Intelligent Edge-Assisted Crowdcast with Deep Reinforcement Learning for Personalized QoE

Fangxin Wang*, Cong Zhang◊*, Feng Wang‡, Jiangchuan Liu*, Yifei Zhu*, Haitian Pang†*, Lifeng Sun†
*School of Computing Science, Simon Fraser University, Canada
◊School of Computer Science and Technology, University of Science and Technology of China, China
‡Department of Computer and Information Science, The University of Mississippi, USA
†Department of Computer Science and Technology, Tsinghua University, China

***Abstract*—Recent years have seen booming development and great success in interactive crowdsourced livecast (i.e., crowdcast). Different from traditional livecast services, crowdcast is featured with tremendous video contents at the broadcaster side, highly diverse viewer side content watching environments/preferences as well as viewers' personalized quality of experience (QoE) demands (e.g., individual preferences for streaming delays, channel switching latencies and bitrates). This imposes unprecedented key challenges on how to flexibly and cost-effectively accommodate the heterogeneous and personalized QoE demands for the mass of viewers.**

In this paper, we propose *DeepCast*, an edge-assisted crowdcast framework, which makes intelligent decisions at edges based on the massive amount of real-time information from the network and viewers to accommodate personalized QoE with minimized system cost. Given the excessive computation complexity in this context, we propose a data-driven deep reinforcement learning (DRL) based solution that can automatically learn the best suitable strategies for viewer scheduling and transcoding selection. To our best knowledge, DeepCast is the first edge-assisted framework that applies the advance of DRL to explicitly accommodate personalized QoE optimization for crowdcast services. We collect multiple real-world datasets and evaluate the performance of DeepCast using trace-driven experiments. The results demonstrate the superiority of our DeepCast framework and its DRL-based solution.

I. Introduction

In recent years, the interactive crowdsourced livecast (or *crowdcast*) has become increasingly popular and seen explosive success in the market, generating billion dollars of revenue for major crowdcast platforms[1]. In a crowdcast service, numerous broadcasters can stream their own contents to the viewers in their channels through crowdcast platforms, such as Twitch.tv[2], Youtube Gaming[3], and Douyu[4], to name but a few. Usually, the source streaming will be sent to the cloud and CDN servers for transcoding and then delivering to massive viewers with different bitrates [1].

Compared to conventional livecast services where professional video producers (e.g., TV channels and Netflix) broadcast well-planned content to viewers, crowdcast is featured with tremendous video contents generated at the broadcaster side, highly diverse viewer side content watching environments/preferences, as well as frequent interactions among broadcasters and viewers. This imposes unprecedented key challenges on how to flexibly and cost-effectively accommodate the heterogeneous and personalized quality of experience (QoE) demands (such as individual preferences for streaming delays, channel switching latencies and bitrates) for different viewers.

For example, many crowdcast platforms allow viewers to send interactive messages in the chatbox of a live channel, and both the broadcaster and other viewers can view these messages and give feedbacks [2]. Some viewers can be very proactive and have frequent interactions, while others can be passive and only watch video contents with the message display closed. In this case, the proactive viewers are much more sensitive to the video latency than the passive viewers because frequently delayed messages will impair their interaction experience. Another example is that viewers usually have different viewing patterns. A considerable part of viewers browse many channels frequently and even stay only a few seconds in each channel. They tend to care more about the channel switching latency rather than the streaming delay and the bitrate. In contrast, there are also faithful viewers who only watch the channel of one particular broadcaster, thereby being insensitive to the channel switching latency.

Traditional solutions for livecast services mainly rely on cloud-CDN architectures for streaming transcoding and delivery. Yet such dedicated architectures cannot well satisfy the viewers' personalized QoE demands since all the QoE metrics are optimized in a monolithic approach. A new emerging network paradigm, i.e., edge computing [3], brings new possibilities to the livecast service by pushing the service closer to the end users. The existing edge-based approaches, however, are still not able to well address the unique challenge from crowdcast to effectively accommodate viewers' diverse QoE demands.

and utilizes

In this paper, we propose DeepCast, a novel edge-assisted framework that is customized for crowdcast to accommodate personalized QoE while minimizing the overall service cost. In DeepCast, distributed edge servers only need to receive

[1]https://www.fool.com/investing/2017/11/26/amazoncoms-twitch-is-dominating-the-game-streaming.aspx
[2]https://www.twitch.tv/
[3]https://gaming.youtube.com/
[4]https://www.douyu.com/

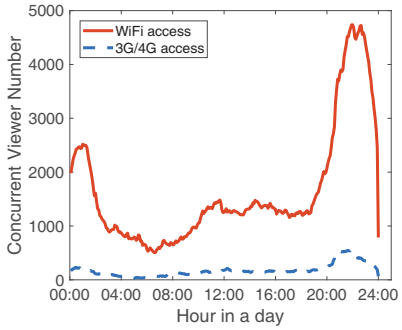


Fig. 1. The number of concurrent viewers in different time periods of a typical day.

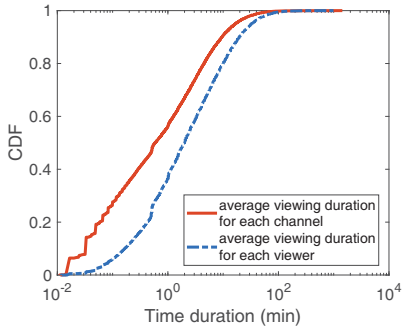


Fig. 2. The CDF plot of average viewing duration for every channel and average viewing duration for every viewer.

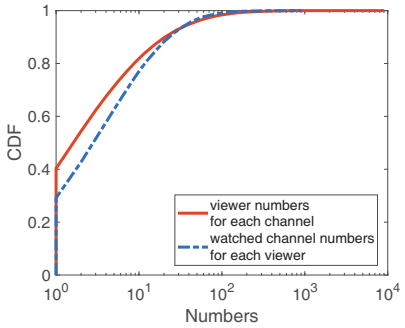


Fig. 3. The CDF plot of viewer numbers in different live channels and watched channel numbers for every viewer during a day.

streaming of high bitrate from CDN servers and then downsample (or transcode) to other bitrates directly requested by various local viewers, where edge servers also work collaboratively to schedule the viewer requests to proper servers based on different QoE demands and current system resource distribution. Although desirable, it is challenging to achieve an optimal viewer scheduling and bitrate transcoding selection due to the massive video contents, diversified QoE demands and uncertain online watching behaviors. To this end, we make effective use of the advanced deep reinforcement learning (DRL) [4]–[8] to tackle the key challenge therein. DRL learns a control policy (i.e., assign viewers to which edge server) purely based on the experience, without any predefined rules. Specifically, the edge system maintains an RL agent, which starts knowing nothing and gradually learns to optimize the viewer assignment policy based on the observed performance of the past assignment.

In particular, DeepCast uses a state-of-the-art Actor-Critic network model (A3C [6]) to train the deep neural network, which extracts the edge system resource usage situation, current viewing situation and the personalized viewer request as a state and selects an optimal action through the network. We collect three real-world datasets including a livecast viewing trace and real edge trace in a major city in China, and a public viewer bandwidth trace in the US, to evaluate the performance of DeepCast by trace-driven experiments. The results demonstrate the superiority of our DeepCast framework and its DRL solution.

In summary, the contributions of this paper are as follows:

- We propose a novel edge-assisted framework called DeepCast that is customized for crowdcast services. We comprehensively consider the personalized QoE targets (e.g., different preferences in streaming delay, channel switching latency and bitrate) and system cost (e.g., computation and bandwidth cost), and integrate them into a viewer scheduling optimization.
- We propose a DRL based experience-driven solution in DeepCast, which can effectively learn the best suitable strategy of viewer scheduling and transcoding selection to achieve high personalized QoE and low system cost.
- We implement the DeepCast framework and collect three real-world datasets for trace-driven experiments. We compare DeepCast with the state-of-the-art solutions and the results further demonstrate the superiority of DeepCast.

To our best knowledge, DeepCast is the first edge-assisted framework that applies DRL to explicitly afford personalized QoE optimization for crowdcast service.

The rest of paper is organized as follows. Section II introduces our motivation through a data trace analysis. Section III describes our edge-assisted crowdcast framework followed by a problem formulation. Section IV details our design of DRL based solution. Section V comprehensively evaluates the performance of DeepCast through trace-driven experiments. We review some related work in section VI with the conclusion in section VII.

II. Motivation

In this section, we first leverage a trace-driven data analysis to better understand the diversity of personalized QoE demands in crowdcast service. We have 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.

Fig. 1 illustrates the number of concurrent viewers through different access approaches in different time periods in a typical day. We can find that about 10% of viewers use the 3G/4G cellular network for crowdcast service. Considering the high prices for cellular data and the huge traffic amount for videos, such viewers may not have a strong favor in high bitrate. Instead, they are often willing to sacrifice the bitrate for a low streaming delay and a smooth watching experience. Besides, the number of concurrent viewers at the evening peak time is 4 to 5 times than other time periods, which can also bring high burden to the platform's bandwidth supply if all viewers are served with very high bitrates.

We also investigate the average viewing time duration for each channel and viewer, as illustrated in Fig. 2. We can

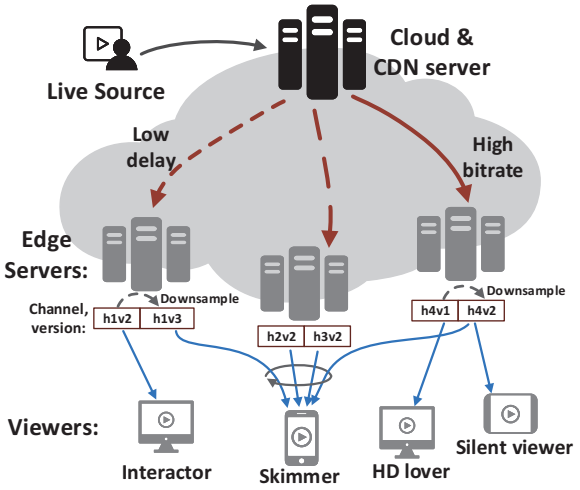


Fig. 4. The edge-assisted crowdcast framework. Edge servers can get channel content of different bitrates from the CDN server and can downsample from high bitrate to low bitrate to serve viewers with various bandwidth situations. Viewers can be served by different servers (edges or the CDN) to optimize their personalized QoE demands.

see that there are about 35% of viewers watching a channel for less than one minute. For this type of viewers, the most obvious impact on viewing experience is the channel switching latency since they usually browse many channels frequently. They tend to care more about this latency rather than the streaming delay and the bitrate. As a comparison, there are about 15% of viewers watching a channel for more than one hour. These viewers are usually more loyal to their favourite channels and have a much higher tolerance for the channel switching latency.

Fig. 3 illustrates the CDF plot of the number of total viewers for every channel and the total watched channels for every viewer in a typical day. We can see that the viewer distribution exhibits a typical long-tailed distribution, where only 2% channels have more than 100 viewers. Besides, there are more than 50% viewers watching more than 3 channels. This observation indicates a highly heterogeneous channel viewing preference for different viewers, where they can have dramatically different QoE preference in different channels.

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 well considered and accommodated 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. Edge-Assisted Crowdcast Framework

Fig. 4 illustrates our edge-assisted crowdcast framework, where 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 through RTMP (Real-Time Messaging Protocol). The original streaming is then encoded and compressed into RTSP (Real-Time Streaming Protocol) streams with multiple bitrates, which are pushed to the CDN.

In a citywide area, the edge servers are distributed much closer to the viewers and each edge server will serve the crowdcast viewing requests within its proximity. The CDN usually only needs to deliver channel content of high-quality versions to the edge servers through HTTP. Then the edge servers can *transcode* (or downsample) the high versions to low-quality versions to serve the viewers with different bitrate requests. A viewer may request a channel content with a specific version based on his/her own bandwidth condition. Given the viewers' personalized QoE demand and the current resource allocation situation, the regional edge can choose to serve the viewer itself or redirect the request to another edge (or the CDN), so as to optimize viewers' personalized QoE demands and minimize the system cost. For example in Fig. 4, the requests of a channel skimmer (a viewer who quickly browses many channels) can be redirected to the nearest available edge server rather than the CDN to achieve low channel switching latency. Tab. I lists the notations used in this paper.

TABLE I
Notations used in our framework

$\mathbf{E}$	the edge server list
c	the CDN server
$\mathbf{H}$	the streaming channels
$\mathbf{V}$	the version list of a channel
$\mathbf{U}$	the set of all the viewers
$\phi(u)$	the target version of viewer u
$X^{(u,j)}_{(h,v)}$	indicate whether user u is scheduled to server j for (h,v)
$Y^{(e)}_{(h,v)}$	indicate whether (h,v) is at edge e
$\mathcal{D}^{(u)}$	the streaming delay of user u
$l^{(u,j)}$	the latency between user u and server j
$T^{(e)}_{R(h,v^*,v)}$	the transcoding latency for channel h when the highest version is v^*
$\mathcal{L}^{(u)}$	the channel switching latency for user u
$\mathcal{B}^{(u)}$	the bitrate mismatch level for user u
$M(\phi(u),v)$	a predefined bitrate mismatch utility function
$\mathcal{C}_T$	the total cost of computation
$(h,v^*)_e$	the highest version of channel h in edge e
$I_{T(h,v)}$	the resource consumption for transcoding to (h,v)
$P^{(e)}_T$	the price of unit computation resource
$\mathcal{C}_B$	the total cost of bandwidth
P^j_B	the unit bandwidth price of server j
$I_{B(h,v)}$	the bandwidth resource consumption for serving (h,v)
$W^{(e)}_T$	the computation capacity of edge e
$W^{(e)}_B$	the bandwidth capacity of edge e
α,β	the weighted parameters to tune the QoE penalty and cost penalty
$\alpha^{(u)}_1,\alpha^{(u)}_2,\alpha^{(u)}_3$	the personalized QoE factors for viewer u
s_t	the state at time t
a_t	the action at time t
r_t	the reward at time t
γ	the discount factor for future reward

B. Problem Formulation

To better understand the challenges of implementing the online viewer scheduling and transcoding selection, we start

from analyzing its offline scenario with known viewer requests and resource situations. We assume that in a city-level region the crowdcast service provider has one CDN server c and a set of edge servers as $\mathbf{E} = \{1, 2, ..., E\}$ distributed across the city. 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 as $\mathbf{U} = \{1, 2, ..., U\}$ 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 scheduling outcome, where $X^{(u,j)}_{(h,v)} = 1$ ($j \in \{\mathbf{E} \cup c\}$) (resp. 0) indicates user u is (resp. is not) scheduled to j for (h, v). And $Y^{(e)}_{(h,v)} = 1$ (resp. 0) denotes that (h, v) is (resp. is not) at edge e.

We consider three QoE metrics for each viewer, 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 the viewer and the actually 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), and $T^{(e)}_{R(h,v^*,v)}$ is the transcoding latency when the highest version is v^*. The first part of the equation represents the latency if the viewer is scheduled to an edge server, while the second part means the viewer is directly scheduled to the CDN server. Note that if $v^* = v$, there is no need to transcode (i.e., $T^{(e)}_{R(h,v^*,v)} = 0$) because the edge receives this version from 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. Then in this situation, there is a bitrate mismatch, which also affects the QoE. We can calculate the mismatch level $\mathcal{B}^{(u)}$ as follows:

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

where $M(\phi(u), v)$ is a predefined bitrate mismatch utility function.

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

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

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

$$\mathcal{C}_B = \sum_{e} \sum_{u} \sum_{h,v} X^{(u,e)}_{(h,v)} I_{B(h,v)} P^{(e)}_B + \sum_{h,v} \sum_{u} X^{(u,c)}_{(h,v)} I_{B(h,v)} P^{(c)}_B + \sum_{e} \sum_{h,v^*} Y^{(e)}_{(h,v^*)} I_{B(h,v^*)} P^{(c)}_B \tag{5}$$

where $I_{B(h,v)}$ is the resource consumption for serving (h, v), $P^{(e)}_B$ and $P^{(c)}_B$ 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 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, including the QoE penalty (penalty for streaming delay, channel switching latency and bitrate mismatch) and the system cost penalty:

$$Min: \alpha \sum_{u} \left(\alpha^{(u)}_1 \mathcal{D}^{(u)} + \alpha^{(u)}_2 \mathcal{L}^{(u)} + \alpha^{(u)}_3 \mathcal{B}^{(u)} \right) + \beta \left(\mathcal{C}_T + \mathcal{C}_B \right) \tag{6}$$

$s.t.$

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

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

$$\sum_{h,v} Y^{(e)}_{(h,v)} I_{T(h,v)} \leq W^{(e)}_T, \forall e \tag{9}$$

$$\sum_{h,v} \sum_{u} X^{(u,e)}_{(h,v)} I_{B(h,v)} \leq W^{(e)}_B, \forall e \tag{10}$$

where $W^{(e)}_T$ and $W^{(e)}_B$ 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 $\alpha^{(u)}_1$, $\alpha^{(u)}_2$, $\alpha^{(u)}_3$ 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 suitable version. Eq. 9 and Eq. 10 ensure that the resource usage does not exceed the capacity.

The offline problem (Ω) can be reduced to a multi-dimensional knapsack problem (by considering all servers can have all the channels and versions), which is an NP-complete problem. Even this problem can have pseudo-poly solution, it is quite challenging to achieve fast and effective scheduling

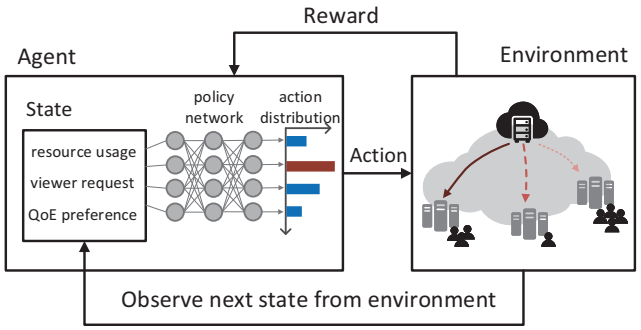


Fig. 5. The workflow of using deep reinforcement learning for crowdcast viewer scheduling.

with such massive amount of viewers and their diversified QoE preferences, not to mention that in practice, the viewers are arriving and leaving dynamically. With the abundant network and viewer information, we believe a data-driven solution is necessary in the edge-assisted crowdcast context. To this end, we turn to the design of a deep reinforcement learning based model to solve the problem, as described in the next section.

IV. Deep Reinforcement Learning Model Design

In this section, we first introduce the basic learning mechanism of applying deep reinforcement learning (DRL) into the scheduling problem. We then describe how we transform the online viewer scheduling and transcoding selection problem into a learning task and design a DRL based model to learn an effective solution.

A. Basic Learning Mechanism

Unlike existing edge resource allocation approaches using predefined rules or model-based heuristics, DRL strives to learn a general action decision from the past experience based on the current state and the given reward. Specifically, in the workflow of DeepCast as illustrated in Fig. 5, a learning *agent* interacts with the *environment* in the DRL setting, where the agent is the main component of scheduling decision and the environment defines the rules, restrictions and reward mechanism. At 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. If the agent continues this process, 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.

B. Model Design

Our DeepCast solution uses a state-of-the-art actor-critic based DRL model A3C [6]. We introduce the detailed functionality design as follows.

State space. We consider the practical online scenario where 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 scheduling 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 the 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 server.

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 the request to other edges or the CDN. We therefore use a viewer connection table $\mathbf{tab} = (e, h, v)$ to record the viewer scheduling 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[5]. 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 scheduling. 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 so that we cannot store them in a tabular form and solve the problem using traditional methods, e.g., Q-learning and SARSA [5]. To address this issue, we use neural network [9] to represent the policy π, where the adjustable parameters of the neural network is referred to as the policy parameters θ. Then we can 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 the 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 the available high-quality version or directly request the version from the CDN (when no available higher version). Note that the computation and bandwidth resource 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 a practical crowdcast scenario, 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 becomes $(|E|+1)^K$, where the model is very challenging especially when K is very large. Hence, we divide

[5]In practice, the personalized QoE preference can be either directly set by each viewer, or learned through analyzing each viewer's watching history (e.g., interaction frequency, channel switching frequency, average watching duration, etc.), which can be easily obtained by crowdcast platforms. A simple example of possible prediction methodologies will be discussed in §V-A.

the time into small slots so that in each slot DeepCast only processes one viewer request.

Reward. When applying an action a_t to the state s_t, the learning agent will receive 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 the reward r_t as the opposite number of the generated overall penalty for the coming viewer u as:

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

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 scheduling viewer u at time t.

Training methodology. We train the DeepCast learning model using the state-of-the-art asynchronous advantage actor-critic (A3C) model [6], [10], [11] since the actor-critic architecture well matches our context and it also has shown successful applications in many other research problems [10], [11]. The DeepCast model 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 of network parameters 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 learning model employs a *policy gradient algorithm* [12]. The key idea of this algorithm is to estimate the parameter gradient direction toward the maximized total reward. In each step, we first compute the gradient of the accumulated discounted reward regarding the parameter θ, with which we are able to update the actor network parameter θ. To avoid the convergence to a suboptimal policy, we also add an entropy regularization term to the actor's update rule as in [7], which helps to encourage more exploration. We train the critic network following a temporal difference method [5]. Once the actor-critic network is well trained, we can select the viewer scheduling action based on the output of the actor network.

V. Performance Evaluation

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

A. Methodology

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

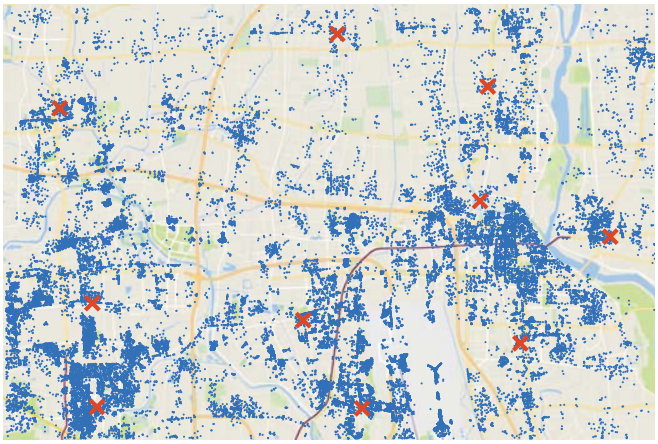


Fig. 6. The sampling rectangular area of viewers and edge servers. The blue dot represents the location of viewers and the red cross indicates the location of edge servers.

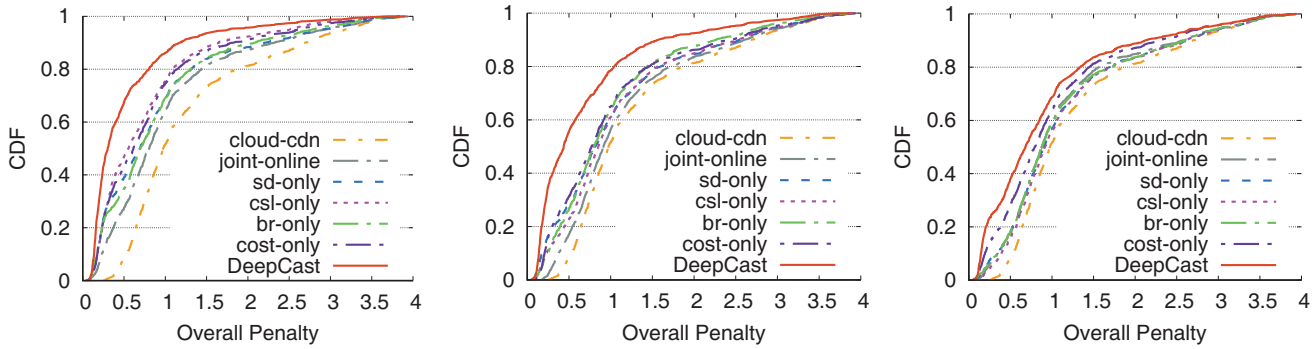
TABLE II
Measurement result of bandwidth, transcoding resource usage and transcoding latency.

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

- **Trace for viewing information.** We collected 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. We obtained the viewer ID, channel ID, start time and end time from this dataset.
- **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 Beijing. We uniformly sample a part of the access point locations as the edge locations.
- **Trace for bandwidth information.** We collected the bandwidth situation from a broadband dataset from FCC [13]. We extract the average bandwidth in this dataset as the viewer bandwidth situation in our experiment.

Since the dataset of Inke.tv does not disclose the location and bandwidth information, we instead extract the location information from the iQiYi dataset and the bandwidth information from the FCC dataset, and use the integrated data for training and evaluation.

Edge-assisted crowdcast measurement and setup. 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. 6. 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.



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

Fig. 7. The CDF plot of overall penalty by different approaches under different edge settings.

TABLE III
VIEWER CLASSIFICATION METHODOLOGY.

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 IV
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

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 [14]. 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 1 second stream from the highest bitrate to lower bitrate versions in Tab. II. We set the latency between edges and CDN randomly from 20 ms to 100 ms considering the good network condition of edge servers. The latency between CDN and viewers is randomly set from 100 ms to 300 ms. And the latency between viewers and edges is proportional to their geo-distance [15], with a maximal value of 100 ms.

Training setup. We implement the DeepCast learning model using tensorflow [16] 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 we set the default neuron numbers in the hidden layer as 4096 plus 2048. We train the network using the viewer watching situation in a day. 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 $\mathcal{B} = log(R^*/R)$, where R^* is the target bitrate for a viewer based on the bandwidth situation and R is the actually allocated bitrate. This logarithmic representation was used by BOLA [17] and can effectively decrease the marginal improvement of high bitrates. With the viewing information dataset, we use a simple method to classify the viewers into different QoE preference categories based on their daily average watching channel numbers ($\bar{n}$) and watching durations ($\bar{t}$). We classify the viewers in our dataset into four categories, i.e., streaming delay preferred (*sd-pref*), channel switching latency preferred (*csl-pref*), bitrate preferred (*br-pref*) and others (*normal*), and set different QoE parameters for each categories, as illustrated in Tab. III. We tune these parameters to balance each part with the normal setting. We use Amazon AWS [14] as a reference for the setting of bandwidth and transcoding cost, and set the edge bandwidth price to be 20% of that 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*): this architecture schedules all viewers to CDN instead of edges. 2) Edge-assisted conventional livecast architecture which uses a state-of-the-art algorithm proposed in [1] for scheduling and transcoding (*joint-online*): the viewers are first scheduled to proper edges and then transcoding are conducted accordingly. 3) Streaming delay only (*sd-only*): we only consider the video latency in the training process. 4) Startup latency only (*csl-only*): we only consider the channel switching latency in the training process. 5) Bitrate only (*br-only*): we only consider the bitrate mismatch level in the training process. 6) Cost only (*cost-only*): we only consider the system cost in the training process.

B. Evaluation Results

We next evaluate the performance of DeepCast and analyze the impact of edge capacity on the experiment results. We set three different edge capacities, i.e., *fat-edge*, *mid-edge* and *thin-edge*, where the detailed capacity settings are shown in Tab. IV. We set both the QoE penalty factor α and the system cost factor β as 0.5 by default. Fig. 7 shows the CDF plot of the overall penalty of each viewer scheduling under different edge capacities. And Fig. 8 provides the normalized average

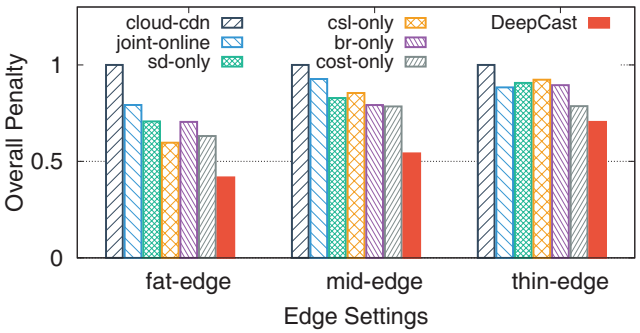


Fig. 8. The normalized overall penalty under different edge settings.

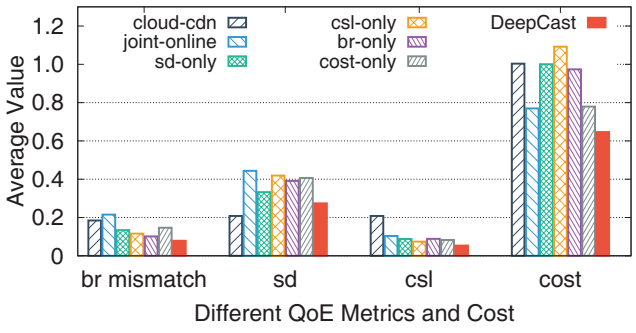


Fig. 9. The value of different QoE metrics under different edge capacity settings.

overall penalty, where we set the comparison method *cloud-cdn* as the baseline method.

We have three key observations from this comparisive experiment. First, DeepCast can easily achieve much lower overall penalty than other existing approaches in all the edge capacity settings. Specifically, in the *mid-edge* setting, DeepCast reduces an average of 45.9% overall penalty than the *cloud-cdn* solution and 41.6% overall penalty than the *joint-online* solution. This result indicates that DeepCast can effectively utilize the edge servers to satisfy viewers' personalized and heterogeneous QoE demands and make intelligent viewer scheduling, while the *cloud-cdn* solution and the *joint-online* solution cannot well schedule each viewer, incurring high overall penalty. Besides, DeepCast performs even better when there is more edge capacity. From Fig. 8 the normalized overall penalty of DeepCast achieves 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 choice, which can further lead to a lower overall penalty. Moreover, DeepCast 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 make proper scheduling accordingly.

Fig. 9 illustrates the detailed QoE metrics and system cost under the *mid-edge* settings. We can find that DeepCast achieves an average of 0.07 bitrate mismatch, which is 57.6% less than *cloud-cdn* and 63.7% less than the *joint-online*. This result shows that DeepCast can provide better bitrate match given edge-assisted crowdcast architecture and the intelligent scheduling. For the channel switching latency, DeepCast only needs an average of 0.05 s, reducing 75% time compared to traditional *cloud-cdn* architecture due to the much smaller latency between viewers and edges than that between viewers and the CDN. For the streaming delay, DeepCast also reduces 38.3% time than *joint-online* that uses a heuristic scheduling strategy. Note that the streaming delay of DeepCast is slightly higher than that of *cloud-cdn* due to the transcoding delay on edge. However, this additional 0.06 s stream delay is still well within human tolerance for the interactions. Regarding the system cost, DeepCast reduces the average cost penalty by 36% than the *cloud-cdn* solution and by 16.7% than the *joint-online* solution.

VI. Related Work

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 service. Wang et al. [1] considered the video transcoding and viewer delivery in a cloud-CDN architecture, where the authors separate the viewer scheduling and content transcoding and solve each with a heuristic algorithm. Yan et al. [18] proposed a transparent network service called LiveJack, which enables CDN to seamlessly integrate edge clouds for live broadcast. The main focus is on system implementation rather than the consideration of optimizing personalized QoE. Pang et al. [19] used edge servers as relays to reduce the loss rate and latency of the first mile video transmission in crowdcast. Ge et al. [20] proposed an edge-based system to achieve 4K live streaming across the global Internet. These works either focused on the cache cost or dedicated QoE target rather than the personalized QoE. In this paper, we focused on developing an edge-assisted crowdcast framework with a data-driven learning to accommodate personalized QoE with minimized system cost.

B. Deep Reinforcement Learning in Networking

In recent years, deep reinforcement learning (DRL) has shown amazing potentials in many applications [4]. Minh et al. [4] first used Deep Q-Network (DQN) to learn policies from sensor input for decision making. In their work, experience replay and target network were introduced to improve the stability and the performance. Double DQN was next proposed by Van Hasselt et al. [21], which was able to reduce the observed overestimations and achieve better performance on several games. Schaul et al. [22] developed a framework for prioritizing experience, so as to replay important transitions more frequently, and therefore learn more efficiently. Wang et al. [23] proposed dueling network to represent two separate estimators: one for the state value function and one for the

state-dependent action advantage function. The main benefit of this factoring is to generalize learning across actions without imposing any change to the underlying reinforcement learning algorithm. There have been recent works on applying state-of-the-art DRL frameworks such as DDPG [24] and A3C [6] in intended services. Toward this direction, DeepRM [8] leveraged DRL to solve the online multi-resource job scheduling problem and can achieve 46% job completion time than heuristic algorithms. Pensieve [7] used A3C algorithm with DRL to select the optimal bitrate for future video chunks purely based on the past experience. DRL-TE [25] applied the DRL to the traffic engineering problem and achieved experience-driven network utility maximization. Different from these aforementioned works, we consider the viewer scheduling and transcoding selection problem in the crowdcast service. To our best knowledge, DeepCast is the first to apply DRL into edge-assisted crowdcast service to cost-effectively accommodate personalized QoE demands.

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 a data analysis and find that it is complicated to achieve the optimal viewer scheduling 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 scheduling 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 solutions.

Acknowledgement

This work is supported by a Canada Technology Demonstration Program (TDP) grant and a Canada NSERC Discovery Grant. The work of Haitian Pang and Lifeng Sun is supported by the Key Research and Development Project under Grant No. 2018YFB1003703, Beijing Key Laboratory of Networked Multimedia. The corresponding author is Jiangchuan Liu.

References

[1] 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 the International Conference on Computer Communications (INFOCOM)*, IEEE, 2014.

[2] C. Zhang and J. Liu, "On crowdsourced interactive live streaming: a twitch. tv-based measurement study," in *Proceedings of the 25th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pp. 55–60, 2015.

[3] 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.

[4] 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.

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

[6] 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 International Conference on Machine Learning (ICML)*, pp. 1928–1937, 2016.

[7] N. adaptive video streaming with pensieve, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pp. 197–210, ACM, 2017.

[8] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNet)*, pp. 50–56, ACM, 2016.

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

[10] 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 5th International Conference on Learning Representations (ICLR)*, 2017.

[11] Y. Wu and Y. Tian, "Training agent for first-person shooter game with actor-critic curriculum learning," in *Proceedings of 4th International Conference on Learning Representations (ICLR)*, 2016.

[12] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems (NIPS)*, pp. 1057–1063, 2000.

[13] "Measureing broadband america 2016 from federal communications commission." https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016.

[14] "Amazon ec2 pricing." https://aws.amazon.com/ec2/pricing/on-demand/.

[15] O. Krajsa and L. Fojtova, "Rtt measurement and its dependence on the real geographical distance," in *Telecommunications and Signal Processing (TSP)*, pp. 231–234, IEEE, 2011.

[16] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: a system for large-scale machine learning.," in *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, vol. 16, pp. 265–283, 2016.

[17] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "Bola: Near-optimal bitrate adaptation for online videos," in *Proceedings of the International Conference on Computer Communications (INFOCOM)*, pp. 1–9, IEEE, 2016.

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

[19] 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, ACM, 2017.

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

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

[22] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *CoRR*, vol. abs/1511.05952, 2015.

[23] 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.

[24] 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.

[25] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *Proceedings of the International Conference on Computer Communications (INFOCOM)*, IEEE, 2018.