Enhancing the Crowdsourced Live Streaming: a Deep Reinforcement Learning Approach

Rui-Xiao Zhang*, Tianchi Huang*, Ming Ma‡§,Haitian Pang*, Xin Yao*,Chenglei Wu*, Lifeng Sun*§
* Department of Computer Science and Technology, Tsinghua University, Beijing, China
‡ Beijing Kuaishou Technology Co., Ltd., China
{zhangrx17,htc17,pht14,wucl18,yaox16}@mails.tsinghua.edu.cn
maming@kuaishou.com, sunlf@tsinghua.edu.cn

ABSTRACT

With the growing demand for crowdsourced live streaming (CLS), how to schedule the large-scale dynamic viewers effectively among different Content Delivery Network (CDN) providers has become one of the most significant challenges for CLS platforms. Although abundant algorithms have been proposed in recent years, they suffer from a critical limitation: due to their inaccurate feature engineering or naive rules, they cannot optimally schedule viewers. To address this concern, we propose LTS (Learn to schedule), a deep reinforcement learning (DRL) based scheduling approach that can dynamically adapt to the variation of both viewer traffics and CDN performance. After the extensive evaluation the real data from a leading CLS platform in China, we demonstrate that LTS improves the average quality of experience (QoE) over state-of-the-art approach by 8.71%-15.63%.

CCS CONCEPTS

• **Information systems** → *Multimedia streaming*; • **Computing methodologies** → *Neural networks.*

KEYWORDS

Crowdsourced Live Streaming, Reinforcement Learning, Scheduling

ACM Reference Format:
Rui-Xiao Zhang*, Tianchi Huang*, Ming Ma‡§,Haitian Pang*, Xin Yao*,Chenglei Wu*, Lifeng Sun*§. 2019. Enhancing the Crowdsourced Live Streaming: a Deep Reinforcement Learning Approach. In *29th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '19), June 21, 2019, Amherst, MA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3304112.3325607

1 INTRODUCTION

Over the past few years, crowdsourced live streaming (CLS) has become a novel video service on the Internet, and many CLS platforms, such as Twitch.tv, have shown unprecedented growth across the world [9]. To guarantee reliable quality of experience (QoE), the

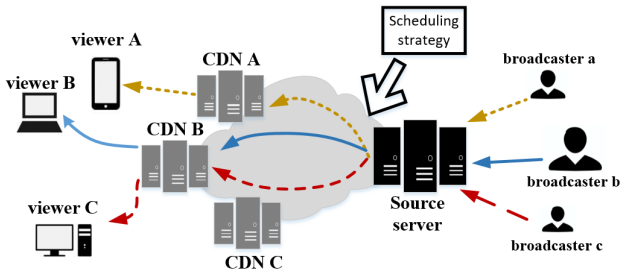


Figure 1: The workflow of CLS scheduling. This figure shows that viewer A, B, and C are watching broadcaster a, b, and c through CDN A, CDN B, CDN B, respectively.

CLS platforms usually employ multiple Content Delivery Network (CDN) providers to serve viewers, and at each time, viewer requests will be scheduled to one of them according to a specific scheduling strategy. We call this practically-motivated scheduling problem as CLS scheduling problem, and a typical workflow for CLS scheduling is shown in Figure 1.

One most unique feature of CLS services is that general users can broadcast their own contents to numerous viewers, and viewers will enjoy the streaming entertainment with different devices (a Mobile phone, an iPad, or a personal computer). Unfortunately, existing algorithms are incapable of handling the above feature. Most of the approaches use prediction-based methods to schedule viewer requests. By making use of historical data, they extract some handcraft features to predict the performance of different CDN providers and redirect viewers to the best one [7, 13]. Nevertheless, these algorithms require significant tuning, and the insights found in one scenario may not generalize well in others. To solve these problems, state-of-the-art work [8] converts the problem into a multi-arms bandit problem to lessen the prediction bias suffered by previous approaches, and uses the *Upper Confidence Bound* (UCB) algorithms to solve it. However, the simplified performance evaluation method (e.g., average) and the ignorance of future dynamics from either viewers or CDN performance make it hard to work.

In this paper, we propose LTS (learn to schedule), a *deep reinforcement learning* (DRL) based approach to deal with CLS scheduling problem. Specifically, we will highlight two most fundamental difficulties: 1) How to design an approach which can make decisions without any human-generated rules and flexibly accommodate to the constantly changing CDN provider performance. 2) How to

§ Lifeng Sun and Ming Ma are the corresponding authors of this paper.

NOSSDAV '19, June 21, 2019, Amherst, MA, USA

ACM ISBN 978-1-4503-6298-6/19/06…$15.00
https://doi.org/10.1145/3304112.3325607

design an approach which can well utilize long-term historical information and automatically cope with different viewer patterns (e.g., flash crowd).

DRL is suitable for CLS scheduling problem since it can well tackle the above difficulties: by using the powerful neural network, it can both make use of the information generated by the environment and generate decisions without any presumption. DRL is also promising as it enables the platforms high-levelly control viewer traffics by just simply defining its own reward function. However, there are still two main challenges for practical application:

Challenge 1: How to define a proper action space for DRL in our problem. The action space is the most concerning part when we apply DRL to practice. In previous work [11], the decision space can be directly converted to action space as it is small and discrete. However, in our problem, the decision to be made is the configuration ratio for requests among m CDN providers in the next time step, which implies that direct mapping from decision space to action space will make the action space too large.

Challenge 2: How to solve "safe exploration" problem. To enhance DRL model, all DRL agents need to sometimes engage in exploration, i.e., taking actions that don't seem optimal in a given state, but help the agent learn about its environment. In previous work, it is usually inexpensive to online train their DRL models. For example, in Atari video games, training a DRL agent needs nothing but a computer and a camera, and there's a limit to how bad the exploration is: getting a low score or losing the game. However, in our problem, it is much less forgiving to train a DRL model in a real live streaming environment, since the chosen poor actions during exploration will make viewers suffer unavoidable QoE degradation, and cause economic losses. In summary, the main contributions of this paper are presented as follows:

1. **Identifying the inefficiencies of scheduling strategy today:** We use a week of real-world live streaming data to profile the viewer dynamics of CLS, after which we use a case study to identify the inefficiencies in nowadays scheduling strategy. The measurement shows that there is a large room for improving the quality of services.

2. **Neural adaptive scheduling system for live streaming:** To well capture the unique features in CLS scheduling problem, we propose LTS (learn to schedule), a novel request scheduling approach based on DRL. In details, we adopt A3C [4], an advanced DRL framework, to solve our problem. Based on the above designs, LTS can make use of historical information and automatically improve itself without any presumptions (e.g., requests patterns).

3. **Data-driven offline simulator:** To alleviate the "exploration safety" problem in real-world training, we build up a realistic offline simulator to train RL agent, which allows us to validate LTS before online deployment and effectively avoid choosing the improper actions. Especially, by using state-of-the-art *Neural Arithmetic Logic Units*(NALU) [16], our simulator can well solve the problem of numerical extrapolation problem caused by data biases.

4. **Evaluating LTS in real-world traces:** Through the extensive evaluation in real-world traces, we find that LTS can significantly outperform state-of-the-art approaches. Specifically, by using a piece-wise linear simulator, which is toy but interpreting, we show that LTS can get almost the optimal solution.

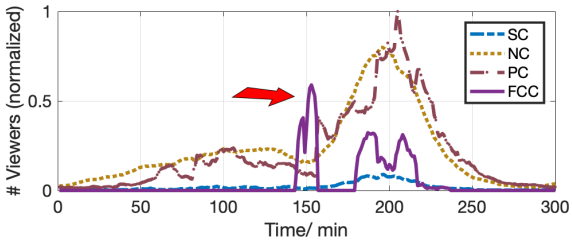


Figure 2: The dynamics of viewers

2 RELATED WORK

Multi-CDN selection: The details of how a particular CLS platform schedules viewers to different CDN providers are uncertain. However, preliminary measurements have alleviated that in most cases, the configuration ratio across different CDN providers is fixed regardless of the dynamics of either CDN or viewer patterns [1, 15]. In recent years, dynamic scheduling has received more attention. [7, 10] propose model-based methods which update their model through historical data and use the prediction of the performance of CDN provider to guide scheduling strategy. However, these methods are significantly affected by the accuracy of modeling and suffer performance degradation caused by data bias. To solve the above problems, [8] uses E2 method to replace traditional model-based methods, which now is the state-of-the-art.

Deep Reinforcement Learning: DRL is now the hot topic in both industry and academia. DQN proposed in [12] makes it possible for RL to learn directly from high dimensional inputs, which is a massive step forward from traditional RL [14]. Based on previous work, [4] extends one agent training to multiple training, and significantly fast the convergence time. Besides, DRL is also successfully applied to other data-driven controlling problem. [19] uses DRL to solve traffic engineering problem; [17, 18] apply DRL to resource management problem. In [11], authors use DRL to select the next time video bitrate. Motivated but different from them, our work is the first to apply DRL in CLS scheduling problem.

There is also some excellent work focusing on other aspects of CLS optimization such as video transcoding and new delivery architectures, and we recommend reader to [5, 20].

3 DATASET AND MOTIVATION

The data used in this paper were collected in cooperation with *Kuaishou*[1], the leading CLS platform in China. The dataset consists of over 500M view sessions from 50M viewers and 0.01M channels. Due to business consideration, we anonymize the name of CDN providers, provinces, and ISPs (internet service providers). As required by the data provider, we also use the normalized viewer number.

We first profile the viewer dynamics of CLS. For all channels, we define them into four different types: the *small channel* (SC), the *normal channel* (NC), the *popular channel* (PC) and the *flash crowd channel* (FCC). SC represents the channel which has a few viewers, while PC represents the popular channel with a lot of viewers. FCC identifies the channel with a large number of viewers arriving in a short period, and NC denotes the channels other than

[1] https://live.kuaishou.com/

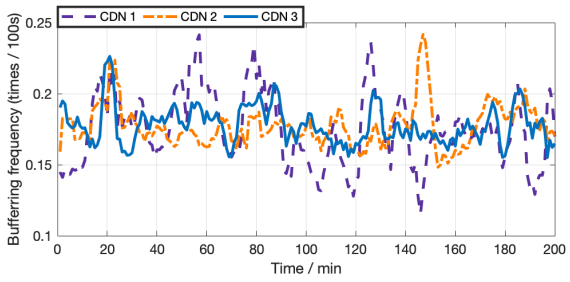


Figure 3: The performance dynamics of different CDN providers

the above three channel types. In details, we identify them based on two standards: 1) the growth rate of viewers within a particular time window. 2) the peak number of concurrent viewers. The first standard is aimed to identify the "burst event", while the second standard is to distinguish channel popularity. After considering all the channel data in our dataset, we classify the FCCs as those with more than 20% viewers growth within 10min and more than 5,000 peak viewers. The evolution of these four type channels is shown in Figure 2, which discloses the following two important observation: first, FCCs can significantly increase the total number of viewers. As pointed by the red arrow in Figure 2, we can see that compared with the peak number of NCs and PCs, the viewer number of FCCs can increase by more than 50% in about 10 mins; second, the burst of FCCs can happen during the off-peak period. As denoted, the burst time (at about 150 min in Figure 2) of FCCs is nearly one hour before the peak time (at about 200 min in Figure 2).

We also present the dynamics of different CDN providers in Figure 3. It should be noted that for the limitations of page space, we only present the average buffering frequency (the details of this metric is described in §4.3) as the performance estimation, and other metrics can lead to similar observations. As shown, we can see that at each time, their performance varies dramatically with time. As a result, to guarantee better experience, more viewers should be serv ed by the best CDN provider (e.g., with the lowest buffering frequency).

These extensive dynamics from viewers and CDN providers make traditional algorithms hard to cope with. We argue that a good CLS scheduling algorithm should make decisions with the inclusion of both viewer patterns and CDN provider performance, and any simplification of these dynamics will separate them from the optimal solution. Therefore, previous algorithms are unable to well solve the problem due to their fixed control laws. To alleviate these problems, we propose *Learn to Schedule* (LTS), a DRL-based approach to solve the CLS scheduling problem.

4 DESIGN OF LTS

We start with a brief introduction to RL: RL is a general framework which at first represents the problem into the *Markov decision process* (MDP), and then learns to make decisions according to the observation of previous transition data. A typical RL consists of following parts: the state space $\mathcal{S}$, the action space $\mathcal{A}$, the reward $\mathcal{R}$, and the policy π. At each time, after observing the state $s_t \in \mathcal{S}$, the RL agent will select an action $a_t \in \mathcal{A}$ based on the policy π, and then get a reward $r \in \mathcal{R}$. The policy π is what an RL agent needs to learn through the reward signal. Now, we would like to formulate CLS scheduling problem by defining its state $s \in \mathcal{S}$, action $a \in \mathcal{A}$ and reward R, and in this paper, we use A3C [4], which is one of the most widely-used actor-critic RL frameworks.

4.1 State

We consider m CDN providers for CLS scheduling with an episode horizon T minutes, and each time step lasts $\Delta t = 1$ min (also works with other Δt settings), which means that after T times decisions, LTS will get the episode terminal, and the system will be reset. In practice, the horizon T is set to be a single day.

At each time step t, since the state s_t should take all m CDN providers into consideration, we denote it as a vector:

$$\overrightarrow{s}_t = [\hat{s}_t^1, \hat{s}_t^2, \ldots, \hat{s}_t^i, \ldots, \hat{s}_t^m] \tag{1}$$

in which $\hat{s}_t^i$ stands for the state of the i-th CDN provider. Moreover, to allow LTS make use of historical information, we represent $\hat{s}_t^i$ as:

$$s_t^i = (\overrightarrow{w}_t^i, \overrightarrow{QoE}_t^i) \tag{2}$$

where the $\overrightarrow{w}_t^i$ is workload (i.e., the viewer number) of i-th CDN provider for past k time steps, and $\overrightarrow{QoE}_t^i$ stands for the mearsurements of past k QoE metrics considered by CLS platforms (e.g., start-up latency).

4.2 Action

The solution space in our problem is the configuration ratio among m CDN providers. However, different from existing DRL application problems which usually directly define the action space as the solution space (e.g., bitrate selection [11]) , the action space in our problem is continuous and too large to converge. To alleviate this problem, we decrease the LTS's action space by discretizing the solution space at intervals of 1% and design LTS's action in a heuristic way. For each CDN provider, there are 3 choices: increase its configuration ratio by 1%, 5%, and 10% on the basis of the ratio at the previous time step. This design enables LTS to schedule viewers through fine-grained action (i.e., 1%) to precisely adjust the configuration ratio, as well as allowing it accommodate to the dynamics responsively through coarse-grained action (i.e., 5% and 10%).

As a result, for m CDN providers, the action space will be $3 \times m$. At the same time, we also need an action to keep the configuration ratio unchanged, so the total action space will be $3 \times m + 1$. Different from the problem in cloud management [18], in which the action is defined to change the different parameters (e.g., CPU number and hard disk storage) independently, the adjustment of the configuration ratio in our problem should jointly consider all the CDN providers, as the sum of configuration ratio should be equal to 1. Therefore, after LTS decides to increase the ratio of one CDN provider, it also needs to select another CDN provider to decrease the corresponding part, and in this paper, we choose the worst CDN provider in the previous time step, since the workload of the worst provider is the most needed to readjust. For example, suppose there are a total of 3 CDN providers denoted as A, B, and C; the configuration ratio in the time step $t-1$ is (20%, 30%, 50%); the measurement shows that the QoE satisfies $QoE_C < QoE_B < QoE_A$ (i.e., C is the worst CDN provider). Then, if LTS selects the action

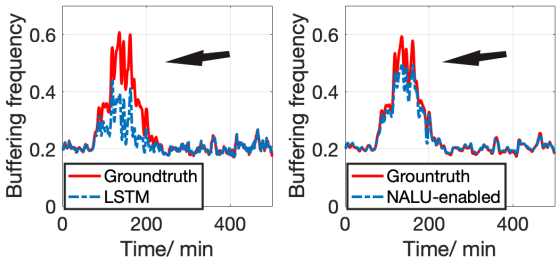


Figure 4: Offline simulator. Comparing with LSTM, our NALU-enabled neural network structure can well solve extrapolation problem.

increase A by 5%, the configuration in the time step t will become (25%, 30%, 45%). In this way, we reserve the relationship of different CDN providers without any pre-selection of the candidates and at the same time reduce the action space effectively.

4.3 Reward

After each action, LTS will get a reward r which reflects the control performance. In details, we will consider the reward based on a weighted sum of following industry-standard QoE metrics [3, 8]:

Startup latency *(SLatency)*: a second-level metric, which represents the duration between a viewer requests the session and the video player gets enough data in buffer to play.

Buffering Ratio *(BuffRatio)*: represented as the rate between the time of buffering and the time of a total session. A buffering event happens when there is no data in buffer, and the display of the content will be paused to wait for the next frame. To make this metric not too small in quantity, here we multiply the ratio with 100, which means the buffering time per 100 seconds.

Buffering frequency *(BuffFreq)*: represented as the average number of buffering events in a live streaming session, which can be calculated as $\frac{\#buffering\ events}{session\ time}$. Note that different from $BuffRatio$, which cares about the time spent in interruptions, $BuffFreq$ focuses on the frequency of the interruptions perceived by viewers. At the same time, we also scale it by multiplying 100, which means the buffering frequency per 100 seconds. Since all the above QoE metrics have negative impacts on viewer engagement [3], we use their negative forms to maximize the total reward. Thus the reward r is formulated as:

$$r(s_t, a_t) = -\alpha SLatency_t - \beta BuffRattio_t - \gamma BuffFreq_t \quad (3)$$

in which the α, β, γ represent the relative importance for CLS platforms. The objective for LTS is to find the policy π to maximize the accumulated reward $R = \sum_{t=1}^{T} r(\overrightarrow{s}_t, a_t)$.

The reward function depends heavily on the mapping of the pair (s_t, a_t) to QoE metrics, which ideally should be provided by the real-world CLS environment. However, training LTS from scratch in a real-world scenario is unacceptable: at the start of the training, it needs to explore some bad choices to learn the dynamics of the environment, but these poorly chosen actions will redirect viewers to improper CDN providers, which decreases overall QoE, thereby making the platform suffer economic losses. To overcome this "safe exploration" problem, we build up a simulator to offline train and evaluate LTS.

Methods	RMSE
ARMA	0.020
Linear Regression	0.064
LSTM	0.012
NALU-enabled LSTM	**0.006**

Table 1: Comparing NALU-enabled method with other widely-used methods using Root Mean Sqare Error (RMSE).

4.4 Offline simulator

Inspired by the recent success of deep neural networks (DNN) in stochastic timeseries forecast, we are determined to build up a DNN-based simulator and train it using supervised learning. However, since our dataset is collected under a fixed strategy used by CLS platforms, the data are biased, and the simulator will suffer frequent failures when the input values lie outside the numerical range used during training. Figure 4 shows an example, and we can find that the most widely used DNN structures (e.g., Long-Short-Time-Memory) cannot deal.

To tackle this problem, we use state-of-the-art *Neural Arithmetic Logic Units*(NALU) [16] module. By reconstructing the fundamental arithmetic operations (e.g., subtraction and multiplication) through a carefully designed network unit, NALU can well solve the extrapolation problem. As shown in Figure 4, our NALU-enabled simulator quite accurately fits real-world data. We also compare with other common time series prediction methods, such as *Auto Regressive Moving Average* (ARMA) and *Linear Regression*. The detailed comparison results can be obtained in Table 1.

5 EVALUATION

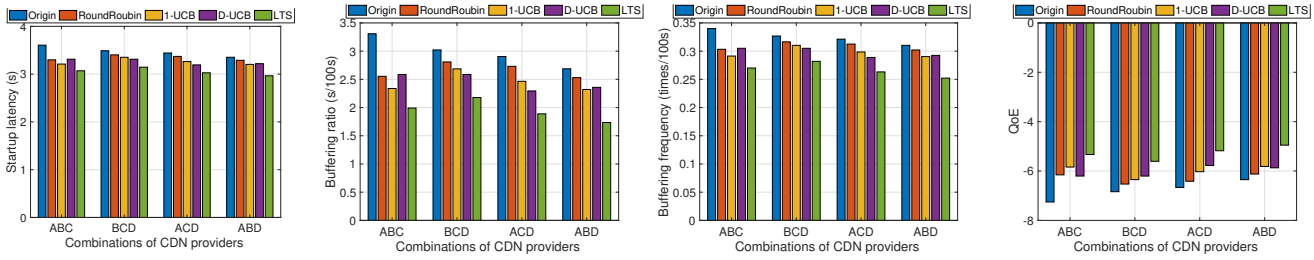
5.1 Methodology

Implementation: We start with providing the implementation details. The parameters for reward function is $(\alpha = 1, \beta = 1, \gamma = 1)$. We consider $m = 3$ CDN providers, which is a common configuration for CLS platforms [2]. At time step t, for each CDN provider i, LTS will input $k = 20$ past workload $\overrightarrow{w}_t^i$ and $\overrightarrow{QoE}_t^i$ measurements to a 2D convolution layer (CNN) with 64 filters, each of size 4 and stride 1. Then the outputs of these layers are aggregated in a hidden layer that uses 64 units to apply the *softmax* activation function. The critic network has the same network structure except for its final output which is a linear activation function. The learning rates for actor and critic networks are 0.0001 and 0.001 respectively. The discounted factor is 0.99 and minibatch size is 20. The simulator has two LSTM layers of size 128 and 64, and then cascaded by a NALU-enabled layer. Training details of A3C can be found in [4, 11].

The experiments are conducted on real-world CLS data, spanning one week (6 days for training and 1 day for test). At each time, we select 3 candidates from a total of 4 different CDN providers, and we fit a separate simulator for each of them.

Baseline algorithms: We compare our DRL-based framework with the following baselines:

- The original algorithm: the strategy used by the data provider. Actually, we don't exactly know its strategy, but the result of which can be directly obtained from collected data.
- Best weighted round-robin: At each time, the requests are redirected to different CDN providers in a static ratio. This algorithm is called *Weighted round-robin* [1] which is the



(a) Startup latency (b) Buffering ratio (c) Buffering frequency (d) Average QoE

Figure 5: Comparison results. The Evaluation on real-world traces shows that LTS outperforms state-of-the-art algorithm by 8.71%-15.63% on QoE. Especially, LTS consistently outperforms baselines on all three QoE metrics.

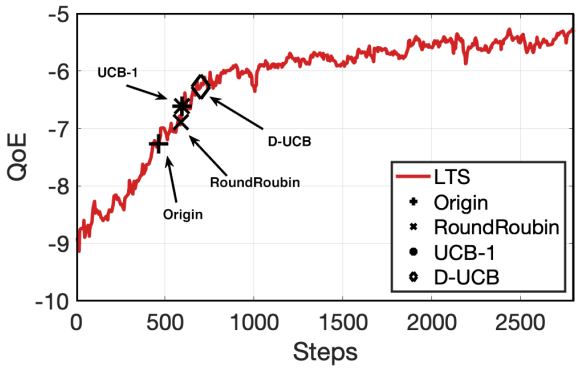


Figure 6: Converge of LTS. We can see that LTS exceeds state-of-the-art baseline in less than 700 steps.

most widely used technique for solving this problem. Remarkably, in our experiment, we use the grid search to find the best distribution ratio. In other words, we use the "best" weighted round-robin algorithm.

- Exploitation and Exploration (E2) algorithm: a state-of-the-art algorithm applied by [8]. At each time, E2 algorithm will use the average-like method to estimate CDN provider performance, and choose the one with the highest upper confidence bound of reward, which will naturally choose the CDN provider with high expected performance or high uncertainty. In our experiment, we use UCB-1 [14] and Discounted-UCB [6], both of which are using moving average to estimate the expected performance, and the difference is the latter gives more weight to more recent measurements.

5.2 Results and discussion

The results are presented in Figure 5, and we can find the following two key observations. First, comparing with the baseline algorithms, LTS exceeds the performance of the best existing algorithms. For example, in combination (CDN provider A, CDN provider B, CDN provider C), which is denoted '**ABC**', LTS outperforms the four baselines by 26.52%, 13.40%, 8.71%, 14.08% respectively in average QoE. Since *the original algorithm* (the blue bar) performs the worst, we guess that the platform still uses the "round-roubin"-like algorithm, namely scheduling viewers to different CDN providers according to a static distribution ratio. Notably, after comparing with *the origin algorithm* with *the best weighted round-roubin* (the orange bar), we find that even for static methods, the performance can vary significantly.

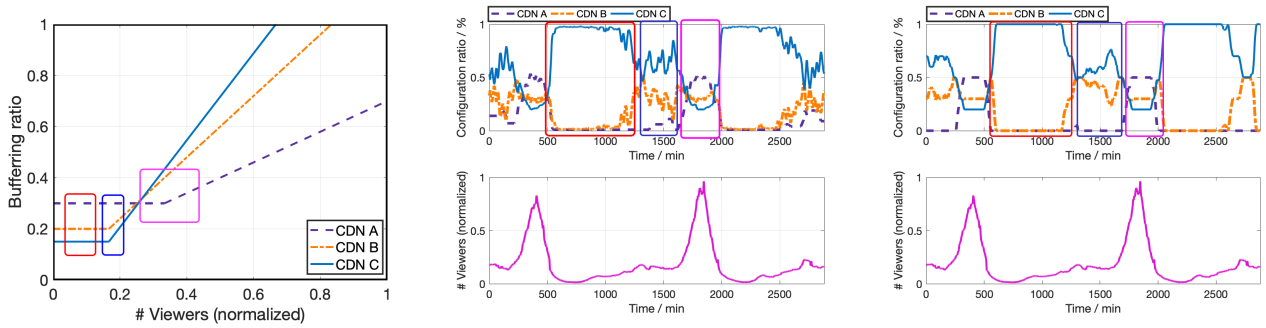
However, these *static* methods still perform unsatisfactorily. The reason is obvious: they ignore the dynamics of the viewers, and cannot capture the performance changes of different CDN providers. For UCB-1 and Discounted-UCB, a critical limitation also separates them from the optimal solution: since they use a straightforward way to estimate the changes of either workload patterns or CDN provider performance (e.g., moving average), they cannot well utilize the historical information. Meanwhile, the ignorance of future dynamics also makes them hard to adapt to the request dynamics responsively. With the help of the deep neural network, LTS can automatically learn the environment dynamics and act accordingly, thus outperforming all baseline algorithms.

Second, we observe that the performance of existing algorithms struggles for different CDN provider combinations. For example, the best weighted round-robin algorithm is the best baseline for the combination **ABC**, while for **BCD**, the best becomes D-UCB. The reason is that these algorithms make decisions in fixed control laws, however for different CDN providers, the strategy should be inherently different: for the small CDN providers, since the performance of them is less stable and more sensitive to viewer numbers, the strategy should be much more conservative to avoid violating the viewers served by them. LTS can capture the dynamics of both viewers and CDN providers and thus, performance with LTS remains consistently high for different CDN combinations.

Besides, we also present the training process in Figure 6. As shown, LTS outperforms the original algorithm in less than 500 steps and exceeds the best D-UCB algorithm in less than 700 steps. In our experiment, LTS converges in less than 3000 epochs, which needs about 3 hours.

5.3 Validation through a toy model

Although the NALU-enabled simulator can well capture the dynamics of CDN performance, the complex network structure makes the prediction results of the simulator hard to understand and incapable of obtaining the optimal solution. To better validate the effectiveness of LTS, we replace the NALU-enabled simulator with a less precise but more explainable one. In details, as presented in [10], CDN performance (e.g., buffering ratio) and the number of serving people (workload) are highly correlated, and the relationship can be approximated by a piecewise function: when the workload exceeds a certain threshold, the CDN performance will degrade



(a) Piecewise linear model (b) The strategy of LTS (c) The strategy of the optimal algorithm

Figure 7: The result of linear piecewise simulator. To better understand the effectiveness of LTS, we use the piecewise linear function as the core of the offline simulator, which is less precise but more explainable. The results show that LTS can get almost the optimal solution.

significantly. For simplicity, we use a piecewise linear model to characterize this relationship and use the greedy search at each time to get the optimal solution.

Note that the network structure of LTS remains unchanged, and LTS has no knowledge about the linearity of our simulator. We present the decisions made by LTS every moment in Figure 7. Figure 7(a) shows the linear model used; Figure 7(b) and Figure 7(c) show the decisions made by LTS and the offline optimal, respectively.

As expected, we can find that LTS can generate almost the optimal strategies in the piecewise linear model. Note the three enclosed parts in Figure 7(a), each of them represents the different environment state. The first part represents the state when total workload is light, and as shown in Figure 7(c), during this period, the best strategy is to schedule as many viewers as possible to the best CDN provider, namely CDN C. The second part represents the state when total workload increases to a middle level, and at this time, redirecting total requests to CDN C will cause its buffering ratio increase significantly. Therefore, the best choice is to ask another CDN provider for help. Since CDN B performs better than CDN A, just like the decision made by the offline optimal, we should choose CDN B. The last part shows the state when the total workload is heavy. At this time, CDN A will perform better than CDN B and CDN C, so more requests should be scheduled to CDN A. Encouragingly, as shown in Figure 7(b), LTS can capture the three pivotal states and make almost the optimal decisions only through raw data without any previous knowledge and presumptions.

6 CONCLUSION

In this paper, we propose LTS, a deep reinforcement learning based approach, to dynamically schedule viewers for CLS. Unlike previous algorithms which use imprecise models or naive strategies, LTS can generate scheduling decisions through raw input data and adapt itself to the environment. Through extensive trace-driven evaluations, we demonstrate LTS outperform the best existing algorithms significantly.

Acknowledgment. This work was done in close cooperation with Kuaishou Technology Co., Ltd., and was part-funded by the National Natural Science Foundation of China (No.61521002), Beijing Key Laboratory of Networked Multimedia (No.Z161100005016051) and Key Research and Development Project (No. 2018YFB1003703).

REFERENCES

[1] Vijay Kumar Adhikari and et al. 2012. Unreeling netflix: Understanding and improving multi-cdn movie delivery. In *INFOCOM, 2012 Proceedings IEEE*. IEEE, 1620–1628.
[2] Vijay Kumar Adhikari et al. 2012. A tale of three CDNs: An active measurement study of Hulu and its CDNs. In *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*. IEEE, 7–12.
[3] Florin Dobrian and et al. 2011. Understanding the impact of video quality on user engagement. In *ACM SIGCOMM Computer Communication Review*, Vol. 41. ACM, 362–373.
[4] Mnih et al. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*. 1928–1937.
[5] Chen Fei et al. 2015. Cloud-assisted live streaming for crowdsourced multimedia content. *IEEE Transactions on Multimedia* 17, 9 (2015), 1471–1483.
[6] Aurélien Garivier and Eric Moulines. 2008. On upper-confidence bound policies for non-stationary bandit problems. *arXiv preprint arXiv:0805.3415* (2008).
[7] Junchen Jiang et al. 2016. CFA: A Practical Prediction System for Video QoE Optimization.. In *NSDI*. 137–150.
[8] Junchen Jiang et al. 2017. Pytheas: Enabling Data-Driven Quality of Experience Optimization Using Group-Based Exploration-Exploitation.. In *NSDI*, Vol. 1. 3.
[9] Jessica Klein. 2018. Twitch Ended 2017 With 15 Million Daily Visitors, 27K Partnered Streamers. https://www.tubefilter.com/2018/02/06/twitch-2017-year-in-review/. Accessed February 6, 2018.
[10] Xi Liu et al. 2012. A case for a coordinated internet video control plane. *ACM SIGCOMM Computer Communication Review* 42, 4 (2012), 359–370.
[11] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 197–210.
[12] Volodymyr Mnih et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.
[13] Mark Stemm and et al. 2000. A network measurement architecture for adaptive applications. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, Vol. 1. IEEE, 285–294.
[14] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.
[15] Ruben Torres et al. 2011. Dissecting video server selection strategies in the youtube cdn. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*. IEEE, 248–257.
[16] Andrew Trask et al. 2018. Neural arithmetic logic units. In *Advances in Neural Information Processing Systems*. 8046–8055.
[17] Yifei Wei and et al. 2018. User Scheduling and Resource Allocation in HetNets With Hybrid Energy Supply: An Actor-Critic Reinforcement Learning Approach. *IEEE Transactions on Wireless Communications* 17, 1 (2018), 680–692.
[18] Cheng-Zhong Xu, Jia Rao, and Xiangping Bu. 2012. URL: A unified reinforcement learning approach for autonomic cloud management. *J. Parallel and Distrib. Comput.* 72, 2 (2012), 95–105.
[19] Zhiyuan Xu and et al. 2018. Experience-driven networking: A deep reinforcement learning based approach. *arXiv preprint arXiv:1801.05757* (2018).
[20] Zhu Yifei and et al. 2017. When cloud meets uncertain crowd: An auction approach for crowdsourced livecast transcoding. In *Proceedings of the 25th ACM international conference on Multimedia*. ACM, 1372–1380.