Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 5562995, 11 pages
https://doi.org/10.1155/2022/5562995

*Research Article*

# Popularity-Guided Cost Optimization for Live Streaming in Mobile Edge Computing

**Tao He, Kunxin Zhu, Zhipeng Chen, Ruomei Wang, and Fan Zhou**

*School of Computer Science and Engineering, Sun Yat-sen University, China*

Correspondence should be addressed to Fan Zhou; isszf@mail.sysu.edu.cn

Received 8 January 2021; Revised 20 August 2021; Accepted 5 November 2021; Published 5 January 2022

Academic Editor: Xiaojie Wang

Live streaming service usually delivers the content in mobile edge computing (MEC) to reduce the network latency and save the backhaul capacity. Considering the limited resources, it is necessary that MEC servers collaborate with each other and form an overlay to realize more efficient delivery. The critical challenge is how to optimize the topology among the servers and allocate the link capacity so that the cost will be lower with delay constraints. Previous approaches rarely consider server collaborations for live streaming service, and the scheduling delay is usually ignored in MEC, leading to suboptimal performances. In this paper, we propose a popularity-guided overlay model which takes the scheduling delay into consideration and utilizes MEC collaboration to achieve efficient live streaming service. The links and servers are shared among all channel streams and each stream is pushed from cloud servers to MEC servers via the trees. Considering the optimization problem is NP-hard, we propose an effective optimization framework called cost optimization for live streaming (COLS) to predict the channel popularity by a LSTM model with multiscale input data. Finally, we compute topology graph by greedy scheme and allocate the capacity with convex programming. Experimental results show that the proposed approach achieves higher prediction accuracy, reducing the capacity cost by more than 40% with an acceptable delay compared with state-of-the-art schemes.

## 1. Introduction

With the rapid popularization of smart devices, the Internet traffic has ushered an explosive growth [1], and almost 82% of all network traffic comes from video traffic [2]. The increasing traffic puts amounts of pressure on the cloud data center, bringing more difficulties for the optimization of the servers [3], especially for some latency-sensitive services, e.g., live streaming. To address this, mobile edge computing (MEC) is brought in as a new technology for live streaming service to reduce the network latency and alleviate the backhaul capacity [4]. Internet service provider (ISP) places nearby MEC servers at the network edge so that users can visit these servers instead of remote cloud servers and get a better experience. Due to limited resources of a single MEC server, multiple servers are usually used to collaborate with each other and form an overlay to deliver the content [5]. The cost of deploying such an overlay mainly comes from the link capacity cost (If an overlay is firstly deployed, it has another cost called *Server Cost* to purchase servers' hardware. Compared with the link capacity cost, *Server Cost* is a one-time deployment cost, so we ignore it in this paper. We think that servers' hardware resources, computing capacities and upload capacities, are enough and will not be an optimization bottleneck). While the link capacity is higher, the source-to-end delay which is defined as the time elapsed from the cloud server to the MEC server is lower but the cost increases rapidly. A critical problem is how to construct the topological graph among these servers and allocate the bandwidth capacity to each link so that the cost will be minimum while the delay is still under a certain bound.

Failed to address the problem above, existing works mainly suffer from following deficiencies.

In MEC environment, most works pay attention to the optimization for static contents [6, 7], such as video on demand (VoD) streaming or image caching, which are delay insensitive. These approaches usually fail to address the rationality of MEC application providers or have different objective functions, leading to improper results in cost optimization for live streaming.

Some works optimize the resource allocation by predicting the popularity of contents [8–10], considering that a few popular videos usually contribute to most of the bandwidth consumption [11]. However, suffering from the similar reason that these models usually focus on the prediction of static contents, it is still hard to meet the real-time requirement of live streaming.

As a considerable impact factor in optimization strategy, server's scheduling delay is usually ignored or not modeled sufficiently in most of existing works [12–14], which renders these models' inaccuracy and inefficiency in reality.

To tackle above challenges, we construct a multisource multichannel overlay model which focuses on the popularity prediction, topology generation, and link bandwidth allocation. We combine the deep neural network and the mathematical model to optimize the overlay deployment cost, i.e., we first predict the popularity of live channel with LSTM model and identify which channel the MEC server should subscribe to. Then, we compute the topology graph and allocate the link capacity by mathematical optimization methods.

In the proposed approach, each channel stream has heterogeneous rate which is constant in transmission, and all packet lengths are equal (the channel rate and the packet length may vary in reality, but they are not variables in our model and do not affect the problem solving. Hence, we simplify them.). Without loss of generality, we suppose that a channel can only originate from one source, and a cloud server could be the source of multiple channels. A subscriber (defined as a MEC server which subscribes to channels) could receive a channel stream from a cloud server, another MEC server subscribing to the same channel or a helper (a helper denotes a MEC server which forwards unsubscribed channel streams.). Therefore, there are some nodes (helpers) that can be included or excluded in a channel transporting path, and all channel trees are combined into a mesh.

In practice, the link cost is usually charged by the maximum rented capacity. And the source-to-end delay actually consists of four type delays, e.g., link propagation delay, server transmission delay, server processing delay, and server queuing delay. More specifically, the link propagation delay is the time that a packet travels over a physical connection, usually reflected by the round-trip time. And the server processing delay can be omitted while the computing resource is sufficient as aforementioned. Then, we combine server transmission delay and server queuing delay into server scheduling delay, defined as the used time that a packet is kept on a server until it is completely transmitted out. In this way, the source-to-end delay consists of link propagation delay and server scheduling delay. As a result, a high scheduling delay will cause network congestion, which should be taken into consideration by all means.

As presented in Figure 1, it depicts an example of proposed overlay model where $S1$, $S2$ are cloud (source) servers and $S3$–$S6$ are MEC servers in different regions. Cost optimization is carried out by a central optimizer which continuously collects network parameters and sends control message flows. The workflow is shown in Figure 2. The optimizer predicts the popularity of live channels and decides the subscribed list of each MEC server. Based on obtained information and the delay requirements, the optimizer computes a topology graph with least deployment cost. Finally, the optimizer informs MEC servers of the optimized information, i.e., the subscribed list, the topology graph, and the link capacity. Once MEC servers receive the control message, they collaborate with each other and form an overlay to deliver live channel $A$ to server $S3$ and $S5$, channel $B$ to $S4$, $S5$, and $S6$. In this overlay, channel stream $A$ is pushed from $S1$ to $S5$ with $S4$ forwarding. $S4$ does not subscribe to channel $A$ but forward its data as a helper. The link between $S4$ and $S5$ transports two channels simultaneously.

In summary, we make the key contributions as follows:

(i) Aiming at live streaming service, we formulate the optimization problem and construct a multisource multichannel overlay model in which the MEC servers collaborate with each other. In addition, the scheduling delay is also taken into consideration to construct an optimized topology graph among MEC servers, achieving lower link capacity with delay constraints

(ii) Instead of using fixed live channel popularity for the optimization, we utilize state-of-the-art LSTM model to learn the features of historical streaming data for adaptive and accurate popularity prediction. Furthermore, we also takes the information of time and weekdays into consideration, employing multiscale input data to make a more accurate and robust prediction

(iii) Cost optimization for live streaming (COLS) is proposed as a complete and efficient cost optimizer framework, which considers the whole systematic flow of the optimization in a logical order, including accurate key parameters prediction, comprehensive overlay model formulation, optimal topology generation, and efficient capacity allocation. Finally, COLS is able to achieve a lower cost in polynomial time while meeting the delay constraint

The remainder of this paper is organized as follows. After reviewing related works in Section 2, we elaborate the popularity prediction of live channel in Section 3. Following the mathematical formulation in Section 4, we compute the topology in Section 5. Illustrative experiment results are presented in Section 6. Finally, we conclude in Section 7.

## 2. Relate Work

In this section, we review related works in the areas of MEC collaboration, popularity prediction, and scheduling delay model.

*MEC collaboration*. In MEC environment, many works [6, 7] realize collaboration mechanism among the servers to achieve higher efficiency. For example, the approach proposed in [6] utilizes the collaboration between the MEC servers to cache the static content in spare time, e.g., midnight, which is impractical for live streaming service. Since these proposed optimization methods aim to allocate

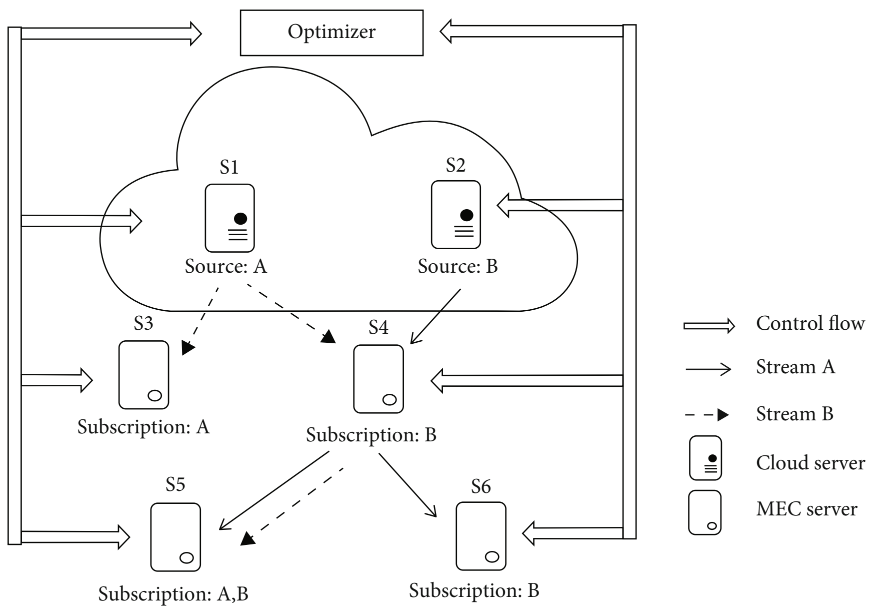

Figure 1: A multisource multichannel overlay model in MEC.

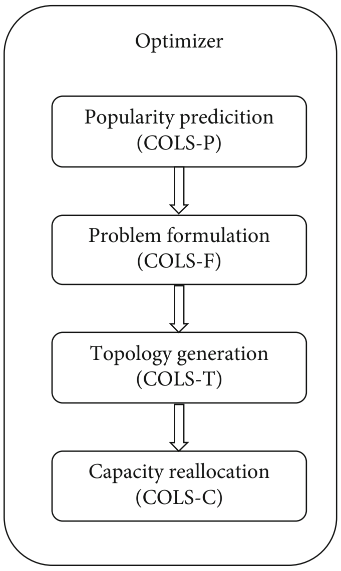
Optimizer
Popularity predicition (COLS-P)
Problem formulation (COLS-F)
Topology generation (COLS-T)
Capacity reallocation (COLS-C)

Figure 2: Workflow of optimizer.

the resource efficiently, they usually have different optimization objective in resource utilization and fail to be applied in live streaming services with real-time requirement.

Recently, some specially designed resource optimization methods [15–18] are proposed for live streaming services. For instance, CCAS [15] proposes an auction-based algorithm to optimize the backhaul capacity and the caching space so as to improve the live video quality. Zhang et al. [16] model the computational and wireless spectrum resource in edge-clouds networks. They propose a Markov decision process to decrease the latency of live streaming services. Nevertheless, these approaches usually focus on resource optimization for a single MEC server and rarely consider the collaboration among multiple servers. It potentially results in a low performance such as the network congestion while the user request is higher.

*Popularity of video stream.* As a key parameter, some works predict the popularity of videos by analyzing the image frame. For example, TLRMVR [8] proposes a novel low-rank multiview embedding learning method to predict the popularity of microvideo. MMVED [9] combines multiple features (image frame, acoustic, and textual info) and considers the randomness for the mapping from data to popularity. Although these approaches are able to achieve efficient prediction, they usually aim at the static complete file and need to parse entire image frame, which is impractical for live streaming. Inspired by success of deep learning techniques, Deepcache [19] predicts the popularity with LSTM Encoder-Decoder to cache contents smartly. And BSPP [10] presents a model for predicting the number of user requests based on Malcov model in MEC and further designs an offloading scheme based on this model. Although effective, these approaches utilize the data in single dimension to predict the popularity, which lack sufficient robustness while facing the random noise and outliers.

On the other hand, some approaches [20, 21] use both the popularity and retention rate of video streams to maximize video bitrate for efficient utilization of bandwidth. Distinguished from these approaches which emphasize bitrate adaptation, this paper focuses on the topology optimization for the cooperation of MEC servers while additionally considering the scheduling delay to achieve lower link capacity with delay constraints. Since our proposed approach and these methods have different focus, respectively, they can be combined to achieve better performances of live streaming services.

*Scheduling delay model.* For the mathematical model about overlay, most existing works [12–14] rarely formulate the relationship between the link capacity and the server scheduling delay. BSUM [12] considers the scheduling delay and constructs the topology graph among MEC servers to optimize the resource. But it studies the scheduling delay insufficiently without considering the impact of different link capacities on the scheduling delay, which limits the effect of optimization consequently.

## 3. Popularity Prediction

In this section, we take advantage of state-of-the-art LSTM model to predict the popularity of live channels, which is the first step called COLS-P of the optimizer as shown in Figure 2.

For time series data, recurrent neural network (RNN) has been widely used to capture the temporal correlations and continuity constraints. As a variant of RNN, long short-term memory (LSTM) model solves the long-term dependence problem of general RNN and enables the network to learn the long-term dependence of time series by selectively memorizing the characteristic information of time series. At each timestep of input time series, LSTM applies the following operations:

$$\begin{aligned} f_t &= \sigma_g\left(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + b_f\right), \\ i_t &= \sigma_g\left(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + b_i\right), \\ o_t &= \sigma_g\left(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + b_o\right), \\ \tilde{c}_t &= \sigma_c\left(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1} + b_c\right), \\ c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t, \\ \mathbf{h}_t &= o_t \circ \sigma_h(c_t), \\ \mathbf{y}_t &= \sigma_o\left(W_y \mathbf{h}_t + b_y\right), \end{aligned} \tag{1}$$

where the operator $\circ$ denotes the Hadamard product (element-wise product). The subscript $t$ denotes the time step. $\mathbf{x}_t$ represents the input at time $t$ and $\mathbf{h}_t$ represents hidden state. $f, i, o$ represent *forget gate*, input and output *reset gates,* respectively, and $c$ denotes *memory cell state*. $W$, $U$, and $b$ are weight matrices and bias which need to be learned during training. And $\sigma$ indicates the activation function.

The consecutive historical popularity data can be regarded as time series data. Intuitively, we employ state-of-the-art LSTM model to predict the live channel popularity based on historical data. Specifically, Figure 3 shows the popularity of one live channel on one MEC server in 300 consecutive hours and the second red box records (in 100-150 hours period) is further detailed in Figure 4, which presents the popularity changes in single day. We observe from these figures that the time data have an impact on the channel popularity:

(i) *Weekday impact.* The data in two red boxes of Figure 3 correspond to the channel popularity in Sunday and Wednesday, respectively. As presented in Figure 3, it is obvious that the popularity of Sunday is higher than that of Wednesday, so the message of weekday can be a valid supplementary information for accurate prediction

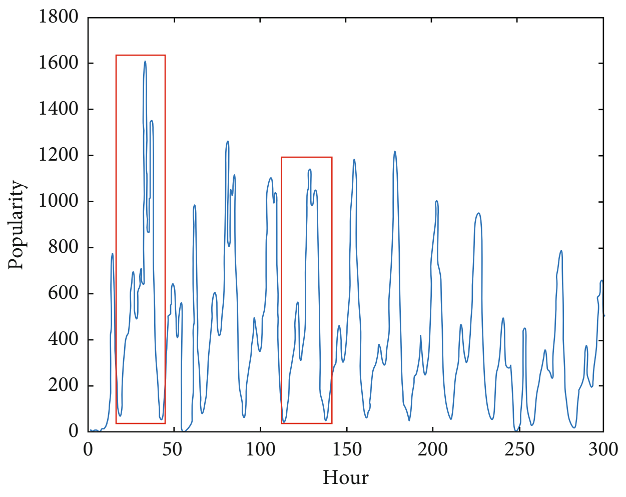

FIGURE 3: Live channel popularity records.

(ii) *Hour impact.* As shown in Figure 4, we can easily distinguish the popularity changes in the third black box from the first two via the trends. However, the first two is hard to be distinguished from each other just with the trends. Fortunately, as we can observed in Figure 4, the different period of time in the day (different hours) can be an efficient indicator to resolve this confusion

Therefore, different from previous works which only take the historical popularity data for training, we also consider the influence of time data (weekday and hour info impacts) and utilize multiscale time data for training to achieve more accurate prediction.

We train the LSTM network for each channel individually. The raw data consists of consecutive tuples, which contain hourly, weekday, and popularity records (i.e., user requests) per hours. The original data is processed in the form of sliding window, in which these tuples are treated as the input sequence. Then, the popularity in next hour is set as a label. As the popularity prediction in this paper is a regression task, the loss function of the network is set to mean square error (MSE) which is the most common and widely used loss function for regression task.

When the training is completed, for a channel $n$, we use the historical popularity records of one MEC server to predict its popularity $p^{(n)}$ in the next time period. Define $q^{(n)} = p^{(n)}/\tau^{(n)}$, where $\tau^{(n)}$ is the streaming rate of channel $n$. Then, we sort the live channel by $q^{(n)}$ and select the first $k$ live channels as the subscribed list $\mathcal{N}_i$ of the MEC server $i$.

## 4. Overlay Formulation

After we learned from LSTM network which channels each MEC server should subscribe to (i.e., $\mathcal{N}_i$), we formulate the overlay model including topology model, cost model, delay model, and joint optimization for live streaming.

The major symbols used in this paper are presented in Table 1. We regard the overlay as a directed complete graph $G = (\mathcal{V}, \mathcal{E})$, where $V$ is the set of all servers (cloud servers

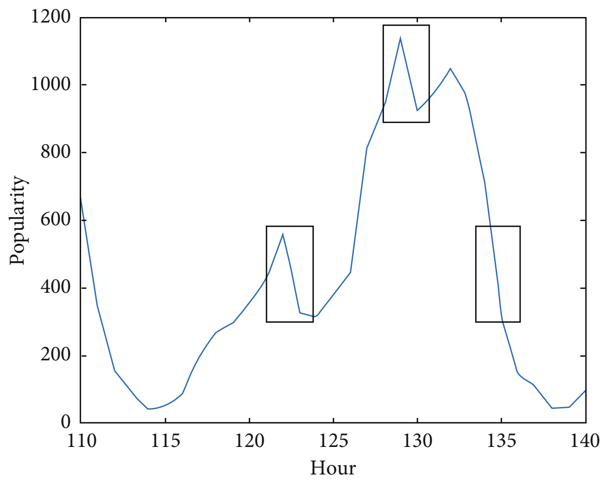

FIGURE 4: Popularity records in specific time period.

TABLE 1: Major notations.

| Notation | Definition |
|---|---|
| $\mathcal{S}$ | Set of sources (cloud servers) |
| $\mathcal{M}$ | Set of subscribers (MEC servers) |
| $\mathcal{V}$ | Set of servers where $\mathcal{V} = \mathcal{S} \cup \mathcal{M}$ |
| $\mathcal{E}$ | Set of links |
| $\mathcal{N}$ | Set of channels |
| $\mathcal{N}_i$ | Set of subscribing channels of server $i$ |
| $\tau^{(n)}$ | Streaming rate of channel $n$ |
| $s^{(n)}$ | Source server of channel $n$ |
| $p^{(n)}$ | Predicted popularity of channel $n$ |
| $\mathcal{M}^{(n)}$ | Set of subscribers of channel $n$ |
| $T^{(n)}$ | Deliver tree of channel $n$ |
| $\langle i, j\rangle$ | Link from server $i$ to server $j$ |
| $b_{ij}$ | Link capacity of $\langle i, j\rangle$ |
| $c_{ij}$ | Capacity cost of $\langle i, j\rangle$ |
| $x_{ij}$ | Indicator indicating whether $\langle i, j\rangle$ is used |
| $d^p_{ij}$ | Propagation delay of $\langle i, j\rangle$, $d^p_{ij} = d^p_{ji}$ |
| $d^s_i$ | Worst-case scheduling delay of server $i$ |
| $D^{(n)}_i$ | Source-to-end delay of server $i$ in tree $T^{(n)}$ |
| $D$ | Total delay constraint |
| $C$ | Total capacity cost |

and MEC servers). Let $\mathcal{S}$ be the set of sources (cloud servers) and $M$ be the set of subscribers (MEC servers). So, $\mathcal{E} = \mathcal{V} \times \mathcal{V}$ is the set of possible overlay connections and $\mathcal{V} = \mathcal{S} \cup \mathcal{M}$. $\mathcal{N}$ is the set of channels. Denote the rate of channel $n$ as $\tau^{(n)}$. The streaming is delivered to subscribers in $\mathcal{M}^{(n)}$ via a tree. A subscriber receives a stream either from a cloud server, a MEC server which subscribes to the same channel or a helper. There are totally $|\mathcal{N}|$ trees, and we denote the tree of channel $n$ as $T^{(n)}$.

*4.1. Topology Model.* Equation (2) shows a variable $x^{(n)}_{ij}$ which indicates whether link $\langle i, j\rangle$ is used in tree $T^{(n)}$. All $x^{(n)}_{ij}$ are combined into a vector solution $\vec{x}$.

$$x^{(n)}_{ij} = \begin{cases} 1, \text{if } \langle i, j\rangle \in T^{(n)}, \\ 0, \text{otherwise}, \end{cases} \tag{2}$$

$$\vec{x} = \left\{ x_{ij} \middle| x_{ij} = \max_n x^{(n)}_{ij}, n \in \mathcal{N} \right\}. \tag{3}$$

Equations (4) and (5) guarantee each channel tree is connected, and there is no isolated server. Also, there is no loop in each tree as shown in Equation (6). $|\mathcal{M}^{(n)}|$ is the number of MEC servers which subscribe to channel $n$. $s^{(n)}$ is the source server of channel $n$. $Y$ is a subset of servers and $E(Y)$ denotes the set of links connecting servers in $Y$.

$$\left|\mathcal{M}^{(n)}\right| \le \sum_{\langle i,j\rangle \in \mathcal{E}} x^{(n)}_{ij} \le |\mathcal{M}|, \quad \forall n \in \mathcal{N}, \tag{4}$$

$$\sum_{\langle i,j\rangle \in \mathcal{E}} x^{(n)}_{ij} \ge 1, \quad \forall i \in \mathcal{M} \cup \left\{ s^{(n)} \right\}, \forall j \in \mathcal{M}^{(n)}, \forall n \in \mathcal{N}, \tag{5}$$

$$\sum_{\langle i,j\rangle \in \mathcal{E}(\mathcal{Y})} x^{(n)}_{ij} \le |\mathcal{Y}| - 1, \quad \forall \mathcal{Y} \subseteq \mathcal{M} \cup \left\{ s^{(n)} \right\}, \forall n \in \mathcal{N}. \tag{6}$$

*4.2. Cost Model.* Denote the capacity of link $\langle i, j\rangle$ as $b_{ij}$. Like $x_{ij}$, all $b_{ij}$ are combined into a vector $\sim b = \{b_{12}, b_{13}, \cdots, b_{ij}\}$. The link cost is $c_{ij}$, which is a linear function of $b_{ij}$. Total cost $C$ is the sum of all link capacity costs, i.e.,

$$\begin{gathered} C = \sum_{i \in \mathcal{V}, j \in \mathcal{M}} x_{ij} * c_{ij}\left(b_{ij}\right), \\ c_{ij}\left(b_{ij}\right) = k_{ij} * b_{ij}, \end{gathered} \tag{7}$$

where $k_{ij}$ is a constant coefficient.

*4.3. Delay Model.* We employ a sequential scheduling model in which a parent node transmits packets into one link after another sequentially [22]. Denote the worst-case scheduling delay of server $i$ as $d^s{}_i$, which is the maximum amount of time that a packet has to wait until it is transmitted out completely (according to a packet, its queuing delay is the sum of other packets' transmission delay.). To avoid the congestion, $d^s{}_i$ should be smaller than the time interval $L/\tau$ between two sequential packets, where $L$ is the packet size [22]. Therefore, we set following congestion constraints:

$$d^s_i = \sum_n \sum_{k \in \Gamma_i} \frac{L \cdot x^{(n)}_{ik}}{b_{ik}}, \tag{8}$$

$$d^s_i \le \frac{L}{\tau^{\max}}, \tau^{\max} = \max_n \tau^{(n)}, \quad n \in \mathcal{N}^{\text{Out}}_i, \tag{9}$$

where $\Gamma_i$ is the set of children (with repetition) of server $i$ in all channel trees, ${N_i}^{\text{Out}}$ is the set of channels that server $i$ is streaming out. $\tau^{\max}$ is the maximum streaming rate of these channels (${N_i}^{\text{Out}}$ may be different from $N_i$ since server $i$ can be the leaf of a tree (it will not stream out such channel) or it acts as a helper to relay an unsubscribed stream.).

We illustrate an example of the scheduling delay in Figure 5. Server $i$ is the parent node while others are son node. $b_1$ and $b_2$ are capacities of link $\langle i, j_1\rangle$ and $\langle i, j_2\rangle$, respectively. Channel streams $A$ and $B$ are pushed from server $i$. Servers $j_1$ and $j_2$ subscribe to channel $\{A\}$ and $\{A, B\}$. Hence, child set $\Gamma_i$ of server $i$ is $\{j_1, j_2, j_2\}$ ($j_2$ is calculated repeatedly). In a scheduling period, server $i$ transmits one packet into each edge $\langle i, j_1\rangle$, $\langle i, j_2\rangle$, and $\langle i, j_2\rangle$. Therefore, it transmits three packets totally. The worst-case scheduling delay is the maximum amount of time that the third packet has to wait until it is transmitted out completely, i.e., $d_i^s = L/b_1 + L/b_2 + L/b_2$. Denote the rate of channel $A$ and $B$ as $\tau^{(A)}$ and $\tau^{(B)}$, respectively, then we have $d_i^s \le L/\max\left(\tau^{(A)}, \tau^{(B)}\right)$.

$$D_j^{(n)} = D_i^{(n)} + d_i^s + d_{ij}^p \le D, \quad \forall j \in \mathscr{M}, \forall n \in \mathscr{N}. \tag{10}$$

Equation (10) shows the source-to-end delay constraint. Denote the propagation delay of link $\langle i, j\rangle$ as $d_{ij}^p$ and the source-to-end delay of server $j$ in tree $T^{(n)}$ as $D_j^{(n)}$. $D_j^{(n)}$ is the sum of the source-to-end delay of its parent $i$ in tree $T^{(n)}$, the scheduling delay of $i$ and the propagation delay of link $\langle i, j\rangle$. To ensure quality of service, the source-to-end delay of each node is bounded by a constant value.

*4.4. Joint Optimization Model.* Combining the above model, we formulate our cost optimization problem as follows:

$$\begin{aligned} &\text{Objective}: \min_{\vec{x}, \vec{b}} (13), \\ &\text{subject to}: (9), (10), (11), (12), (15) \text{and} (16). \end{aligned} \tag{11}$$

Our goal is to find overlay trees among MEC servers for each live channel (optimize $\vec{x}$) and allocate the capacity (optimize $\vec{b}$) to minimize the cost while the delay is still under a boundary. However, it is a mixed integer nonlinear programming which is NP-hard [23]. Besides, from Equations (8) and (10) and Figure 5, we find that the source-to-end delay of server $j_1$ can be affected by link capacity $b_2$, even though link $\langle i, j_2\rangle$ does not connect $j_1$. It means there are correlations among different link capacities. When a tree is constructed completely, the scheduling delay is not determined and is affected by other trees which are constructed later. These factors bring difficulties in solving problems. So, we divide the original problem into two subproblems and solve them sequentially in Section 5.

## 5. Algorithm Design

To simplify the problem, we divide the original problem into two subproblems: topology generation (COLS-T) and capacity allocation (COLS-C). In COLS-T, we ignore the scheduling delay and congestion constraints to construct an overlay that meets the delay bound. In COLS-C, we reassign the capacity to each link so as to reach a lower cost based on the aforementioned topology.

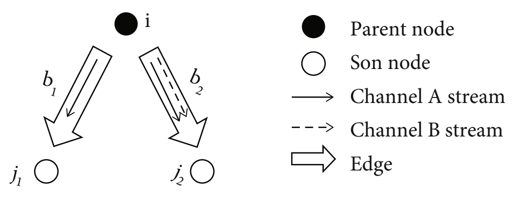

Figure 5: Scheduling delay example.

*5.1. Topology Generation (COLS-T).* In this section, we ignore the scheduling delay and constraint as present in Equation (9), focusing on the propagation delay to construct the tree. Hence, the problem is transformed into how to find a series of Steiner minimum trees [24] under the hop-constraint. We use a greedy scheme to solve it in polynomial time.

There are totally $|\mathscr{N}|$ channel trees. Each tree is initialized and has only a source cloud server. The initial capacity of a tree is its channel rate (In COLS-*C*, we will reallocate the capacity). We expand the tree from the source server to subscribing servers by adding a server into a partially constructed tree in each iteration. We define a metric ($\Delta C_{ij}^{(n)}$ or $\Delta C_{ihj}^{(n)}$) called the **M**arginal **U**nit **C**ost (*MUC*) to determine which server is added. A server has two ways to join in the tree, and their *MUC* is given by:

(i) Server $j$ is directly connected via server $i$ and *MUC* is:

$$\Delta C_{ij}^{(n)} = \begin{cases} \dfrac{c_{ij}\left(t_{ij} + \tau^{(n)}\right) - c_{ij}\left(t_{ij}\right)}{\tau^{(n)}}, D_i^{(n)} + d_{ij}^p \le D \\ +\infty, D_i^{(n)} + d_{ij}^p > D \end{cases}. \tag{12}$$

(ii) Server $j$ is connected through a potential helper $h$ via server $i$. *MUC* is given by:

$$\Delta C_{ihj}^{(n)} = \begin{cases} \Delta C_{ih}^{(n)} + \Delta C_{hj}^{(n)}, D_i^{(n)} + d_{ih}^p + d_{hj}^p \le D \\ +\infty, D_i^{(n)} + d_{ih}^p + d_{hj}^p > D, \end{cases} \tag{13}$$

where $t_{ij}$ is the concurrent throughput of link $\langle i, j\rangle$.

COLS-T is outlined in Algorithm 1. In each iteration, we select link $\langle i, j\rangle$ which incurs the smallest *MUC* and connect the corresponding server $j$ into tree $T^{(n)}$. Then, we update overlay parameters and continue a new iteration until all servers are connected. Finally, we combine all Steiner trees $\{T^{(n)}\}$ into a mesh.

```
1 $T^{(n)} \longleftarrow s^{(n)}, J^{(n)} \longleftarrow \mathscr{P}^{(n)}, \mathscr{H}^{(n)} \longleftarrow \mathscr{P} - \mathscr{P}^{(n)}$;
2 while $\exists n \in \mathscr{N}, J^{(n)} \notin \varnothing$ do
3   foreach$n \in \mathscr{N}, i \in T^{(n)}, j \in J^{(n)}$ do
4     if $\Delta C_{ij}^{(n)} < \Delta C_{ihj}^{(n)}, h \in \mathscr{H}^{(n)}$ then
5       $TC_{ij}^{(n)} \longleftarrow \Delta C_{ij}^{(n)}$
6       $h_{ij}^{(n)} \longleftarrow \varnothing$;
7     else
8       $TC_{ij}^{(n)} \longleftarrow \Delta C_{ihj}^{(n)}$;
9       $h_{ij}^{(n)} \longleftarrow h$
10    end
11  end
12  $i, j, n \longleftarrow \arg\min_{i,j,n} TC_{ij}^{(n)}$;
13    if $h_{ij}^{(n)} \notin \varnothing$ then
14      Add helper $h$, node $j$ into $T^{(n)}$ via node $i$;
15      $\mathscr{H}^{(n)} \longleftarrow \mathscr{H}^{(n)} - h$;
16      $t_{ih} + = \tau^{(n)}$;
17      $t_{hj} + = \tau^{(n)}$;
18    else
19      Add node $j$ into $T^{(n)}$ via node $i$;
20      $t_{ij} + = \tau^{(n)}$;
21    end
22    $J^{(n)} \longleftarrow J^{(n)} - j$;
23 end
```

Algorithm 1: COLS-T

*5.2. Capacity Allocation (COLS-C).* To achieve efficient capacity allocation, we take the scheduling delay into consideration based on the aforementioned overlay $\{T^{(n)}\}$ given by COLS-T. So we reallocate the capacity of each link to make most of the limited capacity and achieve lower cost.

In order to achieve optimal allocation, we first prove the capacity allocation is a convex problem. Then, we take advantage of classical optimization algorithm, sequential least squares quadratic programming (SLSQP) which is widely used to solve the convex optimization problem.

Considering that the overlay topology has been constructed, $\vec{x}$ is constant, i.e., Equations (3), (4), (5), and (6) are always satisfied so we can omit them. In this way, our objective is to find a vector $\vec{b}$ so that (7) get a minimum value subject to constraints (9) and (10).

We prove COLS-C is a convex problem as follows:

(i) Object function (7) is the sum of convex functions. Obviously, (7) is a convex function

(ii) Scheduling delay constraint (9) can be rewritten as:

$$f\left(\vec{b}\right) = \sum_{n}\sum_{k\in\Gamma_i} \frac{L \cdot x_{ik}^{(n)}}{b_{ik}} - \frac{L}{\tau^{\max}} \le 0, \tag{14}$$

where $\Gamma_i$ is the children set of server $i$. $L/\tau^{\max}$ is a constant. The second-order derivative of $f(\vec{b})$ fulfills the condition of being convex:

$$\nabla^2 f\left(\vec{b}\right) \succcurlyeq 0. \tag{15}$$

Hence, $f(\vec{b})$ is a convex function.

(iii) Total delay constraint (10) is similar as (9). It can be rewritten as:

$$g\left(\vec{b}\right) = D_j^{(n)} - D = d_{s^{(n)}j}^{p} + \sum_{i\in\mathscr{A}_j^{(n)}}\sum_{k\in\Gamma_i} \frac{L}{b_{ik}} - D \le 0, \tag{16}$$

where $s^{(n)}$ is the source of channel $n$; $d_{s^{(n)}j}^{p}$ is a constant means the aggregated source-to-end propagation delay of server $j$; $A_j^{(n)}$ is the ancestor set of server $j$; $D$ means the delay upper bound. $g(\vec{b})$ is also a convex function since $\nabla^2 g(\vec{b}) \succcurlyeq 0$.

As demonstrated above, object function (7) and constraints (9) and (10) are all convex, so we have proved that the capacity allocation is a convex problem. Then, we utilize *SLSQP* algorithm (*SLSQP* can be called directly from the library *SciPy*) to seek the minimum solution by iterating over the objective function (7) which denotes the sum of all link capacity costs, while satisfying the delay constraints (9) and (10).

*5.3. Computational Complexity Analysis.* The complexity of COLS is $O(|\mathscr{N}|^2|\mathscr{P}|^3|\mathscr{V}| + |\mathscr{E}|^3)$. In COLS-T step, one server is included into one tree at each round, and there are totally $O(|\mathscr{N}||\mathscr{P}|)$ rounds. In each round, there are $O(|\mathscr{N}||\mathscr{P}||\mathscr{V}|)$ links to be calculated. Each link costs $O(|\mathscr{P}|)$ time to select the helper, hence, adding a node in each round costs $O(|\mathscr{N}||\mathscr{P}|^2|\mathscr{V}|)$ and all rounds cost $O(|\mathscr{N}|^2|\mathscr{P}|^3|\mathscr{V}|)$.

In COLS-C step, we use *SLSQP* to solve the convex problem. It uses the Han-Powell quasi-Newton method with a BFGS update of the B-matrix and L1-test function in the step-length algorithm. It has $O(Q^3)$ overall time complexity where $Q$ is the number of variable [25–27]. There are $|\mathscr{E}|$ variables, so the time complexity is $|\mathscr{E}|^3$. In summary, the overall complexity of COLS-{T + C} is $O(|\mathscr{N}|^2|\mathscr{P}|^3|\mathscr{V}| + |\mathscr{E}|^3)$.

## 6. Illustrative Experiment Results

To evaluate the performance of COLS, we have conducted extensive experiments in two aspects: popularity prediction and mathematical optimization. We present detailed experiment settings and comparison schemes in Section 4.1. Then, we illustrate results in Section 6.2.

*6.1. Simulation Setup.* We compare COLS with following state-of-the-art schemes:

(i) DEEPCACHE [19] builds a LSTM Encoder-Decoder model to predict the popularity of content.

In this network, the dimension of training data is single, and it only has historical playback records

(ii) CCAS [15]. Each MEC server connects the cloud server directly, and there is no collaboration. The MEC server gives up delivering some channels to save the link capacity. The link capacity is a constant in CCAS, and the scheduling delay is not considered. To meet the delay constraint, we adapt CCAS by adding a capacity allocation step. We increase the capacity iteratively with a certain proportion until it meets the constraint

(iii) BSUM [12] constructs an overlay with MEC collaboration. It considers the scheduling delay insufficiently, which ignores the correlation between different link capacities, and hence, the real scheduling delay is higher. Besides, just like CCAS, the link capacity is constant. BSUM only optimizes the topology and does not optimize the capacity. We also add a capacity allocation step which is the same as the step in CCAS

The dataset used for experimental evaluation comes from real scenes, which is provided by the telecom operator. Considering that the size of the source raw data is more than 2 TB including 931964 files (live streaming playback records) [28], we randomly choose a certain number of the records in several consecutive periods of time to construct the dataset used for evaluation. And the detailed baseline parameters are shown in Table 2.

Specifically, experiments are carried out in two parts:

(i) *Popularity prediction*. We compare COLS with DEEPCACHE. We randomly selected 10 sequences as input of designed model and output the corresponding predicted popularity. Mean square error is used to evaluate the prediction performance

(ii) *Cost optimization*. To compare COLS with CCAS and BSUM, we randomly select some MEC servers from the raw data and generate round trip time (RTT, which denotes the propagation delay) matrix among servers. For the selected servers, we use the aforementioned LSTM network to predict the popularity and get the subscribed list of each MEC servers. In order to ensure the accuracy of results, the channel that the MEC server subscribed to in CCAS and BSUM remains the same as COLS. Each scheme is evaluated 20 times and gets an average result. To evaluate the performance of each scheme, we focus on the cost metric, which is the sum of all link capacity costs

*6.2. Simulation Results*. To evaluate the performance of proposed approach, we have conducted extensive experiments. Figures 6 and 7 demonstrate COLS's advantages (MEC collaboration, scheduling delay consideration and capacity convex programming), which make it outperforms other schemes. Figure 6 illustrates the component proportion of maximum server delay before capacity allocation. CCAS does not consider the scheduling delay and BSUM considers insufficiently. Both of them have some missing scheduling delays which are not calculated. All their theoretical delays are lower than the bound but real delays are opposite. To meet the delay constraint (500 ms in this paper), they need to allocate more capacities to reduce the scheduling delay, which bring the cost increases. Just as mentioned above, CCAS makes MEC servers connect the cloud server directly, leading to a higher scheduling delay and a lower propagation delay. On the contrary, COLS and BSUM have collaborations, and thus, their scheduling delays are lower.

Figure 7 depicts maximum server delays after capacity allocation. It is obvious that CCAS reduces most scheduling delay and causes a highest capacity cost. BSUM uses a simple allocation algorithm, and the link capacity is redundant after

Table 2: Baseline parameters.

| Parameters | Value |
|---|---|
| Number of servers | $\|\mathscr{V}\| = 50$ |
| Number of all channels | $\|\mathscr{N}\| = 10$ |
| Length of sliding window in COLS-P | $W = 6$ |
| Selected parameter in COLS-P | $k = 3$ |
| Segment size | $L = 100\text{kb}$ |
| Dalay constraint | $D = 500\text{ms}$ |

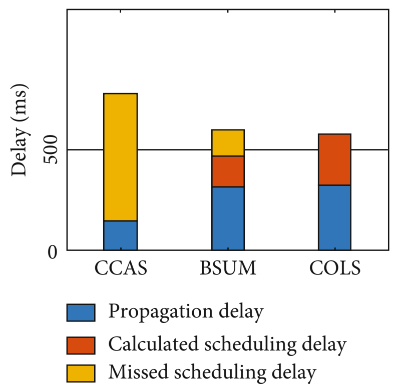

Figure 6: Max delay before reallocation.

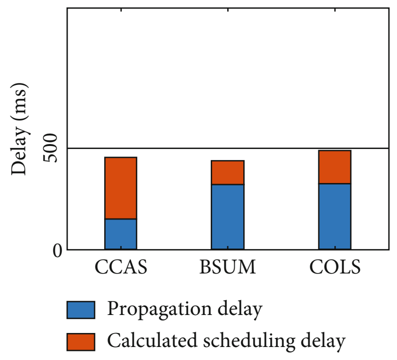

Figure 7: Max delay after reallocation.

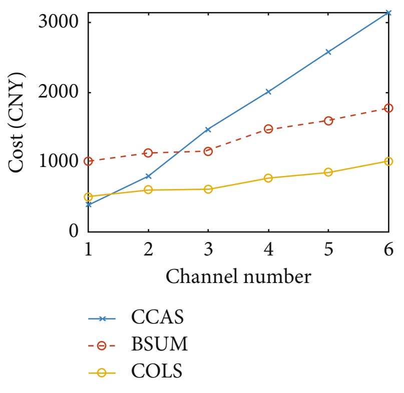

Figure 8: Cost vs. channel number.

the allocation. Hence, the scheduling delay is lower than that of COLS as shown in Figure 7. Compared with other schemes, COLS uses convex programming to allocate capacities, which is more efficiently. It makes capacities less redundant and minimizes the cost while the delay is still under the bound (500 ms).

Figure 8 shows the cost versus the channel number. The results demonstrate that COLS outperforms the other two competing schemes, and the cost of COLS is only around half of BSUM/one-third of CCAS when the channel number reaches 6. CCAS gives up delivering some live channels, i.e., users receive the live stream from the cloud server directly, which increases the cloud server scheduling delay and saves the link capacity. When the channel number is low, the overlay topology is simple, and fewer links connect the cloud server, which provides more growth space for the scheduling delay. However, as the channel number increases, the topology becomes complicated, and more links connect the cloud server. CCAS needs more capacities to reduce the scheduling delay. These capacity costs are higher than costs saved by giving up live channels. Inversely, in this case, COLS has the lowest cost because of MEC collaboration, scheduling delay consideration, and convex programming. It is more suitable for multichannel overlay, which is more common in the reality, i.e., COLS is more practicable than other schemes.

Figure 9 presents the difference results between COLS and DEEPCACHE. The mean prediction error of our method is lower than that of DEEPCACHE. This is because COLS adds more information (hours and weekday) to predict the popularity, making it a more accurate.

Figure 10 illustrates the cost versus the server number. It shows that COLS cost is lowest (outperforms others at least 40%) and increases more slowly than that of competing schemes. The reason is that COLS considers both MEC collaboration, scheduling delay, and efficient capacity allocation. In CCAS, all MEC servers connect the cloud server directly. Section 5 infers that the cloud server connects too much links, leading to a high scheduling delay. To reduce the scheduling delay, CCAS has to increase the link capacity and gets a highest cost. BSUM considers the scheduling delay insufficiently, and the real delay is beyond the bound. It has to allocate more capacity to meet the constraint. The allocate algorithm causes redundant capacities and a higher cost.

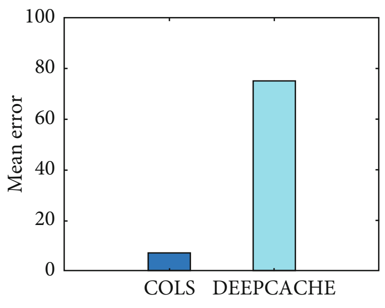

Figure 9: Mean prediction error of popularity.

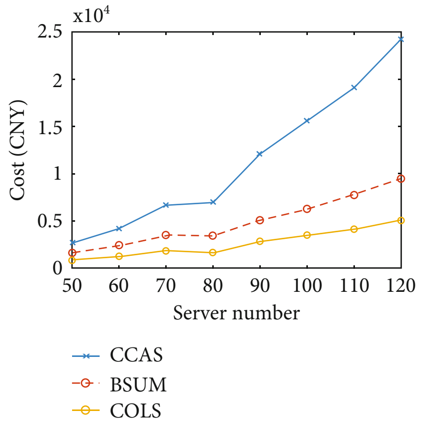

Figure 10: Cost vs. server number.

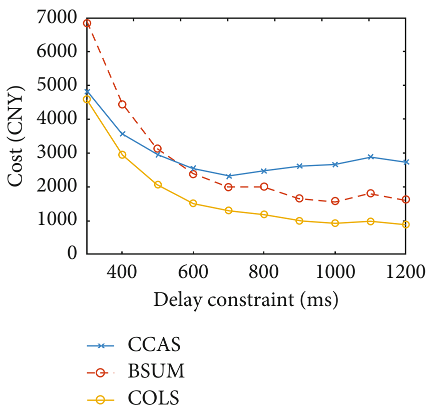

Figure 11: Cost vs. delay constraint.

We plot in Figure 11 the cost versus the delay constraint $D$ with 50 servers. When $D$ decreases, topology graphs constructed by COLS and BSUM gradually become similar as the graph constructed by CCAS, i.e., MEC servers connect the cloud server directly, and there is no MEC collaboration. The reason is that the propagation delay will accumulate and beyond small constraint if there are collaborations. Therefore, CCAS could give up some live channels to get a lower cost. When $D$ increases, all costs of three schemes decrease. COLS has collaborations and convex programming to reduce the cost more quickly than others. In other words, high-cost reductions are achieved by sacrificing a small amount of delay in COLS.

## 7. Conclusion

In this paper, we study the cost optimization of an overlay MEC network for live streaming. Proposed network has a more realistic delaying and topological model, where MEC servers collaborate with each other to delivery streaming. We formulate the problem and propose a framework called COLS, which predicts the popularity by LSTM model and solves optimization problem by greedy scheme and convex programming. Simulation results show that COLS has higher prediction accuracy, reduces the capacity cost by at least 40% compared with state-of-the-art schemes.

## Data Availability

The source data used to support the findings of this study are currently under embargo while the research findings are commercialized. Requests for data, 6 months after publication of this article, will be considered by the corresponding author.

## Disclosure

The abstract was presented at the 8th International Conference on Digital Home (ICDH) 2020.

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## Acknowledgments

This work was supported by Guangxi Innovation Driven Development Special Fund Project (AA18118039).

## References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] R. Pepper, *Cisco Visual Networking Index (VNI) Global Mobile Data Traffic Forecast Update*, Cisco, Tech. Rep., 2013.

[3] H. Lu, C. Gu, F. Luo, W. Ding, and X. Liu, "Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning," *Future Generation Computer Systems*, vol. 102, pp. 847–861, 2020.

[4] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: a survey of the emerging 5g network edge cloud architecture and orchestration," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[5] K. Bilal and A. Erbad, "Edge computing for interactive media and video streaming," in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pp. 68–73, Valencia, Spain, 2017.

[6] P. Yuan, Y. Cai, X. Huang, S. Tang, and X. Zhao, "Collaboration improves the capacity of mobile edge computing," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10610–10619, 2019.

[7] A. Ndikumana, S. Ullah, T. LeAnh, N. H. Tran, and C. S. Hong, "Collaborative cache allocation and computation offloading in mobile edge computing," in *2017 19th AsiaPacific Network Operations and Management Symposium (APNOMS)*, pp. 366–369, Seoul, Republic of Korea, September 2017.

[8] P. Jing, Y. Su, L. Nie, X. Bai, J. Liu, and M. Wang, "Low-rank multi-view embedding learning for micro-video popularity prediction," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 8, pp. 1519–1532, 2017.

[9] J. Xie, Y. Zhu, Z. Zhang et al., "A multimodal variational encoder-decoder framework for micro-video popularity prediction," *Proceedings of The Web Conference*, vol. 2020, pp. 2542–2548, 2020.

[10] Q. Fan and N. Ansari, "On cost aware cloudlet placement for mobile edge computing," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 4, pp. 926–937, 2019.

[11] L. Tang, Q. Huang, A. Puntambekar, Y. Vigfusson, W. Lloyd, and K. Li, "Popularity prediction of facebook videos for higher quality streaming," in *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)*, pp. 111–123, Santa Clara, CA, USA, 2017.

[12] A. Ndikumana, N. H. Tran, T. M. Ho et al., "Joint communication, computation, caching, and control in big data multi-access edge computing," *IEEE Transactions on Mobile Computing*, vol. 19, no. 6, pp. 1359–1374, 2020.

[13] D. Zhang, L. Tan, J. Ren et al., "Near-optimal and truthful online auction for computation offloading in green edge-computing systems," *IEEE Transactions on Mobile Computing*, vol. 19, no. 4, pp. 880–893, 2020.

[14] Z. Zheng, L. Song, Z. Han, G. Y. Li, and H. V. Poor, "A stackelberg game approach to proactive caching in large-scale mobile edge networks," *IEEE Transactions on Wireless Communications*, vol. 17, no. 8, pp. 5198–5211, 2018.

[15] Y. Hung, C. Wang, and R. Hwang, "Optimizing social welfare of live video streaming services in mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 19, no. 4, pp. 922–934, 2020.

[16] Z. Zhang, R. Wang, F. R. Yu, F. Fu, and Q. Yan, "QoS aware transcoding for live streaming in edge-clouds aided hetnets: an enhanced actor-critic approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 11, pp. 11295–11308, 2019.

[17] Y. Hung, C. Wang, and R. Hwang, "Combinatorial clock auction for live video streaming in mobile edge computing," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 196–201, Honolulu, HI, USA, 2018.

[18] W. Chen, P. Chou, C. Wang, R. Hwang, and W. Chen, "Live video streaming with joint user association and caching placement in mobile edge computing," in *2020 International Conference on Computing, Networking and Communications (ICNC)*, pp. 796–801, Big Island, HI, USA, 2020.

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

[20] A.-T. Tran, N.-N. Dao, and S. Cho, "Bitrate adaptation for video streaming services in edge caching systems," *IEEE Access*, vol. 8, pp. 135844–135852, 2020.

[21] N.-N. Dao, D. T. Ngo, N.-T. Dinh et al., "Hit ratio and content quality tradeoff for adaptive bitrate streaming in edge caching systems," *IEEE Systems Journal*, vol. 15, no. 4, pp. 1–4, 2020.

[22] J. Dai, Z. Chang, and S.-H. G. Chan, "Delay optimization for multi-source multi-channel overlay live streaming," in *2015 IEEE international conference on communications (ICC)*, pp. 6959–6964, London, UK, 2015.

[23] J. Hartmanis, "Computers and intractability: a guide to the theory of np-completeness (Michael R. Garey and David S. Johnson)," *SIAM Review*, vol. 24, no. 1, pp. 90-91, 1982.

[24] E. N. Gilbert and H. O. Pollak, "Steiner minimal trees," *SIAM Journal on Applied Mathematics*, vol. 16, no. 1, pp. 1–29, 1968.

[25] D. Kraft, "Algorithm 733: TOMP–Fortran modules for optimal control calculations," *ACM Transactions on Mathematical Software (TOMS)*, vol. 20, no. 3, pp. 262–281, 1994.

[26] M. M. Alves, R. Monteiro, and B. F. Svaiter, *Primaldual Regularized SQP and SQCQP Type Methods for Convex Programming and Their Complexity Analysis*, Preprint, 2014.

[27] D. Kraft, *A Software Package for Sequential Quadratic Programming*, Open Grey, 1988.

[28] N. Liu, H. Cui, S.-H. G. Chan, Z. Chen, and Y. Zhuang, "Dissecting user behaviors for a simultaneous live and vod iptv system," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 10, no. 3, pp. 1–16, 2014.