DeepCast: Towards Personalized QoE for Edge-Assisted Crowdcast with Deep Reinforcement Learning

Fangxin Wang, *Student Member, IEEE,* Cong Zhang, *Member, IEEE,* Feng Wang, *Senior Member, IEEE,* Jiangchuan Liu, *Fellow, IEEE,* Yifei Zhu, *Student Member, IEEE,* Haitian Pang, *Student Member, IEEE,* and Lifeng Sun, *Senior Member, IEEE*

***Abstract*—Today's anywhere and anytime broadband connection and audio/video capture have boosted the deployment of crowdsourced livecast services (or *crowdcast*). Bridging a massive amount of geo-distributed broadcasters and their fellow viewers, such representatives as Twitch.tv, Youtube Gaming, and Inke.tv, have greatly changed the generation and distribution landscape of streaming content. They also enable rich online interactions among the crowd, and strive to offer personalized Quality-of-Experience (QoE) for individual viewers. Given the ultra-large scale and the dynamics of the crowd, personalizing QoE however is much more challenging than in early generation streaming services. The rich interactions among the broadcasters, viewers, and the network system, on the other hand, also offer invaluable data that could be utilized towards informed management. This paper presents *DeepCast*, an edge-assisted crowdcast framework that explores the sheer amount of viewing data towards intelligent decisions for personalized QoE demands. DeepCast seamlessly integrates cloud, CDN, and edge servers for crowdcast content distribution, and advocates a data-driven design that extracts the hidden information from the complex interactions among the system components. Through deep reinforcement learning (DRL), it automatically identifies the most suitable strategies for viewer assignment and transcoding at edges. We collect multiple real-world datasets and evaluate the performance of DeepCast with trace-driven experiments. The results demonstrate its flexibility and effectiveness towards better personalized QoE and lower cost for crowdcast systems.**

***Index Terms*—Crowdsourced livecast, Edge computing, Personalized QoE, Deep reinforcement learning.**

I. Introduction

Interactive crowdsourced livecast (or *crowdcast*) has become increasingly popular and seen great success in the past few years. In a crowdcast service, numerous broadcasters can stream their own contents to the viewers in their channels through such crowdcast platforms as Twitch.tv[1], Youtube Gaming[2], and Inke.tv[3], to name a few. And the massive amount of viewers can both watch the live video contents and interact with the broadcaster and other viewers within the same channel. The crowdcast market is estimated to grow from 30.29 billion in 2016 to more than 70 billion by 2021, and it is reported that 36% of the global Internet users had watched live video as of November 2016 [1]; in Facebook, people spend 3x longer time watching live streaming compared to content that is no longer live [2].

In a livecast service, the source streaming will first need to be transcoded to different bitrates and then be delivered to viewers. Compared to traditional livecast services where professional video producers (e.g., TV channels and Netflix) broadcast well-planned content to viewers at specified time periods, there exist three unique features in a crowdcast service. First, crowdcast platforms are having an increasingly large scale due to the myriad broadcasters and viewers. For example, the monthly unique broadcaster number reached more than 2 million and the daily unique viewers were more than 15 million in Twitch.tv in 2017 [3]. The myriad video contents generated at the broadcaster side require massive resource for video transcoding and delivery, which brings heavy burdens and cost to crowdcast platforms. Second, the content preferences on the viewer side are highly diversified and heterogeneous. Most viewers can be attracted by a very small proportion of broadcasters and the total viewer number at peak time and off-peak time can vary by orders of magnitude [4]. Such a highly skewed viewing patterns and dynamic resource demands make effective viewer and resource management very complicated. Third, given the rich interactions between viewers and broadcasters as well as the diversified watching environments/preferences, viewers have personalized quality of experience (QoE) demands (such as various preferences for streaming delays, channel switching latencies and bitrates), which, if not carefully handled, can be hard to satisfy. For example, the communicative viewers that enjoy interacting with broadcasters can be more sensitive to the streaming latency, while those channel skimmers who only browse each channel for a while usually prefer a low channel switching latency, even sacrificing the bitrate and the absolute streaming latency.

Fangxin Wang, Jiangchuan Liu and Yifei Zhu are with School of Computing Science, Simon Fraser University, Canada. E-mail: {fangxinw, yza323, jcliu}@sfu.ca.
Cong Zhang is with School of Computer Science, University of Science and Technology, China. Email: congz@ustc.edu.cn.
Feng Wang is with Department of Computer and Information Science, The University of Mississippi, USA. E-mail: fwang@cs.olemiss.edu.
Haitian Pang is with Department of Computer Science and Technology, Tsinghua University, China. E-mail: pht14@mails.tsinghua.edu.cn.
Lifeng Sun is with Department of Computer Science and Technology, Tsinghua University, China. E-mail: sunlf@tsinghua.edu.cn.
A preliminary version of this work appeared in IEEE INFOCOM 2019.
Corresponding author: Jiangchuan Liu.

[1]https://www.twitch.tv/

[2]https://gaming.youtube.com/

[3]http://www.inke.cn/

The unique features in crowdcast impose an unprecedented key challenge on how to flexibly and cost-effectively accommodate the heterogeneous and personalized QoE demands for a large crowd of viewers. Traditional approaches for livecast services rely on cloud-CDN architectures for streaming transcoding and delivery. Yet such dedicated architectures cannot well satisfy viewers' diversified personalized QoE demands since all the QoE metrics are usually optimized in a monolithic approach. As a new emerging paradigm, *edge computing* [5], brings more flexibility to livecast services, and has been used to optimize 4K live streaming [6], reduce the first mile delivery latency [7], accommodate flash crowds for popular contents [8], etc. These existing edge-based works, however, are still not able to fully address the unique challenge in effectively accommodating viewers' diverse QoE demands.

In this paper, we propose DeepCast [9], an intelligent edge-assisted crowdcast framework that accommodates personalized QoE with minimized system cost. In DeepCast, distributed edge servers fetch high bitrate from CDN servers and then downsample (or transcode) to multiple bitrates as requested by viewers. Edge servers work collaboratively to assign viewer loads to proper servers based on viewers' personalized QoE demands and the instantaneous system resource distribution.

It is extremely complex to achieve the optimal viewer assignment and transcoding selection due to the numerous video contents, diverse QoE demands and uncertain online watching behaviors. Yet, the rich interactions among the broadcasters, viewers, and the network system also offer invaluable data that could be utilized towards informed management. We argue that the recent advances in DRL apply well in this context. We propose a data-driven DRL-based approach that can automatically learn from the network and viewer information to make intelligent decisions without any predefined rules. Specifically, the edge system maintains a learning agent that gradually learns to optimize the viewer assignment and transcoding policy based on the resulting performance of the past assignment. DeepCast uses state-of-the-art *asynchronous advantage actor-critic* network model (A3C) [10] to train the deep neural network, which extracts the edge resource distribution and the viewer request as a state and selects an optimal action through the network. To our best knowledge, DeepCast is the first edge-assisted framework that applies DRL for personalized QoE optimization in crowdcast services.

We collect three real-world datasets, including a livecast viewing trace and real edge trace in a major city of China as well as a public viewer bandwidth trace in US, to evaluate the performance of DeepCast. The results demonstrate that DeepCast can effectively improve the average personalized QoE by 54.6% than the cloud-CDN approach and by 50.9% than the state-of-the-art edge-based approach. DeepCast also reduces the system cost by 36% and 16.7%, respectively.

The rest of this paper is organized as follows. Section II introduces the background and motivation of our work with data-driven analysis. Section III describes our edge-assisted crowdcast framework followed by a problem formulation. Section IV describes deep reinforcement learning and related applications, and introduces the design of our DRL-based approach in detail. Section V evaluates the performance of DeepCast, as compared with state-of-the-art approaches through trace-driven experiments. We further discuss our work in section VI and conclude it in section VII.

II. Background and Motivation

A. Crowdsourced Livecast

Crowdsourced livecast (or *crowdcast*) has become increasingly popular in recent years in both industry and academia. Many previous efforts have been made to improve the QoE and reduce the cost in crowdcast services. Wang et al. [11] considered the video transcoding and viewer delivery in a cloud-CDN architecture. They separated the viewer assignment and content transcoding and solved each with a heuristic algorithm. Yan et al. [8] proposed a transparent network service called LiveJack to seamlessly integrate edge clouds for live broadcast with a focus on system implementation rather than optimizing QoE. Pang et al. [7] used edge servers as relays to reduce the loss rate and latency of the first mile video transmission in crowdcast. Ge et al. [6] proposed an edge-based system to achieve 4K live streaming across the global Internet. These pioneer works either focused on reducing the cache cost or improving a dedicated QoE target. We however argue that different crowdcast viewers can have quite different QoE preferences. A dedicated viewer serving strategy may discriminate the personalized QoEs for certain viewers and increase the service cost for crowdcast platforms. We next use a trace-driven data analysis to better understand the diversity of personalized QoE demands in a crowdcast service.

B. Data Driven Analysis

We first collected a dataset of users' watching records from Inke.tv (one of the largest crowdcast platforms in China) for 11 days in 2016, with about 7.3 million viewing sessions on each day. Each record contains a viewer ID, channel ID, network type, location, start time and end time. We extract the viewing information in the Beijing area (one of the most active areas) as a case study. To investigate how active viewers behave in different channels, we also collected the viewer interaction information of 300 popular channels of Twitch.tv (the leading crowdcast platforms in US) for two months using the Twitch API [12]. This data trace includes the viewers' arriving and leaving logs, chatting messages, broadcasters' channel content and so on.

We start from analyzing the viewing information from the Inke.tv dataset. Fig. 1 analyzes the accessed network type of different viewers and presents the number of concurrent viewers through different access approaches for all the channels in a typical day. We find that about 10% of viewers use 3G/4G cellular networks for the crowdcast service. Considering the expensive pricing rate for cellular data and the huge traffic amount for videos, such viewers may not have a strong favor in high bitrate. Instead, they sometimes are willing to sacrifice the bitrate for a relatively lower streaming delay but smooth watching experience. Besides, the number of concurrent viewers in the evening peak time is 4 to 5 times higher than others, which also brings high burden to

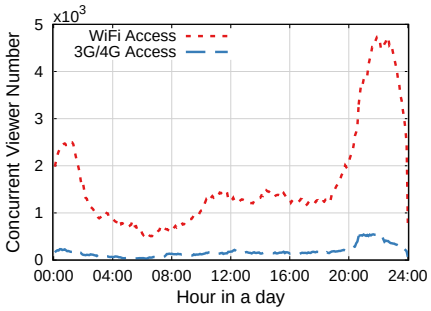


Fig. 1. The number of concurrent viewers in different time periods of a typical day.

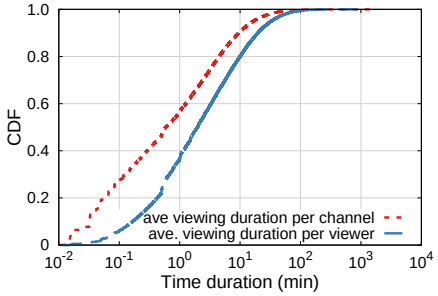


Fig. 2. The CDF plot of average viewing duration for every channel and average viewing duration for every viewer.

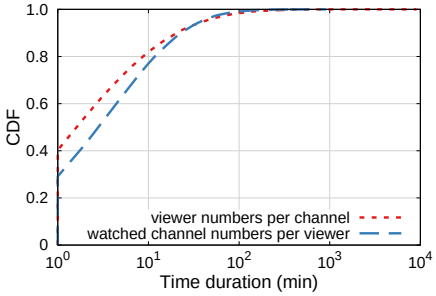


Fig. 3. The CDF plot of viewer numbers in different live channels and watched channel numbers for every viewer during a day.

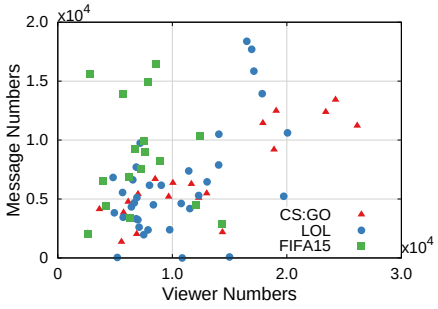


Fig. 4. The distribution of viewer numbers and interaction message numbers of channels in three games.

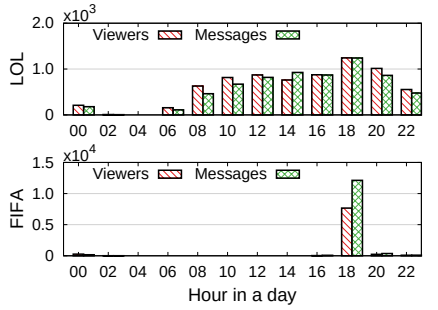


Fig. 5. The average viewing and interaction situations of different time periods in a day for channels of two games.

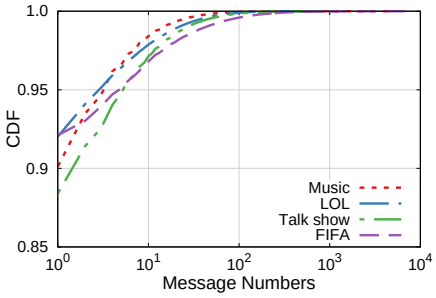


Fig. 6. The CDF plot of interactive message numbers of each session in different channels.

the platform's bandwidth supply if all viewers were served with very high bitrates.

We also investigate the average viewing time duration for each channel and viewer, as illustrated in Fig. 2. We observe that there are about 35% of viewers watching a channel for less than one minute. For these viewers, the most obvious impact on viewing experience is the channel switching latency since they browse many channels frequently. They tend to care more about the channel switching latency rather than the absolute streaming delay and bitrate. As a comparison, there are about 15% of viewers watching a channel for more than one hour. These viewers, being loyal to their favorite channels, have a much higher tolerance for the channel switching latency.

Fig. 3 plots the CDF plot of the number of total viewers for every channel and the total watched channels for every viewer. The viewer distribution follows a typical long-tailed distribution, where only 2% channels have more than 100 viewers. These channels however attract the majority of the viewers, indicating a strongly unbalanced viewer distribution. Besides, there are more than 20% viewers watching more than 10 channels every day, suggesting a highly heterogeneous channel viewing preference for different viewers.

We next study the viewer interactions from the Twitch dataset. We select three games for comparison, i.e., CS:GO, League of Legends (LOL), and FIFA, which are all popular games of different types. Fig. 4 compares the daily average viewer numbers and interaction message numbers of channels for these three games. We define IA-ratio as $\frac{\#\ of\ ave.\ interactions}{\#\ of\ ave.\ viewers}$ and we can observe that IA-ratio of FIFA is much larger than that of CS:GO. This indicates viewers of FIFA channels are more active and interact more frequently.

We select two typical channels from LOL and FIFA and plot the average viewing and interaction situations of different time periods in a day as illustrated in Fig. 5. It clearly shows that the viewer and interaction numbers are relatively stable for the LOL channel, which become higher at evening peak time and much lower at midnight. In contrast, the FIFA channel gathers a large number of viewers with animated interaction in the prime time; yet there are almost no viewers in other time. These results show that people's watching preferences vary with different time periods for different channels.

We randomly select four popular channels and extract the interaction message numbers of each session (a session means the watching process from a viewer coming to the viewer leaving). As shown in Fig. 6, the number of interaction messages distribution is highly skewed, with 87% sessions never generating messages and more than 5% sessions having frequent messages (more than 10 messages per viewing). Those communicative viewers with frequent interactions would prefer low streaming delay since highly delayed messages hurt the interaction between the viewer and the broadcaster, further undermining the QoE of all the parties. In contrast, those silent viewers who barely involve in any interactions are not so sensitive to streaming delay, as long as the video and messages are synchronized [13].

The above data analysis indicates that different viewers can have heterogeneous and personalized demands for various QoE metrics, such as streaming delay, channel switching latency and bitrate, which however are not systematically con-

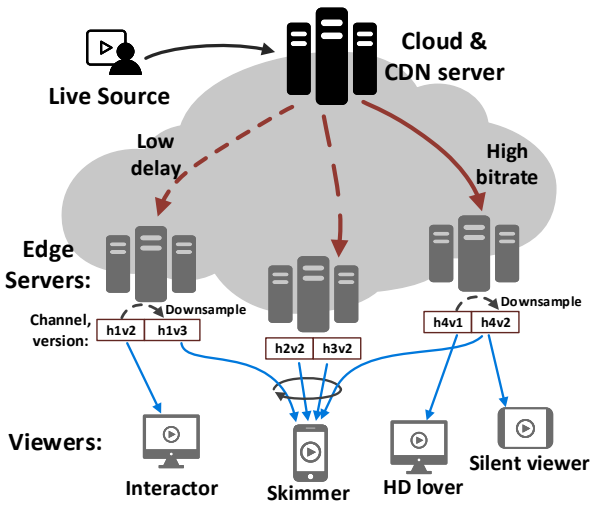


Fig. 7. The edge-assisted crowdcast framework. Edge servers can get channel content of different bitrates from the CDN server and can downsample from high bitrate to low bitrate to serve viewers with various bandwidth situations. Viewers can be served by different servers (edges or the CDN) to optimize their personalized QoE demands.

sidered and satisfied in existing livecast solutions. We therefore propose an intelligent edge-assisted crowdcast framework to address this problem, as discussed in the next section.

III. Edge-Assisted Crowdcast

A. Framework

Fig. 7 illustrates our edge-assisted crowdcast framework. A broadcaster in a channel first builds up a connection with the platform's service center (e.g., usually the cloud) and transmits the raw streaming. The original streaming is then encoded and compressed into streams with multiple bitrates, which are pushed to CDN servers. Given the tight latency demand in such interactive applications as personal livecast and AR/VR streaming, the service providers usually use WebRTC or their proprietary protocol for multimedia streaming [14].

In a citywide area, edge servers are distributed much closer to the viewers and each edge server will serve crowdcast viewing requests within its proximity. The CDN usually only needs to deliver the channel content of high quality versions to the edge servers through HTTP. Then edge servers can *transcode* (or downsample) the high quality versions to low quality versions to serve viewers with different bitrate requests. A viewer may request a channel content with a specific version based on his/her own bandwidth condition. Given viewers' personalized QoE demands and the available resources, the regional edge can serve the viewer itself or redirect the request to another edge (or the CDN), so as to optimize the viewers' personalized QoE and minimize the system cost. For example in Fig. 7, the requests of a channel skimmer (a viewer who quickly browses many channels) is redirected to the nearest available edge server rather than the CDN, so as to achieve a low channel switching latency.

B. QoE and Cost Aware Optimization

To better understand the challenges of implementing the online viewer assignment and transcoding selection, we start from analyzing a simpler offline batch arrival scenario, where the viewer requests and resource situations are known in advance. We assume that in a city-level region the crowdcast service provider has one CDN server c and a set of edge servers, $\mathbf{E} = \{1, 2, ..., E\}$, at fixed locations. All of them are connected via the backhaul network. The CDN is capable of having all the streaming channels $\mathbf{H} = \{1, 2, ..., H\}$ and different versions $\mathbf{V} = \{1, 2, ..., V\}$, where a channel of a particular version is denoted as (h, v). We assume that U viewers, $\mathbf{U} = \{1, 2, ..., U\}$, have viewing requests for different channels and versions. Based on each viewer's watching preference and individual bandwidth condition, the target version of viewer u is denoted as $\phi(u) = v'$. We use a binary variable X to denote the viewer assignment, where $X^{(u,j)}_{(h,v)} = 1$ ($j \in \{\mathbf{E} \cup c\}$) (resp. 0) indicates user u is (resp. is not) assigned to j for (h, v). And variable $Y^{(e)}_{(h,v)} = 1$ (resp. 0) denotes that (h, v) is (resp. is not) at edge e.

We consider three QoE metrics, i.e., *streaming delay*, *channel switching latency* and *bitrate mismatch level*, where the bitrate mismatch level is defined as a function of the difference between the target version of a viewer and the actual assigned version. Then we can calculate the streaming delay $\mathcal{D}^{(u)}$ of viewer u as follows:

$$\mathcal{D}^{(u)} = \sum_{h,v} \sum_{e} X^{(u,e)}_{(h,v)} \left(l^{(u,e)} + T^{(e)}_{R(h,v^*,v)} + l^{(e,c)} \right) + \sum_{h,v} X^{(u,c)}_{(h,v)} l^{(u,c)} \tag{1}$$

where $l^{(u,e)}$ (or $l^{(u,c)}$) indicates the latency between user u and edge e (or the CDN c). $T^{(e)}_{R(h,v^*,v)}$ denotes the latency introduced by transcoding or downsampling from a high bitrate version to a lower version. For each edge e, it can already have the resource for a channel since other viewers may also connect to this server for the same channel. We represent the highest version existing at the edge as v^*. If v^* is higher than v, the server only needs to transcode from the higher version v^* to the target version v. There is no need to transcode if $v^* = v$ (i.e., $T^{(e)}_{R(h,v^*,v)} = 0$) because the edge receives this version from the CDN server. The first part of the equation represents the latency if the viewer is assigned to an edge server, while the second part means the viewer is directly assigned to the CDN server. Similarly, the channel switching latency $\mathcal{L}^{(u)}$ can be calculated as:

$$\mathcal{L}^{(u)} = \sum_{h,v} \sum_{j \in \{\mathbf{E} \cup c\}} X^{(u,j)}_{(h,v)} l^{(u,j)} \tag{2}$$

In practice, the actual assigned version of a channel for a viewer can be different from the target version. For example, the crowdcast system may assign a lower version channel content to a viewer if no enough system resource, or to satisfy the viewer's other preference, such as low streaming delay.

There is then a bitrate mismatch that affects the QoE. We can calculate the mismatch level $\mathcal{B}^{(u)}$ as follows:

$$\mathcal{B}^{(u)} = \sum_{h,v} \sum_{j \in \{\mathbf{E} \cup c\}} X_{(h,v)}^{(u,j)} M(\phi(u), v) \tag{3}$$

where $M(\phi(u), v)$ is a predefined bitrate mismatch utility function. Existing research mostly has suggested three representative penalty functions, e.g., linear function [15], logarithmic function [16] and HD-preferred function [17]. The linear function directly uses the linear difference between the two versions as the mismatch, while the logarithmic function uses the logarithm of the ratio of these two values to decrease a marginal quality improvement. In a HD-preferred function, a tabular mapping is used for different bitrates and higher bitrates are assigned with much higher utility. If $\phi(u)$ equals v, the provided bitrate version for viewer u is exactly the same as the requested one, and so, there is no bitrate mismatch.

Besides the personalized QoE, we also consider the cost for system resource usage, i.e., the *computation cost* and the *bandwidth cost*. The total computation resource cost (for the transcoding at edges) $\mathcal{C}_T$ can be calculated as follows:

$$\mathcal{C}_T = \sum_{e} \sum_{(h,v) \neq (h,v^*)_e} Y_{(h,v)}^{(e)} I_{T(h,v^*,v)} \cdot P_T^{(e)} \tag{4}$$

where $(h, v^*)_e$ indicates the highest version of channel h in edge e, $I_{T(h,v^*,v)}$ is the computation resource consumption for transcoding from (h, v^*) to (h, v), and $P_T^{(e)}$ is the unit computation resource price at edge e. Similarly, we can also calculate the total bandwidth cost $\mathcal{C}_B$ as:

$$\begin{aligned} \mathcal{C}_B = &\sum_{e} \sum_{u} \sum_{h,v} X_{(h,v)}^{(u,e)} I_{B(h,v)} P_B^{(e)} \\ &+ \sum_{h,v} \sum_{u} X_{(h,v)}^{(u,c)} I_{B(h,v)} P_B^{(c)} + \sum_{e} \sum_{h,v^*} Y_{(h,v^*)}^{(e)} I_{B(h,v^*)} P_B^{(c)} \end{aligned} \tag{5}$$

where $I_{B(h,v)}$ is the resource consumption for serving (h, v), $P_B^{(e)}$ and $P_B^{(c)}$ is the unit bandwidth price for different edge servers and the CDN server, respectively. The three parts in (5) indicate the bandwidth cost from viewers to edges, viewers to the CDN, and edges to the CDN, respectively.

Integrating the viewers' personalized QoE demands (Eq. 1, Eq. 2 and Eq. 3) and the system cost (Eq. 4 and Eq. 5) together, we have the following optimization objective (Ω) that minimizes the sum of overall *penalty*, include QoE (i.e., the weighted sum of the streaming delay, channel switching latency and bitrate mismatch) and system cost (i.e., the weighted sum of the computation cost and the bandwidth cost):

$$Min: \alpha \sum_{u} \left(\alpha_1^{(u)} \mathcal{D}^{(u)} + \alpha_2^{(u)} \mathcal{L}^{(u)} + \alpha_3^{(u)} \mathcal{B}^{(u)} \right) + \beta \left(\mathcal{C}_T + \mathcal{C}_B \right) \tag{6}$$

$s.t.$

$$\sum_{h,v} \sum_{j \in \{\mathbf{E} \cup c\}} X_{(h,v)}^{(u,j)} = 1, \forall u \tag{7}$$

$$X_{(h,v)}^{(u,j)} \leq Y_{(h,v)}^{(j)}, \forall (h,v), j \in \{\mathbf{E} \cup c\} \tag{8}$$

$$\sum_{h,v} Y_{(h,v)}^{(e)} I_{T(h,h^*,v)} \leq W_T^{(e)}, \forall e \tag{9}$$

$$\sum_{h,v} \sum_{u} X_{(h,v)}^{(u,e)} I_{B(h,v)} \leq W_B^{(e)}, \forall e \tag{10}$$

where $W_T^{(e)}$ and $W_B^{(e)}$ are the computation and bandwidth capacity of edge e, α and β are the weighted parameters to tune the QoE penalty and system cost penalty, and $\alpha_1^{(u)}$, $\alpha_2^{(u)}$, $\alpha_3^{(u)}$ are the personalized QoE preference factors for viewer u, which can be either specified by viewers or derived from viewers' watching history. Eq. 7 guarantees that a viewer can only connect to one edge or the CDN. Eq. 8 indicates that the target server must have the corresponding channel of a suitable version; Eq. 9 and Eq. 10 ensure that the resource usage does not exceed the capacity.

C. Problem Hardness and Data Driven Approach

Theorem 1. *The problem of viewer assignment and transcoding selection such that the overall penalty can be minimized is NP-hard.*

The proof of this theorem is presented in the Appendix. Though a simplified version of the problem can be transformed into the MKP problem, which has some approximate solutions [18], it is still difficult to achieve an efficient and effective solution using traditional model-based approaches considering the following three issues. First, the original problem Ω is much more complex with more constraints (e.g., the computation capacity restriction), as described in Eq. 9. The high complexity and large solution space bring great challenges to achieve fast or even realtime decisions. Second, in a practical crowdcast system, viewers are arriving and leaving dynamically. A dedicated model-based solution cannot take into account the historical features of viewer patterns to optimize the assignment. For example, in Fig. 1, the concurrent viewer number usually escalates during night time. Moreover, different viewers can have various personalized QoE demands, as discussed in section II-B. An accurate prediction of viewers' personalized QoE demands is necessary for the subsequent assignment.

On the other hand, we notice that the abundant network and viewer pattern information, and the rich interactions between viewers and broadcasters offer invaluable data that could be utilized for a data-driven management. The data-driven approach can not only well capture the implicit viewing features from the historical patterns but also provide an integrated solution from the preliminary prediction to the final viewer assignment and transcoding selection. In particular, the recent advance of the deep reinforcement learning (DRL) has demonstrated great potentials in many fields and fits our context well. To this end, we turn to the design of a DRL-based model to solve the problem, as described in the next

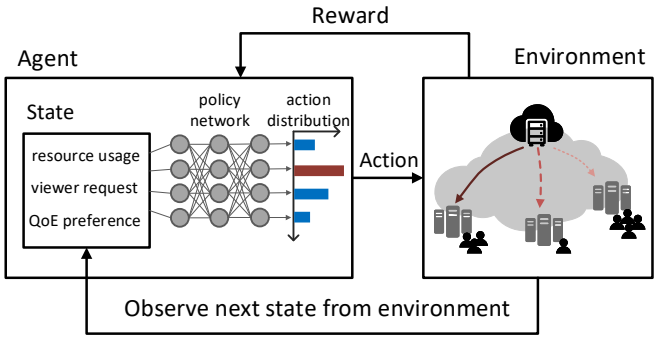


Fig. 8. The workflow of using deep reinforcement learning for crowdcast viewer assignment.

section. It is worth noting that different from the offline batch case described in this section, our DRL-based approach is designed to well handle the online crowdcast scenario with dynamic viewer arriving and leaving.

IV. Deep Reinforcement Learning Model Design

In this section, we begin with a background introduction of deep reinforcement learning and its applications in the networking field. We next introduce the basic learning mechanism of applying deep reinforcement learning (DRL) into the assignment problem. We at last describe how we transform the online viewer assignment and transcoding selection problem into a learning task and design DeepCast, an intelligent DRL-based edge-assisted crowdcast framework, to obtain an effective solution.

A. Deep Reinforcement Learning

In recent years, deep reinforcement learning (DRL) has shown great potentials in many fields, such as computer games, chess and many real-world applications. Minh et al. [19] proposed to use Deep Q-Network (DQN) to learn policies from sensor input for decision making. Experience replay and target network were introduced to improve the stability and the performance. Double DQN [20] was next proposed to reduce the observed overestimations. Wang et al. [21] proposed a dueling network to represent two separate estimators: one for the state value function and one for the state-dependent action advantage function. This factoring generalizes learning across actions without imposing any change to the underlying algorithm. Lillicrap et al. [22] presented deep deterministic policy gradient model (DDPG), an actor-critic and model-free algorithm based on the deterministic policy gradient that can operate over continuous action spaces. Mnih et al. [10] developed A3C model, which allows parallel actor-learners to work asynchronously with fast and stable results.

DRL also sheds a light on many problems in the networking field, where viewer's quality of experience has been the focus. For example, Mao et al. [17] developed a practical rate adaptation framework, Pensieve, which used A3C to select the optimal bitrate for future video chunks. Huang et al. [23] considered the trade-off between the sending bitrate and the video quality and developed a DRL-based algorithm to offer a higher

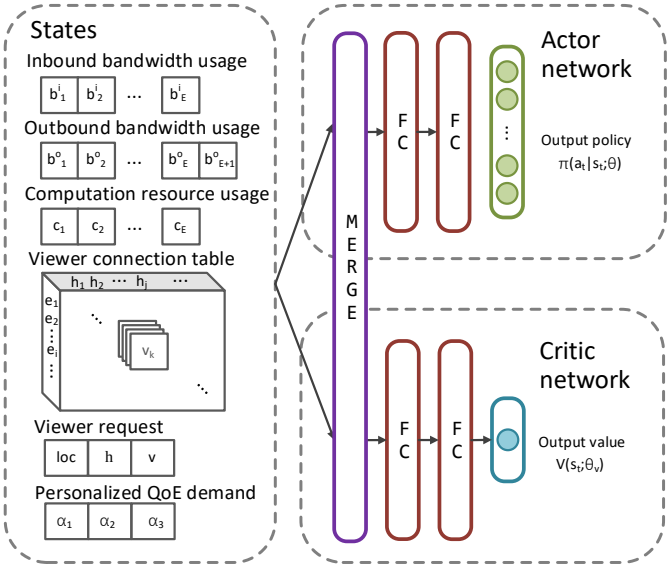


Fig. 9. The DeepCast framework that uses actor-critic model for policy selection.

perceptual video quality with possibly lower sending rate and transmission latency. Different from these aforementioned works that focused on the client-side optimization for better QoE, we consider the viewer assignment and transcoding selection problem from the service providers' perspective in crowdcast services. To our best knowledge, DeepCast is the first to provide a novel scheduling framework that considers viewers personalized QoE in the edge-assisted crowdcast scenario and applies DRL to cost-effectively accommodate personalized QoE demands.

B. Basic Learning Mechanism

Unlike existing edge resource allocation approaches using predefined rules or model-based heuristics, DeepCast strives to learn a general action decision from the past experience based on the current state and the given reward. In the workflow of DeepCast, see Fig. 8, a learning *agent* interacts with the *environment* in the DRL setting, where the agent is the main component of assignment decision, and the environment defines the rules, restrictions and reward mechanism. In each time step t, the agent observes a state s_t and can choose an action a_t. When this action is done, the current state will transit to the next state s_{t+1} and the agent will receive a reward r_t. It will get accumulated rewards after every action until done. The objective of DRL is to find a best policy π mapping a state to an action that maximizes the expected discounted accumulated reward as $\mathbf{E}[\sum_{t=t_0}^{\infty} \gamma^t r_t]$, where t_0 is the current time and $\gamma \in (0, 1]$ is a factor to discount the future rewards.

C. Model Design

As shown in Fig. 9, DeepCast uses a state-of-the-art actor-critic-based DRL model A3C [10]. We introduce the detailed functionality design as follows.

State space. We consider a practical online scenario where the viewers come and leave dynamically. Recall the problem

formulation in §III-B, the state space in the DRL formulation consists of three components, including *resource usage*, *viewer assignment information*, and *current viewer request*. We use two vectors $\mathbf{b^i} = \{b_1^i, b_2^i, ..., b_E^i\}$ and $\mathbf{b^o} = \{b_1^o, b_2^o, ..., b_{E+1}^o\}$ to record the inbound bandwidth and outbound bandwidth of the edges and the CDN, respectively. Note that the vector length of $\mathbf{b^o}$ is $E+1$ because viewers can also connect to the CDN server. We use a vector $\mathbf{c} = \{c_1, c_2, ..., c_E\}$ to record the computation resource usage at each edge. In practical situations, the computation resource of each edge can be nonuniform. The computation resource usage is quantified in units of the occupation of vCPU core numbers; the computation cost can be thereby calculated.

In general, a viewer's request should first be processed by the regional edge. Based on the optimization policy, the edge will decide to serve the viewer itself or redirect it to other edges or the CDN. We use a viewer connection table $\mathbf{tab} = (e, h, v)$ to record the viewer assignment information, indicating how many viewers are served by edge e watching channel h of version v.

Besides the current edge system information, the current viewer request is also included in the state. The viewer request consists of the viewer location loc, requested channel h of version v based on her/his bandwidth condition, and the personalized QoE preference. In practice, a viewer's personalized QoE preference can be either directly set by each viewer (who is familiar with the own preference), or learned through analyzing each viewer's watching history (e.g., interaction frequency, channel switching frequency, average watching duration, etc.) by the service platform (who has the complete watching history).[4] Integrating all these components together, the state input can be represented as $s_t = \{\mathbf{b^i}, \mathbf{b^o}, \mathbf{c}, \mathbf{tab}, loc, h, v, \alpha_1^{(u)}, \alpha_2^{(u)}, \alpha_3^{(u)}\}$.

Policy. When receiving a state s_t, the learning agent of DeepCast needs to take an action a_t for viewer assignment. The action space can be represented as $\{1, 2, ..., E, c\}$, where $a_t = j$ means assigning the current viewer request to server j (an edge or the CDN). Given the continuous values of the edge resource usage, there are infinite $\{state, action\}$ pairs which cannot be stored in a tabular form and then solved using traditional methods such as Q-learning and SARSA [24]. To address this issue, a neural network [25] can be used to represent the policy π, where the adjustable parameters of the neural network are referred to as the policy parameters θ. We then represent our policy as $\pi(a_t|s_t;\theta) \rightarrow [0, 1]$, indicating the probability of taking the action a_t at current state s_t.

Once a viewer is assigned to an edge, the edge selects the requested channel h and version v that leads to the maximal reward to serve the viewer. If the edge does not have (h, v), it will transcode to version v from available high versions or directly request it from the CDN (when no available higher version). Note that the computation and bandwidth resources at each edge are limited. If the edge server's resources are not enough, the request will be redirected to the CDN for help. In practical crowdcast scenarios, many viewers can come within a short time period. If we allow the model to assign K viewers at the same time, the action space can be $(|E|+1)^K$, where the model is very challenging especially when K is very large. To this end, we divide the time into small slots so that, in each slot, DeepCast only processes one viewer request.

Note that the total number of viewers is not fixed in our model. When a viewer arrives or leaves, the crowdcast platform occupies or releases the corresponding resource required by this viewer and the learning system also updates the state accordingly. As to the channels, since only a small portion of channels are popular, we need to only select those popular channels for edge-assisted learning-based scheduling.

Reward. When applying an action a_t to state s_t, the learning agent will get a reward r_t from the environment. Considering the optimization objective Ω in §III-B, we craft the reward to achieve the minimal overall penalty. Specifically, when we assign a viewer request to an edge (or the CDN), the viewer will have a personalized QoE and add an extra cost for the system. We define reward r_t as the opposite number of the overall penalty for the coming viewer u:

$$\begin{aligned} r_t = &-\alpha \left(\alpha_1^{(u)} \mathcal{D}^{(u)(t)} + \alpha_2^{(u)} \mathcal{L}^{(u)(t)} + \alpha_3^{(u)} \mathcal{B}^{(u)(t)} \right) \\ &- \beta \left(\mathcal{C}_T^{(u)(t)} + \mathcal{C}_B^{(u)(t)} \right) \end{aligned} \tag{11}$$

where $\mathcal{D}^{(u)(t)}$, $\mathcal{L}^{(u)(t)}$, $\mathcal{B}^{(u)(t)}$, $\mathcal{C}_T^{(u)(t)}$, $\mathcal{C}_B^{(u)(t)}$ are the streaming delay, switching latency, bitrate mismatch, transcoding cost and bandwidth cost for assigning viewer u at time t, respectively. It is worth noting that r_t here is for one individual viewer. Our learning model aims to maximize the accumulated rewards for all viewers in order to achieve the maximal sum of rewards.

A3C model. We choose to use the state-of-the-art asynchronous advantage actor-critic (A3C) model [10] as the learning model in DeepCast. The A3C model consists of an actor network (i.e., the policy network) and a critic network (i.e., the value network). The former network is trained to make the proper viewer assignment choice based on the current system state and viewer preference. Its output is the probability distribution of assigning the current viewer to different edge servers. For the inference stage, the selection (i.e., action) will be the one with the largest probability. The latter network is trained to learn an estimate of the expected total reward from the empirically observed reward. Its output is the expected reward of the current state. Note that the critic network only involves in the training stage. In the inference stage, only the actor network is required to make viewer assignment. We choose the A3C model rather than RL-based models such as DQN [19] and REINFORCE [26] due to its successful application in many related problems [27], [28]. Besides, its asynchronous architecture supports multiple learning agents to train in parallel, which speeds up the training significantly [10].

Training method. As illustrated in Fig. 9, DeepCast maintains a policy $\pi(a_t|s_t;\theta)$ (the actor network) and an estimate of the value function $V(s_t;\theta_v)$ (the critic network), where θ is the policy parameter and θ_v is the value function parameter. The actor network and the critic network share the previous part

[4] A simple example of possible prediction methodologies will be discussed in §V-A.

of network architectures except for the last output layer. In our model, the learning agent continues to take actions for the coming viewer requests. The model updates both the policy and the value function based on the returns of every t_{max} actions or until done.

The training process of DeepCast adopts a policy gradient method [26]. The key idea is to estimate the parameter gradient direction towards the maximized total reward. The gradient of the accumulated discounted reward regarding the parameter θ can be represented as:

$$\nabla_\theta E_{\pi_\theta}[\sum_{t=0}^{\infty}\gamma^t r_t] = E_{\pi_\theta}[\nabla_\theta log\pi(a_t|s_t;\theta)A^{\pi_\theta}(s_t,a_t)] \quad (12)$$

where $A^{\pi_\theta}(s_t,a_t)$ is an estimate of the advantage function. The empirically computed advantage $A(s_t,a_t)$ can be calculated as:

$$A(s_t,a_t) = Q(s_t,a_t) - V(s_t;\theta_v) \quad (13)$$

where $V(s_t;\theta_v)$ is the output of the critic network. The Q value, $Q(s_t,a_t)$, reflects the effect of applying a_t to state s_t:

$$Q(s_t,a_t) = r_t + \gamma V(s_{t+1};\theta_v) \quad (14)$$

With the policy gradient above, we are able to update the actor network parameter θ. As stated in [10], however, an entropy of policy π to the objective function is necessary to discourage the convergence to a suboptimal policy. We therefore add an entropy form and update θ as:

$$\theta = \theta + \eta\sum_t \nabla_\theta log\pi(a_t|s_t;\theta)A(s_t,a_t) + \delta\nabla_\theta H(\pi(s_t;\theta)) \quad (15)$$

where $H(\cdot)$ is the entropy of the policy, η is the learning rate for the actor network, and δ is also a hyper-parameter.

We train the critic network following the temporal difference method [24] and update the critic network as

$$\theta_v = \theta_v - \eta'\sum_t \nabla_{\theta_v}[r_t + \gamma V(s_{t+1};\theta_v) - V(s_t;\theta_v)]^2 \quad (16)$$

where η' is the learning rate for the critic network. Once the actor-critic network is trained, we select the viewer assignment action based on the output of the actor network.

V. Performance Evaluation

In this section, we compare DeepCast with state-of-the-art approaches and evaluate their performance with real trace-driven experiments.

A. Experiment Setup

Data traces. To better evaluate DeepCast and other approaches, we adopt three real-world data traces and use them together to reconstruct a practical evaluation environment:

- **Trace for viewing information.** We collect the crowdcast viewing information from the network trace of Inke.tv for 11 days in December 2016, which has about 7.3 million viewing sessions every day, including viewer ID, channel ID, start time and end time.
- **Trace for location information.** We collect the viewer location and edge location from a dataset of iQiYi for two weeks in 2015, which includes about 1.8 million viewer locations and 1 million access point locations in the city of Beijing. We uniformly sample a part of the access point locations as the edge locations.
- **Trace for bandwidth information.** We collect the bandwidth situation from a broadband dataset from FCC [29]. We extract the average bandwidth in this dataset as the viewer bandwidth situation in our experiment.

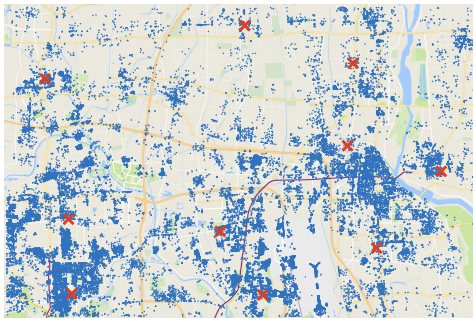


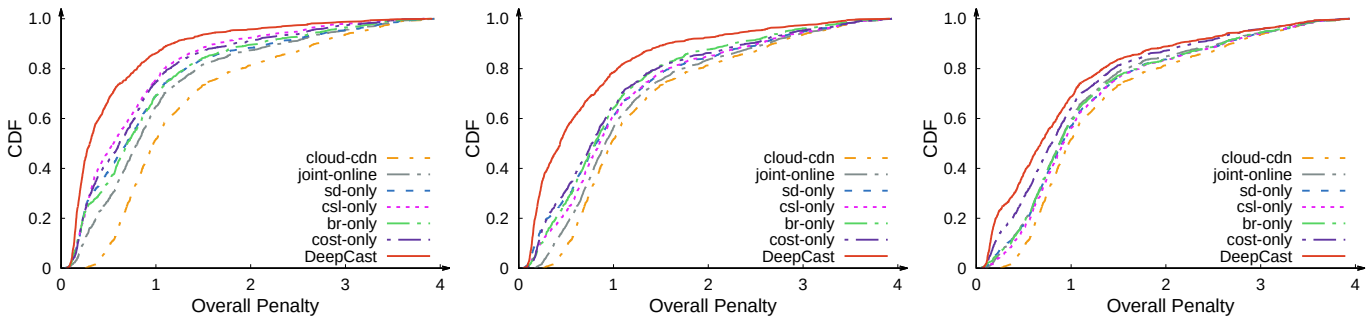
Fig. 10. A sample rectangular area of viewers and edge servers. The blue dots represent the locations of viewers and red crosses indicate the locations of edge servers.

Edge-assisted crowdcast measurement and setup. We use the integrated trace for the training and evaluation of DeepCast. Unless otherwise specified, the default parameters are set as follows: We select a rectangular area in Beijing from our location dataset as the target region (35 km $\times$ 21 km), as shown in Fig. 10. The average session request number is 45 thousand every day and we select the top 50 channels with the most viewers for evaluation. We evenly sample 10 access points as the edge server locations.

We set the inbound and outbound bandwidth of each edge server as 200 and 400 Mbps, respectively. The vCPU core number is 36 based on Amazon AWS c4.8xlarge instance [30]. The CDN server can relay all the channels of different versions and is capable of serving all the viewers. We measure the bitrates and list the transcoding overhead of a 1-second stream from the highest bitrate to lower bitrate versions in Tab. I. We set the latency between edges and CDN randomly from 20 ms to 100 ms considering the relatively good network condition of edge servers. The latency between CDN and viewers is randomly set from 100 ms to 700 ms, and the latency between viewers and edges is proportional to their geo-distance [31], with a maximum of 100 ms.

Training setup. We implement the DeepCast learning model using tensorflow [32] and run the experiment on a desktop with dual GTX 1080 Ti GPU cards, dual Intel I7 3.6 GHz CPU cards and 32GB memory. The default parameters in the actor-critic training phase is set as $\gamma = 0.99$, $\eta = 5e^{-4}$, $\eta' = 1e^{-3}$, and the default neuron numbers in the hidden layer is 4096 plus 2048. We train the network using the first 80% data of the viewer watching information. Based on the settings above, training the actor-critic network needs 4 hours to achieve a stable result.

Personalized QoE metrics. We set the bitrate mismatch function as $M(R^*,R) = log(R^*/R)$ (presented in Eq. 3), where R^* is the target bitrate for a viewer based on the



(a) fat-edge (b) mid-edge (c) thin-edge

Fig. 11. The CDF plot of overall penalty by different approaches under different edge settings.

TABLE I
MEASUREMENT RESULT OF BANDWIDTH, TRANSCODING RESOURCE USAGE AND TRANSCODING LATENCY.

versions (p)	1440	1080	720	480	360	240
bitrate (Mbps)	4.3	2.85	1.85	1.2	0.75	0.3
transcoding (vCPU)	NA	330%	142%	82%	51%	41%
transcoding time (s)	NA	0.27	0.19	0.16	0.13	0.11

TABLE II
VIEWER CLASSIFICATION METHOD.

Categories	Classification criteria	QoE Metrics
sd-pref	$\bar{n} \leq 2$, $\bar{t} \geq 30min$	$\alpha_1 = 2$, $\alpha_2 = 1.5$, $\alpha_3 = 2$
csl-pref	$\bar{n} \geq 5$, $\bar{t} \leq 10min$	$\alpha_1 = 0.5$, $\alpha_2 = 6$, $\alpha_3 = 2$
br-pref	$\bar{n} \geq 4$, $\bar{t} \geq 30min$	$\alpha_1 = 0.5$, $\alpha_2 = 1.5$, $\alpha_3 = 8$
normal	otherwise	$\alpha_1 = 1$, $\alpha_2 = 3$, $\alpha_3 = 4$

TABLE III
DIFFERENT SETTINGS OF EDGE CAPACITY.

Edge setting	bw_in capacity	bw_out capacity	compute capacity
fat-edge	400 Mbps	800 Mbps	64 vCPU
mid-edge	200 Mbps	400 Mbps	36 vCPU
thin-edge	100 Mbps	200 Mbps	16 vCPU

bandwidth and R is the allocated bitrate. This logarithmic representation was used in many works, e.g., BOLA [16], representing diminishing the marginal improvement of higher bitrates. With the viewing information dataset, we use a simple method to classify viewers into four QoE preference categories based on their daily average watching channel numbers ($\bar{n}$) and watching durations ($\bar{t}$). The categories include streaming delay preferred (*sd-pref*), channel switching latency preferred (*csl-pref*), bitrate preferred (*br-pref*) and others (*normal*), as illustrated in Tab. II. We refer to Amazon AWS [30] for the setting of bandwidth and transcoding cost, and set the edge bandwidth price to 20% of the CDN bandwidth price. We also adjust the system cost proportionally to balance the QoE penalty and system cost penalty when $\alpha = \beta$.

Comparison methods. We consider the following methods as the baseline for comparison with DeepCast: 1) Traditional Cloud-CDN architecture (*cdn-only*), which assigns all viewers to CDN instead of edges; 2) Edge-assisted conventional livecast architecture (*joint-online*), which is derived from a state-of-the-art approach proposed in [11] for assignment and transcoding; specifically, the viewers are first assigned to proper edges and the transcoding strategy is then selected accordingly in a heuristic way; 3) Streaming delay only (*sd-only*), which considers the video latency in the training process; 4) Startup latency only (*csl-only*), which only considers the channel switching latency in the training process; 5) Bitrate only (*br-only*), which only considers the bitrate mismatch level in the training process; 6) Cost only (*cost-only*), which only considers the system cost in the training process.

We next evaluate the performance of DeepCast and analyze the impact of different settings on the experiment results.

B. Impact of Edge Capacity

We first consider the impact of edge capacity on the viewer assignment performance. We set three different edge capacities, i.e., *fat-edge*, *mid-edge* and *thin-edge*, as presented in Tab. III. We set the default QoE penalty factor α and the system cost factor β both to 0.5. Fig. 11 shows the CDF plot of the overall penalty of each viewer assignment under different edge capacities. Fig. 12 provides the normalized average overall penalty. For ease of comparison, we set *cloud-cdn* as the baseline and the results of other methods for comparison are normalized accordingly.

We have three key observations from this experiment. First, DeepCast easily achieves much lower overall penalty than others in all the edge capacity settings. Specifically, in the mid-edge setting, DeepCast reduces an average of 45.9% overall penalty than cloud-cdn and 41.6% overall penalty than joint-online. This result indicates that DeepCast can effectively utilize the edge servers to satisfy viewers' personalized and heterogeneous QoE demands and make intelligent viewer assignment, while cloud-cdn and joint-online cannot well assign each viewer, thus incurring high overall penalties. DeepCast performs even better when there is more available edge capacity. From Fig. 12, the overall penalties of DeepCast are 0.7, 0.54 and 0.42 under the setting of thin-edge, mid-edge and fat-edge, respectively. This indicates that more edge capacity will empower DeepCast with more flexible choices, which can further lead to a lower overall penalty. DeepCast also outperforms other learning-based methods that only consider part of the QoE metrics or the cost, such as sd-only by 41.4%, csl-only by 30.5%, br-only by 40.7% and cost-only by 33.9%. This is because DeepCast comprehensively considers all the related metrics and makes proper assignment accordingly.

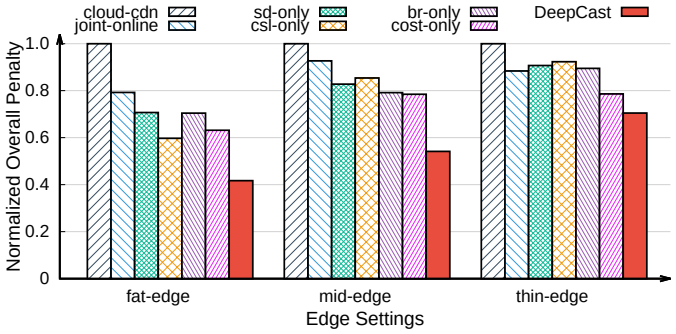


Fig. 12. The normalized overall penalty under different edge settings.

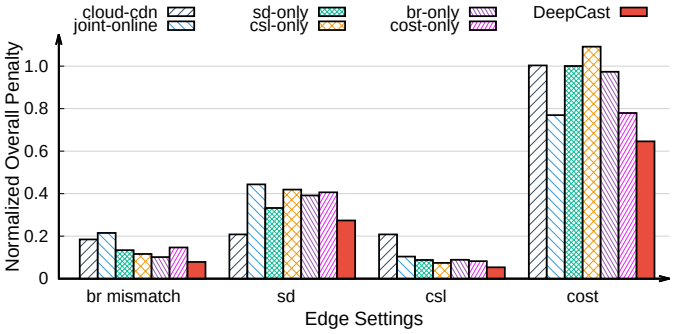
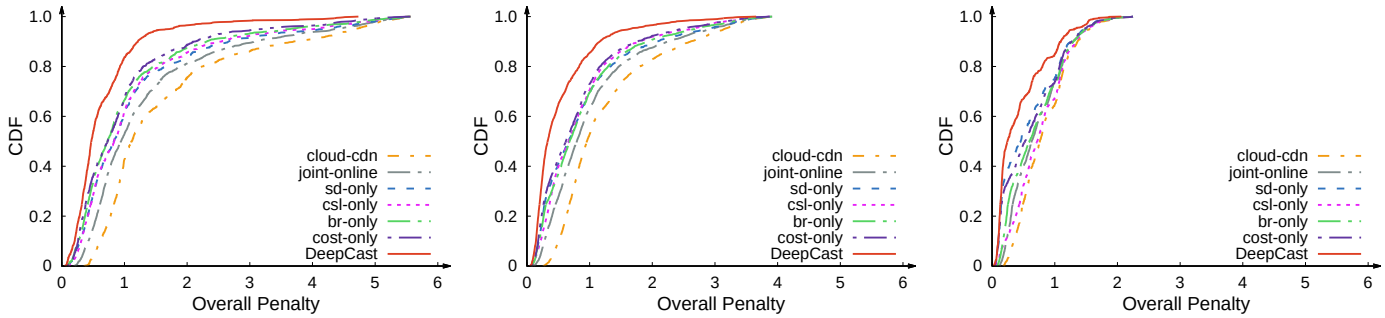


Fig. 13. The value of different QoE metrics under different edge capacity settings.



(a) $\alpha = 0.8, \beta = 0.2$ (b) $\alpha = 0.4, \beta = 0.6$ (c) $\alpha = 0.2, \beta = 0.8$

Fig. 14. The CDF plot of overall penalty by different approaches with various ratios between QoE and cost.

Fig. 13 illustrates the detailed QoE metrics and system cost under the mid-edge setting. We can see that DeepCast incurs an average of 0.07 bitrate mismatch, which is 57.6% less than cloud-cdn and 63.7% less than joint-online. This shows that DeepCast can achieve better bitrate match given the edge-assisted crowdcast architecture and the intelligent assignment. For the channel switching latency, DeepCast only needs an average of 0.05 s, reducing 75% time as compared to the traditional cloud-cdn architecture. For the streaming delay, DeepCast also reduces 38.3% time than joint-online that uses a heuristic assignment strategy. Note that the streaming delay of DeepCast is a bit higher than cloud-cdn due to the transcoding delay on edge. However, this additional 0.06 s stream delay is still well within human tolerance for interactions. Regarding the system cost, DeepCast reduces the average cost penalty by 36% than cloud-cdn and by 16.7% than joint-online.

C. Impact of QC-ratio

Besides different edge settings, different ratios between QoE penalty and system cost penalty (i.e., α/β, defined as QC-ratio) can also affect the performance of DeepCast. Note that if we set $\alpha+\beta = 1$, then a higher α value means the crowdcast platform is more aggressive, caring more about viewers' QoE experience, while a higher β means the crowdcast platform is more economical, having a high priority for saving cost. We focus on the fat-edge setting given it provides sufficient edge capacity, and similar observations can be made in other settings.

Fig. 14 shows the CDF plots of the overall penalty of different approaches when we set different QC-ratios. It is easy to find that the penalty distribution of assigning every viewer is highly skewed when the crowdcast platform cares more about QoE experience (e.g., $\alpha = 0.8$). This is because aggressively satisfying all the QoE demands will prioritize viewers' QoE demands regardless of the system resource usage at first, which easily leads to excessive and unreasonable resource usage. The imbalanced resource usage further results in the assignment of the following viewers with high overall penalties. On the other hand, if the platform cares more about cost (e.g., $\alpha = 0.2$), the penalty distribution of each assignment will be more centralized.

Fig. 15 shows the overall penalty normalized by the cloud-cdn baseline. We can observe that DeepCast can achieve more intelligent viewer assignment from a holistic perspective over all the settings, reducing the overall penalty by 30.8%, 41%, 23.1% than cloud-cdn when we set α as 0.8, 0.4 and 0.2, respectively.

D. Comparison on Different Learning Model

In our design, DeepCast uses the asynchronous advantage actor-critic (A3C) [10] as the learning model. Deep reinforcement learning has many forms of learning models that can apply well in our context. A well-known candidate is deep Q-learning network (DQN) which has breakthrough contribution in achieving human-level control over many computer games [19]. To better understand the compact of learning models, we compared our proposed model to DQN, which further can be divided into 1-step-DQN and n-step-DQN according to the different training settings. The difference

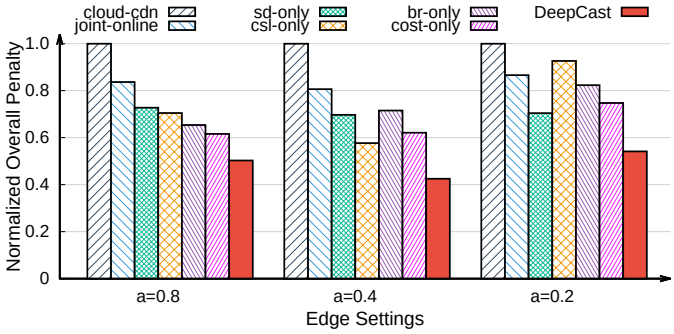


Fig. 15. The normalized average overall penalty with different ratios between QoE and cost.

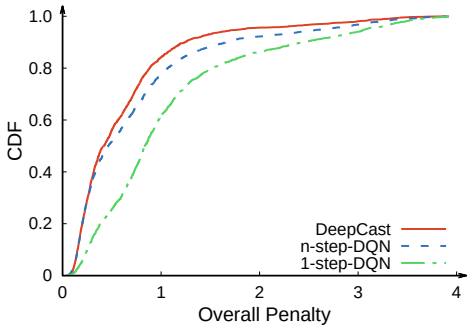
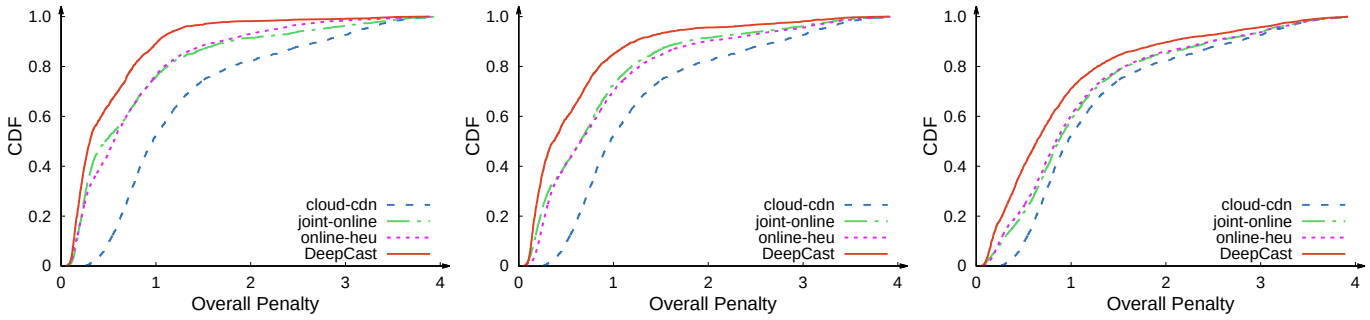


Fig. 16. The CDF plot of overall penalty comparing DeepCast with n-step DQN and 1-step DQN.



(a) fat-edge (b) mid-edge (c) thin-edge

Fig. 17. The CDF plot of overall penalty by different approaches under different edge settings. We train our model with synthetic data trace and test in with real trace to show its generalization ability.

between 1-step-DQN and n-step-DQN is that the latter one is updated towards every n-step with an accumulated reward.

Fig. 16 plots the CDF curve of the overall penalties for different learning models. We can see that DeepCast has a significant performance gain compared with 1-step-DQN, where DeepCast has more than 85% viewers with the penalty less than 1 and this metric is less than 60% for 1-step-DQN. Similarly, n-step-DQN outperforms 1-step-DQN largely in the overall penalty. This is because the n-step update is much more efficient in reward propagating [33]. Even though, DeepCast still outperforms n-step-DQN with about 5% penalty reduction. Compared to Q-learning, actor-critic employs both benefits from the actor and the critic, where the policy gradient strategy enables a faster and more efficient learning process. This comparison indicates that DeepCast is more capable of capturing the intricacies in the viewer scheduling and transcoding context so as to make more effective assignments.

E. Optimality Analysis

Since our target of the allocation problem is to achieve the minimized penalty, it is important to understand how our learning-based policy approaches the optimal solution. We can hardly compute the optimal allocation in the online crowdcast scenario directly; we then consider comparing our model with the offline optimal solution (offline-opt) defined from Eq. 6 to Eq. 10. In the offline scenario, viewers arrive in batches and no viewers leave during its batch. The offline approach tries to allocate these viewers to achieve minimum penalty. The coming viewer numbers in a batch is set as 100. Besides offline-opt, we also compare our model with an online heuristic (*online-heu*) approach, which greedily selects the best server and bitrate for each coming viewer without considering other viewers. Fig. 18 presents the average penalty for these three approaches under different workloads using thin-edge settings. Here the workload is defined as the ratio of occupied bandwidth for all edge servers to the total bandwidth capacity. From this result, we can see that DeepCast incurs an average of 0.3, 0.52 and 1.09 penalty when the edge workload is 20%, 50%, and 80%, respectively. The margins of the penalties between online-heu and offline-opt are 0.191, 0.327, and 0.384, while those between DeepCast and offline-opt are 0.129, 0.20, and 0.252 with different edge workloads, respectively. This comparison indicates that DeepCast is able to reduce the gap between the state-of-the-art approach and the offline optimum by 35.3% on average.

F. Generalization Analysis

In the previous experiments, training is based on the data collected from the same environment with that of the testing process. Our model however can be applied in a new environment (e.g., the viewing patterns are different) with few data or even no real data. We next conduct experiments to evaluate the generalization ability of our DeepCast model.

To this end, we train our model with synthetic data and test the performance with real trace collected from our dataset. Specifically, we first extract a small portion of the data from the whole dataset as a benchmark and obtain the boundary of metrics therein, e.g., the viewer location, the viewing

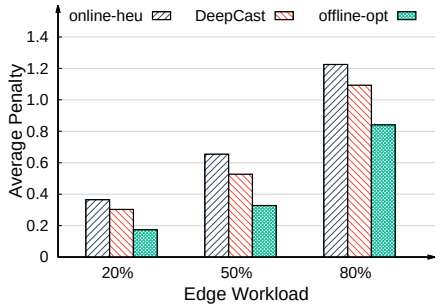


Fig. 18. The comparison with online-heu and offline-opt.

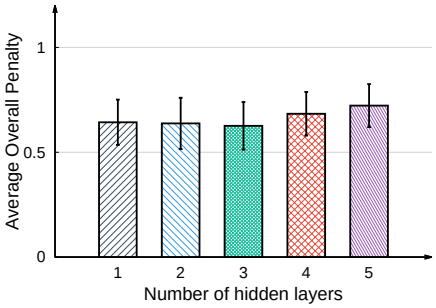


Fig. 19. The average overall penalty with different hidden layer numbers.

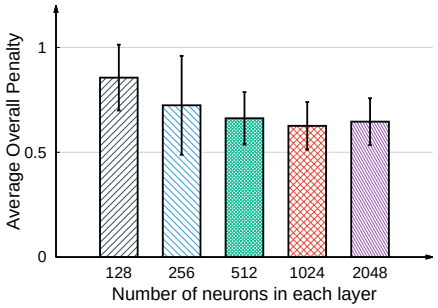


Fig. 20. The average overall penalty with different neuron numbers in each hidden layer.

time length, the bandwidth situations, etc. We then randomly assemble these metrics to form complete viewing requests based on the boundaries. For example, the viewing time length is randomly determined between the longest and the shortest viewing length from the benchmark. After training, we then use the rest dataset to test the performance of our model. It is worth noting that the testing set is different from the benchmark set.

The evaluation results are illustrated in Figure. 17, where each curve indicates the overall penalties of an approach with different edge settings, trained with the synthetic data. Note that we also use online-heu, which is irrelative to the training, as a baseline for comparison. We can find that even with the synthetic data, DeepCast still demonstrates good performance in viewer assignment with low penalties compared with all other approaches. Generally, online-heu performs very closely to joint-online due to the similar heuristic strategy. Compared with these two heuristics and the state-of-the-art approach, DeepCast has an average of 5% to 10% penalty reduction, not to mention even the 25% plus penalty reduction compared to the traditional cloud-CDN approach. Note that even the optimal solution will still incur penalty. Thus the actual penalty reduction of Deepcast will be much higher (see §V-E for more details). Similarly, DeepCast is able to achieve higher performance gain with more available edge capacities due to the flexible selection.

G. Robustness and Sensitivity Analysis

The robustness and sensitivity are also important indicators for a learning system under different configurations. We next conduct experiments to analyze how sensitive is our learning model to different learning architecture and parameters. Specifically, we set a range of neuron numbers and network hidden layer numbers to test the performance of DeepCast.

Hidden layers. To evaluate how sensitive our model is to the hidden layers, we first fix the neuron numbers to 1024 per hidden layer and vary the hidden layer numbers. As illustrated in Fig. 19, we can see that the derived average overall penalties do not have a large difference when the hidden layer number is between 1 and 3. This value achieves 0.626 with a standard deviation of 0.113 when the hidden layer number is 3. When the network architecture has more hidden layers (e.g., 5 layers), the performance only slightly degrades. This is because a deeper network can cause overfitting more easily and needs more parameter tuning. Even in this situation, the average overall penalty increase introduced by adding learning layers is only up to 0.08, which is within 11% of the average penalty.

TABLE IV
EVALUATION WITH DIFFERENT ENTROPY FACTOR SETTINGS.

Entropy factor settings	Average Overall Penalty
dynamic	0.594 ± 0.148
δ=0.1	0.586 ± 0.132
δ=0.2	0.577 ± 0.165
δ=0.5	0.543 ± 0.173

Neuron numbers. We next set the default network layer number to 3 and vary the neuron numbers of each layer to test the performance. The results are shown in Fig. 20. Intuitively, more neurons in each layer will lead to a lower average overall penalty. This trend however comes to a plateau when the neuron number in each layer becomes 2048. This is largely because too many neurons are no longer necessary for improving the representation ability given the problem size. When each layer has too few neurons (e.g., 128), the average overall penalty is much larger, since such neuron size is not sufficient for feature representation. Thus, in our design, we use 1024 neurons per layer as the default setting. The result further indicates that our learning policy is insensitive to the neuron numbers as there are only less than 5% performance difference when setting normal neuron numbers.

Entropy parameters. In our learning model, the entropy term is introduced to encourage more exploration (see Eq. 15), where δ is a hyperparameter to control the level of such exploration. In our experiment setting, we set this parameter as 1 at first and then decrease the term gradually to 0.1, so as to emphasize improving rewards. We compare such dynamic setting with different static settings as presented in Tab. IV. The experiment shows that the dynamic settings slightly outperform other static settings of 0.1, 0.2, and 0.5, respectively. On the other hand, it also demonstrates that our learning model is insensitive to such hyperparameter changes, outputting relatively stable results.

VI. DISCUSSION

Model Updating and Scalability. It is worth noting that our DeepCast can well support the online updating and scalable scheduling. In practical viewer assignment, the well-trained

network can be deployed at each regional server. Once a viewer is allocated, the target server will notify other servers to update their states accordingly. The learning model needs an update when two situations occur. The first one comes from a major update of the edge resources, e.g., more edge servers are introduced for viewer serving, in which a re-training is required given the change of model state. However, this situation only rarely happens and the re-training overhead is little given the low updating frequency. The second one is due to a major change in the viewer access patterns. Our model supports online updating to better adapt to the viewer access patterns. It can periodically updates the network using the past experience. For example, the network can be updated at one server in an hourly or daily basis and the trained network is then distributed to each server.

Personalized QoE Prediction and Setting. The QoE preference of a particular viewer represents one's watching behavior, which is likely a long-term habit rather than an one-time action. Since viewers are familiar with their watching behavior, they can determine the preference among these metrics. On the other hand, viewers' QoE preferences can also be learned by service platforms. We have used a simple method to classify viewers into categories with different QoE demands for evaluation. In practice, however, a crowdcast platform can have long-term watching history data for every viewer and is able to leverage advanced data mining and learning techniques [34] to derive more accurate personalized QoE demands and preferences. It is worth noting that viewers' QoE preference can change over time. Our work is able to well support such preference changes with timely updating.

VII. Conclusion

In this paper, we proposed DeepCast, an intelligent edge-assisted crowdcast framework that applies deep reinforcement learning to afford personalized QoE demands and accommodate cost-effectiveness. We first studied viewers' diversified QoE preferences through data-driven analysis and find that it is complex to achieve the optimal viewer assignment and transcoding selection due to the massive video contents, diverse QoE demands and uncertain online watching behaviors. To better understand the challenges of implementing the online viewer assignment and transcoding selection, we then analyzed its offline scenario with known viewer requests and resource situations. We then proposed DeepCast, an intelligent edge-assisted crowdcast framework that incorporated advanced deep reinforcement learning for dynamic viewer requests and network conditions. We trained the network based on multiple real-world network datasets. Our trace-driven experiments further demonstrated the superiority of DeepCast compared to the state-of-the-art approaches.

Acknowledgement

This work was supported by a Canada Technology Demonstration Program and a Canada NSERC Discovery Grant. Cong Zhang's work was supported in part by NSFC Grant No.61902369. Lifeng Sun's work was supported in part by NSFC Grant No.61936011, Beijing Key Lab of Networked Multimedia.

References

[1] "Live streaming statistics." https://stretchinternet.com/live-streaming-statistics/.

[2] "Facebook live statistics." https://www.socialmediatoday.com/marketing/top-5-facebook-video-statistics-2016-infographic.

[3] "Twitch 2017 year in review." Accessed: July 30, 2017, [online].https://www.twitch.tv/year/2017/.

[4] C. Zhang, J. Liu, and H. Wang, "Towards hybrid cloud-assisted crowdsourced live streaming: measurement and analysis," in *Proceedings of ACM NOSSDAV*, p. 1, 2016.

[5] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing – a key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[6] C. Ge, N. Wang, W. K. Chai, and H. Hellwagner, "Qoe-assured 4k http live streaming via transient segment holding at mobile edge," *IEEE Journal on Selected Areas in Communications*, 2018.

[7] H. Pang, Z. Wang, C. Yan, Q. Ding, and L. Sun, "First mile in crowdsourced live streaming: A content harvest network approach," in *Proceedings of Thematic Workshops of ACM Multimedia*, pp. 101–109, 2017.

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

[9] F. Wang, C. Zhang, J. Liu, Y. Zhu, H. Pang, L. Sun, *et al.*, "Intelligent edge-assisted crowdcast with deep reinforcement learning for personalized qoe," in *Proceedings of IEEE INFOCOM*, pp. 910–918, 2019.

[10] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of ICML*, pp. 1928–1937, 2016.

[11] Z. Wang, L. Sun, C. Wu, W. Zhu, and S. Yang, "Joint online transcoding and geo-distributed delivery for dynamic adaptive streaming," in *Proceedings of IEEE INFOCOM*, 2014.

[12] "Twitch irc." https://help.twitch.tv/customer/portal/articles/1302780-twitch-irc.

[13] X. Ma, C. Zhang, J. Liu, R. Shea, and D. Fu, "Live broadcast with community interactions: Bottlenecks and optimizations," *IEEE Transactions on Multimedia*, vol. 19, no. 6, pp. 1184–1194, 2017.

[14] Z.-N. Li, M. S. Drew, and J. Liu, *Fundamentals of multimedia*. Springer, 2004.

[15] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over http," in *Proceedings of ACM SIGCOMM*, vol. 45, pp. 325–338, 2015.

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

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

[18] M. Dawande, J. Kalagnanam, P. Keskinocak, F. S. Salman, and R. Ravi, "Approximation algorithms for the multiple knapsack problem with assignment restrictions," *Journal of combinatorial optimization*, vol. 4, no. 2, pp. 171–186, 2000.

[19] 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.

[20] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning.," in *Proceedings of AAAI*, vol. 2, p. 5, 2016.

[21] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *CoRR*, vol. abs/1511.06581, 2015.

[22] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

[23] T. Huang, R.-X. Zhang, C. Zhou, and L. Sun, "Qarc: Video quality aware rate control for real-time video streaming based on deep reinforcement learning," in *Proceedings of ACM Multimedia*, 2018.

[24] R. S. Sutton, A. G. Barto, *et al.*, *Reinforcement learning: An introduction*. MIT press, 1998.

[25] M. T. Hagan, H. B. Demuth, M. H. Beale, and O. De Jesús, *Neural network design*, vol. 20. Pws Pub. Boston, 1996.

[26] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proceedings of NuerIPS*, pp. 1057–1063, 2000.

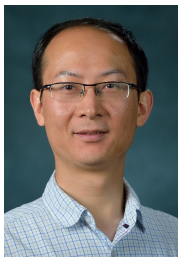
[27] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," in *Proceedings of ICLR*, 2017.
[28] Y. Wu and Y. Tian, "Training agent for first-person shooter game with actor-critic curriculum learning," in *Proceedings of ICLR*, 2016.
[29] "Measureing broadband america 2016 from federal communications commission." https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016.
[30] "Amazon ec2 pricing." https://aws.amazon.com/ec2/pricing/on-demand/.
[31] O. Krajsa and L. Fojtova, "Rtt measurement and its dependence on the real geographical distance," in *Telecommunications and Signal Processing*, pp. 231–234, IEEE, 2011.
[32] M. Abadi, P. Barham, J. Chen, *et al.*, "Tensorflow: a system for large-scale machine learning.," in *Proceedings of USENIX OSDI*, vol. 16, pp. 265–283, 2016.
[33] J. Peng and R. J. Williams, "Incremental multi-step q-learning," in *Machine Learning Proceedings*, pp. 226–232, Elsevier, 1994.
[34] A. M. Elkahky, Y. Song, and X. He, "A multi-view deep learning approach for cross domain user modeling in recommendation systems," in *Proceedings of WWW*, pp. 278–288, 2015.
[35] H. Kellerer, U. Pferschy, and D. Pisinger, *Introduction to NP-Completeness of Knapsack Problems*, pp. 483–493. 01 2004.

APPENDIX

Proof: We prove this theorem by reducing the multiple knapsack problem (MKP), which is a known NP-hard problem [35], to this problem (Ω). As the original problem Ω is complex, we consider a simplified case Ω' of the original problem. We remove the computation constraint, which means each edge has free and sufficient computation capacity so that edges have all versions for every channel. We assume that α_3 is large enough for every viewer, i.e., viewers can not accept bitrate mismatch. We set the penalty of every viewer consistent by setting $\alpha_1^{(u)}$, $\alpha_2^{(u)}$ and $\beta^{(u)}$ the same for every viewer u. In this way, the original problem Ω is simplified to Ω', that is, assigning viewers (i.e., items) to each edge (i.e., knapsack) so that the total profit (i.e., the opposite value of the penalty) is maximized and the total used bandwidth of each edge (i.e., sum of weight for each knapsack) does not exceed the bandwidth capacity (i.e., $W_B^{(e)}$). We therefore can transform the MKP problem to the problem Ω' and vice versa. As problem Ω' is a special case of the original problem Ω, we prove that the viewer assignment and transcoding selection problem is NP-hard. □

Fangxin Wang (S'15) received the B.S. degree from the Department of Computer Science of Technology, Beijing University of Post and Telecommunication, Beijing, China in 2013 and the M.S. degree in the Department of Computer Science and Technology, Beijing, China in 2016. He is currently pursuing the Ph.D. degree in the School of Computing Science, Simon Fraser University, Burnaby, B.C., Canada. His research interests include Internet-of-Things, wireless networks, big data analytics and machine learning.

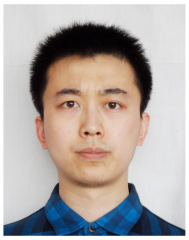
Cong Zhang received M.Sc. from Zhengzhou University, Zhengzhou, China, in 2012, and Ph.D. from Simon Fraser University, Canada, in 2018. Currently, he is an associate researcher in the School of Computer Science at the University of Science and Technology of China. His research interests include multimedia communications and cloud/edge computing.



Feng Wang (S'07-M'13-SM'18) received both the Bachelor's degree and Master's degree in Computer Science and Technology from Tsinghua University, Beijing, China in 2002 and 2005, respectively. He received the PhD degree in Computing Science from Simon Fraser University, Burnaby, British Columbia, Canada in 2012. He is currently an Associated Professor in the Department of Computer and Information Science at the University of Mississippi, University, MS, USA.

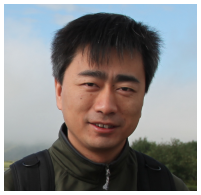
Jiangchuan Liu (S'01-M'03-SM'08-F'17) received B.Eng. (Cum Laude) from Tsinghua University, Beijing, China, in 1999, and Ph.D. from The Hong Kong University of Science and Technology in 2003, both in Computer Science. He is currently a Full Professor (with University Professorship) in the School of Computing Science at Simon Fraser University, British Columbia, Canada. He is an IEEE Fellow, a Fellow of Canadian Academy of Engineering, and an NSERC E.W.R. Steacie Memorial Fellow.

He is a Steering Committee Member of IEEE Transactions on Mobile Computing, and Associate Editor of IEEE/ACM Transactions on Networking, etc. He is a co-recipient of the Test of Time Paper Award of IEEE INFOCOM (2015), ACM TOMCCAP Nicolas D. Georganas Best Paper Award (2013), and ACM Multimedia Best Paper Award (2012).



Yifei Zhu (S'15) received the B.E. degree from Xi'an Jiaotong University, Xian, China, in 2012, and the M.Phil. degree from Hong Kong University of Science and Technology, Hong Kong, in 2015. He is now a Ph.D. student in the School of Computing Science, Simon Fraser University, British Columbia, Canada. His areas of interest are cloud computing, multimedia networking, Internet-of-Things and crowdsourcing.

Haitian Pang (S'16) received the B.E. degree from the Department of Automation, Tsinghua University, in 2014 and Ph.D. Degree from the Department of Computer Science and Technology, Tsinghua University, in 2019. His research interests include network game modeling, cellular-WiFi networking, and mobile networking optimizations.



Lifeng Sun (M'05) was born in 1972. He is currently a Professor with the Department of Computer Science and Technology, Tsinghua University. His research interests include video streaming, video coding, video analysis, and multimedia edge computing. He received the Best Paper Award in the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY in 2010, the Best Paper Award from ACM Multimedia 2012, and the Best Student Paper Award from MMM 2015, IEEE BigMM 2017 and NOSSDAV 2019.