Learning-based Fuzzy Bitrate Matching at the Edge for Adaptive Video Streaming

Wanxin Shi*†, Qing Li†, Chao Wang‡, Longhao Zou§†
Gengbiao Shen*¶, Pei Zhang‖, Yong Jiang*†
* International Graduate School, Tsinghua University, Shenzhen, China
† Peng Cheng Laboratory, Shenzhen, China
‡ Tsinghua-Berkeley Shenzhen Institute, Tsinghua University, Shenzhen, China
§ Southern University of Science and Technology, Shenzhen, China
¶ Sangfor Technologies Incorporation, Shenzhen, China
‖ Beijing University of Posts and Telecommunications, Beijing, China
shiwx17@mails.tsinghua.edu.cn,liq@pcl.ac.cn,wangchaoxixixi@gmail.com
zoulh@sustech.edu.cn,sgb16@tsinghua.org.cn,zhangpei@bupt.edu.cn,jiangy@sz.tsinghua.edu.cn

ABSTRACT

The rapid growth of video traffic imposes significant challenges on content delivery over the Internet. Meanwhile, edge computing is developed to accelerate video transmission as well as release the traffic load of origin servers. Although some related techniques (e.g., transcoding and prefetching) are proposed to improve edge services, they cannot fully utilize cached videos. Therefore, we propose a Learning-based Fuzzy Bitrate Matching scheme (**LFBM**) at the edge for adaptive video streaming, which utilizes the capacity of network and edge servers. In accordance with user requests, cache states and network conditions, LFBM utilizes reinforcement learning to make a decision, either fetching the video of the exact bitrate from the origin server or responding with a different representation from the edge server. In the simulation, compared with the baseline, LFBM improves cache hit ratio by 128%. Besides, compared with the scheme without fuzzy bitrate matching, it improves Quality of Experience (QoE) by 45%. Moreover, the real-network experiments further demonstrate the effectiveness of LFBM. It increases the hit ratio by 84% compared with the baseline and improves the QoE by 51% compared with the scheme without fuzzy bitrate matching.

CCS CONCEPTS

• **Information systems → Multimedia streaming**.

KEYWORDS

DASH, Edge Computing, Reinforcement Learning

ACM Reference Format:
Wanxin Shi*†, Qing Li†, Chao Wang‡, Longhao Zou§†, Gengbiao Shen*¶, Pei Zhang‖, Yong Jiang*†. 2022. Learning-based Fuzzy Bitrate Matching at the Edge for Adaptive Video Streaming. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3485447.3511944

Corresponding author: Qing Li (liq@pcl.ac.cn).

1 INTRODUCTION

Video streaming makes up a great portion of Internet traffic. According to a Cisco report [10], video streaming will take up 80% of Internet traffic by 2022. Although the network infrastructure is upgraded, the requirement for better user experience still brings a significant challenge.

There are two significant factors influencing user experience during the video transmission, i.e., network conditions and terminal characteristics. **Firstly**, the content delivery of high-quality videos (e.g., 4K/8K) calls for more network capacity. However, the limited bandwidth and unstable network affect the user experience which can be measured by objective metrics such as bitrate level, bitrate switching, rebuffering time and startup delay. **Secondly**, terminal characteristics (e.g., screen resolution and network access mode) also play an important role in QoE. On the one hand, the playback devices with different resolutions are compatible with different video qualities. For example, if a high-quality video (e.g., 1080P) is watched on a mobile phone with a screen of low resolution (e.g., 720P), it will consume more network capacity for transmission with slight improvement in QoE. On the other hand, terminals may have different network access modes, resulting in different transmission rates and network dynamicity. For instance, mobile phones, tablets and laptops access the Internet via a relatively unstable wireless connection and acquire lower transmission capacity. Whereas, televisions and desktops usually access the Internet via a relatively stable wired connections. These different features make video transmission more challenging.

To optimize video transmission, researchers and developers explore and exploit edge computing. However, the storage capacity of the edge is limited and not all the presentations of videos are cached. Cache hit occurs only if the presentation with the requested bitrate level is cached. In this paper, we argue that the cached video with a different bitrate level (i.e., fuzzy bitrate matching) can also be used to respond to user requests. For achieving better user experience, the edge needs to decide whether to request the video with a precise bitrate from the origin server or use the fuzzy-matching video in the cache, which is a significant challenge due to the dynamic user requests, edge states and network conditions.

To fit into the dynamic network and be compatible with terminal characteristics, we propose a Learning-based Fuzzy Bitrate Matching scheme (***LFBM***) at the edge. The main contributions of our work are as follows.

- We propose the fuzzy bitrate matching to reveal its benefit from utilizing cached contents. It can avoid drastic bitrate oscillation and improve hit ratio.
- We design a learning-based fuzzy bitrate matching (LFBM) with user and edge states, to decide whether to respond with a different representation of the requested video chunk.
- We utilize an LFBM-based cache policy to be compatible with the proposed response scheme and verify the efficiency of the comprehensive scheme.

To evaluate the performance of *LFBM*, we implement the prototype based on the open-source Apache Traffic Server (ATS) [2] with around 3500 lines of added code that has been shared on Github [3]. Based on the implemented prototype, we conduct the experiments in the lab environment and further deploy it on the Internet. In order to use the WAN path of the real Internet, we deploy the origin and edge servers on Australian and North American nodes on the Amazon EC2 cloud respectively. In addition, the clients are deployed on the Planetlab [9]. In the simulation, compared with the baseline, *LFBM* achieves 228% of cache hit ratio. Besides, compared with the scheme without fuzzy bitrate matching, it improves the QoE by 45%. Moreover, the real-network experiments further demonstrate the effectiveness of *LFBM*, it increases the hit ratio by 84% compared with the baseline and improves the QoE by 51% compared with the scheme without fuzzy bitrate matching.

The rest of the paper is organized as follows. Related work and motivation are respectively described in Section 2 and Section 3. The framework and concrete design of *LFBM* is presented in Section 4. While the experiments and conclusions are provided in Section 5 and Section 6.

2 RELATED WORK

Adaptive Bit Rate (ABR). The research on ABR for Dynamic Adaptive Streaming over HTTP (DASH) can be divided into three categories: 1) the rate-based schemes (e.g., [31]) that choose the most appropriate bitrate according to the predicted available bandwidth. The accuracy of prediction may impact bitrate selection and finally influence QoE. 2) the buffer-based schemes (e.g., [19, 30]) that make decision of bitrate selection according to the current buffer occupancy at the client. These schemes rely on a manual tuning of parameters. It may be difficult to adapt to highly dynamic network. 3) the hybrid schemes that take into account both network capacity and client-side buffer occupancy [4, 40]. Reinforcement learning is also applied to improve the hybrid scheme for its intelligent decision-making ability [18, 26]. Generally, ABR algorithms may lack a global view and be overdependent on the accurate estimation of available bandwidth [23].

Transcoding. To exploit the computing resources at the edge, transcoding is utilized [25, 29, 37] for saving network bandwidth. Content providers transcode videos into multiple representations and choose an appropriate one for the end user. However, this approach limits users to select bitrate levels from a given set, which cannot adapt to the continuously changing network. Meanwhile, most requests concentrate on a few videos while most computing resources are consumed by massive rarely-requested videos. To solve the problems mentioned above, online transcoding is developed. Some works [14, 20, 36] take network conditions and user states into account for transcoding online. But online transcoding requires great computing capability of both hardware and software. So it is difficult to widely deploy the online-transcoding function at the light-weight edges.

Prefetching. Prefetching can also optimize video transmission [13, 15, 17, 27, 32, 39] by fetching the content in advance. Prefetching is mainly divided into two categories: space-based and time-based. Space-based prefetching is generally applied to web objects by hyperlinks [13, 39]. It also plays an important role in video recommendation. As for time-based prefetching, it utilizes the idle network to prefetch corresponding content. The sequential and chunk-based characteristics of DASH are well fit into the time-based prefetching. However, prefetching consumes additional bandwidth, which may be a waste of transmission resources if the prefetched content is not used.

Cache Schemes. To fully utilize the storage capacity of edge servers or client-side devices, various cache policies are proposed, e.g., the TTL-based scheme [21], the utility-based scheme [12] and TLRU [6]. Some learning-based schemes, such as [5] and [41], are designed to meet the needs of users adaptively. For ensuring the QoE, some QoE-based cache schemes are developed such as [24] and [35]. However, even if these solutions perform well, cache miss may still occur due to the limited storage space. Besides, cache hit may also bring bitrate oscillation.

3 MOTIVATION

3.1 Bitrate Oscillation

ABR algorithms adapt to dynamic network but may not react promptly to the possible influence of edge cache. For example, Lee et.al. [23] found that cache hit affects the throughput estimation of video transmission. To explore the existence of bitrate oscillation, we deploy the origin and edge on cloud servers. The video is accordingly pre-encoded into five bitrate levels (i.e., 350, 600, 1000, 2000 and 3000 Kbps) with H.264/MPEG-4 codec [38]. The 1000-Kbps one is cached at the edge in advance. If the client requests a 1000-Kbps video chunk, it will bring cache hit. Besides, the bandwidth between the origin and the edge is less than 2.0 Mbps.

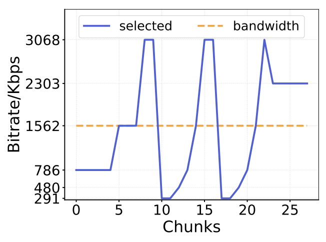


Figure 1: Bitrate oscillation

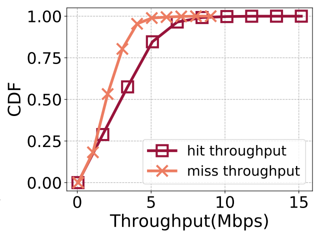
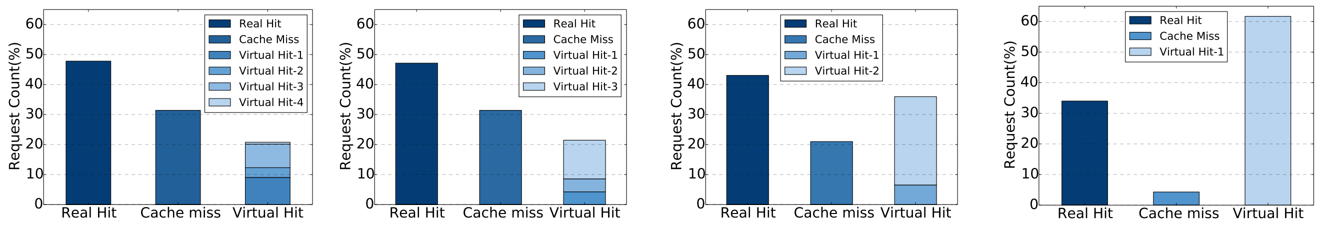


Figure 2: Throughput CDF



(a) No limit (b) Three bitrate levels (c) Two bitrate levels (d) One bitrate level

Figure 3: Rate of virtual hit

The client utilizes Google Chrome to play the video with the default ABR in dash.js [1], an open source JavaScript library for DASH. As Fig. 1 shows, firstly, the client requests the video chunks with low bitrate levels to fill the buffer as soon as possible. The client increases the request bitrate gradually and the cached one triggers cache hit. Then the estimated throughput at the client will be higher due to cache hit, which results in the requests for a higher bitrate level. However, since the chunks with the bitrate higher than 1000 Kbps are not cached at the edge, they will be fetched from the origin server. Meanwhile, the perceived throughput at the client will decrease due to the low capacity of the backhaul path, which forces the client to decrease the request bitrate. Consequently, the continuous bitrate oscillation destroys QoE.

The experiment confirms that the client-side ABR combined with a simple edge-side response policy fails to ensure the QoE. Because the client chooses the representations according to the perceived throughput without considering the cache states at the edge. As Fig. 1 shows, the client requests a high bitrate level under cache hit and turns to a low one under cache miss. So there is necessity to consider client information, edge states and network conditions to avoid the bitrate oscillation.

3.2 Virtual Hit

Here we define ***Virtual Hit*** to indicate the situation that the edge server responds to clients with other bitrate levels instead of the requested one. Opposite to *Virtual Hit*, the regular response with the requested bitrate is *Real Hit*. To verify the feasibility of the response policy based on *Virtual Hit* , we explore user requests and calculate the proportions of *Real Hit* and *Virtual Hit* with the utility-based cache policy. The cached video chunks at the edge are of the highest utility related to benefit and cost. The benefit refers to the requested frequency $REQ(\cdot)$ of a video. The cost indicates the size $SIZ(\cdot)$ of the video. f_m represents the video f with the bitrate m. The utility of the video is formulated as follows.

$$utility = \frac{benefit}{cost} = \frac{REQ(f_m)}{SIZ(f_m)} \tag{1}$$

We set 80 clients to play the videos for 3600 seconds, following a Zipf distribution. There are 100 testing videos encoded into five bitrate levels (204GB totally) in the origin server. Besides, the edge server with a 10GB cache executes the utility-based cache policy. As Fig. 3 shows, there are four instances with different numbers of the other optional representations. Fig. 3a depicts the request proportion of *Real Hit*, cache miss and *Virtual Hit*, without limiting the number of bitrate levels at the edge. While the other three subfigures, Fig. 3b, 3c and 3d, depict that the edge can cache other three, two and one optional representation(s) respectively. That is, if cache miss occurs, the edge server can use the other alternate bitrate level(s) to serve end users. For example, **Virtual Hit-2** means that two alternate bitrate levels are cached and can be utilized. If the edge server cannot respond with any cached content, i.e., cache miss, it will directly fetch the content from the origin server.

It can be concluded that *Virtual Hit* contributes to cache hit. Besides, if more representations are cached for the same content, it will bring more *Real Hit*. The reason is that the clients adapt to the dynamic network by requesting a more proper bitrate level as the number of the available representations increases. However, it also leads to less *Virtual Hit* and more cache miss. We regard that it is of importance to keep a balance between *Real Hit* and *Virtual Hit*, which plays a key role in fully utilizing edge resources and ensuring QoE. So we propose a learning-based fuzzy bitrate matching scheme to solve the problem.

4 DESIGN OF LFBM

4.1 Reinforcement Learning Edge

Based on the reinforcement learning, the edge decides whether to respond with another cached representations, i.e., Virtual Hit. Or it will fetch the exactly requested one from the origin server. The RL edge consists of three layers as Fig. 4 shows: a base layer, a decision layer and a record layer. The base layer defines the fundamental functions of the edge server. It executes the command of responding to users with the cached representations or fetching the content from the origin server in case of cache miss. The decision layer is the intelligent core of the system, making the decision of how to respond based on the information collected from the record layer. Regarding the record layer, it is an auxiliary to the decision layer, collecting user states and network conditions. The workflow of the *LFBM*-based edge is as follows.

(1) The HTTP request arrives at the edge server and will be handled by the cache manager.
(2) The client information collector parses the request header and extracts the playback information.
(3) The cache manager retrieves the requested content from the cache store. Cache hit will trigger Step 10). Otherwise, Step 4) will be executed.
(4) If other representation(s) of the requested content is/are cached, the cache states will be sent to the RL agent for decision making.
(5) The client and network information collectors respectively send user states and network conditions to the RL agent.

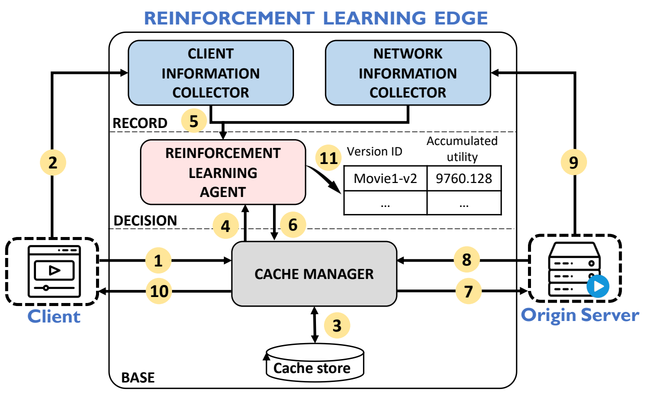


Figure 4: LFBM architecture

(6) The RL agent decides whether to fetch the chunk from the origin server by Step 7) or respond with a substitute representation by Step 10).
(7) If there is no other bitrate level in the cache store or the RL agent decides to fetch the chunk from the origin server, the request will be sent to the origin server.
(8) The origin server sends back the requested content via the backhaul path.
(9) The network information collector gathers the throughput of each session and regards the maximum value as a measurement of network condition (i.e., the throughput under cache miss).
(10) The cache manager sends back the chunk from the cache store or the origin server to the user.
(11) The output of the neural network model will be used to calculate the QoE-related utilities for the cache replacement.

4.2 Regression-based Throughput Prediction

In our work, the throughput of delivering the same chunk under both cache hit and cache miss is necessary for the learning process. However, in one session, the client-side perceived throughput of the same chunk is under either cache hit or cache miss, depending on the cache state at the edge. So we utilize regression methods to calculate the other throughput under cache hit/miss in case of one-time sessions. Before predicting the throughput, we conduct an experiment of playing videos in the Internet to reveal the correlation between the throughput and the other features that are listed in Table 1. The analysis of the collected information helps us figure out the feasibility of throughput regression. The Pearson correlation coefficient $\rho(\cdot)$ is used to measure the correlation between two variables, e.g., X and Y:

$$\rho(X,Y) = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{2}$$

where σ_X and σ_Y represent the standard deviation of X and Y respectively.

Table 1: Term definition

Term	Definition
MTP	the throughput under cache miss at the client
HTP	the throughput under cache hit at the client
SIZ	the size of video chunks
totalMTP	the total throughput under cache miss at the edge in one period
maxMTP	the maximum throughput under cache miss at the edge in one period
avgMTP	the average throughput under cache miss at the edge in one period
reqCT	the number of cache miss occurring at the edge in one period
mTime	the download time under cache miss
hTime	the download time under cache hit
RTT	the round trip time between the client and the edge

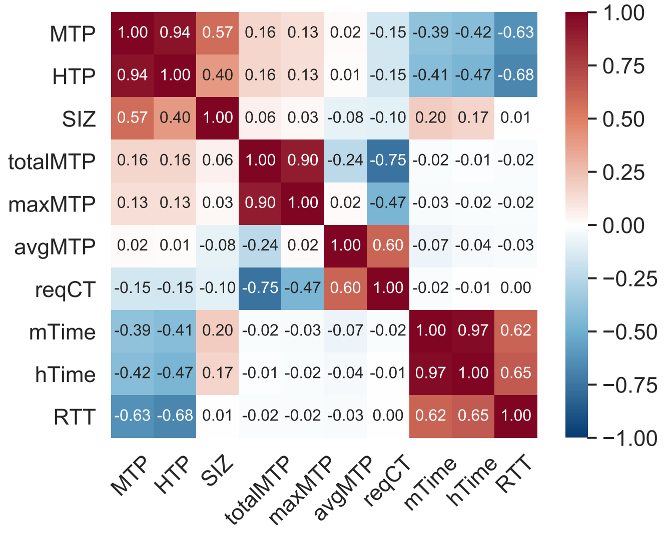


Figure 5: The correlation coefficient between features

Fig. 5 illustrates the correlation coefficients of any two features. The lighter/darker color indicates the stronger negative/positive correlation. Intuitively, *MTP* and *HTP* have the most positive correlation, which assures the feasibility of calculating one of them from the other. Besides, *SIZ*, *hTime* and *mTime* are also highly correlated to *MTP* with the coefficients *0.57*, *-0.42* and *-0.39*, respectively. Because the throughput is an estimated value with the chunk size and download time: $TP = \frac{SIZ}{TIME}$. Another four features are also collected including *totalMTP*, *maxMTP*, *avgMTP* as well as *reqCT*, to estimate the traffic volume under cache miss. In Fig. 5, *reqCT* is negatively correlated to *MTP* with the coefficient *-0.15*. Because the number of cache miss can show how fierce the competition among the clients is. Meanwhile, the client-side throughput decreases due to the competition among the clients. So it is presumable that *reqCT* corresponds to *MTP* and *HTP*. Besides, the correlation coefficient between *RTT* and *MTP* is *-0.63*, confirming that high latency leads to low throughput. According to the analysis above, the following five features should be taken into account for the throughput regression: *mTime/hTime*, *SIZ*, *RTT*, *reqCT* and *MTP/HTP*. Note that, when *MTP* is the input of the regression-based throughput prediction, *HTP* will be the output, and vice versa.

There are multiple regression methods for the throughput prediction. We conduct the experiments to evaluate the efficiency of these methods. Table 2 and Table 3 present the regression efficiency of predicting *MTP* and *HTP* respectively, including Mean Square Error (MSE) and the convergence time of running 200 test traces with/without *RTT*. As Table 2 shows, Light Gradient Boosting Machine *(LightGBM or LGBM)* outperforms the other methods, converging more quickly and obtaining higher precision. Because it is a gradient boosting framework using the tree-based learning algorithms to achieve higher efficiency [22]. *XGBoost* [8] also suffers a low loss in precision but takes more time than *LGBM* does. Although *Lasso* [33], *Ridge* [16], *DNN* and *Linear* have a faster convergence speed, their results are of low precision. Table 3 shows the similar results. Among these algorithms, *XGBoost* achieves the lowest MES but converges very slowly. To balance MSE and convergence time, we choose *LGBM* to predict the throughput under cache hit & miss (as shown in Fig. 2), which is helpful to a more reliable bitrate selection.

4.3 Intelligent Response

The fuzzy bitrate matching model is trained by Asynchronous Advantage Actor-Critic (A3C) based reinforcement learning, including the actor and critic networks. A3C breaks the data association through the asynchronous mode, which utilizes multiple threads to execute multiple agents for exploring and maintaining one target. It is time-saving and resource-efficient for utilizing the raw data directly without processing the input features specifically.

State space. Fig. 6 illustrates the concrete operation process of the intelligent response with fuzzy bitrate matching. The RL agent takes the client information and network state as the input of two neural networks, i.e., the actor and critic networks, at the moment t corresponding to the kth chunk of the video. These inputs include the currently requested bitrate B_k, the last requested bitrate Bl_k, the client-side buffer occupancy Buf_k, the throughput of downloading the last video chunk under cache hit Dt_k^h, the throughput of downloading the last video chunk under cache miss Dt_k^m and the workload of the backhaul path Ld_k. Note that the throughput is predicted with *LGBM*. Ld_k is the maximum throughput under cache miss of the backhaul path. These features are unified as a state vector $s_k \in S_k$ that corresponds to $s_t \in S_t = \{B_k, Buf_k, Dt_k^h, Dt_k^m, Ld_k\}$ at the moment t. Then the processed data are input into the neural networks for training the prediction model.

Table 2: The regression efficiency under cache miss

Algorithm		*LGBM*	*XGBoost*	*Lasso*	*Ridge*	*DNN*	*Linear*
MSE	RTT	**0.0349**	0.0349	0.0483	0.0484	0.0509	0.0484
	No RTT	**0.0351**	0.0352	0.0483	0.0484	0.0512	0.0484
Test time (s)	RTT	**14.681**	30.838	0.0019	0.0009	0.0028	0.0019
	No RTT	**14.172**	28.168	0.0009	0.0010	0.0027	0.0019

Table 3: The regression efficiency under cache hit

Algorithm		*LGBM*	*XGBoost*	*Lasso*	*Ridge*	*DNN*	*Linear*
MSE	RTT	**0.0629**	0.0615	0.1790	0.1791	0.2894	0.1793
	No RTT	**0.2057**	0.1909	0.2369	0.2371	0.3316	0.2375
Test time (s)	RTT	**12.527**	35.240	0.0002	0.0029	0.0023	0.0023
	No RTT	**11.090**	30.492	0.0017	0.0019	0.0025	0.0019

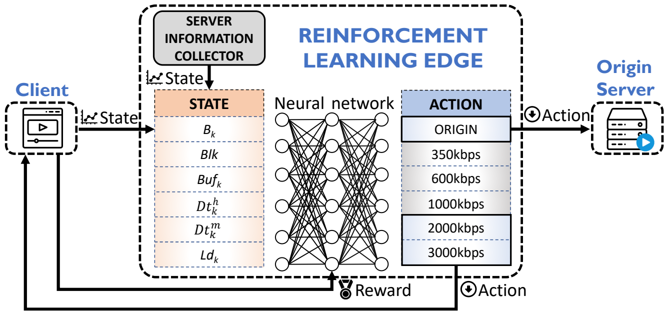


Figure 6: RL model in fuzzy bitrate matching

Action. In accordance with s_t, the RL agent generates the action a_t that benefits the client most. The action $a_t \in A$ indicates whether to respond with the exact requested bitrate (i.e., fetching from the origin server) or the other cached representations. The agent will choose the most appropriate bitrate if directly responding with the cached contents. However, the action decision making is complicated due to the necessity of balancing the interests of all parties. So the executed actions are based on $\pi(s_t|a_t) \in [0, 1]$, indicating the probability of taking the action a under the state s at the moment t. $\pi(\cdot)$ is notated with the gradient parameter θ as $\pi(a_t|s_t; \theta)$ to reveal the optimization process of the actor network. The actor network is updated in accordance with the critic network till θ converges. For more details about A3C, please refer to [26]. We do not traverse all state-action combinations but use the generalization capacity of neural network to evaluate different actions.

Reward. The critic network updates $v(s_t; \omega)$ that denotes the current state value to support the action decision making. The gradient parameter ω is to reveal the optimization process of the critic network. Besides, the historical decisions are of great importance for the future prediction. So the previous action rewards are also included in the learning process. Here, we details the definition of reward. It is known that client-side video players commonly evaluate ABR algorithms to promote their own QoE. From the perspective of the edge server, cache hit ratio is important for improving the response efficiency and finally optimizing the QoE. Therefore, *LFBM* defines the reward of applying a certain action for the kth video chunk as follows.

$$R_k = QoE_k - I \times (log(\frac{B_k}{B_{min}}) + d) \tag{3}$$

where I is set to 0/1 under cache hit or miss respectively. It is more stressful for the origin server to respond to users with a higher bitrate level. Thus, the negative influence is regulated as $log(\frac{B_k}{B_{min}})$. Besides, d represents the punishment due to the cache miss of the lowest bitrate level.

We consider a general QoE metric used in Pensieve [26] and MPC [40]. In our definition, we consider the QoE contains three parts, including the video quality, bitrate variation and rebuffering duration. The QoE metric is defined as:

$$QoE_k = log(\frac{B_k}{B_{min}}) - |log(\frac{B_k}{B_{min}}) - log(\frac{B_{k-1}}{B_{min}})| - \mu T_k \tag{4}$$

Where $log(\frac{B_k}{B_{min}})$ indicates the quality of the video chunk with the bitrate B_k comparing against the lowest-bitrate representation. T_k is the rebuffering time during the playback of the kth chunk. μ is rebuffering penalty compared with video quality decrease and bitrate variation. The learning model can be modified and updated according to the definition of QoE.

LFBM-based cache policy. In Section 3.2, we adopts a utility-based cache policy. To improve the performance of *LFBM*, we propose a cache policy based on the probability distribution of actions in reinforcement learning. We redefine the utility as follows.

$$utiity\_RL = \frac{benefit}{cost} = \frac{\sum_{s=1}^{S} VAL(f_m)}{SIZ(f_m)} \tag{5}$$

The $VAL(\cdot)$ is the accumulative probability of a bitrate level given by the neural network model. When the neural network chooses the action, $VAL(\cdot)$ denotes the possibility of selecting the corresponding representation. Hence, the LFBM-based cache policy have the abilities to capture the popularity and potential reward of content. According to the utility, the videos are listed in a descending order in one period. The cache replacement is executed in accordance with the utility periodically. The results of the experiments in Section 5 demonstrate that the presented cache policy improves the QoE and hit ratio.

5 EXPERIMENT

5.1 Experiment Setup

To evaluate the performance of *LFBM*, we implement a prototype based on the open source of Apache Traffic Server (ATS) [2]. Based on the prototype, the experiments are carried out in the lab environment and the real Internet. *LFBM* is compared with four alternatives:

- A policy without fuzzy bitrate matching (UTILITY)
- A lower bitrate matching approach (LOWER) [36]
- A closest bitrate matching approach (CLOSEST) [24]
- A highest bitrate matching approach (HIGHEST).

The testbed in the lab experiment takes 2000 network traces as the input from HSDPA [28], FCC [11] and a self-collected dataset EC2P. The trace of HSDPA is from the cellular network, whose bandwidth is low and unstable. The FCC dataset is from the fixed broadband, which is more stable and high-speed. We also construct a dataset (EC2P) through collecting the throughput traces between the server deployed on Amazon EC2 and the PlanetLab nodes [9]. The traces from FCC and HSDPA are used as the throughput under cache hit, between the clients and the edge server. Accordingly, we utilize regression methods to calculate the throughput under cache miss, which will be applied into the training process and evaluation experiments. Fig. 7 shows the differences of these bandwidth traces. In another aspect, Fig. 7 also illustrates the bias between the throughput under cache hit and that under cache miss. Finally, we make a DASH video dataset with various types of videos. The dataset contains 50 videos whose length are 5 to 45 minutes. These videos are encoded into multiple representations with the size of 80GB. And the cache size of the edge server is set to be 10GB.

The bitrates of these representations include 350, 600, 1000, 2000 and 3000 Kbps, following the size distribution as Fig 8. Each video chunk lasts for 4 seconds. Different chunks with the same bitrate levels vary in size as Fig 8 shows. Zipf distribution is proved credible in the modeling of video popularity [7]. Therefore, we assume the video popularity following the Zipf distribution, which means that several popular videos contributes most of the requests. In the experiment, the client sends video requests with a probability in accord with the video popularity as Fig 10. On the client side, we adopt a rate-based ABR algorithm which uses Harmonic Mean for throughput prediction. The client buffer is able to contain up to 30-second video. The video will not play until the player accumulates 12-second video in the buffer. So the user experiences the start-up phase. The CDF of RTTs between the users and edge is shown in Fig. 9. The RTTs are varied depending on the distance and bandwidth between the users and edge. So the schemes are evaluated in diverse network conditions. The rebuffering penalty in QoE is generally decided by video bitrate. In the experiment, we set μ as 2.14, which is $log(\frac{B_{max}}{B_{min}})$. B_{max} indicates the maximum video bitrate, and is set to be 3000Kbps. Similarly, the minimum

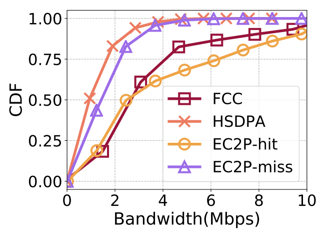


Figure 7: Bandwidth CDF

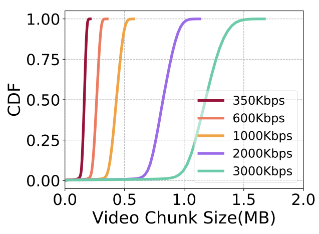


Figure 8: Segment CDF

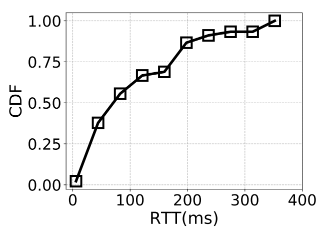


Figure 9: RTT distribution

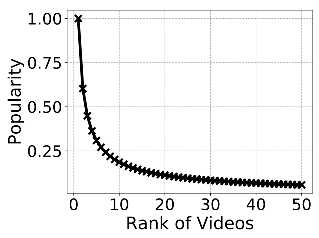


Figure 10: Request pattern

Table 4: Comparison of different schemes

Scenario	Metrics	LFBM	UTILITY	CLOSEST	HIGHEST	LOWER
Lab	Bitrate	0.87	0.71	0.05	1.01	0.66
	Rebuffer	0.004	0.0006	0.13	0.35	0.0005
	Variation	0.05	0.07	0.02	0.03	0.03
	Hit Ratio	0.59	0.18	0.63	0.64	0.24
	QoE	0.74	0.47	0.51	0.14	0.45
Internet	Bitrate	1.13	0.84	1.06	0.97	1.06
	Rebuffer	0.03	0.34	0.19	0.16	0.09
	Variation	0.15	0.04	0.15	0.14	0.15
	Hit Ratio	0.33	0.17	0.30	0.22	0.31
	QoE	0.59	-0.08	0.39	0.29	0.44

bitrate B_{min} is set to be 350Kbps. Obviously, when the video bitrate decreases from maximum to minimum, it exerts a similar negative impact on QoE as one-second rebuffering does.

5.2 Evaluation

The results of different schemes are displayed in Table 4, including the lab and real Internet scenarios.

5.2.1 *Experiment in the Lab.* We conduct two set of experiments to evaluate *LFBM* and the compatible *LFBM*-based cache policy. We analyze the results with multiple metrics, including bitrate distribution, bitrate variation, rebuffering, hit ratio and QoE.

Bitrate Variation. Basically, bitrate plays a key role in video transmission, which influences video quality, viewing smoothness, link utilization, server workload, etc. Without LFBM-based cache policy, *LFBM* exerts similar influence on the request pattern of users' as the other schemes do in Fig. 11a. Comparing Fig. 11a with Fig. 12a, by and large, it can be concluded that clients prefer to request higher bitrate levels after adopting the LFBM-based cache policy. Exploiting the results of bitrate variation in both Fig. 11b and Fig. 12b, all the schemes except *UTILITY* achieve zero-variation by 90%. Although *LFBM* influences bitrate selection of users', it just brings the average 0.05Mbps bitrate variation. Besides, a little bitrate variation is acceptable. For example, in one session, if two successive requested chunks are 350Kbps due to the low capacity of backhaul link, but only one of them is cached. Meanwhile, the alternate 600Kbps chunk is already cached at the edge. In such a case, it is reasonable to respond with the 600Kbps chunk regardless of bitrate variation.

Rebuffering. Fig. 11c exhibits the rebuffering rate of the schemes while Table. 4 includes the statistical rebuffering time in seconds. It can be confirmed that *CLOSEST* does not well fit into the network condition and user requirements. Except *CLOSEST*, the other schemes suffer rebuffering by less than 2%. Compared Fig. 11c with Fig. 12c, none-rebuffering of *UTILITY* increases from 98.4% to 100%. The subtle improvement of none-rebuffering rate benefits the QoE indeed. Therefore, the superiority of the LFBM-based cache policy is also confirmed. To sum up, *LFBM* proves the efficiency of flexible fuzzy bitrate response by achieving the lowest rebuffering rate. LFBM-based cache policy also helps reduce rebuffering time.

Hit Ratio. Fig. 13a and Fig. 14a indicate the average hit ratio for all the users, including *Real Hit* and *Virtual Hit*. Fuzzy bitrate schemes do bring better hit ratio due to *Virtual Hit*. As the baseline, fetching all the cache-miss content from the origin server, *UTILITY* achieves the lowest hit ratio. Without LFBM-based cache policy, the other four schemes achieve approximately 40% hit ratio, which is average because of the relatively long period of cache replacement. In Fig. 14a, LFBM-based cache policy does improve the total hit ratio as well as *Virtual Hit* ratio for some response schemes, except *LOWER* and *UTILITY*. Because LFBM-based cache policy is more compatible with high-bitrate response schemes. Besides, the hit ratio of *LFBM* in Fig. 14a is still inferior to *CLOSEST* and *HIGHEST*. The concrete hit ratio statistics are listed in Table. 4. Because maximizing hit ratio is not the only optimization goal of *LFBM*.

QoE. We respectively plot the average QoE in Fig. 11d and Fig. 12d and the CDF of QoE in Fig. 13b and Fig. 14b. In Fig. 11d, the QoE of *LOWER* is inferior to that of *LFBM* but superior to that of the other schemes due to the limited available bandwidth between the clients and edge server. In such a case, lower bitrate levels can avoid rebuffering, which finally improves QoE. Compared with *UTILITY*, *LFBM* achieves 13% and 56% improvement in QoE respectively as Fig. 11d and Fig. 12d show. Although higher bitrate levels may lead to better QoE, *HIGHEST* is not in line with the expectation because of the limited transmission capacity. Fig. 13b and Fig. 14b illustrate the QoE distribution for clients. Unlike *LFBM*, other response schemes lead to a large number of users acquiring the low QoE because of the unintelligent bitrate selection. Generally, *LFBM* and LFBM-based cache policy do help some users acquire better QoE.

5.2.2 *Deployment in the real Internet.* To further analyze the feasibility and performance of *LFBM*, we utilize Amazon cloud service [34] in Sydney and Virginia, respectively serving as the edge and origin server.

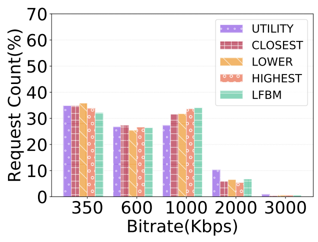
Fig. 15a shows that the most frequently requested bitrate level is 1000Kbps due to the estimated available bandwidth at the client side. Compared with the other schemes, the users with *LFBM* requests the 350Kbps and 600Kbps chunks less. Especially, about 20% of the requests in *UTILITY* is 350Kbps, but only about 10% of the requests in *LFBM* is 350Kbps. Because the users with *UTILITY* still request the lower bitrate levels when the edge server has cached the higher-bitrate chunks for the same content. To sum up, *LFBM* flexibly responds with the cached content.

In Fig. 15b, the performance of *LFBM* is similar to the other schemes. Because the dynamic network state and the difficulty of accurate bandwidth estimation cause bitrate variation. *UTILITY* suffers the highest bitrate variation because cache miss and low bandwidth in the wild lead to drastic bitrate switching. In Fig. 15c, *LFBM* achieves 98% none-rebuffering ratio, outperforming the other schemes. *CLOSEST* and *HIGHEST* result in rebuffering because the responded bitrate levels do not fit into the client-perceived network capacity. *LOWER* leads to less rebuffering but cannot take full use of the available bandwidth. The main reason for the inevitable rebuffering is that the clients set in PlanetLab nodes are too scattered and remote to the edge server.

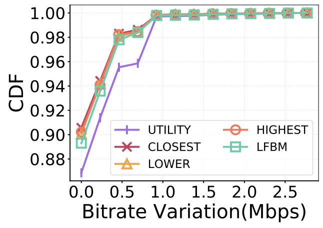
The hit ratio of *LFBM* ranks first in Fig. 16a while *UTILITY* and *HIGHEST* perform the worst. The low hit ratio of *UTILITY* should be attributed to the difficulty of accurate bandwidth estimation and the underutilization of the cached content at the edge, i.e., no *Virtual Hit*. *HIGHEST* also achieves a relatively low hit ratio because the edge with LFBM-based cache policy does not always cache the high-bitrate chunks but the beneficial ones. As shown in Fig. 15d, the overall QoE of *LFBM* is better than the other schemes. Although the QoE of *LFBM* is relatively low due to the startup phase, it keeps growing in the subsequent response process.

6 CONCLUSION

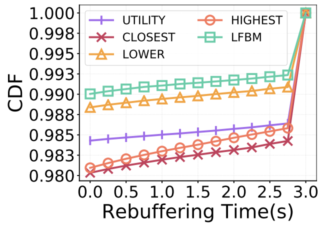
The great volume of network traffic calls for the optimization of adaptive video streaming. Therefore, we propose the Learning-based Fuzzy Bitrate Matching scheme (*LFBM*) at the edge for adaptive video transmission. *LFBM* makes the intelligent decision of either fetching videos from the origin server or responding with the cached content. Through the trace-based lab experiments and the deployment in the Internet, we verify its efficiency and feasibility.



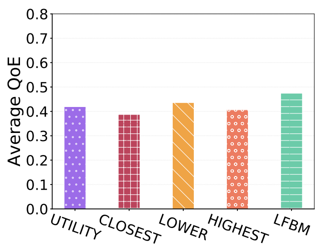
(a) Bitrate



(b) Bitrate Variation

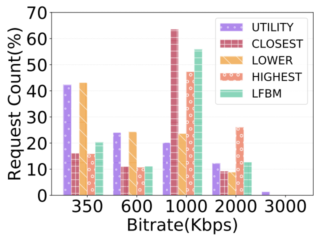


(c) Rebuffering

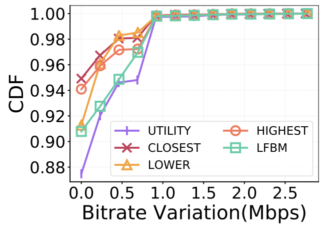


(d) QoE

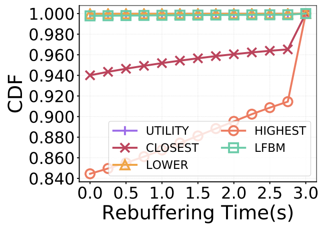
Figure 11: With utility-based cache policy



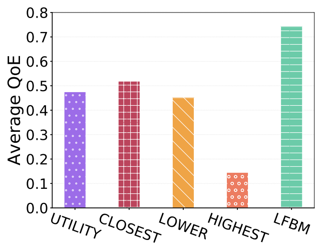
(a) Bitrate



(b) Bitrate Variation

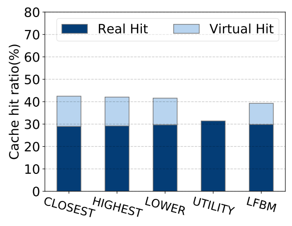


(c) Rebuffering

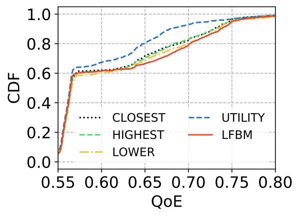


(d) QoE

Figure 12: With LFBM-based cache policy

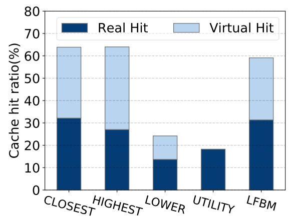


(a) Hit Ratio

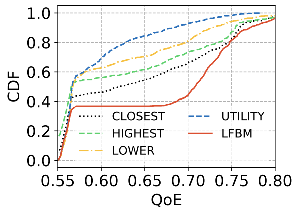


(b) QoE CDF

Figure 13: With utility-based cache policy



(a) Hit Ratio



(b) QoE CDF

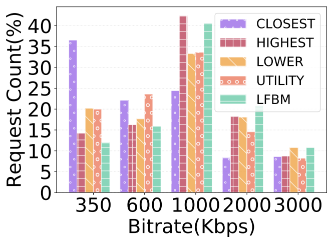
Figure 14: With LFBM-based cache policy

ACKNOWLEDGMENTS

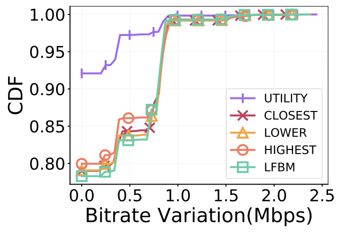
This work is supported by the Key-Area Research and Development Program of Guangdong Province (No. 2020B0101130006), National Natural Science Foundation of China under grant No. 61972189, the Shenzhen Key Lab of Software Defined Networking under grant No. ZDSYS20140509172959989, the Social Science Fund of Shenzhen under grant No. SZ2020D009 and China Postdoctoral Science Foundation under grant No. 2021M700091.

REFERENCES

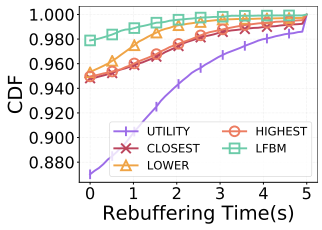
[1] 2016. dash.js. https://github.com/Dash-Industry-Forum/dash.js.
[2] 2018. Apache Traffic Server 8.0. https://trafficserver.apache.org/.
[3] 2019. LFBM. https://github.com/lfbmpaper/lfbm.
[4] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, and et.al Ribeiro. 2018. Oboe: Auto-Tuning Video ABR Algorithms to Network Conditions. In *ACM SIGCOMM*.
[5] Daniel S Berger, Ramesh K Sitaraman, and Mor Harchol-Balter. 2017. AdaptSize: Orchestrating the Hot Object Memory Cache in a Content Delivery Network. In *USENIX NSDI*.
[6] Muhammad Bilal and Shin-Gak Kang. 2014. Time Aware Least Recent Used (TLRU) Cache Management Policy in ICN. In *IEEE ICACT*.
[7] Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. 2007. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *ACM SIGCOMM*.
[8] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A Scalable Tree Boosting System. In *ACM SIGKDD*.
[9] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. 2003. Planetlab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review* 33, 3 (2003), 3–12.
[10] VNI Cisco. 2018. Cisco Visual Networking Index: Forecast and Trends, 2017–2022. *White Paper* 1 (2018), 1.
[11] Federal Communications Commission. 2018. Measuring Broadband America - Eighth Report.
[12] Mostafa Dehghan, Laurent Massoulie, Don Towsley, Daniel Menasche, and Yong Chiang Tay. 2019. A Utility Optimization Approach to Network Cache Design. *IEEE/ACM Transactions on Networking* 27, 3 (2019), 1013–1027.
[13] Josep Domenech, Jose A Gil, Julio Sahuquillo, and Ana Pont. 2010. Using Current Web Page Structure to Improve Prefetching Performance. *Elsevier Computer Networks* 54, 9 (2010), 1404–1417.
[14] Guanyu Gao, Weiwen Zhang, Yonggang Wen, Zhi Wang, and Wenwu Zhu. 2015. Towards Cost-Efficient Video Transcoding in Media Cloud: Insights Learned from User Viewing Patterns. *IEEE Transactions on Multimedia* 17, 8 (2015), 1286–1296.
[15] Chithra D Gracia and S Sudha. 2016. A Case Study on Memory Efficient Prediction Models for Web Prefetching. In *IEEE ICETETS*.
[16] Arthur E Hoerl and Robert W Kennard. 1970. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics* 12, 1 (1970), 55–67.
[17] Wen Hu, Yichao Jin, Yonggang Wen, Zhi Wang, and Lifeng Sun. 2018. Towards Wi-Fi AP-Assisted Content Prefetching for On-Demand TV Series: A Reinforcement Learning Approach. *IEEE TCSVT* 28, 7 (2018), 1665–1676.



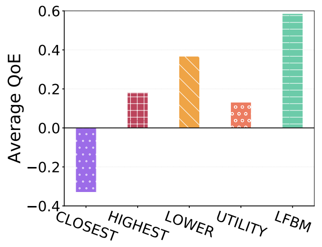
(a) Bitrate



(b) Bitrate Variation

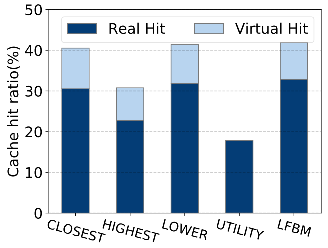


(c) Rebuffering

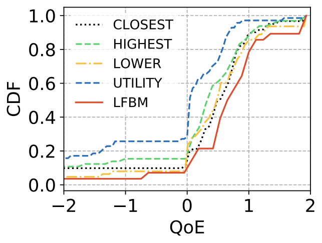


(d) QoE

Figure 15: Comparison in the Internet



(a) Hit Ratio



(b) QoE CDF

Figure 16: Comparison in the Internet

[18] Tianchi Huang, Chao Zhou, Rui-Xiao Zhang, Chenglei Wu, Xin Yao, and Lifeng Sun. 2019. Comyco: Quality-Aware Adaptive Video Streaming via Imitation Learning. In *ACM International Conference on Multimedia (MM)*.

[19] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2015. A Buffer-Based Approach to Rate Adaptation: Evidence from A Large Video Streaming Service. *ACM SIGCOMM Computer Communication Review* 44, 4 (2015), 187–198.

[20] Yichao Jin, Yonggang Wen, and Cedric Westphal. 2015. Optimal Transcoding and Caching for Adaptive Streaming in Media Cloud: An Analytical Approach. *IEEE Transactions on Circuits and Systems for Video Technology* 25, 12 (2015), 1914–1925.

[21] Jaeyeon Jung, Arthur W Berger, and Hari Balakrishnan. 2003. Modeling TTL-Based Internet Caches. In *IEEE INFOCOM*.

[22] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems*.

[23] Danny H Lee, Constantine Dovrolis, and Ali C Begen. 2014. Caching in HTTP Adaptive Streaming: Friend or Foe?. In *ACM NOSSDAV*.

[24] Chenglin Li, Laura Toni, Junni Zou, Hongkai Xiong, and Pascal Frossard. 2018. QoE-Driven Mobile Edge Caching Placement for Adaptive Video Streaming. *IEEE Transactions on Multimedia* 20, 4 (2018), 965–984.

[25] Zhenhua Li, Yan Huang, Gang Liu, Fuchen Wang, Zhi-Li Zhang, and Yafei Dai. 2012. Cloud Transcoder: Bridging the Format and Resolution Gap between Internet Videos and Mobile Devices. In *ACM NOSSDAV*.

[26] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *ACM SIGCOMM*.

[27] Cezar Pleşca, Vincent Charvillat, and Wei Tsang Ooi. 2016. Multimedia Prefetching with Optimal Markovian Policies. *Journal of Network and Computer Applications* 69 (2016), 40–53.

[28] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. 2013. Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications. In *ACM Multimedia Systems Conference (MMSys)*.

[29] Aameek Singh, Abhishek Trivedi, Krithi Ramamritham, and Prashant Shenoy. 2004. PTC: Proxies that Transcode and Cache in Heterogeneous Web Client Environments. *World Wide Web* 7, 1 (2004), 7–28.

[30] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. 2016. BOLA: Near-Optimal Bitrate Adaptation for Online Videos. In *IEEE INFOCOM*.

[31] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. 2016. CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction. In *ACM SIGCOMM*.

[32] Wei-Guang Teng, Cheng-Yue Chang, and Ming-Syan Chen. 2005. Integrating Web Caching and Web Prefetching in Client-Side Proxies. *IEEE Transactions on Parallel and Distributed Systems* 16, 5 (2005), 444–455.

[33] Robert Tibshirani. 1996. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)* 58, 1 (1996), 267–288.

[34] Guohui Wang and TS Eugene Ng. 2010. The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. In *IEEE INFOCOM*.

[35] Yumei Wang, Xiaojiang Zhou, Mengyao Sun, Lin Zhang, and Xiaofei Wu. 2017. A New QoE-Driven Video Cache Management Scheme with Wireless Cloud Computing in Cellular Networks. *Mobile Networks and Applications* 22, 1 (2017), 72–82.

[36] Zhi Wang, Lifeng Sun, Chuan Wu, Wenwu Zhu, and Shiqiang Yang. 2014. Joint Online Transcoding and Geo-Distributed Delivery for Dynamic Adaptive Streaming. In *IEEE INFOCOM*.

[37] Susie J Wee and John G Apostolopoulos. 2001. Secure Scalable Video Streaming for Wireless Networks. In *IEEE ICASSP*.

[38] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra. 2003. Overview of the H.264/AVC Video Coding Standard. *IEEE Transactions on Circuits and Systems for Video Technology* 13, 7 (jul 2003), 560–576. https://doi.org/10.1109/tcsvt.2003.815165

[39] Cheng-Zhong Xu and Tamer I Ibrahim. 2004. A Keyword-Based Semantic Prefetching Approach in Internet News Services. *IEEE Transactions on Knowledge and Data Engineering* 16, 5 (2004), 601–611.

[40] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 325–338.

[41] Chen Zhong, M Cenk Gursoy, and Senem Velipasalar. 2018. A Deep Reinforcement Learning-Based Framework for Content Caching. In *IEEE CISS*.