CoLEAP: Cooperative Learning-Based Edge Scheme With Caching and Prefetching for DASH Video Delivery

Wanxin Shi, *Student Member, IEEE*, Chao Wang, Yong Jiang, *Member, IEEE*, Qing Li, *Member, IEEE*, Gengbiao Shen, *Student Member, IEEE*, and Gabriel-Miro Muntean, *Senior Member, IEEE*

***Abstract*—The outstanding increase in video traffic, puts increasing pressure on network transmission. Since the Dynamic Adaptive Streaming over HTTP (DASH) adjusts the delivery to the dynamic network conditions, it has emerged as a popular approach for video transmissions. However, bitrate switching and video rebuffering may still occur and influence negatively quality of experience (QoE). Additionally the popular videos are transmitted multiple times, which leads to high bandwidth consumption, despite large transmission redundancy. In this context, we propose a Cooperative Learning-based scheme for the smart Edge servers with cAching and Prefetching (CoLEAP) to improve the QoE of adaptive video streaming. CoLEAP employs edge servers which cache the most beneficial contents to reduce redundant video transmissions and prefetches content to decrease network transmission delay. Considering user-related information and the state of network, CoLEAP intelligently makes the most advantageous decisions of caching and prefetching by employing a novel QoE-oriented deep neural network model. To demonstrate the performance of our scheme, we test the proposed solution in comprehensive simulated scenarios and against four alternative solutions. When compared with the existing schemes, CoLEAP increases average bitrate by up to 181.8%, reduces video rebuffering by up to 70.8% as well as decreases response time by up to 28.0%. These values result in minimum improvements of 57.4% and 29.0%, respectively in terms of cache hit rate and QoE.**

***Index Terms*—DASH, caching, prefetching, QoE, edge computing.**

I. Introduction

THERE is an exponential increase in video traffic, which, according to a Cisco report, is expected to exceed 82% of the total IP traffic by 2022 [1]. This video traffic increase puts pressure on existing heterogeneous network environment in terms of its delivery and may as well result in lower viewer quality of experience (QoE). In this context, there is a need for solutions for optimization of video delivery in order to improve user QoE and consequently influence positively user satisfaction and user engagement [2], [3]. The Dynamic Adaptive Streaming over HTTP (DASH) [4] was standardised and is widely being employed to enable highly flexible and dynamic video content adjustment. Diverse client-side adaptive bitrate (ABR) algorithms [5]–[7] were proposed to complement DASH and support in improving the quality of adaptive video delivery. These algorithms can select the most appropriate bitrates for video transmissions, which not only meet user requirements, but also adaptively respond to the rapid change of network state, eventually mitigating the pressure put by the increasing video traffic.

However, even when employing DASH-based solutions, video deliveries still need to overcome some serious challenges. The issues of most concern contain the unstable QoE related to network dynamics and inefficient bandwidth utilization caused by redundant transmissions. First, the highly dynamic Internet can support best-effort content delivery only [8], [9]. For a remote video viewer, the bandwidth fluctuations may result in frequent bitrate switching and even rebuffering, which severely affect QoE and ultimately degrade user satisfaction and engagement [3]. Secondly, video transmissions are strongly correlated temporally and spatially in relation to their content [10]. This double correlation refers to the fact that the end users from some specific area networks tend to request few popular videos during specific time intervals. For instance, 62%-83% of Facebook video transmissions concentrate on its merely top 0.1%-1% videos [11]. Moreover, these clips are always requested during

Manuscript received February 22, 2020; revised July 29, 2020 and September 29, 2020; accepted October 1, 2020. Date of publication October 12, 2020; date of current version October 19, 2021. This work is supported in part by the National Natural Science Foundation of China under Grant 61972189, in part by Guangdong Province Key Area R&D Program under Grant 2018B010113001, in part by the Project "PCL Future Regional Network Facilities for Large-scale Experiments and Applications (PCL2018KP001)", and in part by the Shenzhen Key Laboratory of Software Defined Networking under Grant ZDSYS20140509172959989. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Sanjeev Mehrotra. *(Corresponding author: Qing Li.)*

Wanxin Shi is with Tsinghua Shenzhen International Graduate School, Shenzhen 518055, China, and also with the PCL Research Center of Networks, and Communications, Peng Cheng Laboratory (PCL), Shenzhen 518066, China (e-mail: shiwx17@mails.tsinghua.edu.cn).

Chao Wang is with Tsinghua-Berkeley Shenzhen Institute, Tsinghua University, Shenzhen 518055, China (e-mail: wangchao17@mails.tsinghua.edu.cn).

Yong Jiang is with Tsinghua Shenzhen International Graduate School, Shenzhen 518055, China, and also with Peng Cheng Laboratory (PCL), Shenzhen 518066, China (e-mail: jiangy@sz.tsinghua.edu.cn).

Qing Li is with the Southern University of Science and Technology, Shenzhen 518055, China, and also with the PCL Research Center of Networks and Communications, Peng Cheng Laboratory (PCL), Shenzhen 518066, China (e-mail: liq8@sustech.edu.cn).

Gengbiao Shen is with the Tsinghua Shenzhen International Graduate School, Shenzhen 518055, China (e-mail: sgb16@mails.tsinghua.edu.cn).

Gabriel-Miro Muntean is with the Performance Engineering Laboratory, School of Electronic Engineering, Dublin City University, Galsnevin Campus, Dublin D09, Ireland (e-mail: gabriel.muntean@dcu.ie).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TMM.2020.3029893.

Digital Object Identifier 10.1109/TMM.2020.3029893

peak time (i.e. non-working hours). The redundant transmission of frequently-requested videos contributes to most of bandwidth consumption, inevitably leading to an inefficient use of limited bandwidth resources.

Therefore, optimization of video transmissions continues to attract important research efforts. Based on their principles, existing innovative solutions can be classified in diverse avenues, as follows. A first research direction includes client-side adaptive bitrate adjustment schemes [12]–[14], in which each client making decisions by itself competes for the shared bandwidth [15]. Such solutions may lack global optimization or fairness. The second research avenue focuses on global optimization at the server side. However, it is of difficulty for the server to perceive the whole network state and/or serve all users perfectly in highly dynamic network conditions. The third type of solutions employ cache-based redundancy elimination, including Content Delivery Network (CDN) [16], Information Centric Network (ICN) [17] and cache proxy [18], [19], etc. These techniques have their pros and cons, mostly related to deployment cost, implementation difficulties and function flexibility.

Accordingly, there is necessity to propose a scheme for QoE video delivery improvement with the following properties:

- Optimize globally the video transmission for the clients that share the same bottleneck, instead of allowing each client make decisions locally.
- Reduce content transmission redundancy, which enables that the limited dynamic network bandwidth be utilized efficiently.
- Include mechanisms to reduce network jitter and smooth the viewing experience for end users.
- Perform intelligent differentiation of services for different users and/or videos (including video segments).
- Facilitate cooperation among adjacent edges to reduce unnecessary overhead from redundant transmissions.
- Have commercial sense and be easy to deploy in the current Internet.

In a previous work, we have proposed the smart edge LEAP as a solution to achieve some of the aforementioned goals [20]. However, LEAP does not cooperate with adjacent edges to assist busy backhaul links or utilize the resources available at other edge nodes. Sometimes backhaul links are busy or down, so cooperation between edge servers is beneficial in terms of both supporting services and improving their performance. When the origin server suffers failures, the adjacent edge nodes can act as backup. Additionally fetching content from neighbouring edges helps improve transmission delay.

In this context, the paper introduces **the Cooperative scheme for Learning-based Edge scheme with cAching and Prefetching (CoLEAP)**, which extends LEAP by employing node cooperation and caching in order to improve the performance of video content delivery. CoLEAP considers multiple adjacent edge nodes as part of a large cooperating edge structure, which enables sharing of node storage and supports common utilization of their computing capacity. Similar to LEAP, CoLEAP collects information regarding video sessions from the clients, being aware of video popularity and delivery quality. CoLEAP prefetches the most popular video segments when there is available bandwidth from the origin server. CoLEAP caches video segments from the origin server according to a QoE-weighted popularity. When there is not enough available bandwidth, CoLEAP delivers the prefetched video segments from the edge nodes instead of the origin server and the clients benefit from an improved service quality. The CoLEAP edge nodes cooperate with each other to best share their limited computation and storage resources and support video delivery at improved QoE levels in dynamic network conditions.

This paper makes the following contributions:

- An edge-based scheme for caching and prefetching to reduce video transmission redundancy and mitigate the effect of network jitter.
- A neural network model for the edge nodes to automatically make caching and prefetching decisions such as to maximize the QoE gains by using cooperation.
- A cooperative solution which effectively efficiently utilizes the resources of adjacent edges.

CoLEAP was implemented and tested using simulations in two major scenarios, involving a single and multiple cooperating edge nodes, respectively. The results show how CoLEAP is very efficient in terms of performance as it outperforms significantly other existing approaches in terms of average bitrate, video rebuffering, cache hit rate, transmission duration and QoE levels.

This paper is organized as follows. Section II presents related works. Section III details the network model and some important definitions. The designs of the CoLEAP-related schemes are presented in Section IV. In Section V, the testbed is described and evaluation results are presented. Section VI is the conclusion of the paper.

II. Related Work

Network transmission capacity affects user experience while user experience facilitates the continuous optimization of network transmission capacity. In order to improve user experience in the context of highly dynamic network delivery environments, various schemes are proposed including client-side, server-side and network-based solutions.

A. *Client-Side Adaptive Bitrate Adjustment Schemes*

Lately, there is increased focus on client-side adaptive video bitrate adjustment schemes. The model predictive control algorithm (MPC) [5] assumes accurate prediction of network throughput and utilizes a control-theoretic approach to select the appropriate bitrate at the client in order to achieve the best QoE. Huang *et al.* [21] proposed a buffer-based bitrate selection scheme that explores the influence of buffer occupation on bitrate adaptation. JM Batalla *et al.* [22] exploited a buffer-based tracker to fulfill the guarantees of maximum rebuffering probability. Buffer Occupancy based Lyapunov Algorithm (BOLA) [6] performs local optimization of bitrate selection in a utility-based approach and achieves near-optimal utility by employing the Lyapunov technique. Oboe [7] auto-tunes dynamically the delivery parameters in order to adapt to different network conditions, improve throughput and reduce throughput variability in current network conditions. However, each client makes decisions by itself despite competing for shared bandwidth with other clients and therefore these approaches lack global optimization or fairness.

B. Server-Based Optimization Schemes

The server-based optimization schemes refer to solutions which apply global optimization methods at the server side in order to improve user QoE associated with video transmissions. For instance, the traffic shaping methods in [23] need to maintain the stability and fairness for the users competing for the available bandwidth. The tracker in [24] should manage the clients globally and help them share knowledge with each other. However, assessing the whole network state in real time at the server is extremely difficult, so it is a challenge to support any online global optimization for the fact that decisions are taken remote from end users.

From another perspective, although the server-based approaches are effective, they demand high investment in devices and maintenance due to their computing and storage requirements. S Altamimi *et al.* [25] proposed a server-side QoE-fair rate adaptation method where the learning-based sever consumes great computing resources. The sever-side scheme for 360-Degree video streaming [26] minimizes the overall received video distortion of all users but requires great computing and storage capacity. Therefore, the classic server-based approaches are not strongly recommended mostly due to their coarse-grained optimization and high associated costs.

C. Network-Based Optimization Solutions

Network-based optimization schemes include cache-based solutions, prefetching schemes and hybrid methods. Particularly relevant are some works on caching and prefetching that are vital for improving the performance of DASH streaming.

Cache Schemes: In order to improve the video delivery performance with the limited storage capacity in the network, diverse cache schemes are proposed. The existing cache schemes include offline and online types. Offline schemes adopt some fixed replacement policies without considering the dynamic of clients and servers in a long update period. While online schemes adapt to the real-time dynamic to replace cache content. Some offline cache schemes [11], [27] employ complex algorithms to select and cache the most popular content. In order to improve the response time, online cache schemes are proposed including advanced algorithms or models, e.g., a Markov model based replacement algorithm for popular content [16], a reinforcement learning method based caching method [28] and a real-time dynamic caching for non-popular content [29]. Further improvements were proposed to provide finer granularity caching in a fragmental proxy-caching scheme [30] and to introduce QoE influence into cache replacement policies in some QoE-based cache schemes [31]–[34].

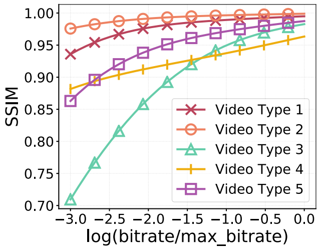


Fig. 1. SSIM: the videos are categorized into five types indicating variance in quality.

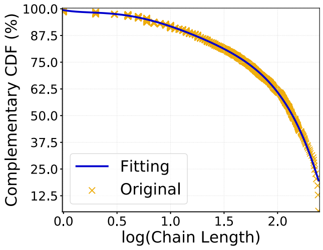


Fig. 2. The distribution of chain lengths: note that chain lengths are denoted in logarithms.

Prefetching Schemes: The current cache-based approaches employ diverse prefetching strategies following a cache miss due to any of the following: first-time request, limited storage space or device failure. First, prefetching was employed mostly for interactive streaming [35]. Now customized video prefetching is widely used. For instance, Google provides a preload-webpack-plugin for web browsers [36], accelerating the loading speed. HotDASH [37] implements a prefetching module in the open source DASH player, which is powered by an optimal prefetching and bitrate decision engine. These works focus on client-side optimization but do not have a global system perspective. Considering network-side HTTP content download, prefetching is performed to the end of the video without considering bitrate level adjustments [13], [14]. Focusing on DASH-based video delivery acceleration, some works construct utility-based models to prefetch content, such as *iPac* [38]. Unfortunately it does not take into account the bitrate switches in the start-up phase [39].

Hybrid Schemes: Lately, following the increasing emphasis on QoE, integrated cache-prefetching schemes have been prevailing [13], [38]. However, despite of the rapid growth of these schemes applied in the network, the existing third party infrastructure is either not practical or smart enough. These schemes are not well compatible with the existing physical infrastructure such as ICNs and CDNs. So it is of great necessity to exploit the potential of smart edges. Smart edges can be regarded as the extension of proxies, working as an intelligent accelerator between client and origin server as Fig. 4. In order to explore the available resources of edges, cooperative caching schemes [40]–[42] are also proposed but without consideration of prefetching.

DASH-aware caching and prefetching schemes are well developed and evaluated. In terms of HTTP-based adaptive streaming (HAS), V Krishnamoorthi et.al [43] quantified the benefits of basic best effort policies and more advanced content quality aware prefetching policies. They proved that policy selection

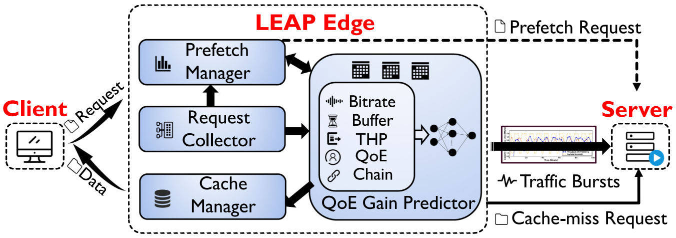


Fig. 3. The architecture of a single smart edge: it works with the client and origin server under the dynamic network environment. The edge includes four key modules detailed in Section IV.

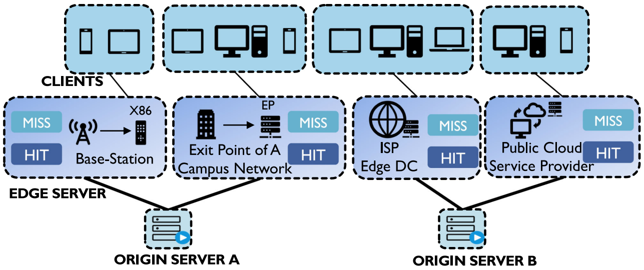


Fig. 4. The architecture of an integrated system: the proposed edge can practically run on the mentioned positions above, serving numbers of users.

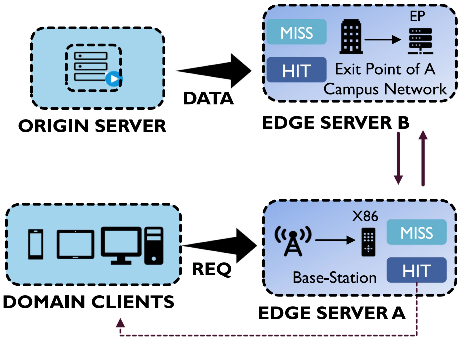


Fig. 5. The cooperative architecture of adjacent edges: each edge serves the covered users with the assistance from the adjacent edge.

is important when trying to enhance HAS performance in edge networks, which also motivates other research on QoE-based prefetching, such as 4 K Video-on-Demand delivery in the mobile network [44]. Besides, S Benno et.al [45] proved the key role of response delay in HAS and the effect of cache on response delay. In summary, these works are enlightening to our research on the cooperative smart edge scheme with caching and prefetching for DASH video delivery, as the architecture of Fig. 5 illustrates, with the purpose of optimizing both user experience and edge performance.

III. Model and Definition

To formulate the problem and solution, we construct the network model and give the concrete definition of notations. We also clarify the definition of QoE which is a key part of defining $Utility$. Besides, the role of $Utility$ playing in our scheme will be illustrated.

A. Network Model

Consider a network model with three types of entities: end users, edge servers and origin servers. End users request video content which by default resides at the origin servers. However the proposed solution CoLEAP employs a cooperative approach, caching and prefetching and involves edge servers to deliver the video content to end users more efficiently. Therefore, the network in our work is defined as a bipartite graph $G_{E,U,O,L_E,L_O,L}$, including the set of edge servers $E = \{e_1, e_2, \ldots, e_N\}$, the set of end users $U = \{u\}$, the set of origin servers $O = \{o\}$, the set of links between end users and edge servers $L_E = \{l_{u,e} | u \in U, e \in E\}$, the set of links between edge servers and origin servers $L_O = \{l_{e,o} | e \in E, o \in O\}$ and the set of links between the edges $L = \{l_{e_i,e_j} | e_i, e_j \in E\}$. Besides, $c_{u,e}(t)$, $c_{e,o}(t)$ and $c_{e_i,e_j}(t)$ represent the transmission capacities of $l_{u,e}$, $l_{e,o}$ and l_{e_i,e_j} at time t, respectively. The proposed scheme requires end users and cooperative edges to supply some local performance-related information to edge servers. For instance, the following information for the end user u requesting the file f_m at time t should be collected such as QoE value $QoE_u(f_m, t)$, buffer length $b_u(f_m, t)$, rebuffering time $r_u(f_m, t)$ and the perceived Round Trip Time (RTT) $\tau_u(f_m, t)$. Since the adaptive algorithm makes the bitrate decision for the next segment sequentially, these four values can be rewritten as $QoE_u(f_m, k)$, $b_u(f_m, k)$, $r_u(f_m)$ and $\tau_u(f_m)$ where k means the moment when requesting the kth segment.

The features of DASH-based video are also illustrated. The set of video files that users can watch is assumed as $F = \{f\}$. Each video f is encoded into $M(f)$ different bitrate versions and split into $K(f)$ segments. In our work, each segment lasts for 4 seconds. Each segment is defined as $f_{m,k}, m \in M(f), k \in K(f)$. The delivery-related values for the requested segments are saved, $f_{m,k}$ they can be used to calculate the value of QoE. As a DASH request arrives, the edge server extracts the required information and calculates the utility with the QoE value. Therefore, we define its utility as $Utility(f_{m,k})$ for the video segment $f_{m,k}$. Then, the prefetching decisions of edge servers are made according to utility values.

B. Qoe

In spite of diverse QoE definitions, its value is influenced by average bitrate, bitrate switching, rebuffering ratio and rebuffering frequency [46] as well as total rebuffering time and start-up stage [5]. QoE definitions describe the experience for a specific period of S segments. Generally, QoE is defined across S segments of video f as follows:

$$QoE_u = \sum_{s=1}^{S} [q(f_{m,s}) - \mu r_u(f_{m,s}) \\ - \lambda |q(f_{m,s+1}) - q(f_{m,s})|]$$

where $q(\cdot)$ indicates the video quality, $r_u(\cdot)$ indicates rebuffering time and $|q(f_{m,s+1}) - q(f_{m,s})|$ indicates the bitrate switching of two sequential video segments. Here QoE_u may depend on several continuous video segments (e.g., 5 or more). Referring to Pensieve [47], we focus on the most essential factors for QoE and

TABLE I
LOGS REQUIRED FOR CALCULATING UTILITY

Content ID	Client	CHL	p(CHL)	QoE Gain	Utility
$R1$	$C1$	2	0.3	5	1.7=0.3*5+0.2*1
	$C3$	1	0.2	1	
$R2$	$C2$	4	0.4	4	1.6=0.4*4
$R3$	$C4$	5	0.8	4	3.2=0.8*4

consider bitrate, rebuffering and smoothness as critical elements. For the user u, the QoE is defined as follows:

$$QoE_u(f_{m,k}) = q(f_{m,k}) - \mu r_u(f_{m,k}) \\ - \lambda|q(f_{m,k}) - q(f_{m,k-1})|$$

where $q(\cdot)$ is used to evaluate the quality related to bitrate. μ and λ are weight factors associated with rebuffering and smoothness, respectively. Here $QoE_u(f_{m,k})$ is only influenced by two continuous segments. Each segment includes a 4-second video clip in our implementation. Two video clips (8 seconds) are able to provide enough QoE knowledge for further decisions in our model. So the defined $QoE_u(\cdot)$ is somewhat different from the conventional one.

In order to provide relevance, $q(\cdot)$ is defined using the structural similarity (SSIM) index instead of simple bitrate value or its variants [5], [6], [47].

SSIM was used as its value reflects better the subjective quality as perceived by users. The larger SSIM value is, the lower compression loss and better video quality are. The SSIM of a video can be formulated as a fourth-degree polynomial of the logarithm of the normalized bitrate as follows, where α is a synthetic representation of the complexity of a video scene, as indicated in [48], [49].

$$SSIM(x) = 1 + \alpha_{v,1}x + \alpha_{v,2}x^2 + \alpha_{v,3}x^3 + \alpha_{v,4}x^4$$

Here $x = log[\frac{B(f_{m,k})}{B_{max}(f_m)}]$ is a logarithmic measure of the normalized bitrate, where $B(f_{m,k})$ indicates the bitrate of the segment $f_{m,k}$ and the $B_{max}(f_m)$ indicates the maximal bitrate of the video f_m. The videos are categorized in one of five types based on their associated SSIM values. These videos include TV serials, movies, documentaries, etc. The video segments from different types with the same bitrate may varies in SSIM that indicates video quality. Fig. 1 shows the SSIM curves for the five video types.

C. Utility

$Utility$ is a comprehensive measure for QoE gain and the being-requested probability of a certain video clip. It influences the decisions of prefetching the next video segment by assessing the benefit of prefetching. The utility associated with the videos is saved at the edge server. The key elements of utility include QoE gain, the probability of bitrate switching and size of the next segment as shown in Table I. Taking $R1$ as a sample, it represents a certain video clip, e.g., $f_{m,k}$. At a certain moment, providing different clients (e.g., $C1$ and $C2$, holding different chain lengths) with the same content (e.g., $R1$) may bring distinct **QoE Gain**s that is elaborated in Section III-C1. **CHL** and p(**CHL**) respectively indicate the chain length of requesting the same bitrate and the probability of keeping a stable chain, which will be further illustrated in Section III-C2. The content with the highest utility is prefetched. Next, we will elaborate in details the utility components used.

1) Qoe Gain: As an important part of the utility, it is employed to reflect the benefit related to fetching the next segment of a video. It also provides input information in the cache replacement process which will be further discussed in Section IV-C. In our scheme, the utility of the probably-requested video segment is updated upon the arrival of new DASH requests, in need of the *QoE Gain* for each segment. In order to calculate *QoE Gain*, we need to determine QoE values of the requested segments if cache hit and miss happens, respectively. For the kth segment in the video f, we assume that the values of QoE if the next segment is cached is $QoE_u^{hit}(f_{m,k+1})$ and if it is not cached is $QoE_u^{miss}(f_{m,k+1})$. Therefore, we describe *QoE Gain* of the probably-requested segment in video f as follows:

$$\Delta QoE_u(f_{m,k+1}) = QoE_u^{hit}(f_{m,k+1}) - QoE_u^{miss}(f_{m,k+1})$$

However, it is very difficult to get the information of $QoE_u^{hit}(f_{m,k+1})$ and $QoE_u^{miss}(f_{m,k+1})$ due to the unknown future network state. We use instead a prediction model to estimate these values. We deploy the prediction model on the edge server and collect user information to be used to calculate *QoE Gain*. A deep neural network is designed to predict the *QoE Gain* for all users, which is detailed in Section IV.

2) Chain-Based Utility: The utility should consider, apart from QoE, also the content popularity to reflect the necessity of fetching the content. The $Utility$ formula, is as follows:

$$Utility(f_{m,k+1}) = \sum_{u \in U} \Delta QoE_u(f_{m,k+1}) \\ \times Popularity(f_{m,k+1})/c(f_{m,k+1})$$

However, the prediction of $Popularity(\cdot)$ for the next segment is usually coarse-grained and not easy to compute. Additionally, in general, $Popularity(\cdot)$ of the content is assessed for a whole video instead of for each segment. Besides, the update period of the content popularity is relatively long, which may be of little use for the timely calculation of utility-based prefetching.

Instead, we focus on the characteristics of content itself. It can be easily concluded that the DASH requests from a client consists of a series of segments with certain bitrate, characterizing bitrate switches. The sequence of the segments with constant bitrate is called a chain. The length of the chain is consistent with the constant requests. Simply according to the historical statistics, the chain length distribution can be obtained [50]. It can be modeled as a piecewise distribution by multiple fitting lines or a special continuous function. As defined, the probability of a chain length larger than x is $p(x)$. With the collected logs from the end users deployed on PlanetLab [38], we propose a 5-degree polynomial approximation as a function of the chain length, i.e. $p(x) = \sum_{i=0}^{5} a_i x^i$, to fit the distribution as shown in Fig. 2.

To clarify the definition of $Utility$, the size of the next segment is defined as $c(f_{m,k+1})$ for the kth segment with a bitrate of m in the file f. Since the video segments have the same duration when employing DASH, segment bitrate represents segment size in the experiment. Therefore, the utility value is namely the expected QoE gain per unit, as follows:

$$Utility(f_{m,k+1}) = \frac{\sum_{u \in U} \Delta QoE_u(f_{m,k+1}) \times p(l)}{c(f_{m,k+1})}$$

where l represents the length of corresponding chain of a single end user, when requesting the kth segment.

To sum up, $Utility$, based on QoE gain and the probability of bitrate switching, influences prefetching decisions in real time. Besides, *QoE Gain* also influences cache replacement as stated in Section IV-C. The concrete use of $Utility$ is further detailed in Section IV-E

IV. CoLEAP Design

A. Overview

Consider a single *Edge* that handles the DASH requests from a *Client* and makes decisions to cache-prefetch the video content from the origin *Server*, based on the QoE. Fig. 3 shows the architecture of the CoLEAP smart edge which consists of four modules including *Request Collector*, *Cache Manager*, *QoE Gain Predictor* and *Prefetch Manager*. *Request Collector* gathers and parses the requests from users and supplies user information to other modules. *Cache Manager* achieves cache update, cache lookup and content reply. *QoE Gain Predictor* utilizes the user information and the network state to predict throughput and *QoE Gain*. *Prefetch Manager* calculates the utility for videos and executes the utility-based prefetch strategy.

These modules cooperate with each other to serve the users covered by the smart edge interactively. This design optimizes the transmission globally instead of making the decision for each client locally, which is beneficial to the efficient utilization of the limited network bandwidth. Additionally, it can achieve intelligent differentiation of services for different users or videos (even video segments). Moreover, the differentiated service does not result in transmission redundancy, but can help the smart edge reduce network jitter in real time, ultimately providing users with a smooth viewing experience.

In order to avoid long public Internet path with no quality assurance, CoLEAP is deployed closer to users at the level of the smart edge server. The corresponding entities should have the motivation to deploy the edge. According to the number of users, the computing and storage capacity of the edge will be different. So the smart edge can be deployed in multiple potential positions as shown in Fig. 4. An option is for the campus or enterprise network to deploy the edge server at the exit point connected to network providers. In this way, edge managers themselves achieve the optimization for their covered users. For example, iQIYI Open Cache Program [51] is exactly an available platform that provides customizable edge functions. Actually, the smart edge can also be compared to a reverse proxy which intercepts the requests before they reach the origin server. This can be achieved by setting up appropriately the DNS entry for the origin server. Alternatively, with the aid of public could, virtualized edge data center of the Internet Service Provider (ISP) and chosen super clients, content providers can deploy the smart edge as a container, virtual machine (VM) or application. For example, content providers, e.g., ByteDance and Youku, utilize the hardware of Internet Data Center (IDC) to serve end users. They depend on the storage and traffic capacity of IDC, which motivates them to customize the edge functions. In this way, the content providers themselves are of necessity to optimize caching and prefetching. It follows that the content provider can both help improve QoE for its users and reduce the required bandwidth.

However, independent edges cannot take full advantage of network capacity. Because certain edges may be too far from the origin server. So even with prefetching, the responses from the origin server to the edge are slow. The backhaul link may not have enough transmission capacity for prefetching or may even break down in some extreme conditions. So it is necessary to make adjacent edges cooperate as illustrated in Fig. 5. The figure shows how fetching content from a nearby edge is more efficient. In our work, we set the first edge e_{first} as the cache server for the geographically concentrated clients. It helps accelerate video transmissions to these users. The second edge e_{second} is an adjacent edge for e_{first}, but not covering the users. If the link between the two edges has high transmission capacity, then e_{first} and e_{second} can work as a *cooperative* cache. However, these two edges cache distinctive content to support the users, being *complementary*. For instance, e_{second} caches the content with high utility, while e_{first} stores content with relatively low utility.

B. Request Collector

In a DASH-based approach, the user performs adaptive video streaming by dynamically selecting the bitrate for next video segments requested. The smart edge collects some key information from the geographically-concentrated DASH requests. When such a DASH request arrives at the edge, the *Request Collector* parses the obtained URL and identify the requested video, and updates the information associated with the requested content. The related modules record or calculate the latest feature values and writes them into these previously mentioned modules including *Prefetch Manager* and *QoE Gain Predictor*. Additionally, the edge periodically updates request information and network status data to train the neural network model which will be discussed later concretely.

As the smart edge takes the QoE-related cache-prefetch decisions, alongside URL, the DASH request needs to provide some relevant information. These requests may originate from either the end user or the adjacent edge. The following information is appended to the HTTP header of each DASH request, including *lastQoE, buffer, bitrate, videoType, chainLength, throughputList_n, downloadTime_n, segmentSize_n* and *ifHit_n, RTT*. In Fig. 9, these variables are respectively abbreviated as *LSTQ, BFR, BTRT, TYP, CL, THP, DWT, IFHIT and RTT*. *Request Collector* extracts this information from the DASH requests for further use. This information only consumes little transmission bandwidth, so the extra overhead in the DASH scenario is low. (e.g., 1 byte for *lastQoE*, 5 byte for *throughputList_n*, 1 byte

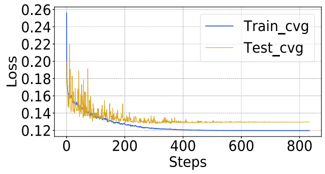


Fig. 6. Prediction convergence.

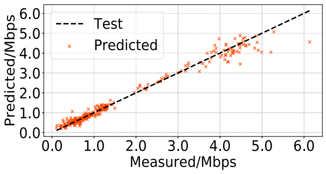


Fig. 7. Throughput regression.

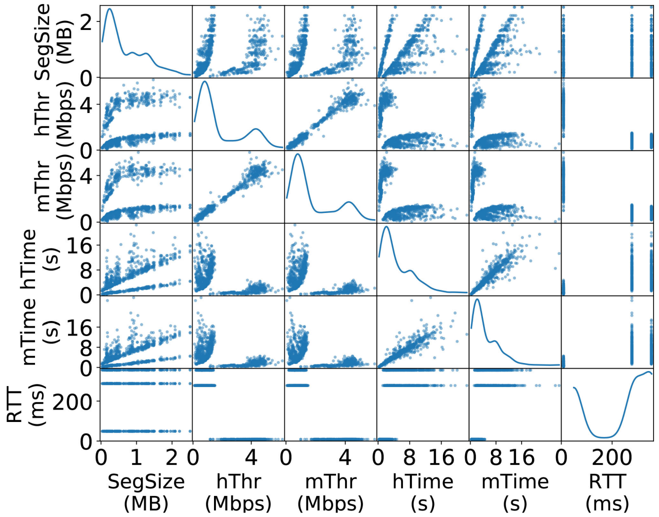


Fig. 8. Scatter matrix of QoE metrics: it shows the correlation between every two metrics. A strong correlation presents regularly-distributed dots.

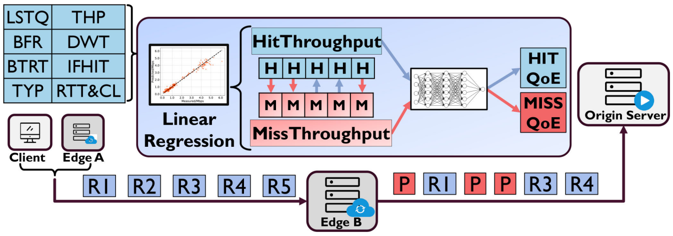


Fig. 9. The deep neural network for QoE prediction: the figure illustrates the operations of the whole predicting process along with generating requests, including collecting client-side information, estimating throughput and predicting QoE gains.

for *ifHit_n*, etc. For instance, n=5, 22 bytes data is sent at each request.)

C. Cache Manager

The cache strategy is relatively independent of the prefetch strategy. This is as caching is performed for the already-requested content, while prefetching is executed for the content which may be requested in the future. The storage space for caching is allocated larger than that for prefetching, due to the relative utilization ratio of content in the two situations. The prefetched content may not be used and be discarded quickly in a short update period of prefetching. So *Cache Manager* performs the designed cache strategy and updates the local cache content during an update period. Replacing different cached segments has a distinct impact on QoE because of their different SSIM values.

The cache update period is defined as T_c. We assume that, for a certain video f during the θth period, there are $H_f(\theta T_c)$ segment IDs in all requests. $h^i_f(\theta T_c)$, in which $i = 1, \ldots, H_f(\theta T_c)$, is the number of requests regarding each segment ID. The average *QoE Gain* for the video segments from historical statistics are maintained and denoted as $\widehat{\Delta QoE}^k_f, k = 1, \ldots, K(f)$. To sum up, the Accumulated QoE gain (AQ) for the received segments in the period can be described as $AQ^i_f(\theta T_c) = h^i_f(\theta T_c) \times \widehat{\Delta QoE}^{x(i)}_f, i = 1, \ldots, H_f(\theta T_c)$, where $x(i)$ is the function to map the ith segment into the real ID.

As illustrated above, AQ is based on the request frequency and QoE Gain, which takes both request characteristics and video features into account. So we regard that AQ is useful to figure out the priority of corresponding videos. If there are multiple contents that have similar priority but without enough storage spaces, we just remove the old ones which were requested early. Even if a small number of cached contents are not so frequently requested, the contents will be updated in the next cache replacement period due to its changing request frequency and QoE Gain.

We design a novel cache strategy which we name **Proportional Accumulated Cooperative QoE gain (PACQ)** to maximize the expected overall QoE gains for CoLEAP. During the cache update period for a domain smart edge, for instance e_{first}, its *Cache Manager* first calculates all values of the AQ gains for the content requested directly by the domain users, i.e. $AQ^i_f(\theta T_c)$. Then the list of $AQ^i_f(\theta T_c)$ is sorted in descending order. After that, *Cache Manager* selects the sorted segments sequentially till it cannot add more contents to the limited cache space. The local cached contents is also checked. Finally, the non-selected segments will be replaced. The periodical update mechanism in *Cache Manager* improves the adaptability of CoLEAP, leading to timely reaction to any dynamic scenario.

At the same time, for an adjacent edge e.g., e_{second}, its *Cache Manager* also calculates the AQ gains for the content requested directly by the clients, as well as AQ gains for the content requested by other edges i.e. e_{first}. $PACQ$ has a tuning parameter η used to adjust the importance of the other edges' requests. $f(u)$ and $f(e)$ are the requests from users and edges, respectively for the file f.

$$PACQ^i_f(sT_c) = h^i_{f(u)}(\theta T_c) \times \widehat{\Delta QoE}^{x(i)}_{f(u)}$$
$$+ \eta \times h^i_{f(e)}(\theta T_c) \times \widehat{\Delta QoE}^{x(i)}_{f(e)}$$

Any CoLEAP edge performs the same process described for e_{first} to replace the items which have the AQ gains. In fact, each edge acts both as primary edge (i.e. e_{first}) and as a cooperative edge (i.e. e_{second}) for other edges. CoLEAP edges work as a large cooperative cache to support the domain clients and offload traffic from the origin servers. This is achieved by edges by dynamically and intelligently storing and serving content using the CoLEAP approach.

To better distinguish different requests, the requests from clients have user labels in the implementation, while those from other edges do not. With the user labels, the edge can execute the differentiated responses to different requests. We regard that $PACQ$ well solves the problem that duplicate contents may be cached by adjacent edge servers. When the edge is relatively idle, it can serve the adjacent edge more by increasing the parameter η, and vice versa. But the η in our implementation is not automatically adaptive, which needs further improvement.

D. Qoe Gain Predictor

Any cache hit is associated with positive QoE gains, since the cache edge server closer to the user than the origin server accelerates the download process as well as decreases the transmission latency. Additionally the edge load may be lighter and network conditions better, so it may provide the user with higher throughput. Therefore, prefetching the segments with high QoE gains is beneficial. However, the actual QoE gains at the moment of content request can only be predicted. We employ a deep neural network approach to predict the QoE gains in two cases. Besides, as network throughput is a vital factor in the QoE gain prediction, we also design a simple prediction model, using linear regression, to forecast future throughput. Next, we elaborate on the two prediction models.

1) Throughput Prediction: Network throughput is greatly influenced by many factors. We have analysed the correlation between the throughput and other factors by plotting the scatter-based correlation diagram in cache hit and cache miss scenarios, as shown in Fig. 8. The data is collected from the nodes on PlanetLab. These nodes act as client players requesting for video segments from the multimedia server deployed on Amazon Cloud. The regular trend in subfigures indicates a strong correlation between two features, while a disperse behaviour reflects weak correlation. It can be concluded that segment size, user-to-edge RTT and download time of video segments are strongly relative to the throughput.

We cannot acquire the network throughput of cache hit when cache miss happens because cache hit and cache miss cannot occur at the same moment, and vice versa. Fortunately, it was noted that there is a strong correlation between the throughput during cache hit and the throughput during cache miss. Thus, linear regression is available to predict throughput in two cases, and the inputs of the regression model are the factors which were identified to have the highest correlation. The fitting results show a good agreement with actual measurement values, as Fig. 7 illustrates. The discrete dots scatter around the standard line, indicating a good performance.

2) Qoe Gain Prediction: QoE gain prediction is performed using a deep neural network approach. QoE gain is defined as the difference between the QoE values associated with cache hit and cache miss, which means that two QoE values need to be predicted. We design a three-layer neural network to calculate two QoE values by instantiating the model twice. The difference between the two instances is in terms of network throughput values, illustrated in Fig. 9. The inputs of the deep neural network model consist of the information obtained from the *Request Collector*. Moreover, the deep neural network adds into the model the predicted throughput of cache hit, calculated by using the historical throughput in cache hit scenarios to determine $QoE_u^{hit}(f_{m,k+1})$. Similarly, it calculates $QoE_u^{miss}(f_{m,k+1})$ by using predicted throughput of cache miss. By subtracting the two values, the QoE gain of the next segment is computed and is used to calculate the corresponding utility.

The QoE gain prediction model is trained at the server side due to the requirement of computing capacity and power consumption. We make use of a back-end server similar to Pytheas [52] to train the model. For deployment in a cache scenario, the edge in CoLEAP is an excellent choice. The edge server is able to train the QoE prediction model, calculate the QoE gain and collect historical statistics with the aid of its computing capacity. In addition, if the prefetched content has already been cached at the edge server, it is not necessary to repeat the calculation of the QoE gain, saving computing resources to some extent.

E. Prefetch Manager

CoLEAP performs the prefetch following utility computation and evaluation. Selecting the prefetch segments that generate the maximum overall benefit result in an optimal result. We design a periodical prefetch process and assume the prefetch period is T_p. The ideal solution involves selecting the prefetch segments such as the sum of the associated utilities is maximised in each period. However, this simple method has two problems. First, a large number of requests received in any period results in excessive maintenance overhead and makes very hard solving the complicated selection process. The second relates to the fact that the fixed period solution inevitably postpones some prefetch decisions and degrades the overall benefit. Hence, there is a need for a more practical and timely method to make the prefetch decision.

This content selection process is similar to the secretary problem [53] which is one of the famous optimal stopping problem. We adopt the classic solution of the secretary problem as it responds to requests dynamically and timely, achieving high efficiency in each period. Note, we do not have to prefetch every content which may be requested, as coarse-grained level selection offers good results. Additionally, caching and prefetching are relatively independent to each other and they both improve the overall results. Caching targets popular content requested in the past and content that may be used in the future. Caching is a process with a long update period and large storage space requirements. Decisions refer to caching the whole video or just certain video segments. Prefetching consumes the transmission capacity and aims at content which may be accessed in a short time in the future. If the prefetched content is not accessed in a short period of time, it will be replaced immediately. Therefore, the update cycle of prefetch is short, and its prefetch granularity is in favor of saving resources. The secretary problem solution is appropriate to the prefetching problem. Once the *Prefetch Manager* has collected enough statistics, it can directly make the prefetch decision for the subsequent requests.

Algorithm 1: p-Tops Mechanism

```
Inputs: $Utility(f_{m,k+1})$, $r(dT_p)$, $Utility^{max}(\eta T_p)$, $n_{count}$
Outputs: $x_f$ - Binary variable denoting the prefetch decision.
 1: $x_f = 0$
 2: if $t \geq (\eta + 1)T_p$ then
 3:     update the value of $r(\eta T_p)$
 4:     $Utility^{max}(\eta T_p) = Utility(f_{m,k+1})$
 5:     $n_{count} = 1, \eta = \eta + 1$
 6: else
 7:     if $n_{count} \leq r(\eta T_p)$ then
 8:         $Utility^{max}(\eta T_p) =$
 9:                 $\max\{Utility^{max}(\eta T_p), Utility(f_{m,k+1})\}$
10:         $n_{count} = n_{count} + 1$
11:     else
12:         if $Utility(f_{m,k+1}) \geq Utility^{max}(\eta T_p)$ then
13:             $x_f = 1$
14:         end if
15:     end if
16: end if
17: return $x_f$
```

We design a novel **Periodical Two-stage Optimal Prefetch Selection mechanism (p-Tops)** to solve the optimal selection problem as shown in Algorithm 1. The details of the p-Tops' two stages are described as follows.

Stage 1: The *Prefetch Manager* performs this stage at the beginning of each period. For the dth prefetch period, the current available bandwidth between the edge server and origin server can be obtained: $c_{e,o}(\eta T_p)$. The obtained average bitrate in the last prefetch period is assumed as $\hat{\pi}(\eta T_p)$. Thus, the possible maximum number of the prefetch segments can be concluded in this period: $\kappa(\eta T_p) = \frac{c_{e,o}(\eta T_p)}{\hat{\pi}(\eta T_p)}$ The weighted average, as a useful approach to alleviate the influence of uncertainty and randomness, is utilize to compute this value, i.e. $\nu(\eta T_p) = \alpha \nu((\eta - 1)T_p) + \beta \kappa(\eta T_p)$, where α and β are weight factors. Similar to [48], it can be figure out that the number of the accumulated segments in Stage 1 is as follows: $r(\eta T_p) = \lfloor \frac{\nu(\eta T_p)}{\kappa(\eta T_p) e^{1/\kappa(\eta T_p)}} \rfloor$. This indicates that the *Prefetch Manager* only gathers request statistics in the first $r(\eta T_p)$ requests to find the maximum utility $Utility^{max}(\eta T_p)$ and does not prefetch segments. This process is detailed in Lines 7-10 of Algorithm 1.

Stage 2: With the collected information in Stage 1, the *Prefetch Manager* implements the utility-based prefetch strategy. The process of filtering prefetch requests is shown in Lines 11-14 of Algorithm 1. Once a request arrives during Stage 2, the *Prefetch Manager* compares the utility of the request with the acquired maximum utility $Utility^{max}(\eta T_p)$ from Stage 1. As defined in Section III, the $Utility_{f'(t)}$ is mapped to the video segment in $Utility(f^{'}_{m,k})$. The possible-to-prefetch segment, i.e. the next segment, will be prefetched if the $Utility(f_{m,k+1})$ is larger than $Utility^{max}(\eta T_p)$.

For more details in **p-Tops**, Lines 2-5 reset the parameters at the beginning of a new prefetch period while Line 17 returns the prefetch decision. By deploying **p-Tops**, we perform sequential prefetching with a minimum influence on delay. Since it is assumed that the bottleneck is between the edge server and the origin server, the number of prefetch decisions in one period does not exceed the upper bound of transmission capacity. In addition, the weighted average method makes full use of the estimated results and historical information, enhancing the performance of prefetch strategy.

For better understanding that how to avoid the overlap of caching and prefetching, we give a brief illustration on the execution. We regard that the cache-miss requests are based on traversing the cache space to ensure no corresponding contents. In fact, the edge makes a quick comparison between the prefetching requests and the cache-miss logs, so as to ensure no overlapping requests. It also ensures that the prefetched contents and to-be-prefetched contents do not exist in the cache space. Besides, the prefetching requests sent during a prefetch period are also recorded to avoid cache-miss requests. We record the prefetching requests by generating a labeled cache-miss log at the edge and dumping it in a prefetch period. The above simple operations are helpful to avoid content redundancy.

However, the problem that the contents being returned may be duplicate with the contents already cached, is difficult to solve in the concurrent network. In spite of it, we believe that the probability of its occurrence is small and the impact of it is little. Because in the next prefetch period, the old prefetched contents will be cleared and replaced depending on the comparison with the real cache-miss requests, which ensures that a content will not be prefetched in an infinite loop or cause a waste of bandwidth. Moreover, prefetching should be executed during the idle time to avoid affecting the response to normal requests. In other words, the inevitable overlapping contents from the concurrent network are trivial. Because the space ratio of caching to prefetching is 1000:1, or even 1000:0.5. Meanwhile, the cache replacement period (e.g., minutes to hours) is generally long, which is not synchronized with the prefetch period (e.g., seconds to minutes). Compared with spending more time and computing resources on redundancy elimination, it works well to wait and update in the next prefetch period. To sum up, we believe that prefetching will not cause serious space waste and bandwidth waste.

V. Implementation and Evaluation

A. Implementation and Setup

1) Implementation: In order to evaluate the performance of our scheme in diverse network scenarios, we deploy CoLEAP in a prototype based on the Apache Traffic Server (ATS) [54]. The associated 4,000 lines of C++ and python code is available open source in Github [55]. ATS uses a state machine to handle HTTP transactions. Based on current states and HTTP transactions, ATS uses HOOKS to call specific plug-ins and perform user-defined functions. That is, the scheme is realized in the form of plug-in which can be flexibly added or removed.

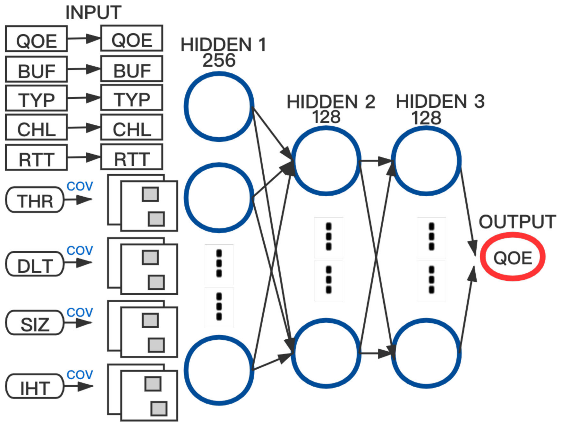


Fig. 10. The architecture of neural network: it is adopted in the prediction of QoE under cache miss/hit.

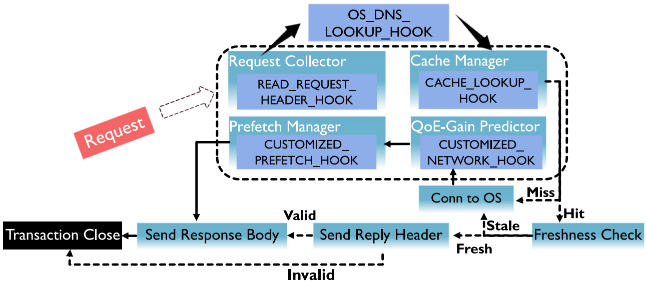


Fig. 11. The correspondence of hooks in ATS: it shows how a request triggers the related actions in ATS which exploits hooks to finish transactions.

The main computing overhead is from training the *QoE Gain* model, which mainly depends on the number of users and requests. Larger user groups, containing more feature information, lead to higher training cost, and vice versa. In our implementation, the server with NVIDIA Tesla M60 (16 GB GPU) is adopted to train the model for hours to support hundreds of clients. We regard that, in different real networks, the model should be modified in accordance with the user scale, thus causing variance in demand for training power. With a well-trained model, the ATS-based plug-in can be recompiled and applied into other networks easily.

Fig. 11 illustrates how CoLEAP is deployed in the ATS context. Four modules of the smart edge is implemented through the plug-ins in ATS. READ_REQUEST_HDR_HOOK parses user requests, achieving the function of *Request Collector*. CACHE_LOOKUP_HOOK implements the *Cache Manager* module to complete cache state feedback and the QoE-based cache replacement. *QoE Gain Predictor*, obtaining the inputs from the caller of Operation System (OS), runs when CUSTOMISED_NETWORK_HOOK triggers it. It is known that Convolutional Neural Network (CNN) is a feedforward neural network including a convolutional layer and a pooling layer. For *QoE Gain Predictor*, the one-dimensional features pass through a fully connected layer with 128 neurons as Fig. 10. The multi-dimensional features pass through a one-dimensional convolution layer with 128 neurons. The convolution kernel size is 3. The past n values of the variables are first processed by the CNN before the hidden layer. The stride and padding of the convolutional network are both 1 respectively. The outputs of the full connection layer and the convolution layer are input into a network composed of three full connection layers. There are three hidden layers which contain 256, 128 and 128 neurons, respectively. Besides, the activation function of the neural network is Relu which has fast convergence and better consistence with the characteristics of biological neurons. In our work, the neural network is implemented in PyTorch with a C++ interface libtorch.

In summary, *QoE Gain Predictor* is a compiled C++ program that is invoked before the *Prefetch Manager* module runs. Then, *Prefetch Manager* deals with prefetch requests and sends the selected ones to the origin server with the aid of the proposed prefetch strategy. The download throughput of the origin server is continuously updated by executing TXN_CLOSE_HOOK.

2) Framework Setup: Next, we will give a complete picture of the evaluation settings. There are five machines simulating client hosts, edge servers and origin server in the experiments. Two client hosts run 80 DASH request programs in total, simulating different users downloading videos. The rate-based ABR algorithm adopted is to choose the maximal bitrate according to the throughput, which is similar to FastMPC [5]. DASH video requests sent to the edge are attached with user state information. Additionally, the distribution of client-side video requests is set conforming to the Zipf law with a parameter, e.g., 0.73.

Two edge servers run ATS to implement the LEAP-based schemes and the origin server serves the video requests with Nginx Web Server [56]. To demonstrate the efficiency of the proposed schemes under different cache sizes, the experiments are executed with 5 GB and 10 GB cache. Additionally, *CoLEAP* has two edges with 5 GB cache each. These two edges have good transmission connectivity. The transmission capacity between the client hosts and the edge server, is set at three levels: good, average and poor by setting the delay on different program ports using Linux Traffic Control tools [57]. The corresponding delay is realized by setting different loss ratio for transmission, i.e. 0, 15%, 30%. As for the origin server, it contains 60 videos which occupy in total about 100 GB. These videos include some popular TV series, movies, documentaries and advertisement videos. The types of these videos are equally distributed across the five SSIM categories.

3) Schemes and Metrics: Six schemes are compared in our experimental evaluation: *CoLEAP*, *LEAP*, *Cache Only*, *Prefetch All*, *iPac* and *No Cache*. *LEAP* refers to a solution which uses the proposed approach, but only one edge adopting smart caching and prefetching strategies to serve the domain clients. *CoLEAP* employs two smart edges to serve users cooperatively. In *LEAP* and *CoLEAP*, one edge has a link to the origin server with a maximum bandwidth of 20 Mbps. In *CoLEAP*, the backhaul link between the second edge and the origin server has a maximum bandwidth of 90 Mbps. *Cache Only* refers to a solution which caches the content with the highest QoE gain. *Prefetch All* employs the aggressive prefetching that keep fetching till the end of videos. *iPac* prefetches two segments for every request according to their utility and *No Cache* performs no caching at all.

TABLE II
COMPARATIVE PERFORMANCE DATA FOR DIFFERENT SCHEMES

Cache Size	Metrics	CoLEAP	LEAP	Cache Only	Prefetch All	iPac	No Cache
10GB	Bitrate	**1650.80**	**787.65**	444.28	461.02	585.88	351.79
	Rebuf	**0.07**	**0.14**	0.24	1.78	0.24	0.84
	Hit Rate	**0.96**	**0.71**	0.39	0.61	0.53	-
	Delay	**2.60**	**3.40**	3.74	5.67	3.61	4.65
	QoE	**0.89**	**0.80**	0.67	-0.86	0.69	0.07
5GB	Bitrate	-	**624.70**	431.63	444.45	524.28	351.79
	Rebuf	-	**0.12**	0.20	1.64	0.34	0.84
	Hit Rate	-	**0.68**	0.37	0.60	0.47	-
	Delay	-	**3.45**	3.62	5.55	3.80	4.65
	QoE	-	**0.82**	0.72	-0.73	0.58	0.07

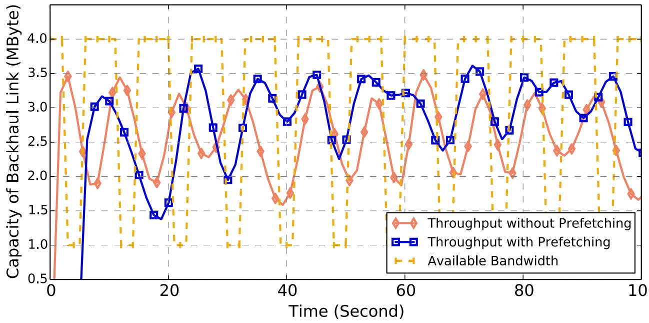


Fig. 12. Throughput variation when there is background traffic.

To better evaluate the performance of the schemes, five important metrics are included in the evaluation, i.e., bitrate, rebuffering, cache hit rate, transmission speed and QoE. These metrics are assessed from two perspectives including the average value in a short period and the global distribution. Among them, bitrate can represent the video definition at the client side. Rebuffering time means the video stalling brought from slow transmission. So the transmission speed is also a key metric for the schemes. Besides, cache hit rate at the edge is fully considered to evaluate the performance of caching and prefetching. Finally, QoE is a defined index to present the effects of the schemes in an intuitive manner. The overall testing results are presented in Table II, and are discussed in the next subsection.

B. Evaluation in the Simulated Scenario

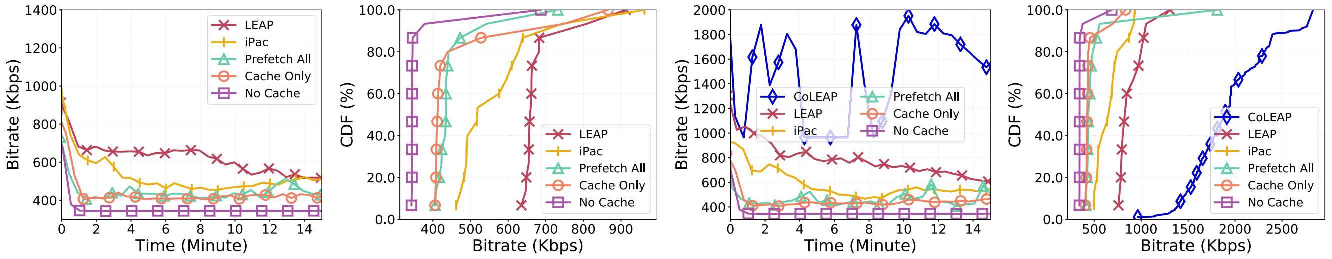
1) Throughput: To further illustrate the motivation of the proposed scheme, the throughput is first analysed when bandwidth variation is experienced by the link between the edge and origin server. This simulates the presence of background traffic that changes the available capacity of the backhaul link. The bandwidth is varied between 1 Mbps and 4 Mbps periodically, i.e., 9 s where high bandwidth availability is maintained for six seconds in each period followed by three seconds of low available bandwidth (or high load time). The schemes compared against are *Cache Only* and *LEAP*, which differ in terms of prefetching.

Fig. 12 shows that the throughput of *Cache Only* is lower than *LEAP*, making better use of the available bandwidth. Additionally, *LEAP* achieves higher throughput than the available bandwidth when *Cache Only* is declining sharply during the high load periods. It can be concluded that, with prefetching, *LEAP* adjusts the utilization ratio of available bandwidth under different network conditions.

2) Average Bitrate and Rebuffering: With the variation of network state, clients play videos and request different segments with various schemes. Client-side average bitrate and rebuffering, as important indexes, are first investigated to compare the effect of the different schemes. Fig. 13 a and Fig. 13 c show how average bitrate varies in time, for the 5 GB and 10 GB cache cases, respectively. The average bitrate slightly decreases in time due to the increasing participation of users. Fig. 13 b and Fig. 13 d show the Cumulative Distribution Function (CDF) of bitrate for different schemes for the two cache sizes. In both scenarios, the *No Cache* scheme acts as the baseline, exhibiting the lowest average bitrate. Its bitrate CDF converges at a low level because all user requests need to be served by the origin server. *Cache Only* and *Prefetch All* schemes show slightly better average bitrate. However, the *Prefetch All* scheme does not intelligently react to the changing bandwidth and sometimes even brings about backhaul congestion. Instead, the smart prefetch schemes, i.e. *iPac*, *LEAP*, and *CoLEAP*, perform better in both average bitrate and rebuffering. Besides, considering bitrate switches, LEAP-based schemes obtains higher average bitrate and lower rebuffering significantly than *iPac* does. Compared with *iPac*, *LEAP* achieves 19.2% and 34.4% improvements of average bitrate in the 5 GB and 10 GB cache scenarios respectively. In the 10 GB scenario, *CoLEAP* obtains high average bitrate, with 109.6% improvement in comparison with *LEAP*. Because *CoLEAP* employs collaboratively an adjacent edge to fetch more beneficial contents and also achieves the best bitrate CDF as the curves show. Moreover, the larger the cache size, the more the efficiency increases for all the schemes, improving their performance.

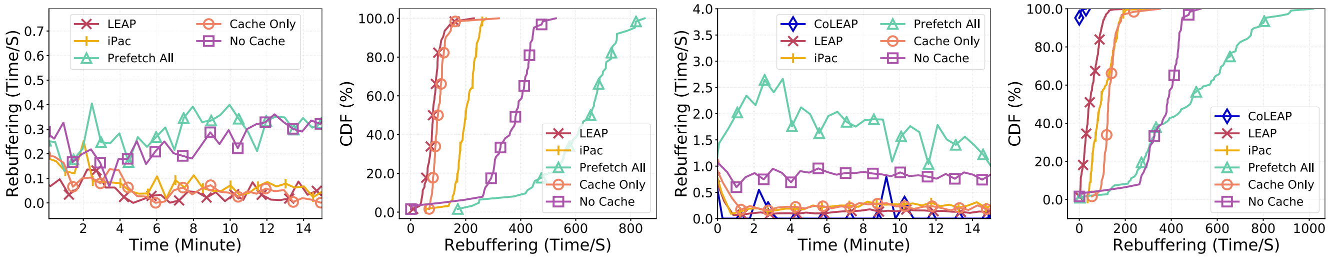
Fig. 14 illustrates the rebuffering time for all schemes with the same two cache sizes: 5 GB and 10 GB. It is obvious that the *Prefetch All* scheme performs the worst due to its aggressive prefetching behaviour while LEAP-based schemes obtain low rebuffering time. Compared with *iPac* in both 10 GB and 5 GB cache scenarios, Table II shows that *LEAP* reduces rebuffering time with 42.7% and 40.0%, respectively. Noteworthy is that *CoLEAP* barely suffers from rebuffering due to the availability of the adjacent edge backhaul path. Compared with *iPac*, *CoLEAP* reduces rebuffering time by 70.8% in the 10 GB cache scenario. However, perhaps the utility-based prefetching does not always work in the cooperative pattern because the statistical information of chain length is collected based on one-edge experiments. Although the chain-based utility works fine in most cases, it still causes rebuffering as Fig 14.(c) around Time 2 and Time 9. To sum up, the LEAP-based schemes learn during the learning process that reducing rebuffering can improve user experience, so they basically give priority to the requests from users who are suffering from rebuffering.

3) Cache Hit Rate: The cache hit rate described here includes not only the traditional cache hit pertinent to the already-requested contents cached by the edge server, but also the prefetch hit related to the prefetch requests in the edge server. Fig. 15 a and Fig. 15 b show the cache hit rate and its CDF with a 5 GB cache. Fig. 15 c and Fig. 15 d illustrate the cache hit



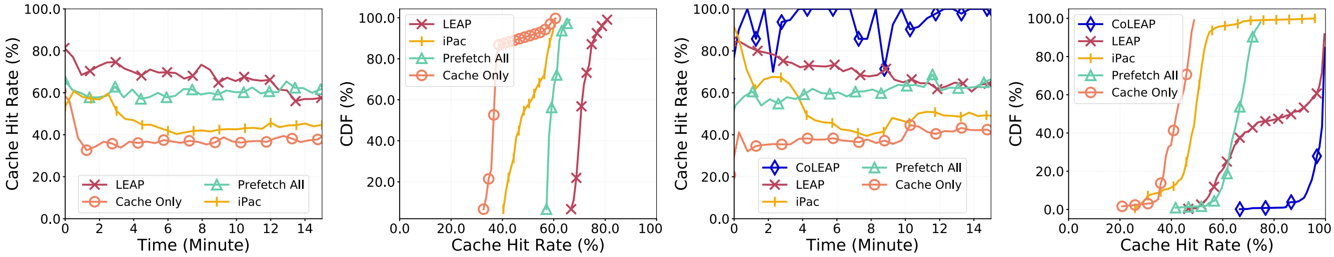
(a) 5GB cache (b) 5GB cache (c) 10GB cache (d) 10GB cache

Fig. 13. Comparison of average bitrate.



(a) 5GB cache (b) 5GB cache (c) 10GB cache (d) 10GB cache

Fig. 14. Comparison of rebuffering.



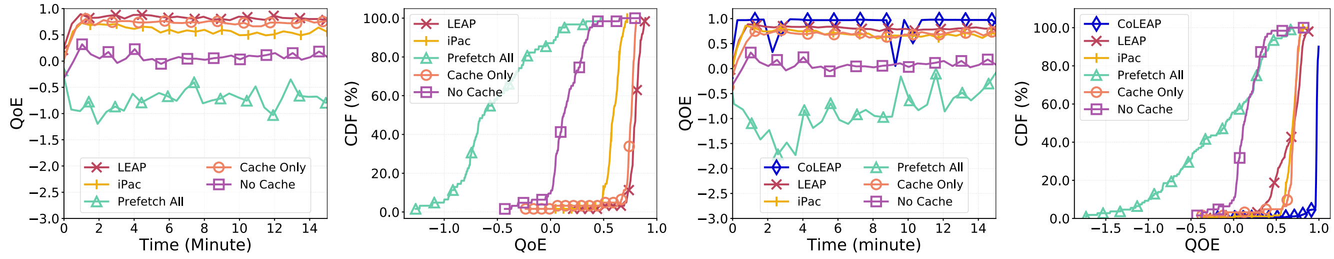
(a) 5GB cache (b) 5GB cache (c) 10GB cache (d) 10GB cache

Fig. 15. Comparison of cache hit rate.

rate and its CDF with a 10 GB cache. As the baseline, *Cache Only* is inferior to *Prefetch All* in both 10 GB and 5 GB scenarios. It can be concluded that *Prefetch All* improves its hit rate by substantially consuming bandwidth resources and may prefetch lots of unnecessary segments with the constant bitrate. In both cache scenarios, the cache hit rate of *iPac* keeps declining till converging with the baseline, which can be attributed to the increasing number of users and video requests. Moreover, *iPac* executes a simple LRU cache policy combined with a conservative utility-based prefetching strategy. So *iPac* still needs improvement.

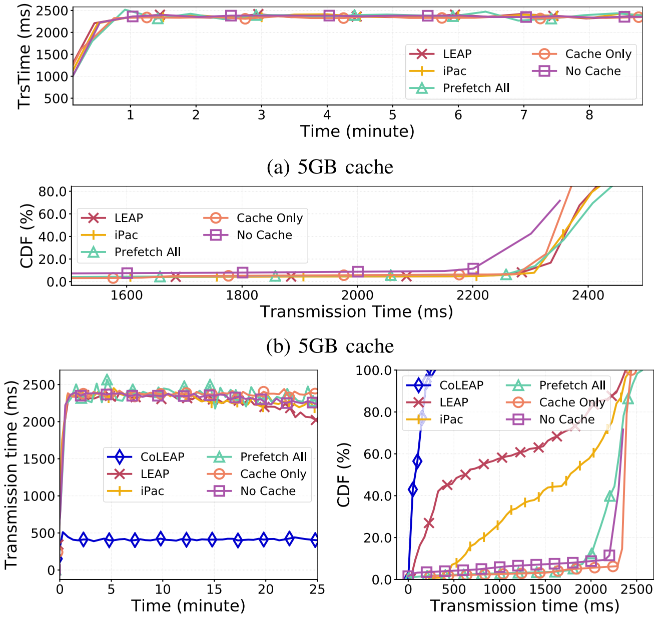
LEAP-based schemes are superior to the other schemes. As Table II shows, *LEAP* improves the cache hit rate by 33.9% and 44.7% in 10 GB and 5 GB cache scenarios in comparison with *iPac*. In the 10 GB cache scenario, *CoLEAP* achieves 96% hit rate 57.4% better than that of *Cache Only*. At the same time, the deep learning model ensures that the prefetched content has larger QoE. Moreover, *CoLEAP* also takes advantage of the available bandwidth of the adjacent edge. However, both the cache hit rates of *iPac* and *LEAP* in both 10 GB and 5 GB cache scenarios decrease gradually and keep stable finally, because the transmission capacity of the backhaul link is exhausted by the increasing number of users and video requests. We also note that, as expected, the larger cache size results in higher cache hit rates. The hit rate of *LEAP* with a 10 GB cache is 4.4% improvement than that with a 5 GB cache. Besides, *CoLEAP* achieves 25.0% higher hit rate than single *LEAP* due to the benefit of the cooperative cache mechanism.

4) *Response Speed and Qoe:* Although the defined QoE does not include explicitly response speed as an influencing factor, the response speed affects viewing experience. Response speed is a metric which helps verify the feasibility of a method and is used to reflect the efficiency of our proposed scheme. The transmission time for every segment and its distribution are shown in Fig. 17a and Fig. 17b for the 5 GB cache edge and in Fig. 17c and Fig. 17d for the 10 GB cache edge. The transmission results are similar for all the schemes tested due to the similar settings in the network simulations. *CoLEAP* transmission results in the 10 GB cache edge case, also shown in Table II, are better than those of the other schemes, as *CoLEAP* uses edge cooperation to provide the requested content. CDF also shows how *CoLEAP* consistently supports low transmission times, while the other



(a) 5GB cache (b) 5GB cache (c) 10GB cache (d) 10GB cache

Fig. 16. Comparison of QoE.



(a) 5GB cache

(b) 5GB cache

(c) 10GB cache (d) 10GB cache

Fig. 17. Comparison of response speed.

schemes have both low and very high transmission times. This is a great result, despite the fact that the connection between the edges in *CoLEAP* also adds to the transmission time.

Next we investigate the effect of all schemes on QoE. When comparing the different schemes, we take *No Cache* as the baseline, so its QoE value fluctuates around the zero line, as shown in Fig. 16 a and Fig. 16 c, for the 5 GB and 10 GB cache size cases, respectively. The CDF of QoE presented in Fig. 16 b and Fig. 16 d also describe the effect of these schemes in the two cases. We find that the aggressive prefetching behavior of *Prefetch All* causes the inevitable transmission delays even rebuffering. *No Cache* shows a relatively better QoE, but the scheme does not make good use of edge server resources. *Cache Only*, *iPac* and *LEAP* obtain even higher values of QoE as the figures show. Of these schemes, *LEAP* improves QoE by at least 15.9% and 13.4% in the 10 GB and 5 GB cache scenarios, respectively as shown in Table II. We also find that cache size has little influence on QoE for single *LEAP*, further demonstrating the efficiency of *LEAP*. However, *CoLEAP* achieves the highest average QoE and the best distribution, which confirms that utilizing adjacent edges is highly beneficial. Note that the initial value of QoE is lower for all the schemes, because waiting delay in the start-up phase is considered as rebuffering and affects QoE. After the buffer fills to a given threshold, the client starts playing and the value of QoE reflects the schemes' behaviour.

VI. Conclusion

The increasing deployments of DASH make smart edges become the preferred target for transmission optimization. In order to provide end users with high QoE, it is of necessity to design an efficient edge scheme with caching and prefetching to take advantage of the available edge-side resources. This paper presents a cooperative learning-based smart edge caching and prefetching solution to improve the QoE of adaptive video streaming. We formalize the network problem and design a deep neural network to predict the QoE gain, which is employed in the proposed cooperative utility-based caching and prefetching strategy. The results of comprehensive simulation-based testing demonstrate the efficiency of our scheme in terms of multiple performance metrics and in comparison with several existing solutions. By smart prefetching, *CoLEAP* mitigates traffic load in the backhaul links, increases average bitrate, reduces transmission time and rebuffering and improves QoE. Further work considers deploying *CoLEAP* in a real Internet environment and testing it in comparison with other classic approaches.

References

[1] V. Cisco, "Cisco visual networking index: Forecast and trends, 2017–2022," *San Jose, CA, USA, Cisco, White Paper*, 2018.

[2] F. Dobrian *et al.*, "Understanding the impact of video quality on user engagement," in *Proc. ACM SIGCOMM*, 2011, pp. 362–373.

[3] J. Li, L. Krasula, Y. Baveye, Z. Li, and P. Le Callet, "AccAnn: A new subjective assessment methodology for measuring acceptability and annoyance of quality of experience," *IEEE Trans. Multimedia*, vol. 21, no. 10, pp. 2589–2602, Oct. 2019.

[4] I. Sodagar, "The MPEG-DASH standard for multimedia streaming over the internet," *IEEE Multimedia*, vol. 18, no. 4, pp. 62–67, Apr. 2011.

[5] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," in *Proc. ACM SIGCOMM*, 2015, pp. 325–338.

[6] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal bitrate adaptation for online videos," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.

[7] Z. Akhtar *et al.*, "Oboe: Auto-tuning video ABR algorithms to network conditions," in *Proc. ACM SIGCOMM*, 2018, pp. 44–58.

[8] D. Wu, Y. T. Hou, W. Zhu, Y.-Q. Zhang, and J. M. Peha, "Streaming video over the internet: Approaches and directions," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, no. 3, pp. 282–300, Mar. 2001.

[9] Q. Zhang, W. Zhu, and Y.-Q. Zhang, "Resource allocation for multimedia streaming over the internet," *IEEE Trans. Multimedia*, vol. 3, no. 3, pp. 339–355, Sep. 2001.

[10] Z. Wang, L. Sun, W. Zhu, S. Yang, H. Li, and D. Wu, "Joint social and content recommendation for user-generated videos in online social network," *IEEE Trans. Multimedia*, vol. 15, no. 3, pp. 698–709, Apr. 2012.

[11] L. Tang *et al.*, "Popularity prediction of facebook videos for higher quality streaming," in *Proc. USENIX ATC*, 2017, pp. 111–123.

[12] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in HTTP-Based adaptive video streaming with festive," *IEEE/ACM Trans. Netw.*, vol. 22, no. 1, pp. 326–340, Feb. 2014.

[13] C. Pleşca, V. Charvillat, and W. T. Ooi, "Multimedia prefetching with optimal markovian policies," *J. Netw. Comput. Appl.*, vol. 69, pp. 40–53, 2016.

[14] W. Hu *et al.*, "Toward Wi-Fi AP-assisted content prefetching for an on-demand TV series: A learning-based approach," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 7, pp. 1665–1676, 2017.

[15] K. T. Bagci, K. E. Sahin, and A. M. Tekalp, "Compete or collaborate: Architectures for collaborative DASH video over future networks," *IEEE Trans. Multimedia*, vol. 19, no. 10, pp. 2152–2165, Oct. 2017.

[16] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, "AdaptSize: Orchestrating the hot object memory cache in a content delivery network," in *Proc. USENIX NSDI*, 2017, pp. 483–498.

[17] L. Zhang *et al.*, "Named data networking," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 66–73, 2014.

[18] W.-G. Teng, C.-Y. Chang, and M.-S. Chen, "Integrating web caching and web prefetching in client-side proxies," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 5, pp. 444–455, May 2005.

[19] C. D. Gracia and S. Sudha, "A case study on memory efficient prediction models for web prefetching," in *Proc. IEEE Int. Conf. Emerg. Trends Eng., Technol. Sci.*, 2016, pp. 1–6.

[20] W. Shi *et al.*, "LEAP: Learning-based smart edge with caching and prefetching for adaptive video streaming," in *Proc. ACM/IEEE IWQOS*, 2019, pp. 1–10.

[21] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," in *Proc. ACM SIGCOMM*, 2014, pp. 187–198.

[22] J. M. Batalla, P. Krawiec, A. Beben, P. Wisniewski, and A. Chydzinski, "Adaptive video streaming: Rate and buffer on the track of minimum rebuffering," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 8, pp. 2154–2167, Aug. 2016.

[23] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. C. Begen, "Server-based traffic shaping for stabilizing oscillating adaptive streaming players," in *Proc. ACM NOSSDAV*, 2013, pp. 19–24.

[24] A. Detti, B. Ricci, and N. Blefari-Melazzi, "Tracker-assisted rate adaptation for MPEG DASH live streaming," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.

[25] S. Altamimi and S. Shirmohammadi, "QoE-Fair DASH video streaming using server-side reinforcement learning," A*CM Trans. Multimedia Comput., Commun., Appl.*, vol. 16, no. 2 s, pp. 1–21, 2020.

[26] J. Zou *et al.*, "Probabilistic tile visibility-based server-side rate adaptation for adaptive 360-degree video streaming," *IEEE J. Sel. Topics Signal Process.*, vol. 14, no. 1, pp. 161–176, Jan. 2019.

[27] S. M. S. Tanzil, W. Hoiles, and V. Krishnamurthy, "Adaptive scheme for caching YouTube content in a cellular network: A machine learning approach," *IEEE Access*, vol. 5, no. 99, pp. 5870–5881, 2017.

[28] C. Zhong, M. C. Gursoy, and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," in *Proc. IEEE 52nd Annu. Conf. Inf. Sci. Syst.*, 2018, pp. 1–6.

[29] M. Leconte, G. Paschos, L. Gkatzikis, M. Draief, S. Vassilaras, and S. Chouvardas, "Placing dynamic content in caches with small population," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.

[30] J. Z. Wang and S. Y. Philip, "Fragmental proxy caching for streaming multimedia objects," *IEEE Trans. Multimedia*, vol. 9, no. 1, pp. 147–156, Jan. 2006.

[31] Y. Wang, X. Zhou, M. Sun, L. Zhang, and X. Wu, "A new QoE-driven video cache management scheme with wireless cloud computing in cellular networks," *Mobile Netw. Appl.*, vol. 22, no. 1, pp. 72–82, 2017.

[32] C. Li, L. Toni, J. Zou, H. Xiong, and P. Frossard, "QoE-Driven mobile edge caching placement for adaptive video streaming," *IEEE Trans. Multimedia*, vol. 20, no. 4, pp. 965–984, Apr. 2018.

[33] W. Zhang, Y. Wen, Z. Chen, and A. Khisti, "QoE-Driven cache management for HTTP adaptive bit rate streaming over wireless networks," *IEEE Trans. Multimedia*, vol. 15, no. 6, pp. 1431–1445, Oct. 2013.

[34] P. L. Vo, L. Van Nguyen, T.-A. Le, and D. N. M. Dang, "A QoE-based caching algorithm for HTTP adaptive streaming contents in radio access networks," in *Proc. IEEE 6th Int. Conf. Commun. Electron.*, 2016, pp. 417–422.

[35] J. Sánchez-Hernández, J. Garcia-Ortiz, V. González-Ruiz, and D. Müller, "Interactive streaming of sequences of high resolution JPEG2000 images," *IEEE Trans. Multimedia*, vol. 17, no. 10, pp. 1829–1838, Oct. 2015.

[36] "preload-webpack-plugin, chrome browser prefetch function," Oct. 2019. [Online]. Available: https://github.com/GoogleChromeLabs/preload-webpack-plugin

[37] S. Sengupta, N. Ganguly, S. Chakraborty, and P. De, "HotDASH: Hotspot aware adaptive video streaming using deep reinforcement learning," in *Proc. Int. Conf. Netw. Protocol*, 2018, pp. 165–175.

[38] K. Liang, J. Hao, R. Zimmermann, and D. K. Yau, "Integrated prefetching and caching for adaptive video streaming over HTTP: An online approach," in *Proc. ACM Multimedia Syst. Conf.*, 2015, pp. 142–152.

[39] P. Juluri and D. Medhi, "Cache'n DASH: Efficient caching for DASH," in *Proc. ACM SIGCOMM*, 2015, pp. 599–600.

[40] D. Wu, Q. Liu, H. Wang, Q. Yang, and R. Wang, "Cache less for more: Exploiting cooperative video caching and delivery in D2D communications," *IEEE Trans. Multimedia*, vol. 21, no. 7, pp. 1788–1798, Jul. 2019.

[41] S. Zhang *et al.*, "Cooperative edge caching in user-centric clustered mobile networks," *IEEE Trans. Mobile Comput.*, vol. 17, no. 8, pp. 1791–1805, Aug. 2018.

[42] H. Pang, J. Liu, X. Fan, and L. Sun, "Toward smart and cooperative edge caching for 5G networks: A deep learning based approach," in *Proc. IEEE 26th Int. Symp. Qual. Service*, 2018, pp. 1–6.

[43] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri, "Helping hand or hidden hurdle: Proxy-assisted HTTP-based adaptive streaming performance," in *Proc. IEEE 21st Int. Symp. Modelling, Anal. Simul. Comput. Telecommun. Syst.*, 2013, pp. 182–191.

[44] C. Ge, N. Wang, G. Foster, and M. Wilson, "Toward QoE-assured 4 K video-on-demand delivery through mobile edge virtualization with adaptive prefetching," *IEEE Trans. Multimedia*, vol. 19, no. 10, pp. 2222–2237, Oct. 2017.

[45] S. Benno, J. O. Esteban, and I. Rimac, "Adaptive streaming: The network HAS to help," *Bell Labs Tech. J.*, vol. 16, no. 2, pp. 101–114, 2011.

[46] A. Ahmed, Z. Shafiq, H. Bedi, and A. Khakpour, "Suffering from buffering? detecting QoE impairments in live video streams," in *Proc. IEEE 25th Int. Conf. Netw. Protocols*, 2017, pp. 1–10.

[47] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. ACM SIGCOMM*, 2017, pp. 197–210.

[48] M. Zanforlin, D. Munaretto, A. Zanella, and M. Zorzi, "SSIM-Based video admission control and resource allocation algorithms," in *Proc. IEEE 12th Int. Symp. Model. Optim. Mobile, Ad Hoc, Wireless Netw.*, 2014, pp. 656–661.

[49] F. Chiariotti, S. D'Aronco, L. Toni, and P. Frossard, "Online learning adaptation strategy for DASH clients," in *Proc. ACM Multimedia Syst. Conf.*, 2016, pp. 1–12.

[50] J. Summers, T. Brecht, D. Eager, and A. Gutarin, "Characterizing the workload of a netflix streaming video server," in *Proc. IEEE Int. Symp. Workload Characterization*, 2016, pp. 1–12.

[51] "iqiyi open cache program," Jul. 2018. [Online]. Available: http://open.iqiyi.com/developer/iocp/iocp.html

[52] J. Jiang, S. Sun, V. Sekar, and H. Zhang, "Pytheas: Enabling data-driven quality of experience optimization using group-based exploration-exploitation," in *Proc. USENIX NSDI*, 2017, pp. 393–406.

[53] P. Freeman, "The secretary problem and its extensions: A review," *Int. Stat. Rev./Revue Internationale de Statistique*, vol. 51, no. 2, pp. 189–206, 1983.

[54] "Apache traffic server," Aug. 2018. [Online]. Available: https://trafficserver.apache.org/

[55] "LEAP," Feb. 2019. [Online]. Available: https://github.com/wcc2wykdb/LEAP.git/

[56] "NGINX-High performance load balancer, web server, & reverse proxy," Jul. 2018. [Online]. Available: https://www.nginx.com/

[57] B. Hubert *et al.*, "Linux advanced routing & traffic control HOWTO," *Netherlabs BV*, vol. 1, pp. 99–107, 2002.

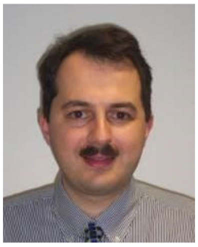
Wanxin Shi (Student Member, IEEE) received the B.S. degree in computer science from China University of Geosicences (Beijing), in 2017. She is currently working toward the M.S. degree with Tsinghua University. She is primarily interested in edge computing, optimization of video transmission, software-defined networking, etc.

Chao Wang received the B.S. degree in software engineering from the Dalian University of Technology in 2017, and is currently working toward the M.S. degree with Tsinghua University. Her research interest is in network-based optimization for adaptive video streaming.

Yong Jiang (Member, IEEE) received the B.S. and Ph.D. degrees from Tsinghua University, Beijing, China, in 1998 and 2002, respectively, both in computer science and technology. He is currently a Full Professor with the Graduate school, Shenzhen, Tsinghua University. His research interests include the future network architecture, the Internet QoS, software defined networks, network function virtualization, etc.

Qing Li (Member, IEEE) received the B.S. degree from the Dalian University of Technology, Dalian, China, in 2008 and the Ph.D. degree from Tsinghua University, Beijing, China, 2013, both in computer science and technology. He is currently an Associate Professor with the Southern University of Science and Technology, China. His research interests include reliable and scalable routing of the Internet, software defined networks, network function virtualization, in-network caching/computing, intelligent self-running network, etc.

Gengbiao Shen (Student Member, IEEE) received the B.S. and M.S. degrees from Beihang University, Beijing, China, in 2013 and 2016, respectively, both in instrument science and technology. He is currently working toward the Ph.D. degree with Tsinghua University, China. His research interests include data center network, in-network caching, intelligent network, software-defined networking, flow scheduling, load balancing, etc.



Gabriel-Miro Muntean (Senior Member, IEEE) received the B.Eng. and M.Sc. degrees from the Politehnica University of Timisoara, Romania, in 1996 and 1997, respectively, and the Ph.D. degree from the School of Electronic Engineering, Dublin City University (DCU), Ireland, in 2003, for his research on quality-oriented adaptive multimedia streaming over wired networks. He is currently an Associate Professor with the School of Electronic Engineering DCU and Co-Director of the Performance Engineering Laboratory DCU. He has authored or coauthored more than 350 papers in prestigious international journals and conferences, has authored four books and 18 book chapters, and has edited six other books. His research interests include quality-oriented and performance related issues of adaptive multimedia delivery, performance of wired and wireless communications, energy-aware networking, and personalized technology-enhanced learning. Dr. Muntean is an Associate Editor for the IEEE TRANSACTIONS ON BROADCASTING, THE MULTIMEDIA COMMUNICATIONS, Area Editor of the IEEE COMMUNICATION SURVEYS AND TUTORIALS, and a Reviewer for other important international journals, conferences, and funding agencies. He is a senior member of IEEE Broadcast Technology Society and coordinated the EU Horizon 2020-funded NEWTON Project (http://newtonproject.eu).