Livesmart: a QoS-Guaranteed Cost-Minimum Framework of Viewer Scheduling for Crowdsourced Live Streaming

Rui-Xiao Zhang*, Ming Ma‡+, Tianchi Huang*, Haitian Pang*, Xin Yao*

Chenglei Wu*, Jiangchuan Liu†, Lifeng Sun§+

* Department of Computer Science and Technology, Tsinghua University
§ BNRist, Department of Computer Science and Technology, Tsinghua University
‡ Beijing Kuaishou Technology Co., Ltd., China
† Simon Fraser University
* {zhangrx17,htc19,pht14,wucl18,yaox16}@mails.tsinghua.edu.cn
maming@kuaishou.com, jcliu@cs.sfu.ca, sunlf@tsinghua.edu.cn

ABSTRACT

Viewer scheduling among different CDN providers in crowdsourced live streaming (CLS) service is especially challenging due to the large-scale dynamic viewers as well as the time-variant performance of the content delivery network. A practical scheduling method should tackle the following challenges: 1) accurate modeling of viewer patterns and CDN performance; 2) intelligent workload offloading to save costs while guaranteeing the quality of service (QoS); 3) and ease of integration with practical CDN infrastructure in CLS platforms.

In this paper, we propose Livesmart, a novel framework that facilitates a QoS-guaranteed cost-efficient approach for CLS services. Specifically, we address the first challenge by carefully designing deep neural networks which make Livestream capture the environment dynamics without any presumptions; we then tackle the second challenge by leveraging the Model Predictive Control (MPC) method which enables Livesmart to make decisions in a long-term way. For the last challenge, we propose a probability shift model based on the realistic CLS delivery structure, thus empowering Livesmart to be practically deployed. We collect real-world data in cooperation with Kuaishou, one of the largest CLS platform in China, and evaluate Livesmart with trace-driven experiments. Comparing with prevalent methods, Livesmart can significantly reduce the CDN bandwidth costs (24.97%-63.45%) and improve the average QoS (5.79%-7.63%).

CCS CONCEPTS

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

KEYWORDS

content delivery networks, crowdsourced live streaming, neural networks

ACM Reference Format:
Rui-Xiao Zhang*, Ming Ma‡+, Tianchi Huang*, Haitian Pang*, Xin Yao* and Chenglei Wu*, Jiangchuan Liu†, Lifeng Sun§+. 2019. Livesmart: a QoS-Guaranteed Cost-Minimum Framework of Viewer Scheduling for Crowdsourced Live Streaming. In *Proceedings of the 27th ACM International Conference on Multimedia (MM '19), October 21–25, 2019, Nice, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3343031.3351013

1 INTRODUCTION

Recent years have witnessed great development in crowdsourced live streaming (CLS). In CLS scenarios, any general user can broadcast his/her contents to numerous viewers, and viewers will enjoy the streaming entertainment with different devices (a Mobile phone, an iPad, or a personal computer). Many platforms like Youtube.game, Twitch.tv and Kuaishou.com have shown unprecedented growth across the world.

To guarantee the viewer engagement, content delivery network (CDN) has been a fundamental support network structure for CLS platforms [2]. Moreover, since one single CDN may suffer performance fluctuation, multiple CDN (multi-CDN) structure has been accepted by more leading platforms [3, 4]. However, the emerging demand for live contents has presented enormous challenges to CLS platforms: on the one hand, since the performance of different CDN providers are both time-variant and workload-related (§3.1), platforms need to dynamically schedule viewers to satisfy the growing demand for higher quality of viewing experience; on the other hand, platforms also need to deliver videos in a more cost-efficient way to make more profits. To guarantee QoS while trimming costs, how to intelligently schedule users to highly dynamic CDN providers has been a critical problem for platforms.

Different from well-prepared traditional live streaming (such as TV channels and sports), the following features make CLS scheduling more complicated. First, viewers in CLS scenario are equipped with more randomization: any broadcaster can suddenly become a celebrity, which can cause a burst of viewers watching him/her (also denoted as flash crowd [23]). Such a highly dynamic viewing pattern, if not well identified, can result in both cost-inefficiency

+ Lifeng Sun and Ming Ma are the corresponding authors.

MM '19, October 21–25, 2019, Nice, France

ACM ISBN 978-1-4503-6889-6/19/10…$15.00
https://doi.org/10.1145/3343031.3351013

and QoS degradation (§3.2). Second, the viewers are much more sensitive to delay and jitter: CLS viewers not only watch the streaming but also participate by interacting with broadcasters [17, 23]. Due to that switching CDN can lead to viewer side buffer and even failure (e.g., DNS resolution failure or HTTP redirection failure), this actually requires that the platform shouldn't proactively reschedule the viewers who are being served and can only control the new viewers.

There have been abundant approaches proposed in recent years focusing on CLS delivery optimization, which unfortunately, cannot handle all the above features. On one hand, most of them only focus on improving user engagement regardless of delivering costs [11, 12], which inevitably results in economic inefficiency. On the other hand, some studies schedule viewers by applying simplified environment settings (e.g., ignoring the temporal dynamics of CDNs and viewers), which fails to get the optimal decision under the real-world conditions [13, 21]. Moreover, since neither of the previous methods pays attention to the unschedulability of the remaining viewers featured by CLS services, they cannot be directly deployed by CLS platforms.

To tackle the above problems, we propose Livesmart, a novel framework that focuses on viewer scheduling for CLS platforms. Different from previous work, Livesmart is enabled with the following properties.

Accurate dynamics modeling: Previous work is based on static network settings, while it is dynamic in reality. So what is essential is to describe the dynamics from both viewer side and CDN side. To tackle this challenge, we propose to use deep neural networks (DNNs) to characterize the dynamics of them. Specifically, since DNN can predict with almost raw input data, our models can well accommodate the heterogeneity among different CDN providers and also different viewer patterns. Moreover, inspired by the similarity between the "data-extra prediction" problem and CLS viewer prediction problem, we propose to use an advanced neuron unit to build up DNN models. By carefully designing the DNN structure, we demonstrate that our model works well.

Effective and economical scheduling: Building on insights from the system control theory, we formulate the QoS-guarantee cost-minimum problem, and propose that model predictive control (MPC) [18] is a suitable algorithm that can optimally combine the prediction results of prediction models and the feedback signals of each decision. Specifically, after receiving a series of decisions, Livesmart predicts the expected QoS and costs for the next few time steps and uses them to generate an optimal decision.

Ease of deployment: Instead of modifying the existing delivering structure, or relying on additional technique support from CDN providers and CLS platforms, Livesmart is designed to be seamlessly integrated into real-world CLS platforms by modeling remaining viewers through a probability-based dynamic shift model.

In summary, the main contributions of this paper are three-fold: 1) We conduct large-scale measurement to get insights into CLS viewer scheduling. Specifically, to the best of our knowledge, we are the first to concern the influence of remaining viewers in CLS services. 2) We propose Livesmart, a novel framework that focuses on the CLS viewer scheduling. By harmonizing the mathematical methods and deep learning methods, Livesmart is both effective and deployable. 3) Through extensive trace-driven experiments, we demonstrate that Livesmart can significantly reduce the costs (24.97%-63.47%) and improve the QoS (5.79%-7.63%) at the same time.

2 RELATED WORK

With the growing popularity among users, how to improve viewer engagement in CLS scenarios has received great interests. Many of them focus on system optimization. For example, [23] proposes a network optimizer which can collaborate the CDN and edge devices. [5, 7, 9, 20] facilitate some cloud-assisted frameworks which optimize the content delivery by adaptively leveraging cloud services. At the same time, some of them also focus on other aspects. For example, [22, 26] concern about transcoding efficiency, and [16] aims at optimizing the first-mile transmission in CLS scenarios. However, due to all of them focus on the new delivery structure which needs additional technique support from CDN providers or platforms, they cannot be directly used in nowadays CLS services.

At the same time, multi-CDN has been a fundamental structure for nowadays video delivery platforms [2]. Previous work has shown that in most cases, the configuration ratio across different CDN providers is fixed regardless of the dynamics from CDN providers or viewers. To alleviate these problems, a lot of work has focused on dynamic viewer scheduling. For example, [3, 4, 11, 14] propose model-based methods which update their model through historical data and use the prediction of CDN performance to guide scheduling strategy, while [12] uses exploration-exploitation method to online measure CDN performance and schedule viewers in a real-time way. However, due to that these methods only consider optimizing the viewer engagement, they can result in economic inefficiency.

Although some work indeed considers the CDN performance and delivery costs, they all simplify the environment settings which may result in suboptimal solutions and separates them from practical deployment. For example, [13] ignores the time-series dynamics of CDN providers, while [21] overlooks the influence of flash crowds, and neither of them pays attention to the impact of remaining viewers. Different from them, Livesmart systematically considers the dynamics from CDNs and viewers. Specifically, by modeling the remaining viewers, Livesmart can be seamlessly integrated into nowadays CLS delivery structure. Besides, our method can also be extended to the cloud provider selection problem [15], in which both the job workload and cloud performance are dynamic.

3 MOTIVATION

3.1 The dynamics of CDN providers

We first profile the dynamics of different CDN providers, and we select out the data from the same group[1]. We take *stalling frequency* as the performance evaluation metric. (For paper consistence, the data used in this paper are described in 6.1) The definition of *stalling frequency* is the average number of stalling events[2] experienced by viewers, and as it increases, the QoS degrades. We first measure the average stalling frequency of each CDN in the process of time, and present the results in the upper part of Figure 1. As shown,

[1] Same group means the viewers/CDN in the same region and use same ISP (Internet Service Provider)

[2] A stalling event happens only when there is no data in playback buffer, and the video has to be paused until the next frame.

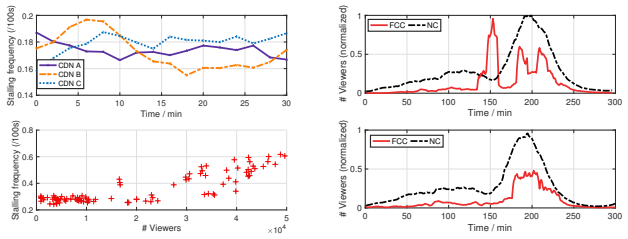


Figure 1: The dynamics of CDN providers

Figure 2: Viewer access pattern

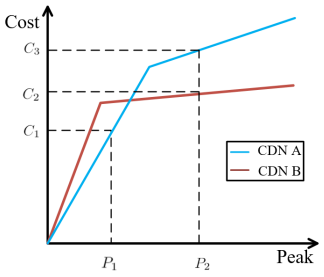


Figure 3: Cost-inefficiency

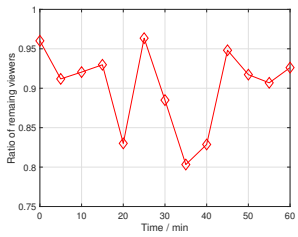


Figure 4: Ratio of remaining viewers

we can see that despite the same group, the QoS of different CDN providers fluctuates drastically over time. For example, during [0 min, 15 min], the best CDN provider switches from CDN C to CDN A and finally to CDN B. At the same time, we also observe that different CDN providers also have different patterns. For example, CDN B fluctuates most violently since it has the largest performance range (from 0.158 to 0.194), while CDN A and CDN C are more stable. Finally, we observe that the QoS is related to the workload scheduled to it. The lower part of Figure 1 shows an example, and we can see that with the number of concurrent viewers increases, the average stalling frequency first keeps stable and then increases significantly.

The above observations demonstrate that the CDN provider performance is both time-variant and workload-related, and there is also heterogeneity between different CDN providers. This can be explained as different CDN providers have different offloading policies and server settings. Existing methods prevalently simplify these dynamics and may cause experience degradation when scheduling viewers.

3.2 The dynamics of CLS viewers

Viewer access pattern We also investigate the viewer access patterns in CLS scenarios. Similar to [23], we first divide all streaming channels into two categories: the *Flash Crowded Channel* (FCC), which represents the channel with a lot of arrivals in a short period, and the *Normal Channel* (NC), which denotes other channels except for FCC. We identify the FCC from NC according to 1) the peak viewer number and 2) the peak growth rate. We present the evolution of them in Figure 2 (as required by the data provider, we normalize the viewer number), from which we have the following observations. First, the crowds can significantly improve the total viewer number. In the upper part of Figure 2, at about 150 min, the viewer number of FCC takes up almost half of the overall workload. Second, the crowds can appear at the off-peak time. For example, also in the upper part of Figure 2, the FCC peak appears at 150 min, while the peak load is at about 200 min. At the same time, in the lower part of Figure 2, we also present the workload evolution in the same period of another day. We can observe that in contrast with the stable pattern of NC, even the same period, the pattern of FCC also varies in different days. The above phenomena can be explained: the most popular broadcaster can stream online at any time of a day, therefore making the crowds appear irregularly.

Actually, these surging workload patterns will significantly influence the outcome of the scheduling decisions. For example, if we don't identify the coming crowd and still regard it as a regular pattern, the scheduling strategy may schedule as much as workload to the best-performance CDN to maximize the average QoS. However, as denoted in Figure 1, too many viewers will overload the CDN and violate its QoS, and the viewers scheduled to it will therefore suffer experience degradation. In addition, the failure of identifying the future workload can also result in the cost-inefficiency: Figure 3 shows an example: there are two CDN providers (denoted as CDN A and CDN B), and if we regard the future workload is P_1, then the most economical way is to schedule all viewers to CDN A. However the real workload is P_2, and apparently scheduling all viewers to CDN A will extend its actual costs to C_3, which is less economical than scheduling all viewers to CDN B (C_2).

The influence of remaining viewers: To practically deploy a scheduling framework for CLS platform, we can only control the new viewers. Suppose a viewer is watching the session through a certain CDN, if the platform switches the CDN at this time, it will inevitably cause the streaming data interrupted or delayed. Since viewers in live broadcasting are much more intolerant of the delay and buffering[17], the real world platform will not proactively reschedule those viewers who are enjoying the streaming and only schedule the new viewers. This actually complicates the scheduling problem due to the following reasons. First, we cannot schedule viewers only according the future one step performance prediction: for example, at time step t, if we have known that CDN A performs the best at time $t+1$, then we assign viewers to CDN A at time step $t+1$. However, from $t+1$ on, these viewers become the remaining viewers and suffer engagement degradation if CDN A becomes the worst in the next time steps. Second, we should consider the remaining viewers if we want to precisely control the number of concurrent viewers at each time step. This is important since both the QoS (denoted in Figure 1) and costs (denoted in Figure 3) are related to the concurrent workload instead of only new viewers. Figure 4 shows an example. We can see that at the different time, the remaining viewers take up a different ratio of total viewers. As a result, we need to separate the new viewers from the remaining viewers and consider the dynamics from both of them.

From the above analysis, we are motivated that an effective scheduling approach for CLS should consider both the dynamics from CDN providers and viewers. Especially, to make it more practical, the approach should also make decisions with consideration of remaining viewers.

4 FORMULATION

Recall that our goal is to minimize the bandwidth costs of CDNs while guaranteeing the QoS. The formulation consists of two parts: the costs calculation and the QoS guarantee.

Cost calculation: Before calculating the overall costs, we first give a brief introduction to the CDN pricing model. The pioneering CDN providers such as Amazon E2, and Tencent CDN offer their charging policies based on the following two strategies:

- **volume-based charging**. The charging policy is based on the volume of total viewers, generated from a group of viewers during a certain period (e.g., one month);
- **peak-based charging**. The charging policy relies on the peak workload of a group of viewers during a certain period.

At the same time, the charging policy also gives the platform a quantitative discount, i.e., the larger the volume (the peak), the lower the unit price you get. Specifically, since most platforms adopt peak-based charging schemes [25], in this paper we also assume that all CDN providers use peak-based charging policies (It is worth noting that our method can still work in volume-based charging).

The charging period is set as T (e.g., a month). For CDN providers, we suppose there are total N CDN providers, and the i-the CDN provider is denoted as CDN_i. The overall costs of a platform are associated with the peak load (or the volume) it consumes, and we define the workload of CDN_i at time step t is w_t^i. At the same time, we represent the charging model of CDN_i as $cost_i$, which is a function of peak load and in general is piecewise and non-decreasing [10]. For convenience, we denote the peak at time t of CDN_i as S_t^i, which can be updated as:

$$S_t^i = \begin{cases} S_{t-1}^i & S_{t-1}^i \geq w_t^i \\ w_t^i & S_{t-1}^i < w_t^i \end{cases} \tag{1}$$

Let x_t^i denote the fraction of *new viewers* distributed to CDN_i. Therefore, the variable x_t^i is the optimization variable, and at each time, the workload w_t^i is updated as:

$$w_t^i = w_t^{i,re} + w_t^{new} * x_t^i \tag{2}$$

in which w_t^{new} denotes the number of all new viewers at time t. Eq. 2 means that at any time, w_t^i are composed of the *remaining viewers* $w_t^{re,i}$ and new viewers scheduled to it. To this end, the costs function will be defined as:

$$C = \sum_{i=1}^{N} cost_i(S_T^i) \tag{3}$$

QoS guarantee: As the viewer engagement is paramount to the platform, the scheduling decision should at the same time guarantee the QoS requirement. In details, we represent the requirement by introducing a threshold QoS_{target} set by the platform. For example, the QoS can be the stalling frequency, startup latency[3] or the combination of them. Notably, QoS is both time-variant and workload-related, and different CDN providers have different QoS patterns (Figure 1), so we denote QoS as $QoS_t^i(t, w_t^i)$.

[3]a second-level metric, which represents the duration between a viewer requesting the session and the video player getting enough data to play.

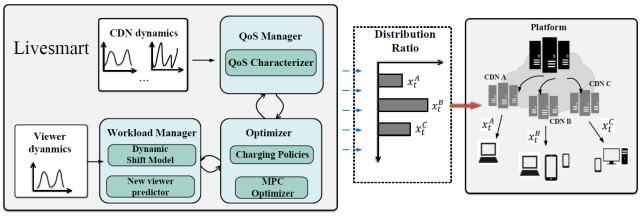


Figure 5: The system architecture of Livesmart. At each time t, Livesmart will characterize the dynamics from CDN and viewers, and then select the most proper distribution ratio of different CDN providers (i.e., x_t^a, x_t^b, x_t^c)

To this end, we have formulated the QoS-guaranteed cost-minimum optimization problem, in which the objective is to minimize the overall usage costs of CDN providers:

$$\min C = \sum_{i=1}^{N} cost_i(S_T^i) \tag{4}$$

$$s.t. \quad \begin{cases} \frac{1}{T*N} \sum_{i=1}^{N} \sum_{t=1}^{T} QoS_t^i(t, w_t^i) \geq QoS_{target} \\ \sum_{i=1}^{N} x_t^i = 1, x_t^i \geq 0 \\ (1), (2) \end{cases} \tag{5}$$

To solve the optimization problem, we should (1) characterize the dynamics from both CDN providers and viewers due to that QoS is time-variant and workload-correlated, and (2) make decisions through a long-term plan since there are remaining viewers (Eq.(2)) which influence the outcomes of the decisions (§3.2).

5 SYSTEM OVERVIEW

The components of Livesmart are depicted in Figure 5. As shown, Livesmart consists of three parts: the ***Workload Manager***, the ***QoS Manager***, and the ***Optimizer***. At each time step t, given the action x_t^i (i.e., the distribution ratio between different CDN providers), the workload manager can model the viewer dynamics and output the future workload, which contains both new viewers (through new viewer predictor) and remaining viewers (through dynamic shift model). Then, after receiving a series of actions, the QoS manager will predict the corresponding QoS (through QoS characterizer). Finally, The Optimizer is responsible for selecting the optimal distribution ratio considering the charging policies through MPC optimizer.

5.1 Workload Manager

In this part, we develop a practical model to represent the dynamics of CLS viewers. Inspired by [24], in which a probability model is proposed to represent the mobile shift, we are also determined to use the historical data to describe the temporal shift of viewers to distinguish the remaining viewers and a deep neural network (DNN) to predict the new viewers.

Dynamic shift model: We first present how to distinguish the remaining viewers. To better illustrate the key idea of the model, we present an example in Figure 6. As can be seen, the remaining viewers of CDN_i at t_0 are classified into two parts, the new viewers

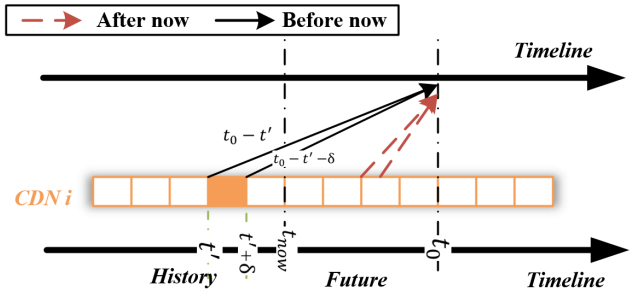


Figure 6: The core design of dynamic shift mode.

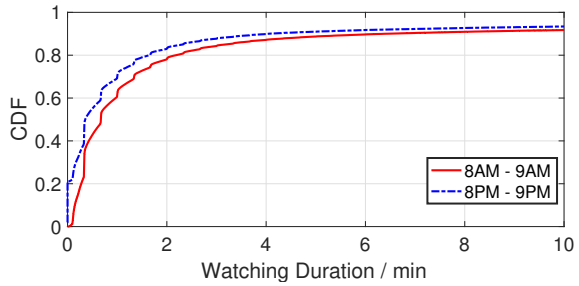


Figure 7: The CDF of watching duration in different period.

1) before t_{now} (solid line) and 2) after t_{now} (dash line). We first modele how many remaining viewers at t_0 are those who come before t_{now}. For those after t_{now}, we can still use the same approach after estimating the viewers between $[t_{now}, t]$.

The staying duration is a key factor, as the viewers may depart out of the target time step if the staying duration is too small. Therefor, we use the variable P_t to denote the probability that the viewers remaining at the target time, and the remaining viewers can be formulated as

$$w^{i,re} = w^{i,new} * P_t \tag{6}$$

It is notable that P_t is not only related to the target period but also relates to the joining time. For example, we set the target time as t_0, then the appropriate staying duration for the new viewers allocated to CDN_i at $t_0 - 10$ min should satisfy $t > 10$ min, while for those who are allocated at $t_0 - 20$ min, the feasible duration is $t > 20$ min. To address this problem, we develop the following approximation method.

As shown in Figure 6, we first discretize the time (e.g., 1 min) with interval δ. We start by studying the new viewers allocated to CDN_i during $[t', t' + \delta]$, if they are still in CDN_i in target time t_0, the feasible duration should be $t > t_0 - t'$. Therefore, we can get the number of remaining viewers at target time t_0 by accumulating all the remaining viewers generated at each time interval before t_0:

$$w^{i,re}_{t_0} = \sum_{k=1}^{\infty} w^{i,new}_{t_k} P(t \geq t_0 - t_k) \tag{7}$$

in which t_k represents the $k-th$ interval before t_{now}, i.e., $t_k = t_{now} - k\delta$. It is notable that both $P(t \geq t_0 - t_k)$ and $w^{i,new}_{t_k}$ can be easily obtained from historical data.

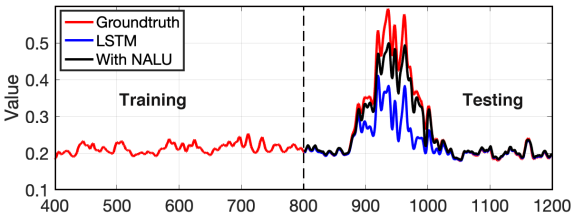


Figure 8: A case study for data-extra problem. As denoted, the model with NALU works better.

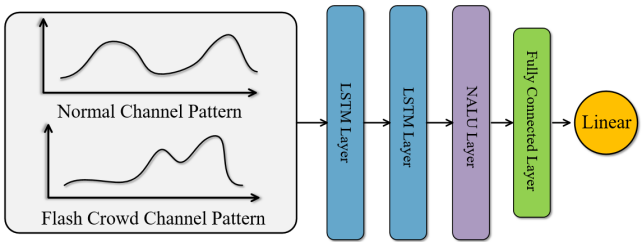


Figure 9: New viewer predictor

In fact, there is no need to calculate k from 0 to ∞. After the measurement from our collected data, we find that the new viewers usually keep staying one CDN for a short time, which indicates that we can perform a temporal pruning to cut off the k.

We show the example in Figure 7, in which we present the cumulative distribution function (CDF) of watching duration for viewers. A straightful thought is to dynamically set k, which, however, may lead to more computation overhead. So we are determined to use a fixed k. As depicted, we find that about 90% of viewers will depart within 10 min. Therefore, we only calculate Eq. (7) by setting k as 10 min. Remarkably, the staying duration is also correlated with the period of the day (denoted in Figure 7). It is reasonable since there will be more broadcasters in the evening, and viewers therefore have more alternatives and are more likely to switch among them. Therefor, we need to update the model every period, and in our experiment, we update it every 1 hour.

New viewer predictor: After that, we need to estimate the new viewers between $[t_{now}, t]$, i.e., new viewer prediction. Specifically, a good viewer prediction method need to address following two important problems: 1) the ability of generalizing to different viewer patterns, and 2) the ability to predict "unexpected" events.

To tackle the first problem, we are determined to use the DNN-based prediction model. Instead of using any handcraft feature engineering and data pre-processing, DNN can make prediction by using almost "raw" data and thus being able to better generalize to different input patterns.

For the second problem, it requires that the prediction model can be empowered to foresee some unexpected events, or in other words, the abnormal events. Back to the neural networks, it means that the DNN model is expected to work still when inputting the data which it seldom encounters in the training set. Traditional DNN structures are incapable of addressing the second problem. Figure 8 shows an example. We can see that long-short-term-memory (LSTM) network, which is the most widely used DNN structure for time series

prediction, suffer performance degradation when inputting data that lie outside of the numerical range during training.

Inspired by the most recent advantages in deep learning, we use the Neural Arithmetic Logic Units (NALU) [19] as one of the fundamental units in our prediction model. By reconstructing the basic arithmetic operations, such as addition and multiplication, NALU can well solve the data-extra problem. Figure 9 shows the structure of our prediction model, and we also plot its prediction result in Figure 8, which works much better than LSTM. We compare the performance of some common time sequence prediction methods, such as *auto regressive moving average* (ARMA) and *linear regression*(LR). In addition, we also compare with different network structures (e.g., different NALU layers and neuron numbers). The results are shown in Table 1, and we finally choose the best one. It is notable that, instead of completely using NALU units to build up the DNN model, the combination of LSTM and NALU performs better, which can be explained as LSTM helps to capture the time sequence pattern.

After combining the dynamic shift model with new viewer predictor, we can draw a full picture of CLS viewer dynamics.

structure (methods)	layers	neuron numbers	RMSE
ARMA	-	-	0.012
LR	-	-	0.014
NALU	2	(32, 32)	0.014
NALU	3	(64, 32, 32)	0.017
LSTM	3	(64, 32, 32)	0.006
LSTM+NALU	3 (2+1)	(64, 32, 32)	**0.0003**
LSTM+NALU	3 (1+2)	(64, 64, 32)	0.0008
CNN+LSTM+NALU	5 (2+2+1)	(64, 64, 32, 32, 32)	0.0011

Table 1: Comparing new viewer predictor with other methods and DNN structures in Root Mean Square Error (RMSE)

5.2 QoS Manager

QoS characterizer: The performance of a CDN provider is related to two kinds of factors: 1) the offloading strategy used by CDN providers themselves, which we call the *inner factors*, and 2) serving workload allocated by platforms, which we call *outer factors*.

To address the two above factors, we are determined to use a DNN model and fusion them by carefully designing the DNN structure. The structure is shown in Figure 10. As depicted, the model consists of two 2D-convolution layers (CNN) and two LSTM layers. The thought behind this design is that, we use 2D-CNN to capture the relationship between workload and QoS metrics, and use LSTM to capture the temporal information. It is notable that instead of training separate DNN model for each CDN provider, which result in large overhead, we input the sequential information from all CDN providers and output the QoS metrics for each of them. Besides, we also cascade the model with one NALU layer to enable its extra-data generalization ability. We compare our model with other methods in Table 2.

5.3 Optimizer

Since we use DNN model to characterize workload and QoS dynamics, the optimization problem is complex and cannot be directly solved by using traditional algorithms. To alleviate this problem, we first discretize the solution space at a specified interval (e.g., 5% or 10%). This method enables us to generate a set of solution candidates, from which we can then use effective searching algorithms (e.g., Alpha-beta pruning [8]) to find the optimal one.

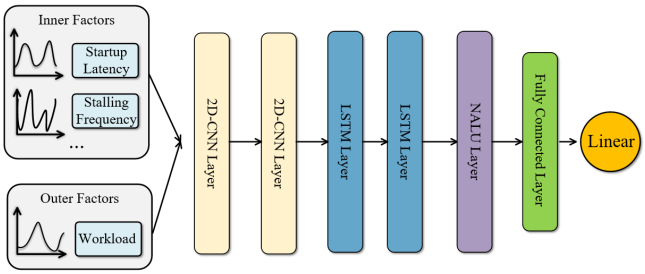


Figure 10: QoS characterizer

structure (methods)	layers	neuron numbers	RMSE
ARMA	-	-	0.020
LR	-	-	0.064
NALU	2	(32, 32)	0.022
NALU	3	(64, 32, 32)	0.016
LSTM	3	(64, 32, 32)	0.012
LSTM+NALU	3 (2+1)	(64, 32, 32)	0.014
LSTM+NALU	3 (1+2)	(64, 64, 32)	0.011
CNN+LSTM+NALU	5 (2+2+1)	(64, 64, 64, 32, 32)	**0.006**

Table 2: Comparing QoS characterizer with other common methods and DNN structures in RMSE

Ideally, given the entire information of dynamics from future workload and CDN performance, which is unavailable in practice, the best distribution ratio at each time can be calculated by searching the solution space. However, even though the perfect knowledge cannot be obtained, it is reasonable that for a short horizon to the future $[t_j, t_{j+N}]$, the prediction can be accurate. Therefore, we run the optimization process inputting the prediction results in this horizon, apply the first decision (i.e., the distribution ratio), and move forward to the next horizon $[t_{j+1}, t_{j+N+1}]$. This is so-called model predictive control (MPC). By well utilizing the predictions and constraints on the manipulated variables, MPC can optimize a time-variant optimization problem in complex and dynamic system.

We also notice that whether the optimization problem is solvable depends heavily on the QoS constraint. To eliminate the possible illness of the optimization problem, we introduce a hyper-parameter k. In details, at each time step t, we input predicted variables $(w^i_{new,t}, QoS^t_i)$ and select out k distribution decisions which can obtain the highest average QoS, i.e., the top-k decisions (x^{topk}_t). Then we find the most cost-effecient decision among them. It is easy to see that here k is equivalent to the QoS_{target}.

Actually, the hyper-parameter k in top-k (i.e., the QoS_{target}) reflects the preference of CLS platform. For example, if we set $k = 1$, i.e., regardless of the costs, Livesmart will make decisions in a QoS-only way. Also, if we set $k = \infty$, i.e., regardless of the QoS, Livesmart will only optimize the overall costs (i.e., greedy).

Algorithm 1 shows how the Optimizer part works. As denoted, the optimizer makes the decision (i.e., x^i_t) by looking forward N steps. At each time, it first selects out the top-k QoS_{max} candidates with the CDN performance prediction $\{QoS_{t_j,t_{j+N}}\}$ from QoS manager and workload dynamics $\{w^i_t(x^i_t), t \in [t_j, t_{j+N}]\}$ from workload

Algorithm 1 The core control of Optimizer

Initialize
for $j = 1$ to T **do**
  $w^{new}_{[t_j, t_{j+N}]} = WorkloadPred(w_{his})$
  for All possible $x_{[t_j, t_{j+N}]}$ **do**
    $w^{i,new}_{[t_j, t_{j+N}]} = x_{[t_j, t_{j+N}]} * w^{new}_{[t_j, t_{j+N}]}$
    $W^{i,re}_{[t_j, t_{j+N}]} = DyShift(w^{i,new}_{[t_j, t_{j+N}]})$
    $w^{i}_{[t_j, t_{j+N}]} = W^{i,re}_{[t_j, t_{j+N}]} + w^{i,new}_{[t_j, t_{j+N}]}$
    update $S^i_{[t_j, t_{j+N}]}$ **according to Eq.(1)**
    $QoS_{[t_j, t_{j+N}]} = QoSCharac(w_{[t_j, t_{j+N}]}, QoS_{his})$
    $x^{topk}_{[t_j, t_{j+N}]} = TopK(QoS_{[t_j, t_{j+N}]})$
  end for
  $x_{t_k} = \arg \min_{x^{topk}_{[t_j, t_{j+N}]}} \sum_{i=1}^{m} cost(S^i_{t_{j+N}})$
end for =0

manager. Then it chooses the most cost-efficient decision among these k candidates.

6 EVALUATION

6.1 Methodology

Dataset: The dataset is composed of two parts: the viewer data and the CDN data, and all of them are provided by Kuaishou[4], a leading CLS platform in China. The viewer data consist of more than 500M view sessions from 50M viewers, each consists of the detailed session information including user ID, start time, end time, CDN provider, ISP, and province. The CDN data consist of the performance statistic information including startup latency, stalling frequency, and stalling rate[5].

Comparing metrics: The comparison consists of two parts: the QoS and the costs. For QoS, we consider three industrial standard metrics including *startup latency* (denoted as $startup$), *stalling frequency* (denoted as $stallFreq$) and *stalling rate* (denoted as $stallRate$), and similar to [6, 12], we also use the weighted sum of them, which is simple and interpretable (other QoS settings can still work):

$$QoS = -\alpha * startup - \beta * stallFreq - \gamma * stallRate \quad (8)$$

we set $\alpha = 5$, $\beta = 1$ and $\gamma = 1$ to scale them into close value. For the costs, we use charging policies obtained from the CDN provider website and calculate the overall costs.

Implementation: Despite there is a total of 5 CDN providers in our dataset, a normal setting is 3 CDN providers [1]. Therefore, without loss of generality, we select 3 of them in our experiment. At the same time, we choose the viewer and CDN provider data in the same group. For dynamic shift model (update once an hour), we discretize the time interval into $\delta = 1min$ (we choose $1min$ as too short period will introduce noise when doing data statistics), and set $k = 10min$ when calculating Eq. (7). For the new viewer predictor, we use two LSTM layers of size 64 and 32 (*sigmoid* as inner activation and *tanh* as activation function), and one NALU layer of 32 NALU units (the same setting with [19]). For the QoS characterizer, we use two layers 2D-CNN with 64 filters (*relu* as the activation function), each of size 2, stride 1. Both QoS characterizer and new viewer predictor use Adam optimizer and uniform initializer, and they have the same layer configuration. For the MPC solver, we set the horizon $N = 3$ and top-k as top-20%. The decision space is discretized into interval 10% (e.g., the ratio of 3 CDN providers t can be $[0.1, 0.1, 0.8]$). Therefore, there are 36 possible distribution ratios ($\binom{9}{2}$) at each time step. Considering horizon number, there are totally $36^3 = 46656$ choices.

[4]www.kuaishou.com
[5]stalling rate represents the rate between the stalling event and the time of a total session.

k	5%	10%	20%	50%	100%
Cost (norm)	0.72	0.40	**0.37**	0.34	0.30
Average QoS	−4.41	−4.43	**-4.46**	−4.87	−5.14

Table 3: Performance of Livesmart under different *top-k*

Baseline algorithms: We compare Livesmart with following algorithms:

The original algorithm: The strategy used by the data provider, the result of which can be directly obtained from collected data.

RoundRoubin: The most widely used scheduling method by platforms. Viewers are scheduled in a uniformly random order.

Cost only: This algorithm minimizes the overall costs regardless of the CDN performance.

QoS only: This algorithm maximizes the QoS regardless of the costs, i.e., the parameter k in top-k is set to be $k = 1$.

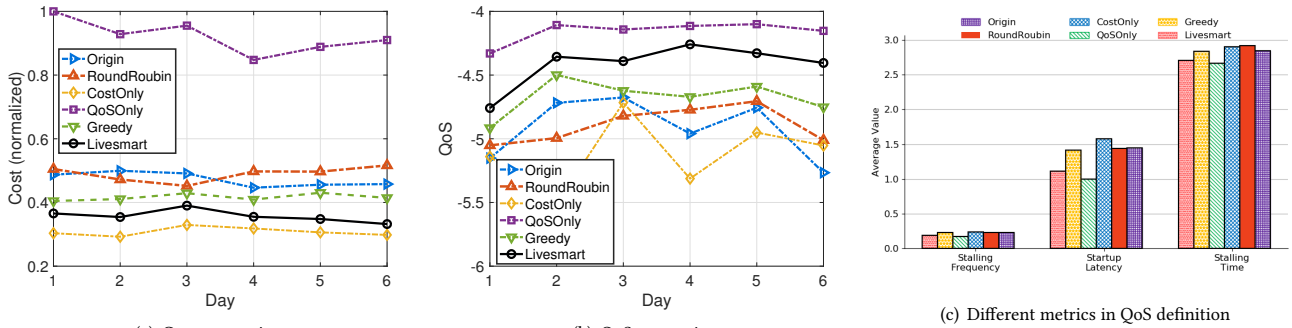
Gready: This algorithm has the same configuration parameters with Livesmart except for the MPC part, in which the prediction horizon is set to be $N = 1$.

6.2 Results and discussion

We first compare the overall costs of each algorithm. We present the results in Figure 11, from which we derive the following two observations: 1) Considering the real-world constraints (Origin, RoundRoubin, QoSOnly, and Greedy), our proposed Livesmart outperforms the baselines in all six days. For example, we can see that on the first day, Livesmart saves costs by 24.97%, 27.60%, 63.45% and 9.67% against four baselines respectively. It is reasonable since Livesmart considers the overall costs and makes the decisions in a long-term way. 2) Among all baselines, *QoSOnly* is the most cost-inefficient algorithm. For example, as presented in Figure 11(a), QoS-Only method increases the overall costs $2.5 - 3.3\times$ comparing with Cost-Only method. The reason is: QoS-Only algorithm is more attentive to schedule viewers to the best-performance CDN providers, which makes them more likely to get higher peak load and therefore higher costs.

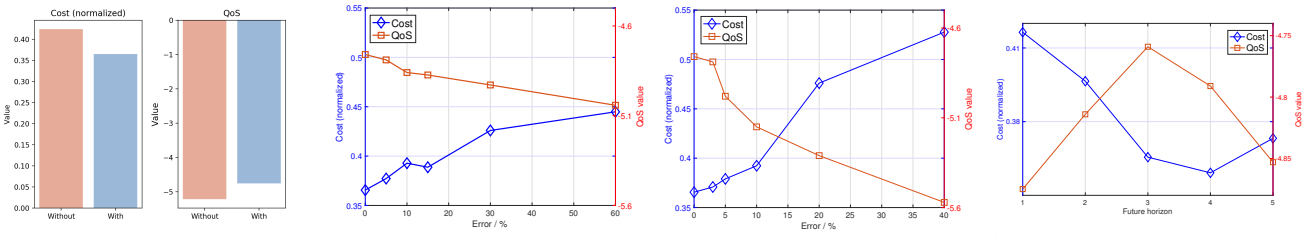
Moreover, we also investigate the QoS value of different algorithms in each day, and present the results in Figure 11(b). We can find that except for the QoS-Only algorithm which is cost-inefficient, Livesmart achieves the highest QoS value against other baselines. For example, in the third day, Livesmart outperforms the four baselines (i.e., Origin, RoundRoubin, Cost-Only, and Greedy) by 7.63%, 5.79%, 7.42%, and 3.18%, respectively. Notably, the Greedy method performs consistently well when comparing with other baselines, since it still makes decisions considering both cost and QoS dynamics even if only one step further.

To better understand the QoS achieved by Livesmart, we also analyze the individual component of our QoS definition (i.e., Eq.(8)). In details, we present the average value of each QoS metric in



(a) Cost comparison (b) QoS comparison (c) Different metrics in QoS definition

Figure 11: Comparison results. The Evaluation on real-world traces shows that Livesmart outperforms prevalent algorithms by 24.57%-63.45% decrease on overall costs and 5.79%-7.63% improvement on average QoS. Especially, Livesmart consistently outperforms baselines on all three QoS metrics.



(a) Sensitivity experiments of dynamic shift model (b) Sensitivity experiments of new viewer predictor (c) Sensitivity experiments of QoS characterizer (d) Sensitivity experiments of lookahead horizon

Figure 12: Sensitivity experiments. The results show that all three parts are essential for Livesmart.

Figure 11(c). We can see that except for the QoS-Only algorithm, Livesmart performs the best in all three QoS metrics.

We also present the relationship between Livesmart performance and parameter k in top-k. As denoted in Table 3, we can see that as the k increases, Livesmart is more attentive to make decisions in a cost-efficient way, which reflects the preference of platforms.

6.3 Sensitivity experiments

In this section, we will study the influence of the following parts: (1) the dynamic shift model and new viewer predictor in workload manager; (2) the QoS characterizer in QoS manager; and (3) the look-ahead horizon N in the optimizer.

Workload Manager: This component consists of two parts: the workload dynamic shift model and new viewer predictor. For the first part, we compare the performance of Livesmart with shift model and without shift model. We present the results in Figure 12(a). Comparing with Livesmart, Livesmart without shift model increases the costs about 11.2% and decreases the QoS about 7.4%. This is reasonable since without shift model, Livesmart cannot control the viewers in an overall way. Then, we analyze how new viewer prediction error influences the performance by adding different level of random noise. As denoted in Figure 12(b), a 10% increase of prediction error will result in about 4% QoS drop and 5% costs increase.

QoS Manager: Figure 12(c) analyzes the impacts of QoS characterizer. We can observe that with the increase of the prediction error, Livesmart performs worse both on costs and average QoS. It is also notable that comparing with workload predictor error, Livesmart is more sensitive to QoS characterizer error. For example, from Figure 12(c), we see that 20% error of QoS manager increases overall costs by 12% and decreases the average QoS by 25%, while for new viewer prediction error, the costs increase by 9% and QoS decreases 11%. This can be explained as: Livesmart makes decisions in a QoS-first manner, which makes the results more likely to be influenced by the QoS manager.

Optimizer: Figure 12(d) shows how the look-ahead horizon impacts the performance of Livesmart. As the horizon number N increases, the performance of Livesmart starts to increase. This is because Livesmart uses more future information and thus planning in a long-term way. At the same time, when Livesmart predicts too far (i.e., more than 4 next time steps), as the prediction accuracy reduces significantly, the performance starts to drop. Considering QoS priority and calculation overhead, we finally choose $N = 3$.

7 CONCLUSION

We propose Livesmart, a QoS-guaranteed cost-minimum approach for CLS services. Specifically, we address several critical challenges in the framework design including a) accurate modeling of viewer pattern and CDN performance; b) intelligent workload offloading to save costs while guaranteeing the quality of service (QoS); c) and ease of integration with practical CDN infrastructure in CLS service. The experiments demonstrate that Livesmart schedules viewers both effectively and economically.

Acknowledgement: We sincerely thank all reviewers and our shepherd Yuansong Qiao. This work was supported by the National Key R&D Program of China (No. 2018YFB1003703), NSFC under Grant 61521002, Beijing Key Lab of Networked Multimedia, and Kwai-Tsinghua Joint Project (No. 20192000456).

REFERENCES

[1] Vijay Kumar Adhikari and 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.

[2] Vijay Kumar Adhikari, Yang Guo, Fang Hao, Matteo Varvello, Volker Hilt, Moritz Steiner, and Zhili Zhang. 2012. Unreeling netflix: Understanding and improving multi-CDN movie delivery. (2012), 1620–1628.

[3] Yonghwan Bang, June-Koo Kevin Rhee, KyungSoo Park, Kyongchun Lim, Giyoung Nam, John D Shinn, Jongmin Lee, Sungmin Jo, Ja-Ryeong Koo, Jonggyu Sung, et al. 2016. CDN interconnection service trial: implementation and analysis. *IEEE Communications Magazine* 54, 6 (2016), 94–100.

[4] Timm Böttger, Felix Cuadrado, Gareth Tyson, Ignacio Castro, and Steve Uhlig. 2018. Open connect everywhere: A glimpse at the internet ecosystem through the lens of the netflix cdn. *ACM SIGCOMM Computer Communication Review* 48, 1 (2018), 28–34.

[5] Fei Chen, Cong Zhang, Feng Wang, and Jiangchuan Liu. 2015. Crowdsourced live streaming over the cloud. In *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2524–2532.

[6] Florin Dobrian, Asad K Awan, Dilip Antony Joseph, Aditya Ganjam, Jibin Zhan, Vyas Sekar, Ion Stoica, and Hui Zhang. 2013. Understanding the impact of video quality on user engagement. *Communications of The ACM* 56, 3 (2013), 91–99.

[7] Chongwu Dong, Yin Jia, Hua Peng, Xiaoxing Yang, and Wushao Wen. 2018. A Novel Distribution Service Policy for Crowdsourced Live Streaming in Cloud Platform. *IEEE Transactions on Network and Service Management* 15 (2018), 679–692.

[8] Daniel James Edwards and TP Hart. 1961. The alpha-beta heuristic. (1961).

[9] Jian He, Di Wu, Yupeng Zeng, Xiaojun Hei, and Yonggang Wen. 2013. Toward optimal deployment of cloud-assisted video distribution services. *IEEE transactions on circuits and systems for video technology* 23, 10 (2013), 1717–1728.

[10] Nicolas Herbaut, Daniel Négru, Yiping Chen, Pantelis A Frangoudis, and Adlen Ksentini. 2016. Content delivery networks as a virtual network function: A win-win ISP-CDN collaboration. In *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 1–6.

[11] Junchen Jiang, Vyas Sekar, Henry Milner, Davis Shepherd, Ion Stoica, and Hui Zhang. 2016. {CFA}: A Practical Prediction System for Video QoE Optimization. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*. 137–150.

[12] Junchen Jiang, Shijie Sun, Vyas Sekar, and Hui Zhang. 2017. Pytheas: Enabling data-driven quality of experience optimization using group-based exploration-exploitation. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*. 393–406.

[13] Hongqiang Harry Liu, Ye Wang, Yang Richard Yang, Hao Wang, and Chen Tian. 2012. Optimizing cost and performance for content multihoming. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. ACM, 371–382.

[14] Xi Liu, Florin Dobrian, Henry Milner, Junchen Jiang, Vyas Sekar, Ion Stoica, and Hui Zhang. 2012. A case for a coordinated internet video control plane. *acm special interest group on data communication* 42, 4 (2012), 359–370.

[15] Zoltán Ádám Mann. 2015. Allocation of virtual machines in cloud data centersâĂŤa survey of problem models and optimization algorithms. *Acm Computing Surveys (CSUR)* 48, 1 (2015), 11.

[16] Haitian Pang, Zhi Wang, Chen Yan, Qinghua Ding, and Lifeng Sun. 2017. First Mile in Crowdsourced Live Streaming: A Content Harvest Network Approach. (2017), 101–109.

[17] Haitian Pang, Cong Zhang, Fangxin Wang, Han Hu, Zhi Wang, Jiangchuan Liu, and Lifeng Sun. 2018. Optimizing Personalized Interaction Experience in Crowd-Interactive Livecast: A Cloud-Edge Approach. In *2018 ACM Multimedia Conference on Multimedia Conference*. ACM, 1217–1225.

[18] James Blake Rawlings and David Q Mayne. 2009. *Model predictive control: Theory and design*. Nob Hill Pub. Madison, Wisconsin.

[19] Andrew Trask and et al. 2018. Neural arithmetic logic units. In *Advances in Neural Information Processing Systems*. 8046–8055.

[20] Feng Wang, Jiangchuan Liu, Minghua Chen, and Haiyang Wang. 2016. Migration towards cloud-assisted live media streaming. *IEEE/ACM Transactions on networking* 24, 1 (2016), 272–282.

[21] Jason Min Wang, Jun Zhang, and Brahim Bensaou. 2014. Content multi-homing: An alternative approach. In *2014 IEEE International Conference on Communications (ICC)*. IEEE, 3118–3123.

[22] Zhi Wang, Lifeng Sun, Chuan Wu, Wenwu Zhu, and Shiqiang Yang. 2014. Joint online transcoding and geo-distributed delivery for dynamic adaptive streaming. (2014), 91–99.

[23] Bo Yan, Shu Shi, Yong Liu, Weizhe Yuan, Haoqin He, Rittwik Jana, Yang Xu, and H Jonathan Chao. 2017. LiveJack: Integrating CDNs and Edge Clouds for Live Content Broadcasting. In *Proceedings of the 25th ACM international conference on Multimedia*. ACM, 73–81.

[24] Zidong Yang, Ji Hu, Yuanchao Shu, Peng Cheng, Jiming Chen, and Thomas Moscibroda. 2016. Mobility modeling and prediction in bike-sharing systems. In *Proceedings of the 14th annual international conference on mobile systems, applications, and services*. ACM, 165–178.

[25] Zheng Zhang, Ming Zhang, Albert G Greenberg, Y Charlie Hu, Ratul Mahajan, and Blaine Christian. 2010. Optimizing cost and performance in online service provider networks. (2010), 3–3.

[26] Yifei Zhu, Jiangchuan Liu, Zhi Wang, and Cong Zhang. 2017. When Cloud Meets Uncertain Crowd: An Auction Approach for Crowdsourced Livecast Transcoding. (2017), 1372–1380.