driven content caching (replication) for live videos.

information of the newly-encoded video segments, and 2) the time it takes for the clients to send their first segment requests. However, in the current pull-based CDN architecture, it is difficult to shorten the time gap (refer to § II for more details). This motivates us to rethink the caching strategy for live videos. Ideally, the cloud CDN server should adopt a video push model to proactively replicate newly-encoded video segments into the appropriate edge servers, such that video replication commences before client requests are received.

Although desirable, it is challenging to achieve such a goal due to the large number of video requests, the large number of edge servers, and the stringent real-time requirements. First, to adopt the proactive caching strategy, we must solve the allocation problem between edge servers and live viewers (i.e., assign viewers to proper edge servers). This is because 1) the video segments that need to be replicated in an edge server are determined by the live viewers served by this edge server, and 2) the bandwidth of edge servers is quite limited (much smaller than that of CDN servers), so some edge servers may be heavily loaded while others are under-utilized. Conventional load balancing solutions [13], [14] assume that content replicas are stored over *all* CDN servers. This assumption is unrealistic in our context due to the massive number of edge servers. Second, since the service capability of each edge server is limited, newly-encoded video segments should be replicated to multiple edge servers to mitigate the problem caused by spiky live video traffic. For each live video segment encoded in real-time from the cloud, we need to make a fast decision on selecting appropriate edge servers to cache the segment.

In this paper, we propose a proactive live video edge replication scheme (*PLVER*) to solve the cache miss problem in live video delivery. *PLVER* first conducts a *one-to-multiple* stable allocation between edge clusters and user groups to balance the load of live requests over edge servers. This way, each user group is assigned to its most preferred edge cluster whenever possible. Based on the allocation result, *PLVER* then uses an efficient proactive live video edge replication (push) algorithm to speed up the edge replication process in each edge cluster by using the real-time statistical viewership of the user groups in the edge cluster.

In summary, this paper makes the following contributions:

- *PLVER* implements a stable *one-to-multiple* allocation between edge clusters and user groups (i.e., one user group is served by one edge cluster, and one edge cluster can serve multiple user groups), under the constraint that the QoE of end users is guaranteed.
- Aiming at speeding up the edge replication process, *PLVER* identifies the unique traffic demand of live videos and develops a proactive video replication algorithm to periodically provide a fast and fine-grained replication schedule. To our knowledge, this is the first research work to provide proactive video replication algorithms (with details disclosed to the public) tailored for edge-assisted live video delivery.
- We perform comprehensive experiments to evaluate the performance of *PLVER*. Trace-driven allocations between 641 edge clusters and 1253 user groups are conducted, covering 64 ISP providers and 470 cities. Based on the allocation results, we further evaluate the performance of the video replication algorithm using traces of 0.3 million Twitch viewers and more than 300 Twitch channels. Performance results demonstrate the superiority of *PLVER*.

The rest of the paper is organized as follows: § II discusses the motivation and related work. § III illustrates the system design of *PLVER*. In § IV and § IV, we introduce two main components of *PLVER*, namely *stable one-to-multiple allocation* and *proactive edge replication*. In § VI, we set up an experimental environment by using Twitch live video viewership dataset and extracting the coverage information of realistic ISP networks. Extensive experiments are conducted in § VII to demonstrate the advantages of *PLVER* and evaluate its performance and robustness under different situations. § VIII concludes the paper.

II. Motivation and Related Work

A. Live Video Delivery Background

A live stream is usually encoded into multiple predetermined bitrates once it is generated and uploaded by broadcasters. For each bitrate, the stream is further split into a sequence of small video segments with equal playback length. The stream can be fetched sequentially by playback clients (e.g., via `HTTP GET`) using a suitable bitrate matching their network conditions [15].

In HTTP-based live video delivery, when a client joins a live channel, she first requests and accesses the stream's playlist file (generated by the original streaming server). This file contains the information of currently available segments (i.e., segments that have been encoded in the cloud) and the bitrates of the stream. Based on the manifest information, the clients send HTTP requests to their local edge servers. Afterwards, the playback clients fetch the live video segments in sequence and periodically access the newest playlist file (manifest) to check if any new segments have been produced. When live videos are delivered over edge servers (as shown in Fig. 1), these video segments will be replicated (cached) to the edge caches when the edge HTTP proxy receives the response (segment) from the cloud.

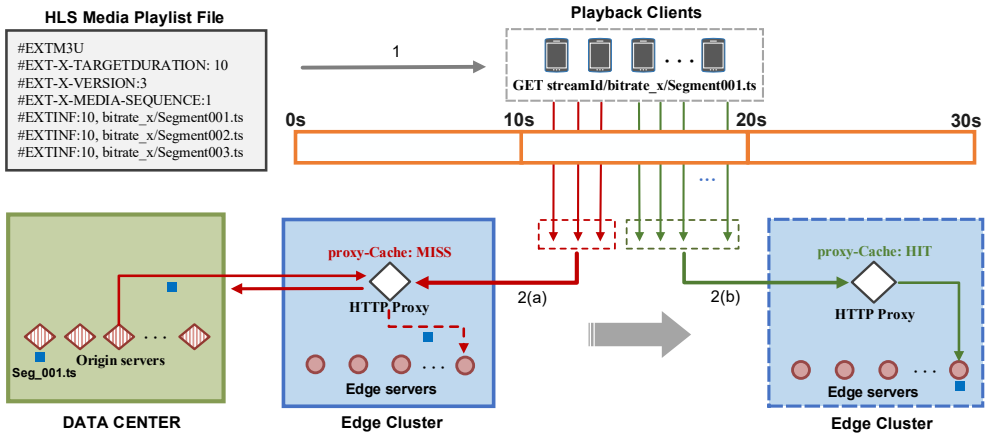


Fig. 2. Illustration of the cache miss problem in edge-assisted live video delivery.

B. Observation and Motivation

To better explain the cache miss problem, we use Apple HLS (HTTP Live Streaming) [16] as an example to illustrate the live video delivery process. As shown in Fig. 2, starting from a certain time after the first 10 seconds of a live stream, the first three video segments were generated from the cloud. By first accessing the playlist file, clients (with geographical proximity) realize the segment update and begin to request segment $001.ts$ via `HTTP GET` during 10 to 20 seconds. These requests are first handled by one of the HTTP proxies in an edge cluster, which checks if the requested segment is already in an edge cache. If the segment is in the edge cache, it could be readily fetched from there (step $2(b)$). If not, the proxy will issue an HTTP request to the original server in the cloud (step $2(a)$). Note that there exists another layer of cache, proxy and encoding servers inside the data center. As our system design does not change the current structure within the data center, these components are omitted in Fig. 2.

We can easily find that an earlier fraction of requests before the segment is fully cached in the edge (shown as step 2(a) in Fig. 2) would miss the edge cache. The current client-driven caching architecture creates a time gap before the caching process is started, which is critical for the live video delivery with real-time requirements. The gap mainly consists of two parts. The first part contains the time that the playback clients request and access the playlist file, which is inevitable in client-driven content caching since the clients must know the segment information (i.e., the URI) before sending the requests. Once the playback clients obtain the video segment availability information, the clients need another period of time before sending the first request for the segment. This second part of time exists because current live streaming protocols (e.g., HLS or MPEG-DASH) generally start live streaming with a relatively "older" video segment instead of the newest one to avoid playback stalls [16]. As shown in Fig. 2, the playback clients would start the live streaming by first requesting segment $001.ts$ rather than segment $003.ts$, thus further postponing the replication of segment $003.ts$ in the client-driven caching architecture.

C. Improving the QoE of Live Streaming

In order to solve the cache miss problem as well as to improve the QoE of 4K live videos, Ge et al. [11] proposed ETHLE, which "holds back" the availability of newly encoded video segments from the playback clients so that the playback clients could send their live requests to a certain segment after it was cached in the edge server. While this work has shown considerable QoE improvement, it may pose extra undesirable latency to the live streams.

In the industry, Facebook proposed two alternative methods to solve the cache miss problem for delivering live video over edge servers [4]. In the first scheme, their solution uses a similar "holding back" idea as that in [11]: the edge proxy returns a cache miss for the first request while holding the rest requests in a queue. Once the segment for the first request is stored in the edge cache via the HTTP response, the requests in the queue can be answered from the edge as cache hits. Similar to the work in [11], this design would incur undesirable latency to the live stream. The second scheme adopts a video push model where the server continuously pushes newly generated video segments to the proxies and the playback clients. This is the only reported design that adopts proactive content push for live video over edge servers. Nevertheless, the exact details of their video replication algorithm are unknown to the public.

In [17], Yan et al. proposed LiveJack, a network service that allows CDN servers to leverage ISP edge cloud resources to handle the dynamic live video traffic. Their work mainly focuses on the dynamic scheduling of Virtual Media Functions (VMFs) at the edge clouds to accommodate dynamic viewer populations. Wang et al. [1] proposed an edge-assisted crowdcast framework, which makes smart decisions on viewer scheduling and video transcoding to support *personalized* QoE demands. Mukerjee et al. [18] performed end-to-end optimization of the live video delivery path, which coordinates the delivery paths for higher average bitrate and lower delivery cost. This work, however, mainly focuses on optimizing the routing of live video delivery. In [19], Zhang et al. provided a video push mechanism to lower the bandwidth consumption of CDN by proactively sending the videos to competent seeds

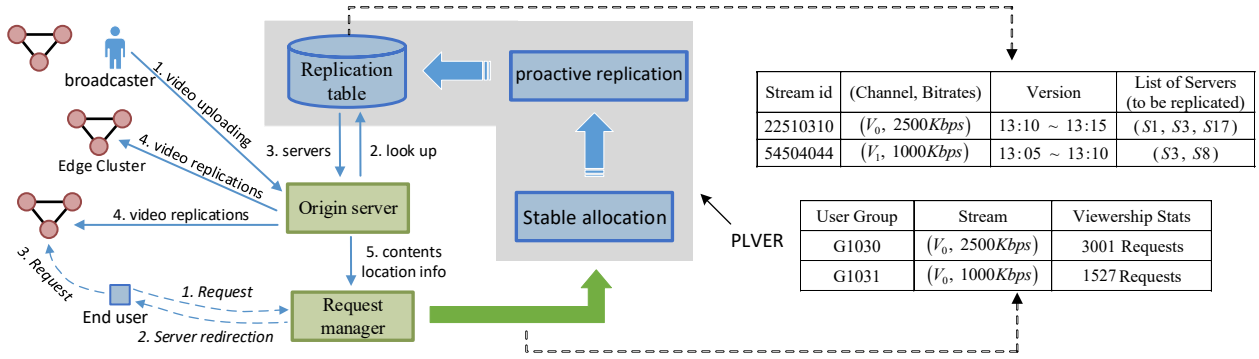


Fig. 3. System architecture: solid lines denote the procedure for video replication; dash lines denote the procedure that a user accesses live video.

in a hybrid CDN-P2P VoD system. This work uses proactive video push, but it does not target live videos. In [12], Ray et al. proposed a live-streaming upload solution that improves the overall QoE by assuming that live streams are typically watched by live viewers at different delays. The main focus of this work, however, is to optimize the bandwidth usage during the video upload process (i.e., from broadcasters to the cloud) instead of the video replication from the cloud to the edge servers. The optimization for non-live video delivery was investigated in [20], [21] and [22].

D. Generic Video Replication Techniques

Different content replication strategies were developed in [23], [9], [24], [25], and [26]. In [23], Hu et al. considered both video replication and request routing for social videos. Their algorithm focuses on social videos and the watching interests of different communities. In [9], Ma et al. considered the video replication strategies in edge servers. They proposed a content replication algorithm to jointly minimize the accumulated user latency and the content replication cost. In [24], Zhou et al. investigated how the popularity of videos changes over time and then designed video replication strategies based on the dynamics in video popularity. In [25], Applegate et al. presented an approach for intelligent video placement that can optimize the bandwidth usage for IPTV VoD services. Different from ours, the above works mainly focus on video-on-demand (VoD) services.

III. System Overview

PLVER's system design is shown in Fig. 3. Once a live viewer sends an `HTTP` request to the *request manager* of the system, the request manager resolves the URL and the IP address to identify the key information of the request, including the requested channel, bitrates, and the user group it belongs to. The above information is used by the request manager to redirect the request to an appropriate edge server. This procedure is denoted by the blue-dash lines in Fig. 3. The request manager also generates the viewership information (e.g., the number of viewers of each stream in each user group) and feeds the information to *PLVER* for edge server selection.

As shown in Fig. 3, there are three main components in *PLVER*: i) the *stable allocation* module assigns the global user groups to their desired edge server clusters and balances the load of live traffic, ii) the *proactive replication algorithm* periodically computes the edge replication schedule within each edge cluster in the near future (e.g., next 5 minutes) based on the viewership information from the request manager, and iii) the *replication table* which contains the directly available information of replication servers for each live video segment.

When the new live video segments of a stream are encoded and generated, the system first checks the most up-to-date replication schedule from the replication table by identifying the key information of the segment. It then proactively replicates these video segments into the guided edge servers, even though these videos are currently not requested by the users. In this way, the replication schedule can be easily obtained by using the stream ID and the version number of the target video segment as the search key. Note that the process of replicating video segments into edge servers and delivering the video contents from edge servers to end users are conducted concurrently, since the video segments are sequentially generated from the broadcasters.

The core component of the system is *PLVER*, denoted in the grey box in Fig. 3. Its main goal is to provide a replication schedule that can be readily used for live video replication over edge servers. To be more specific, it considers the traffic demand from different areas as well as the resource capacity of edge servers so as to provide the replication schedule that maximizes the traffic served by the edges. Note that the tasks of tracking each viewer's requests and directing the requests to edge servers or to the original server belong to real-time request redirection. These tasks happen *after* the replication schedule is generated and need to consider the dynamic content availability in edge servers. They are thus beyond the scope of this paper. Nevertheless, they will be considered in the performance evaluation of our algorithm in § VII.

In the following sections, we present the two main components of our solution, namely *stable one-to-multiple allocation* and *proactive replication algorithm*.

IV. Stable One-to-multiple Allocation

A. The Allocation Problem

Instead of making the request routing decisions individually for each client, we allocate servers at the granularity of user groups. Users in the same group generally have the same network features (*e.g., subnet, ISP, locality*) and thus are likely to experience similar QoE (e.g., startup latency and video buffering ratio) when dispatched to the same server [27], [28]. Similar to conventional content delivery problems, it is generally necessary to first consider the load balance problem between edge server clusters (consisting of a number of edge servers with the same network features) and the user groups.

We consider a target network of a number of edge server clusters and user groups. Each user group i originates an associate live traffic demand d_i, and each edge cluster j has a capacity C_j to serve the demands. In order to satisfy the QoE of users, the network must maintain a list of candidate edge clusters in descending order of preference for each user group. A higher preference denotes the clusters that can provide better-predicted performance for the viewers in the group (e.g., lower latency and packet loss). Likewise, each edge cluster j also has preferences regarding which map units it would like to serve [29].

An allocation of edge clusters to user groups is said to be a *stable marriage* if there is no pair of participants (i.e., edge clusters and user groups) that would both be better off individually than they are with the element to which they are currently matched [30]. By conducting stable allocation, each user group is assigned to its most preferred server cluster (if possible). In other words, stable allocation implies the most desirable matching between user groups and server clusters. The goal of our allocation problem is to assign user groups to edge clusters, such that the capacity constraints are met and the bidirectional preferences are accounted for.

B. Stable Allocation Implementation Challenges

In the context of live video delivered over edge servers, however, stable allocation has practical implementation challenges.

1) Expensive many-to-many assignment: Conventional allocation used by CDN vendors normally generates a many-to-many assignment, i.e., the traffic demand from each map unit could be served by several edge clusters that are geographically near the map unit. Many-to-many assignment makes sense when there are only a small number of server clusters globally. However, in our context, the number of edge clusters is much more than that of conventional CDN clusters. A many-to-many assignment under this situation becomes unnecessarily expensive. For example, given that there are tens of thousands of edge clusters, it is expensive to measure the performance of all the edge clusters and split the traffic demand from a single user group to many edge clusters. Note that the traffic demand generated by a user group in our context is also much smaller than the demand generated by a conventional map unit in conventional CDN.

2) Partial preference lists: Considering a large number of edge clusters and user groups, it is unnecessary to measure and rank the preference of every edge cluster for each user group. Thus, a given user group only needs a preference list of partial edge clusters that are likely to provide the best performance for the user group. Similarly, the edge clusters also only need to express their preferences for the top candidate user groups for the assignment.

3) Integer demands and capacities: The canonical stable marriage problem considers unit demand and capacity, while in our case the demands of user groups as well as the capacities of server clusters could be arbitrary positive integers.

C. Solution Methodology

To address the above challenges, we propose a new allocation algorithm: *Integer Stable One-to-multiple Allocation (ISOA)*, which extends the Gale-Shapley algorithm used for solving the canonical stable allocation problem. *ISOA* works in rounds. In each round, each free user group (all user groups are free initially) proposes its most preferred edge cluster, and the edge cluster could (provisionally) accept the proposal. Let G_i denote the list of user groups assigned to edge cluster i. In the case that the capacity of the edge cluster is violated, we perform a binary search on G_i to identify the user groups that need to be evicted.

Algorithm 1 shows the details of *ISOA*, where uP and cP are the preferred list of edge clusters (by user groups) and the preferred list of user groups (by edge clusters), respectively. C_j denotes the service capacity of edge cluster j. Note that in practice, C_j could be a *resource tree* instead of a single value [29]. To find a stable one-to-multiple allocation, we first set all user groups as free and set the initial user groups to each edge cluster as empty (line 1). Then, we pick up a free user group i in each round and find its most preferred edge cluster j (lines 2-3). Based on the preference of the edge cluster, we insert i into the temporarily assigned user group list of

Algorithm 1: *ISOA*

Input: Preference list by user group and edge cluster: uP, cP;
C_j: resource capacity of an arbitrary edge cluster j;
$D(.)$: traffic demand of given user groups.
Output: G_j: List of allocated user groups to edge cluster j.

1 Initialize all user groups as free; $G_j = \{j : [], \text{for } j \text{ in } E\}$;
2 **foreach** $i \in$ *free user groups* **do**
3 $j \leftarrow$ head of uP_i;
4 Insert i into G_j (rely on cP_j);
5 **if** $D(G_j) <= C_j$ **then**
6 continue;
7 $start \leftarrow$ bSearch(G_j, C_j, i); $k \leftarrow start + 1$;
8 **while** $k \leq length(G_j)$ **do**
9 **if** $D(G_j^{0 \sim k}) > C_j$ **then**
10 remove G_j^k from G_j;
11 **if** $G_j^k == i$ **then**
12 remove j from uP_i; goto line 3;
13 **else**
14 label G_j^k as free user groups;
15 $k \leftarrow k + 1$;
16 **return** G;

Algorithm 2: bSearch(G_j, C_j, i)

1 $left \leftarrow$ position of i in G_j; $right \leftarrow$ length of G_j;
2 $mid = \frac{left+right}{2}$;
3 **while** *True* **do**
4 **if** $D(G_j^{0\sim mid}) \leq C_j$ **then**
5 **if** $D(mid+1) > C_j$ **then**
6 **return** mid;
7 **else**
8 $mid = \frac{mid+right}{2}$;
9 **else**
10 $mid = \frac{left+mid}{2}$;

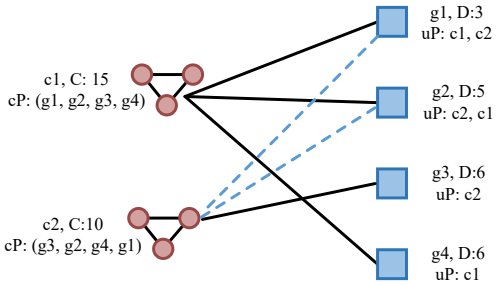


Fig. 4. An example of the stable one-to-multiple allocation containing four user groups and two edge clusters, where the service capacity of each edge cluster is denoted by 'c' and the traffic demand of each user group is denoted by 'D'

edge cluster j (i.e., user groups in G_j are sorted according to cP_j). After adding a new user group to G_j, the current traffic demand needed by G_j may or may not violate the capacity C_j. If C_j is not violated, we go back to propose another free user group for proposing (lines 5-6).

In case C_j is violated, we conduct a binary search (refer to Algorithm 2) to find the first user group $(start+1)$ in G_j which causes the capacity violation. We further go through all user groups from $G_j^{start+1}$ to the end of G_j: if adding a user group (G_j^k) would cause the capacity violation, then we remove this user group from G_j (lines 8-10). If the removed user group is i itself, it suggests that i cannot get its most preferred edge cluster. In that case, i will propose its next-most preferred edge cluster (lines 11-12). Otherwise, the evicted G_j^k will be labelled as a free user group, waiting for a second-chance proposal.

As a simple example, Fig. 4 illustrates the stable one-to-multiple allocation, where we have two edge clusters $c1$ and $c2$, with a service capacity of 15 and 10, respectively. There are 4 user groups ($g1$, $g2$, $g3$ and $g4$), which generate traffic demands of 3, 5, 6 and 6, respectively. The preferred edge cluster list by each user group as well as the preferred user group list by edge cluster are shown in this figure with uP and cP, respectively. We need to match each user group to their most preferred edge clusters. Running *ISOA* with the simple example in Fig. 4, user groups $g1$, $g3$, $g4$ can propose and be matched to their most preferred edge cluster ($c1$, $c2$ and $c1$, respectively). Group $g2$, however, will trigger the capacity violation of $c2$, thus it can only be matched to its second preferred cluster $c1$. The matching results are marked with the solid lines in Fig. 4.

TABLE I
SUMMARY OF MAIN NOTATIONS IN § V

Notation	*Description*
F	Target edge cluster within which the replication problem to be solved
E	Set of all edge servers over F
U	Online viewers from the user groups assigned to F
T	Target time window in the near future
A_j	Set of viewers that are served by edge server j
B_j	Bandwidth capacity of edge server j
a_i	The edge server that serve viewer i during T
b_i	Bandwidth consumed by viewer i
s_i	The live stream that viewer i is watching
c_j	Cache capacity of edge server j.
$\mathcal{D}_i^T$	The video segments to be generated by s_i in T
L_{A_j}	Non-redundant set of streams accessed by viewers in A_j
$S(\cdot)$	Size function that calculates the total data volume in a set of video segments
$\mathcal{V}_j^T$	Live video segments that should be replicated into edge server j during T
$L(v_i^t)$	The replication schedule: list of edge servers that video segments v_i^t should be replicated into

Let N denote the number of user groups in the target network. The time complexity of *ISOA* is $O(N * log(N))$. This is based on the fact that i) the number of edge clusters is much less than the number of user groups, and ii) the algorithm consists of N rounds of proposals and the bottleneck within each round is on the binary search and insert. It is worth mentioning that ISOA uses the peak traffic demand from a certain user group rather than the demand in real-time as the algorithm input, since the stable allocation is normally performed on a regular basis (e.g., weekly). Using peak traffic demand as the input of the allocation algorithm results in guaranteed QoE. In other words, if the allocation algorithm could satisfy viewers' demand during peak traffic hours, it would also perform well during non-peak hours.

V. Proactive Replication over the Edge Servers

A. Notations and Assumptions

Once the allocation problem is solved, we only need to consider the replication problem within each single edge cluster and its assigned user groups. Note that the QoE of the assigned user groups is guaranteed with stable allocation. We formulate the single cluster replication problem by considering a given edge cluster F and the user groups assigned to F. The main notations used in our problem formulation are listed in Table I. Without loss of generality, we make the following assumptions:

- We consider a target time window T in the near future that we need to generate the video replication schedule. During T, a number of live streams are watched by the live viewers U distributed across the user groups allocated to F.
- Each end user is served by one edge server at a time, and clients that cannot be served by the edge servers will be directed to the cloud.
- The cache in an edge server can be shared by multiple viewers who are accessing the video, but each viewer consumes their exclusive bandwidth of the edge server.

B. Resource Constraints

We divide time into a series of short, consecutive time windows, and try to generate a video replication schedule for each time window based on the feed of the most up-to-date viewership of live videos. The goal of the replication schedule is to maximize the traffic served by the edge servers so as to improve the QoE of end users.

Since clients generally access the video segments of a live stream sequentially, users' demands for the video segments to be generated in the next time window can be roughly estimated by the current viewership of the stream. Note that the live viewers might change their video quality (bitrates) during the watching process, and the distributed design and a fine-grained time window allow the system to quickly respond to such stream demand changes. We use a_i to represent the edge server that serves viewer i in T, and use A_j to denote the set of viewers served by server j, i.e.,

$$A_j := \{i | a_i = j, \forall i \in U\}. \tag{1}$$

Since each viewer only needs to be served by one edge server, we have

$$A_{j_1} \cap A_{j_2} = \varnothing, \forall j_1 \neq j_2. \tag{2}$$

For an arbitrary edge server j, it should have enough bandwidth to serve A_j. Thus, the following constraint should be posed:

$$\sum_{i \in A_j} b_i \leq B_j, \forall j \in E, \tag{3}$$

where b_i is bandwidth consumed by viewer i, and B_j is the total bandwidth of edge server j.

Let s_i denote an arbitrary live stream of one live channel with a certain bitrate. We denote the video segments to be generated by s_i in T as $\mathcal{D}_i^T$ (i.e., $\mathcal{D}_i^T = \{v_i^{t_1}, v_i^{t_2}, \ldots, v_i^{t_n}\}$, where $(t_1, t_2, \ldots, t_n)$ are the timestamps of the video segments $(v_i, v_i, \ldots, v_i)$ generated in T, respectively). If we use L_{A_j} to denote the non-redundant set of streams accessed by viewers in A_j (note that $|L_{A_j}| \leq |A_j|$ as one stream is normally watched by more than one viewers), the following constraint on cache capacity should be posed:

$$\sum_{i \in L_{A_j}} S(\mathcal{D}_i^T) \leq c_j, \forall j \in E, \tag{4}$$

where $S(\cdot)$ is the function that calculates the total caching size of a given set of video segments, c_j denotes the cache capacity of edge server j.

C. Cost of Content Replication

While edge servers benefit the live viewers, we may need to generate multiple replicas of single video content over the edge servers. More replicas on the edge servers generally mean more cost of cache resources at the edge as well as extra delivery cost from the cloud to the edge servers.

To reach a good balance, we pose the following constraint to limit the overall replication cost:

$$\sum_{j \in E} \sum_{i \in L_{A_j}} S(\mathcal{D}_i^T) \leq \alpha \cdot \sum_{j \in E} c_j, \tag{5}$$

where $\sum_{j \in E} \sum_{i \in L_{A_j}} S(\mathcal{D}_i^T)$ is the total size of overall replicas cached in the edge servers, and $\sum_{j \in E} c_j$ represents the total size of videos that could be cached globally. We use α, a percentage variable, to bound the total amount of videos that could be replicated, so as to limit the video replication cost.

D. Problem Formulation

The problem to be solved by our proactive replication algorithm can be formulated as:

$$\max_{\{a_i, A_j\}} \quad \sum_{j \in E} \sum_{i \in A_j} b_i \tag{6a}$$

$$\text{s.t.} \quad (2), (3), (4), \textit{ and } (5) \tag{6b}$$

By solving (6), we obtain a_i and A_j, with which the video replication schedule could be easily derived. The video segments that should be replicated into edge server j during T are given by:

$$\mathcal{V}_j^T = \sum_{i \in L_{A_j}} \mathcal{D}_i^T. \tag{7}$$

Based on $\mathcal{V}_j^T$ of each edge server, we can do a simple reverse transformation to get the video replication schedule. For an arbitrary video segment from stream i with timestamp t (i.e., v_i^t), the list of edge servers to which it should be replicated in T is given by:

$$L(v_i^t) = \{j | v_i^t \in \mathcal{V}_j^T, \forall j \in E, \forall t \in T\}. \tag{8}$$

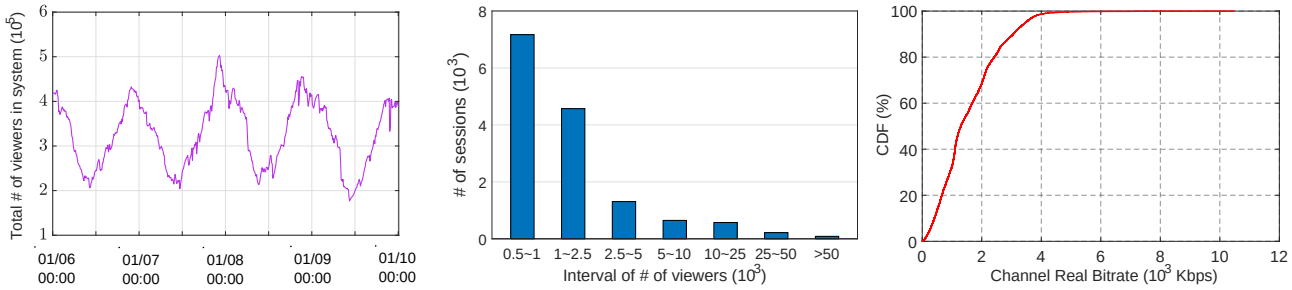
This video replication schedule is then inserted into the *replication table* in Fig. 3, by identifying the channel and bitrate of stream i. Problem (6) is an integer linear program with a massive number of design variables. In the rest of this section, we present a two-step heuristic algorithm to solve this problem.

E. Solution Methodology

Since live video traffic is network intensive [29], the non-shareable bandwidth constraint is a harder constraint compared with the shareable cache capacity constraint. Therefore, *PLVER* first considers the replication problem while temporarily ignoring the constraint on the cache capacity (**Step 1**). It then conducts adjustments by moving workloads from the edge servers where the cache capacity constraint is violated to the edge servers that have available cache and bandwidth resources (**Step 2**).

Step 1 (Edge Replication for Maximum Traffic): By temporarily ignoring the cache capacity constraint, the replication problem could be transformed into the *Multiple Knapsack Problem (MKP)* [31]. This problem is defined as a pair $(\mathcal{B}, \mathcal{S})$ where $\mathcal{B}$ is a set of m bins and $\mathcal{S}$ is a set of n items. Each bin $j \in \mathcal{B}$ has a capacity c_j, and each item i has a weight w_i and a profit p_i. The objective is to assign the items to the bins such that the total profit of the assigned items is maximized, under the constraint that the total weight assigned to each bin does not exceed the corresponding capacity.

If we treat the profit of each viewer i as the bandwidth consumption b_i of that viewer, the *MKP* problem is equivalent



(a) Number of viewers in the system over time. (b) The distribution of number of sessions with different number of viewers. (c) The CDF of the bitrates of live channels.

Fig. 5. The statistical information of the experimental dataset.

to our replication problem with unlimited cache capacities, i.e.,

$$\max_{\{x_{ij}\}} \quad \sum_{j\in\mathcal{B}}\sum_{i\in\mathcal{S}} b_i x_{ij}, \tag{9a}$$

$$\text{s.t.} \quad \sum_{i\in\mathcal{S}} b_i x_{ij} \le B_j, \forall j \in \mathcal{B} \tag{9b}$$

$$\sum_{j\in\mathcal{B}} x_{ij} \le 1, \forall i \in \mathcal{S} \tag{9c}$$

$$x_{ij} \in \{0,1\}, \forall i \in \mathcal{S}, j \in \mathcal{B}, \tag{9d}$$

where $\mathcal{B}$ and $\mathcal{S}$ are defined as the set of edge servers in a given edge cluster and the set of viewers assigned by the stable one-to-multiple allocation, respectively, and x_{ij} is defined as follows:

$$x_{ij} := \begin{cases} 1 \text{ , if viewer } i \text{ is served by edge server } j, \\ 0 \text{ , otherwise.} \end{cases} \tag{10}$$

The *MKP* problem has been well investigated and has a polynomial-time approximation solution (*PTAS*) [31]. After solving Problem (9), we could further calculate the set of video segments that should be replicated into each edge server based on the value of x_{ij}. Let $\mathcal{P}_j$ denote the set of video segments that should be stored in edge server j after Step 1.

Step 2 (Workload Adjustment): The solution $\mathcal{P}_j$ can maximize the total amount of traffic served by the edge cluster under the assumption of unlimited cache capacities of edge servers. Posing the constraint of limited cache capacity, we need to further adjust the solutions by moving part of the replication workloads from the edge servers whose cache capacity is violated, to the edge servers that have spare cache and bandwidth.

The whole proactive replication algorithm is shown in Algorithm 3, including three phases. Phase 1 represents Step 1, and phases 2 and 3 represent Step 2 introduced above. Once phase 1 is finished, there might be some viewers whose demand cannot be satisfied (i.e., $a_i = \varnothing$) if we pose the cache capacity constraint. For each of these unassigned viewers, in phase 2 we try to redirect it to an edge server that has available bandwidth capacity and the required video segments. There are no directly available edge servers that could be used to serve the rest of the unassigned viewers after phase 2. Thus, in phase 3, the algorithm offloads the incomplete video caching tasks to the edge servers that have residual resources. The algorithm returns when all traffic demands are completely satisfied or all edge servers in the given edge cluster are fully loaded.

Algorithm 3: *proactive replication algorithm*

Input: $\mathcal{F}$: given edge cluster; cc_j and bd_j: available cache capacity and bandwidth of edge server j ($j \in \mathcal{F}$), respectively; $\mathcal{M}$: set of user groups allocated to $\mathcal{F}$; s_i: the live stream accessing by viewer i; b_i: bandwidth consumption of s_i;

Output: Replication schedule $\mathcal{S}_j$, i.e., for each edge server j, the set of video segments should be replicated during the near time window T.

```
/* Phase 1: Generate initial replication schedule by solving MKP. */
x_ij ← solvingMKP(bd_j, b_i); S_j ← ∅ , ∀j ∈ F
a_i = ∅, for all viewer i ∈ M;
foreach edgeServer j ∈ F do
    I = {i|x_ij = 1} //viewers that are served by server j;
    L ← the set of unique streams consumed by the viewers in I;
    sort L according to R(s, j); (refer to (11))
    foreach stream s ∈ L do
        if D_s^T.size() > cc_j then
            continue;
        S_j.append(D_s^T); cc_j ← cc_j − D_s^T.size();
        a_i = j, (for all i ∈ I and s_i = s); update bd_j;
/* Phase 2: Redirecting viewers. */
foreach viewer i with a_i = ∅ do
    if exists server j ∈ F with bd_j ≥ b_i && s_i ∈ S_j then
        a_i ← j; bd_j ← (bd_j − b_i);
/* Phase 3: Offloading replication tasks. */
while exits edge server j with available resources do
    if a_i ≠ ∅ for all i ∈ M then
        break;
    pick the stream s with max R(s, j) and cc_j ≥ D_s^T.size();
    if s ≠ ∅ then
        S_j.append(D_s^T); cc_j ← cc_j − D_s.size();
        bd_j ← bd_j − R(s, j); φ = R(s, j)/b;
        update φ (number) unassigned viewers with a_i = j, ∀i
          with s_i = s;
return S_j;
```

The time complexity of our algorithm is $O(n + m)$ (n and m are the number of viewers in $\mathcal{M}$ and the number of edge servers in F, respectively), without considering the first step of solving the *MKP* problem. Since there are different approximation schemes for solving *MKP* in polynomial time and *PLVER* is a decentralized algorithm with viewers and edge servers from a single edge cluster (i.e., n and m are in a small

magnitude), *PLVER* could be solved easily.

Remark 1. *Note that PLVER does not need to track the real-time information of each viewer (e.g., stream being watched, bandwidth consumption). Instead, it only needs the statistics on the number of viewers of each stream (viewership) in each user group. In this sense, a "viewer" in PLVER actually means the corresponding resource demand to each live stream.*

Remark 2. *We introduce the reward for caching a stream s in a certain edge server j (line 7 in Algorithm 3). It implies the traffic demand that could be served by caching the video segments of this stream in server j (during T). Let b denote the bitrate of stream s; then the reward of s could be defined as following:*

$$\mathcal{R}(s,j) = b * min\{\lfloor \frac{\bar{B}_j}{b} \rfloor, N\}, \tag{11}$$

where $\bar{B}_j$ is the current available bandwidth of edge server j, and N is the number of viewers on stream s that have not been assigned to a server (i.e., $a_i = \varnothing$). The reward is in accord with our objective (6a), i.e., maximizing the amount of traffic served by edge servers.

VI. Experimental Setup

A. Live Video Viewership Dataset

Twitch provides developers with a RESTful API to obtain live video information. In our experiment, we use a public dataset [32] that consists of the traces of thousands of live streaming sessions on Twitch [33]. The dataset contains the information of all live channels in the Twitch system, with a sampling interval of 5 minutes. Detailed information includes the number of viewers of each channel, bitrates of each channel, and the duration of live sessions. We select the live channels that have more than 100 viewers and extract the required information for these channels.

Fig. 5(a) shows the total number of viewers in the system from Jan. 06 to Jan. 09. During a certain time period, a channel can be either *online*, which means that it is broadcasting a live video, or offline. When a channel is online, we say that it corresponds to a *session*. Fig. 5(b) shows the distribution of sessions with the different average number of viewers.

Fig. 5(c) illustrates the distribution of bitrates of channels in the dataset. Based on the video encoding guidelines [34], we assume that the video streams can be encoded with multiple standard resolutions (or bitrates): 240p, 360p, 480p and 720p (or $400, 750, 1000, 2500$ *Kbps*). Obviously, while a channel broadcasts with bitrate b, the viewers of this channel cannot select the video quality with a bitrate exceeding b.

B. Target Network & User Groups

geoISP [35] collected the detailed performance and region coverage information of $2,317$ Internet Service Providers (ISPs) in the US. Based on the information, we build a target network over two US states (Washington and Oregon). We further develop a web crawler to collect the ISP coverage information of 470 cities over 70 counties in the two states from the website of geoISP.

TABLE II
Different levels of preferred edge cluster by user groups.

Preference priority	Clusters description
Lv. 1	clusters that are within the same ISP and located in the same city.
Lv. 2	clusters that are within the same ISP and located in the same county.
Lv. 3	clusters that are located in the same city while with different ISPs.
Lv. 4	clusters that are within the same ISP and located in the same state.
Lv. 5	clusters that are located in the same county while with different ISPs.
Lv. 6	clusters that are located in the same state while with different ISPs.

We divide all the viewers from these two states (about 0.3 million on average) into 1253 user groups (based on the combination of ISP and city) within our target network. Note that one ISP can cover multiple cities and one city can be covered by multiple ISPs. From the dataset, we know the percentage of users in a city that is supported by a particular ISP. For each live stream, we distribute its viewers among these user groups based on the population of each user group (calculated based on each city's population and the ISP coverage percentage of the city).

C. Edge Server Clusters

1) Setup of edge clusters & servers: Among all user groups, we further extract 641 city-ISP combinations as the target for deploying the edge clusters. Each edge cluster in our experiments consists of five types of edge servers with 5, 10, 20, 40 and 80 Mbps bandwidth capacity, respectively. The servers are randomly deployed at each edge cluster. The total bandwidth of all the deployed edge clusters is set to the total traffic demand of all viewers. Note that such bandwidth setting may not always guarantee the full satisfaction of all viewers' demands because the bandwidth of each edge cluster may not be fully utilized or the cache capacity and QoE of each edge cluster may be different. Nevertheless, our later experiment shows that such a setting is appropriate to evaluate the performance of different edge caching strategies.

2) Setting of cache capacity: For an edge server with bandwidth capacity b Mbps, it should have at least $b*T$ Mb (T is the considered time period) cache capacity to ensure enough resources in our edge replication strategy. For simplicity, we use $\hat{b}$ to denote $b * T$ hereafter. Since video traffic delivery is network intensive, the cache capacity of edge servers is normally larger than $\hat{b}$ Mb. In our experiments, we assume that the cache capacity (variable X) of all edge servers is uniformed distributed within the range of $(0.5 * \hat{b}, 2 * \hat{b})$. In the following section, we will further adjust the capacity that could be used in each edge server by setting different values of replication cost constraint factor α.

VII. Performance Evaluation

A. Evaluations on Stable One-to-multiple Allocation

1) Evaluation Methodology: We compare the performance of *ISOA* with those of two other edge cluster allocation

TABLE III
ALLOCATION RESULTS OF ISOA.

Preference rank of the allocated cluster	# of user groups at each preference level		
	Greedy Allocation	ISOA	MALB
Lv.1	393	438	438
Lv.2	494	456	441
Lv.3	120	117	128
Lv.4	135	130	133
Lv.5	56	57	58
Lv.6	36	37	37
un-allocated	19	18	18

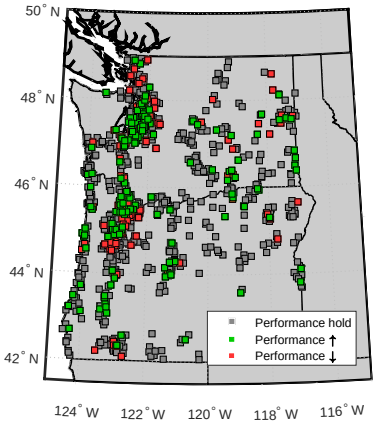


Fig. 6. Performance change with ISOA over *greedy allocation*.

strategies: *greedy allocation* and *MALB*. *MALB* is the server matching algorithm proposed in [36] to solve a similar server matching problem (with preferences taken into consideration) in the LTE network, and it is developed based on a college admission model. *MALB* has a time complexity of $O(N^2)$ compared with $O(N * log(N))$ of *PLVER*. With the greedy allocation, each of the user groups selects its most preferred edge cluster iteratively, until all user groups get allocated or there is no available edge cluster.

2) Preference List Generation: We define the rank of preferred edge clusters of user groups (introduced in § IV), as listed in Table II. Note that the preference list defined in Table II is just an example paradigm to generate the input of our stable allocation algorithm, and it can be altered by the CDNs themselves, e.g., according to the contract terms under which the cluster is deployed, the granularity of the user groups partition, and so on [29].

3) Performance Evaluation of ISOA: We conduct the stable one-to-multiple allocation between user groups and edge clusters based on the data introduced in VI-B and VI-C, using *ISOA* and the aforementioned *MALB* and *greedy allocation* methods. The detailed allocation results are summarized in Table III. Since there are 1253 user groups in our experiment, the table shows the distribution of these user groups allocated with different levels of preferred edge clusters. For example, there are 393 user groups that are allocated with their first ranked (most preferred) edge cluster with *greedy allocation*, while the number is increased to 438 with *ISOA*. Compared with the *greedy allocation* and *MALB*, *ISOA* can allocate more user groups with their higher-ranked (i.e., more preferred) edge clusters.

The performance improvement with *ISOA* over the greedy allocation for every user group in our experiment is further illustrated in Fig. 6, where the performance of each user group is marked with a colored square.

B. Evaluations on Proactive Edge Replication

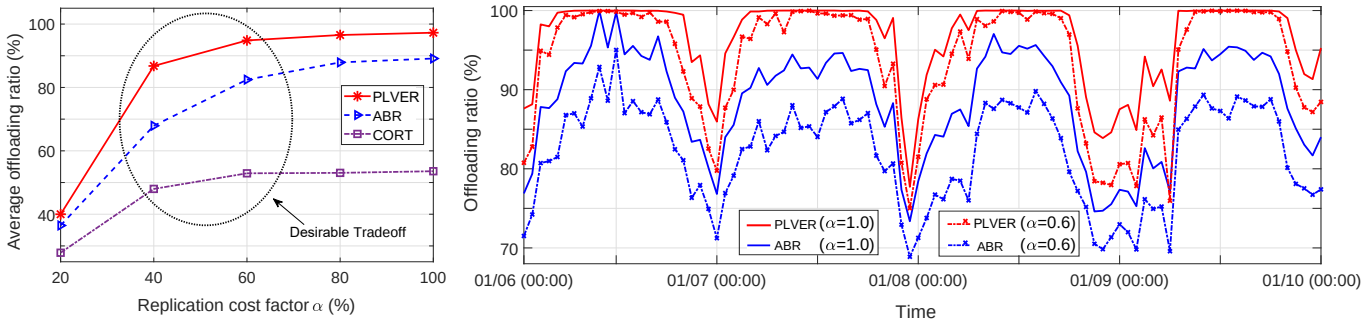
1) Evaluation Methodology: We evaluate *PLVER* by comparing it with the following replication strategies:

- *Auction Based Replication (ABR):* Each edge server conducts a simple "auction" to determine the cached videos: live videos with the largest number of viewers via the edge server win the auction and are cached, and the auction repeats until the edge server uses up its cache capacity [37]. In other words, this method replicates the videos into an edge server based on their current number of viewers in decreasing order.
- *Caching On Requested Time (CORT):* This strategy does not adopt pre-replication but instead uses request triggered caching. It caches the videos into the edge servers in real-time when video segments are truly requested by the end users. When content is requested, it first checks if there are available edge servers to serve this request; if not, it replicates the video segments of this stream into a new edge server.
- *Offline Optimal:* Since *PLVER* aims at solving an integer (binary) linear optimization problem proposed in (6), in our experiments we also compared the performance of *PLVER* with the optimal video replication strategy. The optimal solution is obtained by conducting offline optimization using the CVX Gurobi solver [38], without considering the cache capacity constraint.

To evaluate the performance of different strategies, we use the ***offloading ratio*** metric, which is calculated by the amount of traffic served by the edge servers divided by that of the overall traffic in a time period of length T. The performance is evaluated under different values of the ***replication cost factor*** α (refer to § V-C) so that we can investigate the tradeoff between performance and replication overhead.

2) Overall Performance of PLVER: Based on the Twitch viewership data, we conduct experiments on an hourly basis. By setting the value of α to $20\%, 40\%, 60\%, 80\%$ and 100%, we compute the average offloading ratios of the three strategies in each case. The results are shown in Fig. 7(a), from which we can see that *PLVER* outperforms *ABR* and *CORT* in all five cases. The overall performance improvement by *PLVER* for the five cases are $9\%, 10\%, 15\%, 28\%$ and 10% over *ABR*, respectively, and $82\%, 82\%, 79\%, 81\%$ and 44% over *CORT*, respectively.

Furthermore, we find from Fig. 7(a) that the overall performance is considerably improved when the replication cost constraint (α) is increased from 20% to 60%. However, the performance improvement fades when α continues to increase past 60%. This situation holds for all three replication strategies. Therefore, our experiment maintains a good tradeoff



(a) Avg. traffic offloading ratio with different α.

(b) The hourly performance of *PLVER* and ABR.

Fig. 7. Experiment results of performance of *PLVER* and other methods.

between performance and replication costs when α is between 40% and 60% (as shown in Fig. 7(a)).

We compare the offloading ratios from *PLVER* after Step 1 (based on the initial allocation results by solving MKP) and Step 2 to the optimal allocation results (by solving (9)), respectively. The results are listed in Table IV ($\alpha = 0.5$), where the 100% bandwidth constraint denotes the original bandwidth of edge clusters we used in previous experiments, which can also fully satisfy the traffic demand from viewers in the offline optimal case. We can see that the overall performance of *PLVER* decreases when Step 2 is finished. Nevertheless, such performance degradation can be alleviated or avoided if the edge servers have sufficient cache capacities.

3) Detailed Performance of PLVER: Referring to the overall performance, *ABR* is more comparable to *PLVER* (than *CORT*). We thus investigate the detailed performance behaviors of *PLVER* and *ABR*. The traffic offloading ratios for i) each hour and ii) each user group are shown in Fig. 7(b) and Fig. 8, respectively.

Fig. 7(b) shows the hourly traffic offloading ratio of *PLVER* and *ABR* with the replication cost constraint factor α equal to 100% and 60%, respectively. It shows that even within non-peak hours (when the resources of edge servers are sufficient), it is hard for *ABR* to yield satisfying performance. In contrast, when α decreases from 100% to 60%, the performance degradation of *PLVER* is much smaller than that of *ABR*.

Fig. 8 shows a heat map indicating the performance of *PLVER* and *ABR* at each edge cluster (with $\alpha = 40\%$), where the traffic offloading ratios are represented by different colors. Since there are no edge clusters with performance less than 30% or greater than 90%, our color bar denotes the traffic offloading ratio from 30% to 90%. An edge cluster of better performance is in green, and that of worse performance is in red. Under this setting, the average traffic offloading ratio of each edge cluster over the target network is 65% and 78% for *ABR* and *PLVER*, respectively, and the variance is 0.006 and 0.004 for *ABR* and *PLVER*, respectively.

We also investigate the performance when requesting different video qualities (240p, 360p, 480p, 720p). The satisfaction ratios of requests (i.e., the ratio of requests that are successfully directed to corresponding edge servers) with different replication strategies are shown in Fig. 9(a). We can observe that *PLVER* outperforms *ABR* and *CORT* for all types of quality requests. Among the four different quality

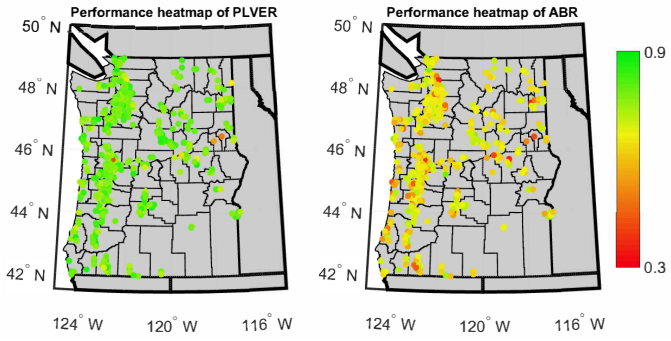


Fig. 8. The performance of *PLVER* and ABR in each edge cluster with $\alpha = 0.4$.

requests, the high-quality request of 720p has a relatively low traffic offloading ratio than the other three. However, as high-quality requests generate more traffic than the others, it has a greater impact on the final performance. Nevertheless, *PLVER* provides a satisfaction ratio of 36% for the 720p requests, which is higher than those of *ABR* (19%) and *CORT* (30%).

4) Impact of Viewership Fluctuation: As *PLVER* makes use of the viewership information (i.e., the number of viewers) in the current time window to make decisions for the next time window, the viewership fluctuation in consecutive time slots may impact the performance of replication algorithms. To investigate, we first generate the replication schedules from different replication strategies referring to the viewership data in peak traffic hours of § VI-A, and then manually generate new viewership data to test the performance of these replication schedules. The new viewership data is generated by introducing different levels of fluctuations on the former viewership data that was used to generate the replication schedules. To be specific, the number of viewers of each channel is added with different percentages of fluctuation (e.g., randomly up or down by 20%).

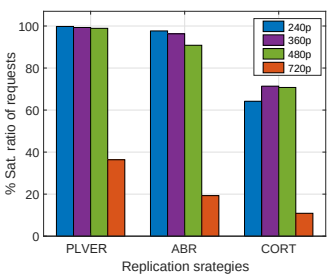
The performance of *PLVER* under different viewership fluctuations is shown in Fig. 9(b). We can see that the performance curve (representing the traffic offloading ratios) decreases slightly from 75% to 64% with fluctuations ranging from 10% to 70%. Nevertheless, according to the statistical analysis of our dataset, viewership fluctuations higher than 30% are quite rare (less than 15%). Hence, its impact on *PLVER* is quite small.

TABLE IV
AVG. OFFLOADING RATIO OF PLVER AND OFFLINE OPTIMAL

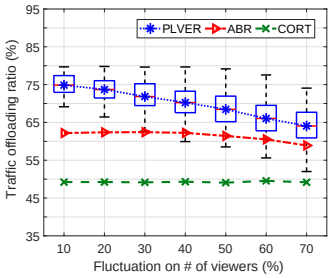
Bandwidth Constraints	*Offline Optimal*	*PLVER (after step 1)*	*PLVER (finished)*
100%	100%	96.50%	90.18%
80%	86.79%	83.18%	75.76%
60%	62.40%	60.47%	51.50%
40%	40.15%	39.90%	28.73%

5) Efficiency Analysis: We evaluated the processing performance of the edge replication algorithm of *PLVER* on a desktop computer without specific performance optimizations. The desktop ran Windows 10 with an Intel©Core™i7-2600 CPU and 4 GB of memory. The average processing time of *PLVER* is 0.47 seconds on the desktop. Typically, the computation of the edge replication schedule should be performed every time period (e.g., one minute) during which the live stream viewership has been changed (to a certain extent). Considering that the duration of a single video segment is normally between 2-10 seconds [11], [16], a replication schedule would normally be able to provide a replication guidance (i.e., the target edge servers to replicate a live stream) for the next 10 (approximately) video segments of a live stream.

It is worth mentioning that our experiments show that the workload adjustment of Algorithm 3 does not lead to big changes compared to the replication guidance derived from solving the initial MKP problem. With the replication cost factor α set as 60%, each edge server could fulfill 82.5% (on average) of the original video replication tasks (derived from solving MKP) at phase 1 without workload adjustment. When α is changed to 80%, indicating a larger cache capacity, the traffic offloading ratio increases to 93.5%.



(a) The satisfaction ratio for video requests with different qualities.



(b) The variation of performance with the fluctuation change on the number of stream viewers.

Fig. 9. Performance results of *PLVER* and other replication strategies.

VIII. CONCLUSION

Live video services have gained extreme popularity in recent years. The QoE of live viewers, however, suffers from the cache miss problem occurring at the edge layer. Solutions from current live video products as well as state-of-the-art research may incur undesirable latency to live streams. In this paper, we propose *PLVER*, an efficient edge-assisted live video delivery scheme aiming at improving the QoE of live viewers. *PLVER* first conducts a one-to-multiple stable allocation between edge clusters and user groups, and then adopts a proactive video replication algorithm to speed up video replication over edge servers. Trace-driven experimental results demonstrate that *PLVER* outperforms other edge replication methods.

REFERENCES

[1] F. Wang, C. Zhang, J. Liu, Y. Zhu, H. Pang, L. Sun *et al.*, "Intelligent edge-assisted crowdcast with deep reinforcement learning for personalized qoe," in *Proceedings of Int'l Conf. on Computer Communications (INFOCOM)*. IEEE, 2019, pp. 910–918.

[2] "Twitch Statistics and Charts:Twitch Tracker," https://twitchtracker.com/statistics, accessed in April 2020.

[3] P. Dogga, S. Chakraborty, S. Mitra, and R. Netravali, "Edge-based transcoding for adaptive live video streaming," in *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[4] "Under the hood: Broadcasting live video to millions," https://code.fb.com/ios/under-the-hood-broadcasting-live-video-to-millions/, accessed in April 2020.

[5] "How Facebook Live Streams To 800,000 Simultaneous Viewers," http://highscalability.com/blog/2016/6/27/how-facebook-live-streams-to-800000-simultaneous-viewers.html, accessed in April 2020.

[6] B. Wang, X. Zhang, G. Wang, H. Zheng, and B. Y. Zhao, "Anatomy of a personalized livestreaming system," in *Proceedings of the ACM Internet Measurement Conference*, 2016, pp. 485–498.

[7] G. Yi, D. Yang, A. Bentaleb, W. Li, Y. Li, K. Zheng, J. Liu, W. T. Ooi, and Y. Cui, "The acm multimedia 2019 live video streaming grand challenge," in *Proceedings of the 27th ACM International Conference on Multimedia*, 2019, pp. 2622–2626.

[8] H. Pang, C. Zhang, F. Wang, H. Hu, Z. Wang, J. Liu, and L. Sun, "Optimizing personalized interaction experience in crowd-interactive livecast: A cloud-edge approach," in *Proceedings of the 26th ACM International Conference on Multimedia*, 2018, pp. 1217–1225.

[9] M. Ma, Z. Wang, K. Yi, J. Liu, and L. Sun, "Joint request balancing and content aggregation in crowdsourced cdn," in *Proceedings of Int'l Conf. on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 1178–1188.

[10] B. Rainer, D. Posch, and H. Hellwagner, "Investigating the performance of pull-based dynamic adaptive streaming in ndn," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 34, no. 8, pp. 2130–2140, 2016.

[11] C. Ge, N. Wang, W. K. Chai, and H. Hellwagner, "Qoe-assured 4k http live streaming via transient segment holding at mobile edge," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 36, no. 8, pp. 1816–1830, 2018.

[12] D. Ray, J. Kosaian, K. Rashmi, and S. Seshan, "Vantage: optimizing video upload for time-shifted viewing of social live streams," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 380–393.

[13] H. Xu and B. Li, "Joint request mapping and response routing for geo-distributed cloud services," in *Proceedings of Int'l Conf. on Computer Communications (INFOCOM)*. IEEE, 2013, pp. 854–862.

[14] S. Narayana, J. W. Jiang, J. Rexford, and M. Chiang, "To coordinate or not to coordinate? wide-area traffic management for data centers," *Dept. Comput. Sci., Princeton Univ., Princeton, NJ, USA, Tech. Rep. TR-998-15*, 2012.

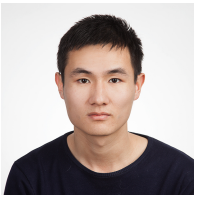
[15] I. Sodagar, "The mpeg-dash standard for multimedia streaming over the internet," *IEEE MultiMedia*, vol. 18, no. 4, pp. 62–67, 2011.

[16] J. Roger Pantos, William May, "RFC 8216: HTTP Live Streaming," https://tools.ietf.org/html/rfc8216.

[17] B. Yan, S. Shi, Y. Liu, W. Yuan, H. He, R. Jana, Y. Xu, and H. J. Chao, "Livejack: Integrating cdns and edge clouds for live content broadcasting," in *Proceedings of the 25th ACM International Conference on Multimedia*, 2017, pp. 73–81.

[18] M. K. Mukerjee, D. Naylor, J. Jiang, D. Han, S. Seshan, and H. Zhang, "Practical, real-time centralized control for cdn-based live video deliv-

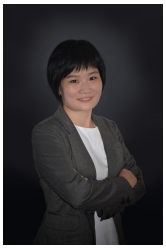
ery," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 311–324, 2015.
[19] Y. Zhang, C. Gao, Y. Guo, K. Bian, X. Jin, Z. Yang, L. Song, J. Cheng, H. Tuo, and X. Li, "Proactive video push for optimizing bandwidth consumption in hybrid cdn-p2p vod systems," in *Proceedings of Int'l Conf. on Computer Communications (INFOCOM)*. IEEE, 2018, pp. 2555–2563.
[20] V. Joseph and G. de Veciana, "Nova: Qoe-driven optimization of dash-based video delivery in networks," in *Proceedings of Int'l Conf. on Computer Communications (INFOCOM)*. IEEE, 2014, pp. 82–90.
[21] J. Kim, G. Caire, and A. F. Molisch, "Quality-aware streaming and scheduling for device-to-device video delivery," *IEEE/ACM Transactions on Networking*, vol. 24, no. 4, pp. 2319–2331, 2016.
[22] Z. Lu and G. De Veciana, "Optimizing stored video delivery for mobile networks: The value of knowing the future," *IEEE Transactions on Multimedia*, 2018.
[23] H. Hu, Y. Wen, T.-S. Chua, J. Huang, W. Zhu, and X. Li, "Joint content replication and request routing for social video distribution over cloud cdn: A community clustering method," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 7, pp. 1320–1333, 2016.
[24] Y. Zhou, L. Chen, C. Yang, D. M. Chiu *et al.*, "Video popularity dynamics and its implication for replication." *IEEE Transaction on Multimedia*, vol. 17, no. 8, pp. 1273–1285, 2015.
[25] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan, "Optimal content placement for a large-scale vod system," *IEEE/ACM Transactions on Networking*, vol. 24, no. 4, pp. 2114–2127, 2015.
[26] A. O. Al-Abbasi and V. Aggarwal, "Edgecache: An optimized algorithm for cdn-based over-the-top video streaming services," in *Int'l Conf. on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2018, pp. 202–207.
[27] X. Nie, Y. Zhao, D. Pei, G. Chen, K. Sui, and J. Zhang, "Reducing web latency through dynamically setting tcp initial window with reinforcement learning," in *IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, 2018.
[28] J. Jiang, S. Sun, V. Sekar, and H. Zhang, "Pytheas: Enabling data-driven quality of experience optimization using group-based exploration-exploitation," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017, pp. 393–406.
[29] B. M. Maggs and R. K. Sitaraman, "Algorithmic nuggets in content delivery," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 3, pp. 52–66, 2015.
[30] D. Gusfield and R. W. Irving, *The stable marriage problem: structure and algorithms*. MIT press, 1989.
[31] C. Chekuri and S. Khanna, "A polynomial time approximation scheme for the multiple knapsack problem," *SIAM Journal on Computing*, vol. 35, no. 3, pp. 713–728, 2005.
[32] "Twitch dataset," http://dash.ipv6.enstb.fr/dataset/live-sessions/, accessed in April 2020.
[33] K. Pires and G. Simon, "Youtube live and twitch: a tour of user-generated live streaming systems," in *Proceedings of the 6th ACM Multimedia Systems Conference (MMSys)*. ACM, 2015, pp. 225–230.
[34] "Youtube. Live encoder settings, bitrates, and resolutions," https://support.google.com/youtube/answer/2853702?hl=en, accessed in April 2020.
[35] "geoisp," https://geoisp.com/us/, accessed in April 2020.
[36] L. Li, Y. Zhang, B. Fan, and H. Tian, "Mobility-aware load balancing scheme in hybrid vlc-lte networks," *IEEE Communications Letters*, vol. 20, no. 11, pp. 2276–2279, 2016.
[37] Y.-H. Hung, C.-Y. Wang, and R.-H. Hwang, "Optimizing social welfare of live video streaming services in mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 19, no. 4, pp. 922–934, 2019.
[38] Gurobi Optimization, http://www.gurobi.com/.



Huan Wang received the Bachelor and Master degrees in Computer Science from Southwest Jiaotong University and the University of Electronic Science and Technology of China, in 2013 and 2016, respectively. He is currently pursuing a Ph.D. degree in the Department of Computer Science, University of Victoria, BC, Canada. His research interests include content/video delivery, edge caching and computing, network traffic anomaly detection.

Guoming Tang (S'12-M'17) is currently a research fellow at the Peng Cheng Laboratory, Shenzhen, Guangdong, China. He received his Ph.D. degree in Computer Science from the University of Victoria, Canada, in 2017, and the Bachelor's and Master's degrees from the National University of Defense Technology, China, in 2010 and 2012, respectively. He was also a visiting research scholar at the University of Waterloo, Canada, in 2016. His research mainly focuses on cloud/edge computing, green computing, and intelligent transportation systems.

Kui Wu (S'98-M'02-SM'07) received the BSc and the MSc degrees in Computer Science from the Wuhan University, China, in 1990 and 1993, respectively, and the Ph.D. degree in Computing Science from the University of Alberta, Canada, in 2002. He joined the Department of Computer Science, University of Victoria, Canada, in 2002, where he is currently a Full Professor. His research interests include network performance analysis, mobile and wireless networks, and network performance evaluation. He is a senior member of the IEEE.



Jianping Wang is a professor of the computer science department at the City University of Hong Kong, Hong Kong. She received the B.E. degree and the MSc degree in Computer Science from Nankai University, Tianjin, China, in 1996 and 1999, respectively and the Ph.D. degree in Computer Science from the University of Texas at Dallas, USA, in 2003. Her research interests include cloud computing, service-oriented networking, and data center networks.