Towards Real-Time Video Caching at Edge Servers: A Cost-Aware Deep Q-Learning Solution

Laizhong Cui, Erchao Ni, Yipeng Zhou, Zhi Wang, Lei Zhang, Jiangchuan Liu, Yuedong Xu

***Abstract*—Given the rapid growth of user-generated videos, internet traffic has been heavily dominated by online video streaming. Caching videos on edge servers in close proximity to users has been an effective approach to reduce the backbone traffic and the request response time, as well as to improve the video quality on the user side. Video popularity, however, can be highly dynamic over time. The cost of cache replacement at edge servers, particularly that related to service interruption during replacement, is not yet well understood. This paper presents a novel lightweight video caching algorithm for edge servers, seeking to optimize the hit rate with real-time decisions and minimized cost. Inspired by recent advances in deep Q-learning, our DQN-based online video caching (DQN-OVC) makes effective use of the rich and readily available information from users and networks. We decompose the Q-value function as a product of the video value function and the action function, which significantly reduces the state space. We instantiate the action function for cost-aware caching decisions with low complexity so that the cached videos can be updated continuously and instantly with dynamic video popularity. We used video traces from Tencent, one of the largest online video providers in China, to evaluate the performance of our DQN-OVC and to compare it with state-of-the-art solutions. The results demonstrate that DQN-OVC significantly outperforms the baseline algorithms in the edge caching context.**

***Index Terms*—Reinforcement Learning, Deep Q-Learning, Edge Cache, Video Popularity**

I. Introduction

With the advances in broadband networking, video coding, and mobile computing in the past two decades, videos can now be generated and consumed by any person anytime and anywhere. Video applications, such as Netflix and YouTube [1], have become essential internet services, and streaming video has long been dominating traffic, accounting for 70% of internet traffic [2]. It is envisioned by the Cisco Visual Networking Index that global mobile video services will grow nine-fold from 2017 to 2022, accounting for 79% of total mobile data traffic by the end of the forecast period [3].

Laizhong Cui, Erchao Ni and Lei Zhang are with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, PR. China and with the Guangdong Laboratory of Artificial-Intelligence and Cyber-Economics(SZ), Shenzhen University. (Email: cuilz@szu.edu.cn; nierchao5@qq.com; leizhang@szu.edu.cn)
Yipeng Zhou is with the Department of Computing, Faculty of Science and Engineering, Macquarie University, Sydney, NSW 2109, Australia, and also with Peng Cheng Laboratory, Shenzhen, Guangdong 518000, China. (Email: yipeng.zhou@mq.edu.au).
Zhi Wang is with Tsinghua Shenzhen International Graduate School. (Email: wangzhi@sz.tsinghua.edu.cn)
Jiangchuan Liu is with the School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada (Email: jcliu@cs.sfu.ca)
Yuedong Xu is with the Department of Electronic Engineering, Fudan University, Shanghai, China (Email: ydxu@fudan.edu.cn)
Corresponding author: Yipeng Zhou.

Given the massive amount of traffic generated by video streaming applications, it is challenging to meet the stringent quality of service (QoS) requirement of streaming videos through remote cloud servers [4], [5]. An emerging trend is to deploy edge servers in close proximity to streaming users to shorten the transmission latency and reduce the backbone traffic load [6]–[8]. A typical edge-empowered streaming system consists of three entities: the cloud, edge servers, and end users. A cloud server or cluster of servers stores all the videos in a remote datacenter and can respond to requests from users and edge servers. An edge server provides streaming service to a limited number of nearby users with its cached videos that are fetched from the cloud. The cache space of an edge server, however, is very limited compared to that of the cloud [7], which is virtually unlimited. Therefore, the performance of an edge server is usually gauged by the cache hit rate, that is, the rate of the requests served by the edge server to the total requests from users associated with this edge server.

Two challenges stand in the way of optimizing the hit rate on edge servers, which have not been well studied in the literature. On the one hand, video popularity is highly volatile, particularly for today's short videos that are likely to be popular for only a few hours. An edge server needs to replace stale videos in time to capture the latest trend [9]. On the other hand, downloading a new video that is not in the edge cache requires time [9], [10], during which the service of the superseded video is not available. In other words, replacing a stale video is not cost free.

This paper presents an in-depth and systematic study on the video caching problem with edge servers, seeking to address the aforementioned challenges. We demonstrate that it can be modeled as a decision-making problem within a reinforcement learning framework, in which caching a video can be regarded as an action and maximizing the hit rate is equivalent to maximizing the reward over time. We then present a lightweight solution that is inspired by recent advances in deep Q-learning [11]. Our online video caching solution, namely, DQN-OVC, considers the unique characteristics of online videos to effectively reduce the computational complexity of deep Q-learning while retaining high prediction accuracy. We decompose the basic Q-value function into a video value function and an action function. Instead of learning the Q-value function for each pair of states and actions, we employ a small neural network to learn the value of individual videos.[1] This is an essential simplification so that DQN-OVC can be executed in real time to update the cached videos in a timely manner. Our DQN-OVC also accounts for the replacement cost, which was largely neglected in early studies. Specifically, when making a decision to replace stale videos, the edge server considers both a) the time cost to download new videos and b) the interruption cost to interrupt the requests for stale videos.

The main contributions of this paper can be summarized as follows:

- We model the video caching problem on edge servers and design a novel lightweight solution inspired by advanced deep Q-learning;
- Our solution, DQN-OVC, achieves low complexity and seamlessly integrates the time cost to download new videos from the remote cloud and the interruption cost into the overall replacement cost.

We carried out extensive experiments using both the request records collected from Tencent, one of the largest video service providers in China and a synthetic dataset. The results demonstrate that DQN-OVC achieves a superior hit rate compared to state-of-the-art solutions.

The rest of the paper is organized as follows. Sec. II discusses the related caching algorithms. Sec. III introduces the background of the video caching system with an edge server and formally formulates our problem. Sec. IV elaborates the key designs in DQN-OVC, the improved cache decision algorithm inspired by deep Q-learning. Sec. V introduces the experimental settings and presents the results of our experiments. Finally, Sec. VI summarizes this paper and discusses our future works.

II. Related Work

In this section, we introduce related works from three perspectives: traditional heuristic caching algorithms, machine learning-based caching algorithms, and caching on edge servers.

A. Traditional Caching Algorithms

The LRU (least recently used) and LFU (least frequently used) are two classic algorithms in the literature [9], [12]. Their implementations are simple yet effective, and hence, they are still widely used in modern caching systems, including those for videos [13]. To accommodate the unique video access patterns, Zhou *et al.* [14] proposed a new reactive caching strategy, which mixed LFU and FIFO and updated new content and old content in different ways. There have also been recent works to accommodate varying object size distributions and request characteristics [15] and to increase the hit rate by maximizing the hit density [16].

B. Machine Learning-based Caching Algorithms

Due to the latest advances in artificial intelligence, various machine learning-based caching algorithms have been developed for online video systems in recent years [17], [18]. It is known that the caching problem can be turned into a popularity prediction problem and then solved through supervised learning. Narayanan *et al.* [19] applied a recurrent neural network (LSTM) to predict the number of video requests in a historic time period. Song *et al.* [20] further demonstrated a cache policy based on the Belady MIN algorithm for content distribution.

Recently, the reinforcement learning (RL) framework has been suggested for making cache decisions. RL learns the reward of each action so that the action can achieve the highest rewards. Sung *et al.* [21] exploited the reinforcement learning framework to determine the edge caching for wireless base stations. They further extended the decision space of their algorithm to be applicable to multiple wireless edge base stations and studied the edge base stations with a small graph structure. Zhong *et al.* [22] applied DRL to obtain the cache strategy based on historical video request time points and new video request time points in the future. Compared to supervised learning, DRL is more flexible in accommodating multiple influential factors. Its action space, however, can be exponential with the video population, incurring significant computational overhead.

C. Caching on Edge Servers

It has been shown that active caching on edge devices can save up to 22 % of the backhaul traffic [23] and significantly reduce transmission latency [24]. Most recent edge caching algorithms for videos have employed machine learning to predict the videos' future popularity [24]–[26]. The limited storage and computing capacity of edge servers and the high demand of advanced learning models, however, can conflict. It is also worth noting that today's machine learning remains unreliable and unstable in terms of the prediction accuracy [27], and it is necessary to continuously monitor the system dynamics and perform timely updates [9], which unfortunately can be challenging for RL with a large action space, particularly at the edge.

Our work is inspired by RL-based algorithms, particularly the recent advances in deep Q-learning cite. Our algorithm considers multiple influential factors, including the replacement cost that has yet to be addressed in previous studies, and significantly reduces the action space to facilitate real-time updates.

III. System Model and Problem Formulation

In this section, we introduce the system model of video streaming services assisted by edge computing. Then, we discuss the replacement cost and formulate our problem with the objective of maximizing the hit rate of the cached videos.

A. Architectural View

We consider a video streaming system with a remote cloud server and a set of edge servers. An edge server is deployed between end-user devices and the cloud server [5], [28], serving its nearby users. Without loss of generality, we focus

[1] Note that the structure of the neural network is identical for all videos.

on the interaction among the cloud server, an edge server, and its users. With virtually unlimited storage space, the cloud server maintains a complete set of all videos [28], whereas the edge server closer to end users can only cache a small number of videos given its limited storage.

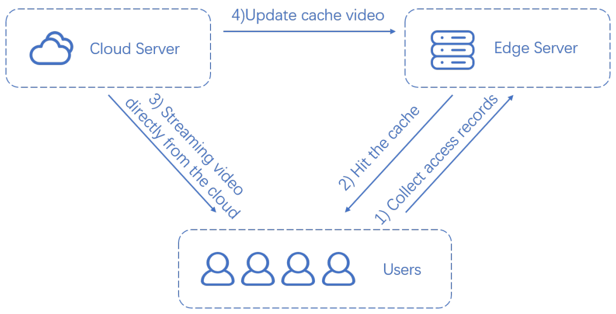

Cloud Server
4)Update cache video
Edge Server
3) Streaming video directly from the cloud
2) Hit the cache
1) Collect access records
Users

Fig. 1: A conceptual architecture of video caching with edge servers.

In Fig. 1, we present a high-level illustration of how the online video caching system with the edge server works. The interactions of the three key players, namely, cloud server, edge server, and end user, are briefly described below.

1) For a video of interest, a user first sends a request to the edge server in its close proximity to fetch the video;
2) If the requested video has been cached by the edge server, the video is streamed to the user directly; otherwise, the user's request is redirected to the cloud server [29]. In both cases, the request is logged to help the edge server make future decisions;
3) The edge server executes the caching algorithm periodically to update the cached videos. Once a stale video is replaced with a new video, it downloads the new video from the cloud server. Meanwhile, requests for the stale video are interrupted and redirected to the cloud server. Before the new video is completely downloaded by the edge server, any request for the new video is still served by the cloud server.

Ideally, both the cloud server and an end user prefer that a request from the user is served by the corresponding edge server. For the former, the edge server can effectively reduce its service load and bandwidth cost. For the latter, the edge server potentially improves the QoS given its closer distance.

As in previous studies [30], we assume that each video stream is divided into a series of segments of the same size. A request for a particular video can then be converted into sequential requests for different video segments. To simplify the system management, we assume that different segments of the same video have similar popularity characteristics, and hence, the experience of a segment can be used to facilitate the prediction of other segments. Our solution, however, will also work with segments with heterogeneous lengths and popularity. Hereafter, we use “video" and “video segment" interchangeably in our discussion.

B. Replacement Cost

Assume that the edge server has decided to replace stale video i_1 with a new video i_2 at time t; we need to consider the following two costs:

- Download Cost: The edge server needs to download video i_2 from the cloud server into local storage before serving user requests;
- Interruption Cost: The streaming for video i_1 will be interrupted, and its ongoing users need to rebuild connections with the cloud server to resume their downloading, as illustrated in Fig. 2.

These costs can be massive in a system with a large number of requests. For instance, in our dataset collected from Tencent Video, nearly a hundred requests are intercepted by the cache in each millisecond during peak hours. More specifically, the download costs consist of three components.

1) The running time of the caching decision algorithm, as it may take milliseconds to predict the popularity of a single video [9].
2) The communication time between the edge and the cloud, including the latency for request communication [30] and the time for video transmission [10];
3) The time to write new videos to the memory of the edge server [31]–[33].

Let B denote the size of each video segment. The time taken by the edge server to complete the replacement can be calculated as:

$$t_d = t_p + \frac{B}{d_e} + \frac{B}{w_e}. \tag{1}$$

Here, t_p is the time to run the replacement decision algorithm, d_e is the download speed from the cloud server by the edge server, and w_e is the speed of writing video content into the memory. [2] Note that an implication here is that the requests for video i_2 during period $[t, t+t_d)$ cannot be served by the edge server.

The interruption cost is measured by the hit misses due to the replacement of stale video i_1, as illustrated in Fig. 2. Clearly, if the replacement is not well decided, the interruption cost can be prohibitive.

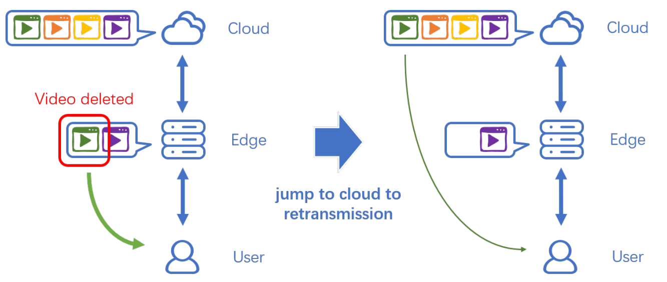


Fig. 2: The interruption cost of replacing a stale video segment.

[2] If the download and write operations are executed in parallel, we can change the download cost to $t_d = t_p + \frac{B}{d_e}$, and our later solution framework will still apply.

C. Problem Formulation

Given the replacement costs, our objective for the cache decision is to maximize the hit rate of the edge server. As we discussed in the last subsection, a request may be interrupted and redirected to the cloud server before streaming is completed. Thus, a video segment request j is successfully hit if the following two conditions are met: 1) The requested video segment is cached by the edge server when request j arrives; and 2) The streaming of this segment is completely served by the edge server.

Assume that there are m requests in total for different video segments. We split the timeline into a series of time slots of identical length. A slot is the basic time unit in our model, and to accommodate dynamics, our video caching algorithm is executed in each slot from 0 to T. Let q_j^t denote whether request j has been successfully hit by the edge server at time slot t. If the video streaming for request j is completed during the period from $t-1$ to t and the request is fully served by the edge server, we have $q_j^t = 1$; otherwise $q_j^t = 0$. The objective, *i.e.*, the hit rate of the edge server, can be defined as:

$$CHR = \sum_{t=1}^{T}\sum_{j=1}^{m} q_j^t. \tag{2}$$

To maximize the hit rate, two constraints should be considered. The first is the cache space limit at the edge server. Assume the cache decision of video segment i at time slot t is represented by a_t^i, where $a_t^i = 1$ if the edge server decides to cache or has cached video segment i, and $a_t^i = 0$ otherwise. Assume that the edge server can cache at most C video segments. We have $\sum_{j=1}^{m} a_t^j \leq C$. The second constraint comes from the finite bandwidth capacity of the edge server. Assume the edge server can download up to L video segments from the cloud server in each time slot. Here, L is determined by the network capacity between the cloud server and the edge server, which in general will not be $L=\infty$ unless this constraint is relaxed in a resource-rich network.

We use c^i to represent the caching state of video segment i, whose value is given by

$$c_t^i = \begin{cases} 0, & \text{Segment } i \text{ is not cached at time slot } t \\ \epsilon, & \text{Segment } i \text{ is being downloaded, where } 0<\epsilon<1 \\ 1, & \text{Segment } i \text{ is cached at time slot } t. \end{cases} \tag{3}$$

For request j to access video i, q_j^t can be calculated by

$$q_j^t = \begin{cases} 1, & j \text{ completed at } t \text{ and } c_{t'}^i = 1, \forall t' \in (t_j^s, t), \\ 0, & \text{otherwise,} \end{cases} \tag{4}$$

where t_j^s is the starting time slot of request j.

As we discussed, it takes t_d time slots to complete the operation to cache a video. It is possible that $t_d > 1$. When $0 < c_t^i < 1$, it indicates that downloading video segment i is in progress.

The caching state c_t^i of segment i can then be calculated as follows:

$$c_{t+1}^i = \begin{cases} 0, & \text{if} \quad a_t^i = 0, \\ min(1, c_t^i + \frac{1}{t_d}), & \text{if} \quad a_t^i = 1. \end{cases} \tag{5}$$

If $a_t^i = 0$ and $a_{t-1}^i = 1$, it means that the video is deleted from the edge server at time slot t; If $a_t^i = 1$ and $a_{t-1}^i = 0$, then the edge server caches video i at time slot t.

We also define a binary indicator with 1 if the video is being downloaded by the edge server, and 0 otherwise. That is

$$I\left(c_t^i\right) = \begin{cases} 1, & 0 < c_t^i < 1; \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

We can then define the optimal video cache replacement problem for the edge server as

$$\begin{aligned} \max_{a_t^i, \forall i, t} \quad & CHR = \sum_{t=1}^{T}\sum_{j=1}^{m} q_j^t \\ \text{s.t.} \quad & C_1 : a_t^i \in \{0,1\}, \quad \forall i, t, \\ & C_2 : \sum_{i}^{n} a_t^i \leq C, \quad \forall t, \\ & C_3 : \sum_{i=1}^{n} I\left(c_t^i\right) \leq K, \quad \forall t, \\ & C4 : (4), (5). \end{aligned} \tag{7}$$

IV. Optimizing Caching with Deep Q-Learning

To solve the problem in Eq. (7), we need to identify the relation between the caching decisions a_t^i's and the hit rate CHR, which unfortunately is extremely complicated. On the one hand, the caching decisions of different videos affect the hits of each other because the caching space is limited and the videos have to compete with each other; on the other hand, there is a latency between a caching decision and the resulting request hits given the download cost, making the real-time decision during peak hours very difficult.

Inspired by recent advances in data-driven optimization, we resort to deep reinforcement learning (DRL) for this decision-making problem [11]. DRL can learn the impact of multiple environmental factors and iteratively refine the actions to be taken in a complex environment to maximize the cumulative rewards over time. For the video caching problem, we can map each edge server's cache to an agent, and caching a particular video can be regarded as an action; the environment can be defined as the network transmission process that is affected by the agent's action, such as where the user should obtain the video of their request. The objective of maximizing the hit rate then becomes identifying the actions towards maximizing the reward. Due to the complicated interaction between the environment and each action, it is impractical to derive the exact relation between each action (*i.e.*, caching a video) and the hit rate objective. Rather than modeling this complicated relation explicitly,

TABLE I: Notation list

Notation	Meaning
$\mathbf{s}_t$	The state of the system at time slot t
$\mathbf{a}_t$	The action of the edge server at time slot t
R^t	Rewards harvested by the edge server at time slot t
π	Cache policy of the edge server
$\mathbf{s}_t^i$	The state of video i at time slot t
a^i	The caching action for video i
c^i	The cache state of video i
q_j^t	Indicator on whether request j is completely served by the edge server at time slot t
h_t^{ik}	k time window of video i at time slot t
C	The maximum number of video segments the edge server can hold
$V^\pi(\mathbf{s})$	The value of video i with policy π and initial state $\mathbf{s}$
$\Gamma(\mathbf{s}\|\theta)$	The neural network to fit the value function $V^\pi(\mathbf{s})$ with parameter θ
$Q^\pi(\mathbf{s},\mathbf{a})$	The Q-value of the executed action $\mathbf{a}$ at state $\mathbf{s}$ with policy π

the recent deep Q-learning (DQN) algorithm can utilize a neural network to learn the reward of each action [11]. The parameters of the neural network can be tuned through the trial of different caching decisions, *i.e.*, experience groups. A remarkable advantage of DQN is that an approximately optimal strategy can be found even if the system encounters a new state. This is done by utilizing the trained neural network and fits the video caching scenario well with continuous video and request arrivals.

In this section, we detail mapping the video caching problem to the DRL framework and the solution with DQN. We summarize the key notations used in our model in Table I.

A. Problem Mapping to DRL and DQN

Consider a general case in which the timeline is split into discrete slots of identical lengths and the decision is made at the end of each time slot based on the system's state with the aim of maximizing the total reward.

At time slot t, the system's state is denoted by $\mathbf{s}_t$, and the adopted action is denoted by $\mathbf{a}_t$. In the next time slot, the environment (*i.e.*, the system) evolves into state $\mathbf{s}_{t+1}$, and the obtained reward is $R_t = R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$. This notation implies that the reward at time t is determined by the current state, the action, and the next state. The objective is to find a policy π that can make an appropriate action at each time slot to maximize the total reward over time. Policy π (also called strategy) can be formally defined as $\pi = [\mathbf{a}_1, \mathbf{a}_2, \dots]$, where $\mathbf{a}_1$ is the first action for the state s_1 at time slot 1. The state space and the action space are represented by $\mathcal{S}$ and $\mathcal{A}$, respectively.

The total expected reward of a policy π (also called the V-value function) since the state $\mathbf{s}$ at time t is defined as

$$V^\pi(\mathbf{s}) = E\left[\sum_{k=0}^{\infty} \gamma^k r_{k+t} | \mathbf{s}_t = \mathbf{s}, \pi\right]. \tag{8}$$

where $\gamma < 1$ is a discount factor, and r_{k+t} is the expected reward for policy π at time slot $t+k$, *i.e.*, $r_{t+k} = E_{\mathbf{a}\sim\pi(\mathbf{s}_{t+k})}\left[R(\mathbf{s}_{t+k}, \mathbf{a}, \mathbf{s}_{t+k+1})\right]$. $V^\pi(\mathbf{s})$ is the expected reward if policy π is adopted and the initial state is $\mathbf{s}$.

The optimal policy is denoted as π_*, which should achieve the highest V-value function since the state $\mathbf{s}$. Thus we have

$$\pi^* = \arg\max_{\pi\in\Pi} V^\pi(\mathbf{s}). \tag{9}$$

Here, Π is the set of all policies. Solving Eq. (9) is very complicated given the difficulty of deriving the transition probabilities from the current state to the next state, particularly for a large-scale system with large state and action spaces.

Now, we employ the DQN algorithm [11] to solve the problem defined in Eq. (9). For a policy π, let us define the Q function as $Q^\pi(\mathbf{s},\mathbf{a}) : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$, which represents the expected reward of policy π by adopting the action $\mathbf{a}$ with the initial state $\mathbf{s}$. It means that

$$\begin{aligned} Q^\pi(\mathbf{s},\mathbf{a}) &= E\left[\sum_{k=0}^{\infty} \gamma^k r_{k+t} | \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}, \pi\right]. \\ &= \sum_{\mathbf{s}'\in\mathcal{S}} P_{\mathbf{a}}(\mathbf{s},\mathbf{s}')\left[R(\mathbf{s},\mathbf{a},\mathbf{s}') + \gamma V^\pi(\mathbf{s}')\right] \end{aligned} \tag{10}$$

For a Markov decision problem (MDP), e.g., our caching decision problem, the Q function can be rewritten recursively as

$$\begin{aligned} Q^\pi(\mathbf{s}_t,\mathbf{a}_t) = \sum_{\mathbf{s}_{t+1}\in\mathcal{S}} & P_{\mathbf{a}_t}(\mathbf{s}_t,\mathbf{s}_{t+1})\Big(R(\mathbf{s}_t,\mathbf{a}_t,\mathbf{s}_{t+1}) \\ & + \gamma Q^\pi(\mathbf{s}_{t+1},\mathbf{a}_{t+1})\Big). \end{aligned} \tag{11}$$

Here, $P_{\mathbf{a}_t}(\mathbf{s}_t,\mathbf{s}_{t+1})$ is the transition probability from state $\mathbf{s}_t$ to state $\mathbf{s}_{t+1}$ given action $\mathbf{a}_t$. Similar to the definition of the V-value function, the optimal Q-value function is defined by the maximum value of Eq. (11) achieved by any policy, which is

$$Q^*(\mathbf{s},\mathbf{a}) = \max_{\pi\in\Pi} Q^\pi(\mathbf{s},\mathbf{a}). \tag{12}$$

An optimal policy should always choose the action maximizing the Q-value function, and thus

$$\pi^*(\mathbf{s}) = \arg\max_{\mathbf{a}\in\mathcal{A}} Q^*(\mathbf{s},\mathbf{a}). \tag{13}$$

If we know the Q-value function for any state and action pair, we can choose the optimal action that maximizes the Q-value. However, it is usually difficult to precisely derive the expression of Eq. (11) for a large-scale system. Instead, we resort to approximately learning the Q-value function with a neural network. It has been proven that a neural network can extract information effectively from high-latitude input [34]. The optimal policy π^*should maximize the Q-value function, and we have

$$Q^*(\mathbf{s}_t,\mathbf{a}_t) = R_t + \gamma \max_{\pi(\mathbf{s}_{t+1})\in\mathcal{A}} Q^*(\mathbf{s}_{t+1}, \pi(\mathbf{s}_{t+1})). \tag{14}$$

Here, $R_t = R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ is the reward at time slot t. Let $Q^\pi(\mathbf{s},\mathbf{a}|\theta)$ denote the neural network that can approximate the Q-value function with parameter θ. The neural network can be trained with the experience groups obtained by trials with different actions. Specifically, we define an experience group as $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}))$ for a trial by choosing action $\mathbf{a}_t$ given that the current state is $\mathbf{s}_t$ and the resulting next state is $\mathbf{s}_{t+1}$. According to Eq. (14), we can leverage an experience group to define the loss function as

$$Loss(\theta) = \|Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t|\theta) - R_t - \gamma \max_{\mathbf{a}_{t+1} \in \mathcal{A}} Q^{\pi}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}|\theta)\|^2. \tag{15}$$

The optimal policy should minimize this loss function. Considering this principle, the DQN algorithm trains the neural network, maximizes the Q-value function, and refines the policy iteratively. The neural network can be updated according to the rule

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} Loss(\theta). \tag{16}$$

where α is the learning rate. Once the neural network is updated, the policy can be updated as

$$\pi(\mathbf{s}) = \arg\max_{\mathbf{a} \in \mathcal{A}} Q^{\pi}(\mathbf{s}, \mathbf{a}|\theta). \tag{17}$$

If the number of trials is large enough, *i.e.*, the number of experience groups is sufficient, the Q-value function $Q^{\pi}(\mathbf{s}, \mathbf{a})$ gradually approaches $Q^{*}(\mathbf{s}, \mathbf{a})$, while policy π gradually approaches π^{*}.

B. DQN-OVC: Online Video Caching with DQN

Although we mapped the video caching problem into the DRL framework with the DQN algorithm, there remain significant challenges for practical real-time implementation, particularly the large sizes of the state space and the action space.

Intuitively, the edge server should cache the video segments with the most need to maximize the hit rate, which is related to the caching state and request history. The state of video segment i at time slot t can be defined as a vector

$$\mathbf{s_t^i} = \begin{bmatrix} c_t^i & y_t^i & h_t^{i1} & h_t^{i2} & \dots & h_t^{ik} \end{bmatrix} \tag{18}$$

Here, c^i represents the caching state of video segment i; y_t^i is the number of requests for video i at time t; $h_t^{i1}, \dots, h_t^{ik}$ represents the number of requests for video i in the past k different time windows. The time windows can be flexibly set, such as the past 5 or 10 time slots. We use k different time windows to characterize the request patterns, where k is a relatively small number, *e.g.*, a default value of 6 in our experiments. Although the current definition video segment focuses on past request records, more parameters can be incorporated into our model as they become available.

For video i, there are two possible actions: cache or not, which can be represented by a binary number, i.e.,

$$a^i = \begin{cases} 1, & \text{Segment } i \text{ is (to be) cached by the edge server;} \\ 0, & \text{Otherwise.} \end{cases} \tag{19}$$

If we apply the DRL framework directly, the system state will be $\mathbf{s} = [\mathbf{s}^1, \mathbf{s}^2, \dots, \mathbf{s}^n]$ given that there are a total of n videos. Likewise, the action of the entire system is $\mathbf{a} = [a^1, a^2, \dots, a^n]$. Apparently, the sizes of both the state space and the action space are exponential, and a feasible solution can rarely be found in real time given a limited number of experience groups and polynomial training time.

1) Decoupling Value and Action: We address this issue by decoupling the value functions and the action functions in the Q-value function. We revise the value function defined in Eq. (8) according to the video caching problem as

$$\begin{aligned} V_1^{\pi}(\mathbf{s^i}) &= E\left[\sum_{k=0}^{\infty} \gamma^k R_{k+t}^i | \mathbf{s}_t^i = \mathbf{s}^i, a_t^i = 1, \pi\right] \\ V_0^{\pi}(\mathbf{s^i}) &= E\left[\sum_{k=0}^{\infty} \gamma^k R_{k+t}^i | \mathbf{s}_t^i = \mathbf{s}^i, a_t^i = 0, \pi\right]. \end{aligned} \tag{20}$$

$V_0^{\pi}(\mathbf{s^i})$ and $V_1^{\pi}(\mathbf{s^i})$ represent the expected reward when $a_t^i = 1$ and $a_t^i = 0$, respectively. Here, R_{k+t}^i is the reward at time slot $k+t$, which is largely determined by the number of requests for video i in each time slot, reflecting the value of video i.

It remains difficult to explicitly derive the expression of $V_0^{\pi}(\mathbf{s^i})$/$V_1^{\pi}(\mathbf{s^i})$ defined in Eq. (20). Different from the original DQN algorithm, we use the neural network denoted by $\Gamma(\mathbf{s}^i|\theta)$ to fit the value function $V^{\pi}(\mathbf{s}^i) = \begin{bmatrix} V_1^{\pi}(\mathbf{s}^i) & V_0^{\pi}(\mathbf{s}^i) \end{bmatrix}$. Through decoupling the value function and the action of a video, the Q-value function of video i can now be simplified as the product of i's value function and the action a^i. That is

$$\begin{aligned} Q^{\pi}(\mathbf{s}^i, \mathbf{a}^i|\theta) &= V_1^{\pi}(\mathbf{s}^i)a^i + V_0^{\pi}(\mathbf{s}^i)(1 - a^i) \\ &= \Gamma(\mathbf{s}^i|\theta) \cdot \begin{bmatrix} a^i & 1 - a^i \end{bmatrix}. \end{aligned} \tag{21}$$

The Q-value function of the entire system is defined as

$$Q^{\pi}(\mathbf{s}, \mathbf{a}|\theta) = \sum_{i=1}^{n} \Gamma(\mathbf{s}^i|\theta) \cdot \begin{bmatrix} a^i & 1 - a^i \end{bmatrix}. \tag{22}$$

We use the number of cache hits for the cached videos on the edge server as the reward. If a video is not cached, the reward offered by the video is 0, that is, $V_0^{\pi}(\mathbf{s}^i) = 0$. At this point, the neural network only needs to predict $V_1^{\pi}(\mathbf{s}^i)$. We further let $\Gamma_1(\mathbf{s}^i|\theta) = V_1^{\pi}(\mathbf{s}^i)$. If $a^i = 1$, the Q-value function is the same as the video's value; If $a^i = 0$, the Q-value of video i is 0. The Q-value function of a particular video i can then be defined as

$$Q^{\pi}(\mathbf{s}^i, \mathbf{a}^i|\theta) = V^{\pi}(\mathbf{s}^i)\mathbf{a} = \Gamma_1(\mathbf{s}^i|\theta)a^i. \tag{23}$$

The Q-value function of the entire system is calculated as

$$Q^{\pi}(\mathbf{s}, \mathbf{a}|\theta) = \sum_{i=1}^{n} \Gamma_1(\mathbf{s}^i|\theta)a^i. \tag{24}$$

The decoupling operation is an essential simplification that greatly reduces the computational complexity of our algorithm, which can be explained from two perspectives:

- To reduce the computational complexity, we fit the value function of all video segments with a single neural network. Specifically, our approach sets up a neural network to fit the value of each video segment. The neural network is trained with experience groups contributed by all video segments.
- The size of the action space is still exponential in our algorithm. However, the optimal policy is straightforward if the values of videos are known. By considering the constraints C_2 and C_3, the optimal policy should use the most valuable video that has yet to be cached to replace the cached video with the least value.

2) Neural Network Training: With the newly defined Q-value function, we can design DQN-OVC by revising the original DQN algorithm. According to the DRL framework, the optimal Q-value function should satisfy the relation in Eq. (14). For the online video caching problem, let $\mathcal{A}_{\mathbf{a}_t}$ denote the set of available actions we can take based on the current action $\mathbf{a}_t$ with constraints C_2 and C_3. The maximum Q-value function should meet the revised relation as

$$Q^*(\mathbf{s}_t, \mathbf{a}_t) = R_t + \max_{\pi(\mathbf{s}_{t+1}) \in \mathcal{A}_{\mathbf{a}_t}} Q^*(\mathbf{s}_{t+1}, \pi(\mathbf{s}_{t+1})). \quad (25)$$

Here, we set $\gamma = 1$ as we focus on a fixed period, and the numbers of hits at different time slots are identical. The cardinality of $\mathcal{A}_{\mathbf{a}_t}$ is much smaller than that of $\mathcal{A}$. As we will show later, the best action in $\mathcal{A}_{\mathbf{a}_t}$ can be determined without computing each action's Q-value. R_t is the reward obtained at time slot t by choosing an action $\mathbf{a}_t$. For our video caching problem, R_t is defined as the number of successful hits at time slot t. Recall that we count the number of requests that have completed their video streaming from the edge server at time slot t as the number of successful hits at time t. Thus we define the rewards at time t as

$$R_t = \sum_{j=1}^{m} q_j^t. \quad (26)$$

where q_j^t is defined in the last section with value 1 if the request j has completed its video streaming from the edge server at time slot t, and 0 otherwise.

With an experience group, the loss function is defined as

$$\begin{aligned} Loss(\theta) =& \| \sum_{i=1}^{n} \Gamma_1(\mathbf{s}_t^i|\theta) a_t^i - R_t - \\ & \max_{\mathbf{a}_{t+1} \in \mathcal{A}_{\mathbf{a}_t}} \sum_{i=1}^{n} \Gamma_1(\mathbf{s}_{t+1}^i|\theta) a_{t+1}^i \|^2. \\ =& \|\Gamma_1(\mathbf{s}_t|\theta) \cdot \mathbf{a}_t - R_t - \max_{\mathbf{a}_{t+1} \in \mathcal{A}_{\mathbf{a}_t}} \Gamma_1(\mathbf{s}_{t+1}|\theta)\mathbf{a}_{t+1}\|^2. \end{aligned} \quad (27)$$

where $\mathbf{a}_{t+1} = [a_{t+1}^1, \ldots, a_{t+1}^n]$. The neural network $\Gamma_1(\mathbf{s}^i|\theta)$ can be trained by substituting the new loss function of Eq. (27) back to Eq. (16).

3) Action Choice: To obtain the optimal caching policy, we still need to revise Eq. (17). Recall that there are constraints C_2 and C_3, which restrict that at most L, videos be downloaded at the same time from the cloud server. Intuitively, if the caching space is not fully occupied, the edge server only needs to download the missing videos of the highest values to fill up the caching space. Otherwise, the cached videos with the lowest values are replaced. If their values are less than the values of certain missing videos, they are replaced by these videos. The value of each video is estimated by the neural network $\Gamma_1(\mathbf{s}_t|\theta)$ at different states.

Alg. 1 presents the action choice algorithm, which describes the cache decisions based on constraints of C2 and C3. Briefly, Alg. 1 has two stages.

1) Stage 1 (from line 2 to line 16): In this stage, the edge server collects candidate videos that may be cached in the edge server, namely, the videos already cached by the edge server and the best L videos that are missing from the edge server. Since $C + L \ll n$, this stage can significantly reduce the computational load to select videos in the next stage;
2) Stage 2 (from line 17 to line 31): The edge server selects the videos to be cached and the videos to be replaced from the candidate list suggested in Stage 1.

4) The DQN-OVC Algorithm: Finally, we present our DQN-OVC algorithm in Alg. 2. Alg. 2 has three stages, which are briefly described below.

1) Stage 1 (from line 4 to line 7): The edge server collects necessary information from the environment, such as state s and reward R, which are used later for decision-making and model training;
2) Stage 2 (from line 8 to line 18). The edge server selects action a based on its state s with Alg. 1. The cached video is updated by action a.
3) Stage 3 (from line 19 to line 28): The edge server obtains a new experience group based on its video caching decision in Stage 2. A recent experience group is randomly selected from the experience group pool to update the neural network (which is used to predict the value of each video).

V. Performance Evaluation

We evaluated our DQN-OVC algorithm under diverse configurations with both real-world trace data and synthetic data and compared it with state-of-the-art solutions. In this section, we present representative results to demonstrate the effectiveness of our solution.

A. Dataset

1) Real Dataset: We collected user request records from Tencent Video, one of the largest online video service providers in China, for a one-month period. There are 162,597,411 records in total, including requests for assorted video categories such as TV series, movies, news, MV, and UGC. As mentioned earlier, to simplify caching management, all videos are divided into segments with a default size of

Algorithm 1: The action choice algorithm

```
Input:
    The state of all the videos $\mathbf{s}_t^i$;
    The cache size of the edge server $C$;
Output:
    The best action of this time slot $\mathbf{a}_t$;
1: Get the value of each video $\gamma = \Gamma(\begin{bmatrix}\mathbf{s}_t^1 & \mathbf{s}_t^2 & \dots & \mathbf{s}_t^n\end{bmatrix})$
2: /* Collect the cached video and the best $L$ uncached video*/
3: Build List $\mathcal{V}_c$
4: Build Heap $\mathcal{V}_{uc}$ where the head node is the minimum value node
5: for $v_i$ in $\mathcal{V}$ do
6:     if $c^i = 1$ then
7:         $\mathcal{V}_c.append(v_i)$
8:     else
9:         if $\gamma_i > \mathcal{V}_{uc}.head$ then
10:            $\mathcal{V}_{uc}$.push($v_i$)
11:            if $\mathcal{V}_{uc}$.len $> L$ then
12:                $\mathcal{V}_{uc}$.pop()
13:            end if
14:        end if
15:    end if
16: end for
17: Build Heap $\mathcal{V}'$ with $\mathcal{V}_c \cup \mathcal{V}_{uc}$ where the head node is the maximum value node
18: /* Make decisions based on the limit of C2 and C3 */
19: Initialize feasible action $\mathbf{a}_t = \begin{bmatrix}0 & 0 & \dots & 0\end{bmatrix}$
20: $c = 0, k = 0$
21: while $\mathcal{V}'.len > 0$ and c<C do
22:     /* Cache decisions are made from the most profitable videos in turn */
23:     $v_i$ = $\mathcal{V}'$.pop()
24:     if $c^i = 1$ then
25:         $\mathbf{a}_t[i] = 1; c = c + 1$
26:     else
27:         if k<K then
28:             $\mathbf{a}_t[i] = 1; c = c + 1; k = k + 1$
29:         end if
30:     end if
31: end while
32: return $\mathbf{a}_t$
```

Algorithm 2: The DQN-OVC algorithm

```
1: Initialize $\Gamma()$ with $\theta$.
2: Initialize experience pool $\mathbb{E}$
3: for every time slot $t$ do
4:     /* Get status $\mathrm{s}$ and reward $R$ from environment */
5:     Respond to user requests
6:     Statistical $R_t = \sum_{\forall j} q_j^t$ by Eq. (4)
7:     Update $\mathbf{s}_t$ with current request status
8:     /* Agent makes decisions and updates the cached video based on $\mathbf{a}_t$ */
9:     Get the best action $\mathbf{a}_t$ by Alg. 1 base on $\mathbf{s}_t$
10:    for $a_t^i$ in $\mathbf{a}_t$ do
11:        if $a_t^i = 1$ and $a_{t-1}^i = 0$ then
12:            caches video $i$
13:        end if
14:        if $a_t^i = 0$ and $a_{t-1}^i = 1$ then
15:            deleted video $i$ from the edge server
16:            Update $c_t^i$ by Eq. (6)
17:        end if
18:    end for
19:    /* Save experience and training agents */
20:    $E_t = (\mathbf{s}_t, \mathbf{a}_t, R_t, \mathbf{s}_{t+1})$
21:    Save experience group $\mathbb{E}$.append($E_t$)
22:    if $\mathbb{E}$.len > max experience pool size then
23:        Remove the earliest experience group
24:    end if
25:    Pick a random sample $E_r = (\mathbf{s}_r, \mathbf{a}_r, R_r, \mathbf{s}_{r+1})$
26:    Get the best action $\mathbf{a'}_{r+1}$ by Alg. 1 based on $\mathbf{s}_{r+1}$
27:    Update
       $\theta = \theta - \alpha\nabla_\theta\Gamma_1(\mathbf{s}_r)\cdot\mathbf{a}_r - R_r - \Gamma_1(\mathbf{s}_{r+1})\cdot\mathbf{a'}_{r+1}$
28: end for
```

100 MB, which results in a total number of 26,903,816 video segments for the requests.

The distribution of the popularity of these video segments is highly skewed. There is only a small portion of segments (approximately 15%) that are popular, attracting most of the requests. In contrast, 85% of the video segments were requested less than once a day, and 40% of the segments were requested only once in the 30-day trace collection period. Given the large user base, the total number of user requests was quite large — during peak hours, there could be approximately 37,000 requests in 30 seconds.

2) Synthetic Dataset: We also tested our algorithm with a synthetic dataset as in set [35]. The dataset consists of 5,000 files and 10,000 requests, which were generated according to the Zipf distribution:

$$f(i;\beta,N) = \frac{1/i^\beta}{\sum_{n=1}^{N} 1/n^\beta} \tag{28}$$

where i is the file rank, N is the total number of videos, and β is the Zipf parameter that shapes the distributions, which is set to 1.3 in our experiments.

B. Experimental Settings

1) Settings of Edge Server: We consider a typical scenario in which an edge server is deployed to serve users in a municipality. The average bandwidth for users to access the cloud/edge server is set to 8/25 Mbps, and the average bandwidth for the edge server to fetch a video from the cloud server is set to 100 Mbps.

The edge server makes caching decisions every second. The maximum number of videos that the edge server can simultaneously download from the cloud server is set to $L = 50$. We set the cache capacity of the edge server to 100, 200, 300, 400, 500, 600, 700, 800, and 1,000 segments in our experiments, which are all constrained in comparison to the total number of video segments.

2) Settings of DQN-OVC: The number of requests over the past 30 seconds is kept as the system state, which is divided into 6 time windows of equal duration. In other words, the duration of each time window h_t^{ik} in the system state lasts 5 seconds.

The number of hit requests R_t on the edge server is used as the reward at time slot t. We use the record in the past hour as experience groups. Since the edge server performs an action every second, we maintain the latest 3,600 experience groups to train the neural network. In every second, 2 to 5 experience groups are randomly selected for training, and the neural network model used by the edge server is updated every 10 minutes.

Since we use 6 time windows, there are 8 elements in the system state. Thus, the neural network has 8 input nodes and 1 output node. We set up two hidden layers, and there are 10 nodes in each hidden layer. We set the learning rate as $1e-9$, which is a relatively low rate that avoids overfitting.

3) Performance Results: We report the experimental results, *i.e.*, hit rates, for the first 4 days because the pattern approximately repeats for the 30 days.

C. Baseline Algorithms

We compare the DQN-OVC algorithm with three baseline algorithms: heuristic caching algorithms, popularity prediction-based algorithms, and advanced DRL-based caching algorithms. An offline optimal algorithm (OFF) is also implemented by assuming that the edge server can always make the best caching decisions with the knowledge of future requests. For a fair comparison, all caching algorithms can obtain user request records in the past 30 seconds.

1) Heuristic Algorithms: For heuristic algorithms, we implement LFU and AViC as representatives. LFU [36] selects the segment with the most historical requests for caching based on the number of user requests in the past 30 seconds. The AViC algorithm [37], removes the segment with the furthest request time in the future through request prediction. In the streaming mode with continuous access to the video segments, the algorithm estimates the expected access time of the next segment based on the length of the segment.

The FIFO algorithm needs to maintain the start caching time of each video. Whenever a new uncached video is requested, the original cached video is replaced. The LRU algorithm is similar, except that it records the last request time of each video. Each time a new uncached video is requested, the originally requested video is replaced.

By considering the replacement cost, it is not feasible to apply LRU and FIFO with the Tencent Video dataset because these two algorithms update cached videos with every new request. The request rate in the Tencent Video dataset is so high that the caching replacement cannot be finished before a new request arrives. Thus, we only compare LRU and FIFO with the synthetic dataset.

2) Supervised Learning-Based Algorithms: We implement LSTM and FNN as two representatives for supervised learning-based algorithms. Both the LSTM algorithm [19] and the FNN algorithm [38] utilize neural networks to predict future popularity changes and make caching decisions based on the predicted popularity. In our experiments, both algorithms predict the number of video requests in the next 20 seconds.

For both algorithms, there are 18 input nodes and 6 output nodes. The FNN sets up a fully connected neural network, while the other uses an LSTM network. There are two hidden layers, and each hidden layer has 10 nodes.

D. Experimental Results

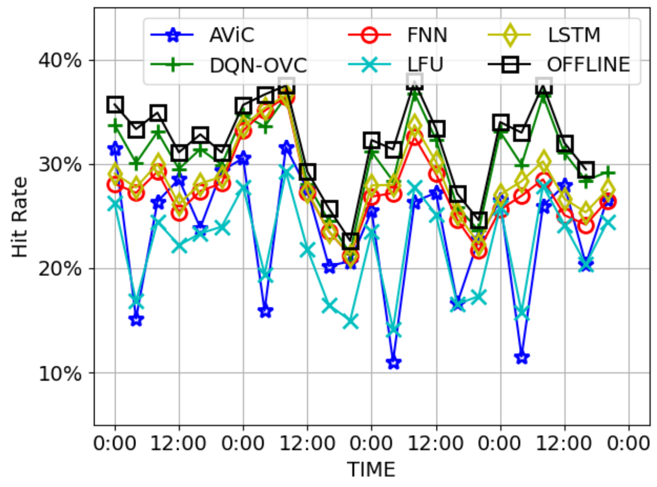


Fig. 3: Comparing hit rates when the cache size is 500

1) Hit rate over time: The first experiment is designed to compare the hit rate performance of different caching algorithms over time by fixing the caching capacity at $C = 500$. We plot the experimental results in Figs. 3, in which the x-axis represents the time over the 4 days while the y-axis represents the hit rate. Although our algorithm makes caching decisions every second, we only plot the average hit rate every 4 hours in case the data points are too dense in the figure.

Note that OFF is best since it knows future requests in advance, which is impractical in the real world; however, it offers a performance bound for evaluating the remaining room for improvement. By comparing the hit rate performance of these caching algorithms, we make the following observations:

- DQN-OVC outperforms all other caching algorithms and is very close to OFF. One of the reasons that DQN-OVC outperforms these baselines is that the replacement cost is considered in our algorithm, which has been largely neglected by these baselines. In particular, the performance gap between DQN-OVC and other algorithms increases during peak hours from 9:00 to

TABLE II: Comparing the running time of different algorithms

Algorithm	Run time per decision
DQN-OVC	6.669ms
LSTM	25.790ms
FNN	5.230ms

10:00 and 18:00 to 24:00. This indicates that DQN-OVC can rapidly adjust cached videos during peak hours according to the latest request trend.

- LSTM and FNN are better than LFU and AViC, and LSTM is slightly better than FNN because of its more advanced model. The AViC algorithm also makes caching decisions based on video popularity prediction; it is a simple heuristic algorithm. It requires user continuous access to the video segment to be effective. At peak hours, it achieves a higher hit rate than other algorithms based on neural networks. However, in the early morning hours, when the number of requests is relatively small, it is worse than the LFU algorithm, possibly because users frequently pause or stop video playback.

In addition, we compare the average running time of DQN-OVC with that of other algorithms. As we can see in Table II, the running time of DQN-OVC is comparable to that of LSTM and FNN at the scale of milliseconds. The low running time of DQN-OVC can support the edge server in making caching decisions every second.

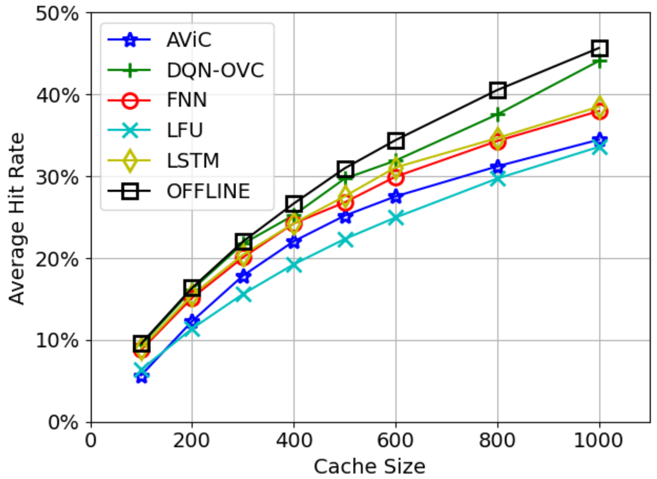


Fig. 4: Comparing different caching algorithms with different caching capacities.

2) Varying Caching capacity: In Fig. 4, we compare the average hit rates of the caching algorithms by varying the caching capacity from 100 to 1,000. The x-axis is the caching size, whereas the y-axis is the average hit rate. This experiment is designed to demonstrate how the caching capacity influences the hit rate performance. The average hit rate increases monotonically with caching capacity for all algorithms. It is worth noting that the DQN-OVC curve is very stable, which indicates it always considerably outperforms other algorithms except for OFF.

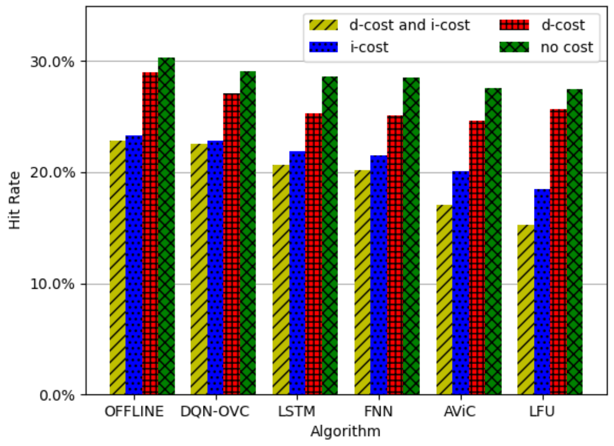


Fig. 5: Comparing the average hit rates of different algorithms by different settings of the replacement cost.

3) Comparison with Different Settings of Replacement Cost: To evaluate how the replacement cost affects the hit rate performance, we conduct this experiment to compare the average hit rate of each algorithm with different settings of the replacement cost.

As we have discussed in Sec. III, there are two kinds of replacement cost, based on which we setup the replacement cost in four different ways:

1) I-cost and d-cost: Both download cost and interruption cost are included.
2) D-cost: Only download cost is included.
3) I-cost: Only interruption cost is included.
4) No cost: No replacement cost is included.

No cost means that the replacement operation of a particular video can be completed instantly. Although ignoring the replacement cost is impractical in real systems, we can better evaluate other baselines since the replacement cost has not been considered in their design.

The cache size of each algorithm is fixed at 300. The experiment results are presented in Fig. 5, from which we can observe that:

- The hit rate performance of DQN-OVC is always better than other baselines with different settings for the replacement cost. The reason is that the deep reinforcement learning inside DQN-OVC can automatically learn experiences under different environments such as different settings of the replacement cost.
- The replacement cost lowers the hit rate performance for all algorithms because the hits during the period of video replacement are removed from the statistics of the hit rates.
- The improvement of the hit rate achieved by DQN-OVC is more significant when the replacement cost is considered. This result indicates that DQN-OVC can

outperforms baselines better in real systems in which the replacement cost must be included.

4) Comparison of Improvement Room: To visualize the remaining improvement room of each caching algorithm, we normalize the hit rate of each algorithm to the best hit rate achieved by the OFF algorithm. The normalized hit rate is in the range from 0 to 1. If it is closer to 1, it implies that the remaining improvement room is smaller, and vice versa.

The experimental results are plotted in Fig. 6, in which the normalized hit rates are compared over time. Again, DQN-OVC achieves the best-normalized hit rate. Interestingly, we find that the normalized hit rate of DQN-OVC is close to 1 during the peak hours, especially from 18:00 to 23:00 each day. In contrast, there is room for performance improvement during light-load periods, such as early morning. We speculate that a large number of user requests emerge during the peak hours only for a few popular videos. In this case, our DQN-OVC can rapidly react according to the latest trend and consequently achieves the best hit rate performance. However, during the light-load periods, there is less concentration on the user requests, and thus, the hit rates of all caching algorithms degrade, even though the total number of user requests is lower.

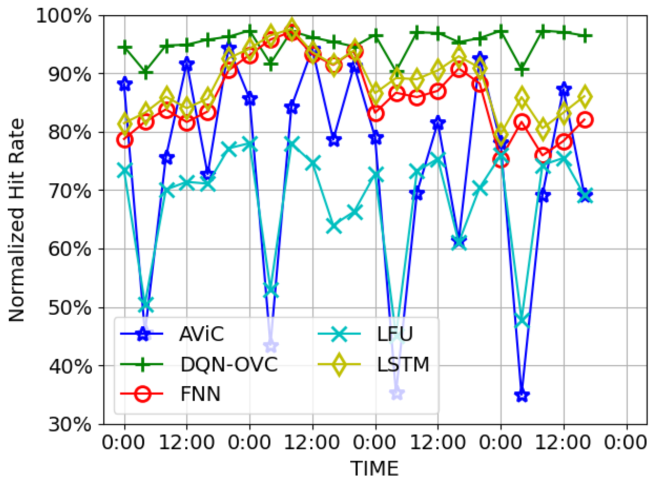


Fig. 6: Comparing the normalized hit rates of different algorithms over time.

5) Comparing DRL-based Algorithms: Finally, we further compare DQN-OVC with other DRL-based algorithms; in particular, the Wolpertinger algorithm developed by Zhong *et al.* very recently [35]. The algorithm adopts the Wolpertinger structure to make caching decisions and, based on the actor output, uses the K-neighbor algorithm to find the executable set of discrete actions. The expected reward of each executable discrete action is calculated, and the action with the highest reward is selected for execution.

In its implementation, the Wolpertinger algorithm considers the number of requests for a file within the most recent 10, 100, 1,000 requests to make caching decisions and uses two neural networks with 384 and 96 hidden nodes. In contrast, we only use a small neural network with 6 hidden nodes.

TABLE III: Comparing the running time of DRL-based algorithms

Algorithm	Runtime per decision
DQN-OVC	0.207ms
Wolpertinger	1.513ms

Because of the large action space and the high complexity of neural networks, the Wolpertinger algorithm cannot be directly applied to the Tencent Video dataset with high-speed request rates. For this comparison, instead of using a real dataset as in the above experiment, we use the synthetic dataset generated according to reference [35].

The comparison of the hit rate with the synthetic dataset is presented in Fig. 7. We can observe that the performance of DQN-OVC is almost identical to that of Wolpertinger, and both of them significantly outperform other heuristic algorithms.

The advantage of DQN-OVC, however, lies in the low computational complexity. We further compare the running time of DQN-OVC and Wolpertinger in Table III. Notably, the running time of our algorithm is much shorter than that of Wolpertinger. This much shorter running time enables us to apply highly adaptive and responsive DQN-OVC, particularly in large-scale systems with a massive number of user requests.

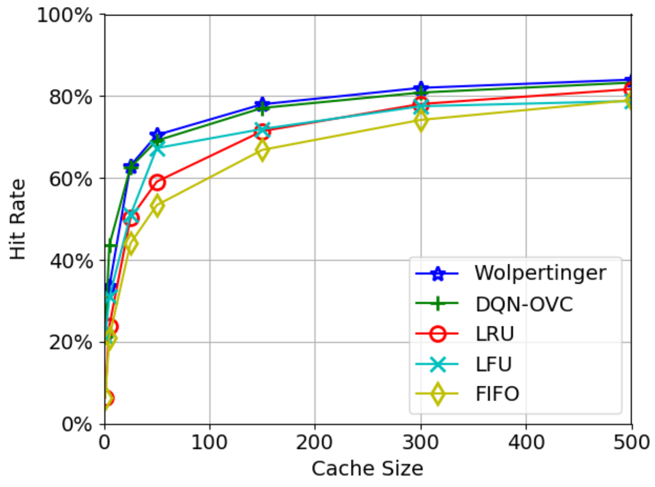


Fig. 7: Comparing different caching algorithms with different caching capacities.

VI. Conclusion

In this paper, we examined the cost of online video caching at edges and proposed a novel algorithm for real-time cost-aware caching. Our algorithm, leveraging the deep reinforcement learning framework, optimizes the cache hit rate with more realistic constraints than existing studies. We enhanced the basic deep Q-learning through smart space reduction, which minimizes the computational time, making the implementation highly adaptive and responsive. Experimental results show that our algorithm can achieve a much better hit rate than baseline algorithms under diverse configurations. In the future, we will seek to deploy our algorithm in real systems, address various practical challenges, and closely investigate how to further improve its hit rate during light-load periods.

Acknowledgement

This work has been partially supported by National Key R&D Program of China under Grant No.2018YFB1800302 and No.2018YFB1800805, National Natural Science Foundation of China under Grant No.61772345, 61772139, 61902257 and 61872215, Natural Science Foundation of Guangdong Province under No. 2021A1515012633, Shenzhen Science and Technology Program under Grant No. RCYX20200714114645048, No. JCYJ20190808142207420, No. GJHZ20190822095416463 and No. RCYX20200714114523079, the Pearl River Young Scholars funding of Shenzhen University, the project "PCL Future Greater-Bay Area Network Facilities for Large-scale Experiments and Applications (LZC0019)" and Australia Research Council under DE180100950.

References

[1] Sandvine, "Global internet phenomena report," Tech. Rep., 2019.

[2] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5g networks: New paradigms, scenarios, and challenges," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 54–61, April 2017.

[3] Cisco, "Cisco visual networking index: Global mobile data traffic forecast update 2015-2020," Tech. Rep., 2016.

[4] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. Leung, "Cache in the air: Exploiting content caching and delivery techniques for 5g systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 131–139, 2014.

[5] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[6] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, Fourthquarter 2017.

[7] L. Chen, L. Song, J. Chakareski, and J. Xu, "Collaborative content placement among wireless edge caching stations with time-to-live cache," *IEEE Transactions on Multimedia*, vol. 22, no. 2, pp. 432–444, Feb 2020.

[8] M. Peng, S. Yan, K. Zhang, and C. Wang, "Fog-computing-based radio access networks: Issues and challenges," *Ieee Network*, vol. 30, no. 4, pp. 46–53, 2016.

[9] H. S. Goian, O. Y. Al-Jarrah, S. Muhaidat, Y. Al-Hammadi, P. Yoo, and M. Dianati, "Popularity-based video caching techniques for cache-enabled networks: A survey," *IEEE Access*, vol. 7, pp. 27 699–27 719, 2019.

[10] S.-H. Park, O. Simeone, and S. Shamai, "Joint cloud and edge processing for latency minimization in fog radio access networks," in *2016 IEEE 17th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, 2016, pp. 1–5.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[12] Z. Ye, F. D. Pellegrini, R. El-Azouzi, L. Maggi, and T. Jimenez, "Quality-aware dash video caching schemes at mobile edge," in *2017 29th International Teletraffic Congress (ITC 29)*, vol. 1, Sep. 2017, pp. 205–213.

[13] H. Pang, J. Liu, X. Fan, and L. Sun, "Toward smart and cooperative edge caching for 5g networks: A deep learning based approach," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, June 2018, pp. 1–6.

[14] Y. Zhou, L. Chen, C. Yang, and D. M. Chiu, "Video popularity dynamics and its implication for replication," *IEEE Transactions on Multimedia*, vol. 17, no. 8, pp. 1273–1285, Aug 2015.

[15] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, "Adaptsize: Orchestrating the hot object memory cache in a content delivery network," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 483–498.

[16] N. Beckmann, H. Chen, and A. Cidon, "Lhd: Improving cache hit rate by maximizing hit density," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 389–403.

[17] P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang, and X. Shen, "Content popularity prediction towards location-aware mobile edge caching," *IEEE Transactions on Multimedia*, vol. 21, no. 4, pp. 915–929, April 2019.

[18] P. Zhou, K. Wang, J. Xu, and D. Wu, "Differentially-private and trustworthy online social multimedia big data retrieval in edge computing," *IEEE Transactions on Multimedia*, vol. 21, no. 3, pp. 539–554, March 2019.

[19] A. Narayanan, S. Verma, E. Ramadan, P. Babaie, and Z.-L. Zhang, "Deepcache: A deep learning based framework for content caching," in *Proceedings of the 2018 Workshop on Network Meets AI & ML*. ACM, 2018, pp. 48–53.

[20] Z. Song, D. S. Berger, K. Li, A. Shaikh, W. Lloyd, S. Ghorbani, C. Kim, A. Akella, A. Krishnamurthy, E. Witchel *et al.*, "Learning relaxed belady for content distribution network caching," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 529–544.

[21] J. Sung, K. Kim, J. Kim, and J. K. Rhee, "Efficient content replacement in wireless content delivery network with cooperative caching," in *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec 2016, pp. 547–552.

[22] C. Zhong, M. C. Gursoy, and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," in *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*, March 2018, pp. 1–6.

[23] M. Zhang, H. Luo, and H. Zhang, "A survey of caching mechanisms in information-centric networking," *IEEE Communications Surveys Tutorials*, vol. 17, no. 3, pp. 1473–1499, thirdquarter 2015.

[24] Z. Piao, M. Peng, Y. Liu, and M. Daneshmand, "Recent advances of edge cache in radio access networks for internet of things: Techniques, performances, and challenges," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 1010–1028, Feb 2019.

[25] J. Kwak, Y. Kim, L. B. Le, and S. Chong, "Hybrid content caching in 5g wireless networks: Cloud versus edge caching," *IEEE Transactions on Wireless Communications*, vol. 17, no. 5, pp. 3030–3045, May 2018.

[26] Y. Guo, B. Zou, J. Ren, Q. Liu, D. Zhang, and Y. Zhang, "Distributed and efficient object detection via interactions among devices, edge, and cloud," *IEEE Transactions on Multimedia*, vol. 21, no. 11, pp. 2903–2915, Nov 2019.

[27] D. Liu, B. Chen, C. Yang, and A. F. Molisch, "Caching at the wireless edge: design aspects, challenges, and future directions," *IEEE Communications Magazine*, vol. 54, no. 9, pp. 22–28, Sep. 2016.

[28] A. Mehrabi, M. Siekkinen, and A. Ylä-Jääski, "Qoe-traffic optimization through collaborative edge caching in adaptive mobile video streaming," *IEEE Access*, vol. 6, pp. 52 261–52 276, 2018.

[29] J. Yao, T. Han, and N. Ansari, "On mobile edge caching," *IEEE Communications Surveys Tutorials*, vol. 21, no. 3, pp. 2525–2553, thirdquarter 2019.

[30] S. Kumar, D. S. Vineeth, and A. F. A, "Edge assisted dash video caching mechanism for multi-access edge computing," in *2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, Dec 2018, pp. 1–6.

[31] M. Mukherjee, L. Shu, and D. Wang, "Survey of fog computing: Fundamental, network applications, and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 1826–1857, 2018.

[32] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," *IEEE access*, vol. 6, pp. 6900–6919, 2017.

[33] J. Ren, H. Guo, C. Xu, and Y. Zhang, "Serving at the edge: A scalable

iot architecture based on transparent computing," *IEEE Network*, vol. 31, no. 5, pp. 96–105, 2017.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[35] C. Zhong, M. C. Gursoy, and S. Velipasalar, "Deep reinforcement learning-based edge caching in wireless networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 1, pp. 48–61, 2020.

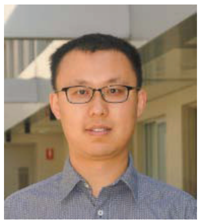
[36] L. Maggi, L. Gkatzikis, G. Paschos, and J. Leguay, "Adapting caching to audience retention rate: Which video chunk to store?" *arXiv preprint arXiv:1512.03274*, 2015.

[37] Z. Akhtar, Y. Li, R. Govindan, E. Halepovic, S. Hao, Y. Liu, and S. Sen, "Avic: a cache for adaptive bitrate video," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, 2019, pp. 305–317.

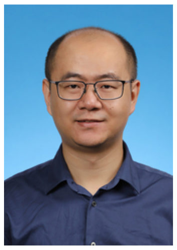
[38] V. Fedchenko, G. Neglia, and B. Ribeiro, "Feedforward neural networks for caching: n enough or too much?" *ACM SIGMETRICS Performance Evaluation Review*, vol. 46, no. 3, pp. 139–142, 2019.

Laizhong Cui is currently a professor in the College of Computer Science and Software Engineering at Shenzhen University, China. He received a BS degree from Jilin University, Changchun, China, in 2007 and a PhD degree in computer science and technology from Tsinghua University, Beijing, China, in 2012. His research interests include future internet architecture and protocols, edge computing, multimedia systems and applications, blockchain, Internet of Things, cloud and big data computing, computational intelligence and machine learning. He led more than 10 scientific research projects, including the National Key Research and Development Plan of China, National Natural Science Foundation of China, Guangdong Natural Science Foundation of China and Shenzhen Basic Research Plan. He has published more than 70 papers in journals including IEEE Transactions on Knowledge and Data Engineering, IEEE Transactions on Multimedia, IEEE IoT Journal, IEEE Transactions on Industrial Informatics, IEEE Transactions on Vehicular Technology, IEEE Transactions on Network and Service Management, ACM Transactions on Internet Technology, IEEE Transactions on Computational Biology and Bioinformatics and IEEE Network. He serves as an associate editor or an editorial board member for several international journals, including IEEE Transactions on Network and Service Management, International Journal of Machine Learning and Cybernetics, International Journal of Bio-Inspired Computation, and Ad Hoc and Sensor Wireless Networks. He is a senior member of the IEEE and a senior member of the CCF.

Erchao Ni received a BEng degree from Shenzhen University, Shenzhen, China, in 2018. He earned his MPhil degree from the College of Computer Science and Software Engineering of Shenzhen University in 2020. His research interests include edge computing and reinforcement learning.



Yipeng Zhou is a Senior Lecturer in computer science with the Department of Computing at Macquarie University and the recipient of the ARC DECRA in 2018. From August 2016 to February 2018, he was a research fellow with the Institute for Telecommunications Research (ITR) of the University of South Australia. From 2013 to 2016, he was a Lecturer at the College of Computer Science and Software Engineering, Shenzhen University. He was a Postdoctoral Fellow with the Institute of Network Coding (INC) of the Chinese University of Hong Kong (CUHK) from Aug. 2012 to Aug. 2013. He earned his PhD degree and MPhil degree from the Information Engineering (IE) Department of CUHK, and his Bachelor degree in computer science from the University of Science and Technology of China (USTC). His research interests lie in federated learning, differential privacy and edge computing



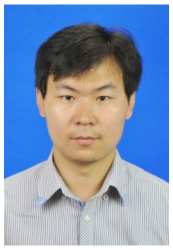
Zhi Wang is currently an associate professor at Shenzhen International Graduate School, Tsinghua University. He received his Ph.D. in 2014 and his B.E. in 2008, both from Tsinghua University. His research areas include edge computing, distributed machine learning, and multimedia networks. He is a recipient of the Natural Science Award of the Ministry of Education (First Prize) in 2017, the National Natural Science Award (Second Prize) in 2018, the Shenzhen Youth Science and Technology Award in 2019, and the Technology Invention Award of the Chinese Institute of Electronics (First Prize) in 2020. In addition, his research won the Best Paper Award of ACM Multimedia, the Outstanding Doctoral Thesis Award of China Computer Federation, the Best Student Paper Award of MMM, and the Best Paper Award of ACM Multimedia, HUMA Workshop.

Lei Zhang (S'12-M'19) received a BEng degree from the Advanced Class of Electronics and Information Engineering, Huazhong University of Science and Technology, Wuhan, China, in 2011, and an MS degree and the PhD degree from Simon Fraser University, Burnaby, BC, Canada, in 2013 and 2019, respectively. He is a recipient of the C.D. Nelson Memorial Graduate Scholarship (2013) and Best Paper Finalist at IEEE/ACM IWQoS (2016). He is currently an assistant professor at the College of Computer Science and Software Engineering, Shenzhen University. His research interests include multimedia systems and applications, mobile cloud computing, edge computing, social networking, and the Internet of Things.

Jiangchuan Liu (S’01-M’03-SM’08-F’17) is a university professor in the School of Computing Science, Simon Fraser University, British Columbia, Canada. He is a Fellow of The Canadian Academy of Engineering, an IEEE Fellow, and an NSERC E.W.R. Steacie Memorial Fellow. He was an EMC-Endowed Visiting Chair Professor of Tsinghua University (2013-2016). In the past, he worked as an assistant professor at the Chinese University of Hong Kong and as a research fellow at Microsoft Research Asia.

He received a BEng degree (cum laude) from Tsinghua University, Beijing, China, in 1999, and a PhD degree from Hong Kong University of Science and Technology in 2003, both in computer science. He is a corecipient of the inaugural Test of Time Paper Award of IEEE INFOCOM (2015), ACM SIGMM TOMCCAP Nicolas D. Georganas Best Paper Award (2013), and ACM Multimedia Best Paper Award (2012).

His research interests include multimedia systems and networks, cloud and edge computing, social networking, online gaming, and Internet of things/RFID/backscatter. He has served on the editorial boards of IEEE/ACM Transactions on Networking, IEEE Transactions on Big Data, IEEE Transactions on Multimedia, IEEE Communications Surveys and Tutorials, and IEEE Internet of Things Journal. He is a steering committee member of IEEE Transactions on Mobile Computing and Steering Committee Chair of IEEE/ACM IWQoS (2015-2017). He is a TPC Cochair of IEEE INFOCOM’2021.



Yuedong Xu is an associate professor in the School of Information Science and Technology, Fudan University, China. He received his PhD in computer science from the Chinese University of Hong Kong (with Prof. John C.S. Lui), MS degree from Huazhong University of Science and Technology, and BS degree from Anhui University. From late 2009 to 2012, he was a postdoc with INRIA Sophia Antipolis and Universite d’Avignon, France (with Prof. Eitan Altman). His areas of interest include performance evaluation, data analytics, machine learning and economic analysis of computer networks.