MultiLive: Adaptive Bitrate Control for Low-delay Multi-party Interactive Live Streaming

Ziyi Wang*, Yong Cui*, Xiaoyu Hu*, Xin Wang†, Wei Tsang Ooi‡, Yi Li§
*Department of Computer Science and Technology, Tsinghua University, China
†Department of Electrical and Computer Engineering, State University of New York at Stony Brook, USA
‡School of Computing, National University of Singapore, Singapore
§PowerInfo Co., Ltd., China

Abstract*—In multi-party interactive live streaming, each user can act as both the sender and the receiver of a live video stream. Designing adaptive bitrate (ABR) algorithm for such applications poses three challenges: (i) due to the interaction requirement among the users, the playback buffer has to be kept small to reduce the end-to-end delay; (ii) the algorithm needs to decide what is the bitrate to receive and what is the set of bitrates to *send*; (iii) the delay and quality requirements between each pair of users may differ, for instance, depending on whether the pair is interacting directly with each other. To address these challenges, we first develop a quality of experience (QoE) model for multi-party live streaming applications. Based on this model, we design *MultiLive*, an adaptive bitrate control algorithm for the multi-party scenario. MultiLive models the many-to-many ABR selection problem as a non-linear programming problem. Solving the non-linear programming equation yields the target bitrate for each pair of sender-receiver. To alleviate system errors during the modeling and measurement process, we update the target bitrate through the buffer feedback adjustment. To address the throughput limitation of the uplink, we cluster the ideal streams into a few groups, and aggregate these streams through scalable video coding for transmissions. We conduct extensive trace-driven simulations to evaluate the algorithm. The experimental results show that MultiLive outperforms the fixed bitrate algorithm, with 2-5× improvement in average QoE. Furthermore, the end-to-end delay is reduced to about 100 ms, much lower than the 400 ms threshold recommended for video conferencing.

***Index Terms*—Multi-party interactive live streaming, Adaptive bitrate control**

I. INTRODUCTION

Live streaming platforms, such as YouTube Live and Twitter's Periscope, have attracted millions of daily active users [1], [2], [3]. Since user engagement increases the revenue, these platform providers are increasingly interested in supporting interactive live streaming experience for their users, leading to *multi-party interactive live video streaming* as an emerging class of applications. In such an application, a user not only acts as a source of video, but also receives one or more streams from other users in the same session simultaneously. An example is collaborative talent show, where various geographically distributed online streamers perform the arts of singing, acting, playing instruments, or other activities together and interact with each other by exchanging streams [4], [5]. Platforms that support such application (e.g., Inke.tv[1] and Douyu.tv[2]) have attracted hundreds of millions of users in recent years.

Three challenges arise from this new class of applications. First, applications such as collaborative talent show require a much tighter synchronization among the users. Schuett [6] reported a delay above 30ms would disrupt the tempo, but a delay of up to 70ms can be tolerated. This delay requirement is stricter than other multi-party live applications, such as multi-party video conferencing, where some existing work has achieved a tolerable delay of around 400 ms [7]. Such low tolerance to end-to-end delay means that the buffer has to be kept small at the sender, the server, and the receiver, increasing the chances of buffer underflow and stalls in the presence of network jitters and inaccurate throughput estimation.

Second, given the heterogeneity of the devices and the network conditions of the users, it is important for a receiver to receive a stream at a bitrate that is best suited for its requirement to maximize its quality of experience (QoE). There are many existing studies on receiver-driven adaptive bitrate (ABR) algorithms [8], [9], [10], [11], [12], [13] that address this question. Since each receiver is also a sender, however, a new question that arises here is: *at what bitrates should each sender encode its video stream to meet the requirement of the receivers?* Many existing solutions, in the context of multi-party video conferencing, call for the use of a transcoding server (e.g., [14], [15], [16], [17]), in which case the sender only needs to send a single stream at a high-enough bitrate and the transcoding server transcodes the stream to the required bitrate for the receiver. Such solution, however, requires additional computation in the cloud and increases the infrastructure cost. Furthermore, the transcoding step introduces additional delay. We therefore consider the scenario where the server only relays the stream without transcoding and the sender encodes the video into multiple streams at the bitrates required by the receivers. Since the uplink of a sender is likely a bottleneck, a sender can only generate a limited number of streams and may not meet the needs of every receiver.

Third, for a receiver, the delay and quality requirements may differ for each sender. For instance, in a collaborative talent show, the end-to-end delay between a performer and a spectator can be higher than between two performers; This spectator may require higher video quality from the performers than that of other spectators.

[1] https://www.inke.com
[2] https://www.douyu.com

In this paper, we present a system for multi-party live streaming to address the challenges above. Our system has the following salient features. First, to minimize delay and stall duration, the streamer can increase or decrease the playback speed. Second, to more effectively utilize the uplink bandwidth, we adopt scalable video coding (SVC) to aggregate multi-rate streams sent from a sender. The server can then distribute different layers to different receivers. Third, we develop a QoE model integrates the different considerations of multi-party interactive live streaming applications. By assigning different weights to different terms, the system is able to personalize the preferences between each pair of users. Finally, we introduce the core component to this system, MultiLive, which is an adaptive bitrate control algorithm that is run centrally in the server, considering the various constraints in a many-to-many scenario as a non-linear programming problem that maximizes the overall QoE. MultiLive periodically solves the non-linear programming problem to obtain the target bitrate for each sender and each receiver and, in the interim, uses a feedback control algorithm to adjust the target bitrate to react to fluctuating throughput and variations in video encoding rate. The target bitrates are clustered into what the senders and the server actually send while minimizing QoE loss. To the best of our knowledge, we are the first to design and implement an adaptive bitrate control algorithm for multi-party interactive live streaming that considers the many-to-many ABR problem while maximizing the QoE considering delay, smoothness, quality, and stall, holistically.

We conducted extensive trace-driven simulations to evaluate the algorithm. We collected and released, as an open dataset, more than 72 hours of uplink and downlink throughput measurements from live streaming servers. In addition, Belgium 4G/LTE dataset [18] is used to test the performance. The results show that MultiLive outperforms the fixed bitrate algorithm, with 2-5× improvement in average QoE. Furthermore, the delay has been reduced to about 100 ms.

The rest of the paper is structured as follows. Section II presents the system architecture. Section III formulates the problem. Section IV elaborates the details of MultiLive. Simulation results are presented in Section V. Section VI discusses the related work. Finally, we conclude the paper in Section VII.

II. System Architecture

We first present the architecture of our system (See Fig. 1). The three major logical entities are: the senders, the server, and the receivers. We only focus on the senders and receivers that participate in the same live streaming session here. In the current multi-party interactive live streaming applications, the number of streamers in a session is in the order of tens or less. Note that, despite our distinction of senders and receivers, in practice, the same streamer acts as both a sender and a receiver.

Each sender generates an SVC-coded, multi-rate, video stream and transmits it to the server. We adopt a *push*-based approach, where a frame is sent as soon as it is generated. This approach avoids the request-response overhead used in pull-based approach, commonly used in DASH-based video-on-demand streaming.

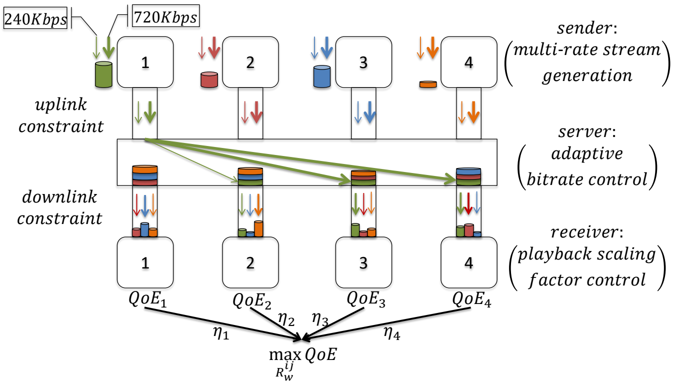

Fig. 1. The architecture of multi-party interactive live streaming.

Upon receiving a frame, the server buffers it and relays the appropriate SVC layers of the frame to each receiver, also using the push-based approach. The decisions of: (i) what is the set of bitrates that each sender should produce, (ii) what is the bitrate that each receiver should receive, are determined by the server through an *adaptive bitrate controller*. The adaptive bitrate controller takes, as inputs, the uplink throughput of each sender, the downlink throughput of each receiver, the state of buffer occupancy at each receiver, and makes these decisions to maximize the total QoE of each receiver. The server also additionally collects the state information (buffer occupancy, downlink throughput, receiver's preferences for each sender) from each receiver and disseminates its decision (the set of bitrates) to the corresponding sender.

The receiver maintains a playback buffer for each sender. This partitioning of frames from each sender allows the receiver to manage the priorities and preferences across the senders (e.g., higher quality for a sender, lower delay for another). During playback, the receiver may also adjust the playback speed. When the buffer occupancy rises above a high threshold, the receiver plays back the video at a faster rate to “catch up” and reduce the end-to-end delay of subsequent frames. When the buffer occupancy falls below a low threshold, the receiver plays back the video at a slower rate, to delay the onset of stalls and thus reducing the duration of stall.

As noted above, we adopt a frame-level granularity in transmission. This approach is commonly used in DASH-based live streaming system using chunk-encoding in Common Media Application Format (CMAF). Transmitting, buffering, and playing back at a frame-level granularity, instead of segment-level granularity commonly used in video-on-demand systems, keeps the end-to-end delay small.

To summarize our design decisions, our system architecture supports the following objectives: First, by processing at a frame-level granularity, adopting push-based transmission, increasing playback speed when needed, and avoiding transcoding at the server, the delay is kept small. Second, using SVC, the sender sends only a single stream to meet the bandwidth requirements of multiple heterogeneous receivers. Finally, by optimizing the QoE (through finding the best bitrate configuration), we maintain a high quality of experience for the users. This final component is the focus for the rest of this paper. In the next section, we will first present some preliminaries and state the optimization problem. This presentation is followed by Section IV, where we will present how the system optimizes the QoE.

III. Problem Formulation

In this section, we present a model of the network constraint, buffer occupancy, and QoE. We end this section with a statement of the optimization problem to be solved at the server to maximize the QoE. To facilitate our presentation, the major notations are summarized in Table I.

A. Network Constraint

Let $C_w^{up(i)}$ denote the average uplink throughput when sender i sends the w-*th* frame and $C_w^{down(j)}$ denote the average downlink throughput when receiver j receives the w-*th* frame. Let R_w^{ij} denote the bitrate of the w-*th* frame from sender i to receiver j.

Considering the uplink of the sender i, the total throughput of the streams it generates must be less than the limit of the uplink throughput after aggregating the streams with SVC. Considering the characteristics of clustering and SVC, the constraints of the uplink can be relaxed as:

$$\max_j \left\{R_w^{ij}\right\} \le C_w^{up(i)}. \tag{1}$$

Similarly, considering the downlink of the receiver j, the sum of the bitrates of several streams it receives must be less than the total downlink throughput:

$$\sum_i R_w^{ij} \le C_w^{down(j)}. \tag{2}$$

B. Buffer Model

The buffer in this paper refers to the queue of video frames to be consumed, used mainly to cope with network jitters. The senders, server, and receivers each have one or more buffers. Among them, the receiver's buffers are directly related to the playback condition, and are more important for the QoE of the receiver. So we focus on the receiver's buffers in our model.

The receiver's playback buffer contains video frames that have been received but yet to be played back. We let $B_w^{down(ij)}$ be the receiver buffer occupancy (in unit of time) when receiver j starts to receive the w-*th* frame of sender i. Let $C_w^{down(ij)}$ be the average uplink throughput when sender i sends the w-*th* frame of i. Let S_w^{ij} be the size of the w-*th* frame and D_w^{ij} be the duration of the w-*th* frame. The time taken to fully receive the w-*th* frame is $S_w^{ij}/C_w^{down(ij)}$. The buffer occupancy increases by D_w^{ij} seconds after the w-*th* frame is received and decreases as the receiver plays back the video.

TABLE I
Major notations used in this paper.

Notation	Description
$C_w^{up(i)}$	Average uplink throughput when i sends the w-*th* frame
$C_w^{down(j)}$	Average downlink throughput when j receives the w-*th* frame
$C_w^{down(ij)}$	Average downlink throughput when j receives the w-*th* frame of i
$B_w^{down(ij)}$	Current receiver buffer occupancy when j starts to receive the w-*th* frame of i
$\hat{B}_w^{down(ij)}$	Target receiver buffer occupancy when j starts to receive the w-*th* frame of i
$B_{fast}^{down(ij)}$	Buffer occupancy threshold of fast playback from i to j
$B_{slow}^{down(ij)}$	Buffer occupancy threshold of slow playback from i to j
$T_w^{up(i)}$	System time when i starts to generate the w-*th* frame
$T_w^{down(ij)}$	System time when j starts to receive the w-*th* frame of i
R_w^{ij}	The real bitrate of the w-*th* frame from i to j
$\hat{R}_w^{ij}$	The target bitrate of the w-*th* frame from i to j
S_w^{ij}	The size of the w-*th* frame from i to j
D_w^{ij}	The duration of the w-*th* frame from i to j
P^{ij}	The playback scaling factor from i to j
Q_K^{ij}	Cumulative video quality of K frames from i to j
V_K^{ij}	Cumulative video quality variations of K frames from i to j
E_K^{ij}	Cumulative rebuffer duration of K frames from i to j
L_K^{ij}	Delay of K frames from i to j

So the evolution of the receiver buffer occupancy level can be derived as:

$$\hat{B}_{w+1}^{down(ij)} = \left(B_w^{down(ij)} - \frac{S_w^{ij}}{C_w^{down(ij)}}\right)_+ + D_w^{ij}, \tag{3}$$

where $\hat{B}_{w+1}^{down(ij)}$ is the target buffer occupancy when receiver j starts to receive the next frame and $(x)_+ = max\{x, 0\}$. Note that if $B_w^{down(ij)} < S_w^{ij}/C_w^{down(ij)}$, the buffer will become empty while the receiver is still receiving the w-*th* frame, leading to stalls (i.e., the receiver's playback buffer does not have frames to render). So the first term in the equation above cannot be negative.

In addition, we consider adjusting the playback speed to achieve fine-grained delay control, according to the current buffer occupancy $B_w^{down(ij)}$ and two thresholds: $B_{fast}^{down(ij)}$ and $B_{slow}^{down(ij)}$. When the buffer occupancy is more than the fast playback threshold, $B_w^{down(ij)} > B_{fast}^{down(ij)}$, the receiver plays back faster to reduce the delay; When the buffer occupancy is less than the slow playback threshold, $B_w^{down(ij)} < B_{slow}^{down(ij)}$, the receiver slows down the playback to alleviate stall. Let P^{ij} be the scaling factor that controls the video playing speed from the sender i to the receiver j, with $P^{ij} = 1$ means normal playback. Then the actual duration of video to playback per second is P^{ij}. The duration of video received from the downlink per second is $C_w^{down(ij)}/R_w^{ij}$. The net consumption rate of video duration is the difference between the two values. After fast/slow playback, the video delay changes from L_w^{ij} to $L_{w'}^{ij}$. The catch-up time we need is $(L_w^{ij} - L_{w'}^{ij})/(P^{ij} - 1)$. Multiplying the catch-up time by the net consumption rate is the amount of change in the buffer. So the evolution of the receiver buffer occupancy level considering playback speed adjustment ($P^{ij} \neq 1$) can be derived as:

$$\hat{B}_{w'}^{down(ij)} = B_w^{down(ij)} - \left(P^{ij} - \frac{C_w^{down(ij)}}{R_w^{ij}} \right) \cdot \frac{L_w^{ij} - L_{w'}^{ij}}{P^{ij} - 1}. \tag{4}$$

Eq. (4) is used to predict the buffer occupancy only if the fast/slow playback occurs. The playback scaling factor P^{ij} can be flexibly selected according to actual conditions. In the case of normal playback, Eq. (3) is used.

C. QoE Model

In multi-party interactive live streaming, the overall session QoE should be considered at two different levels: (i) the session QoE should be the weighted sum of each receiver's QoE, and (ii) a receiver's QoE then depends on the factors in which it receives data from others. We mainly refer to the QoE defined by Yin *et al.* [12] and Ahmed *et al.* [19]. Specifically, each receiver's QoE includes the following four aspects:

(1) Cumulative video quality Q_K^{ij}: Let $q(\cdot)$ be a non-decreasing function that maps bitrate R_w^{ij} to the perceived video quality $q(R_w^{ij})$. Then the cumulative video quality of K consecutive frames from sender i to receiver j is:

$$Q_K^{ij} = \sum_{w=1}^{K} q(R_w^{ij}). \tag{5}$$

(2) Cumulative video quality variations V_K^{ij}: From the receiver's perspective, frequent bitrate switching is undesirable. Therefore, the QoE model should add video quality variations as penalty. The cumulative video quality variations of K consecutive frames sent from sender i to receiver j is:

$$V_K^{ij} = \sum_{w=1}^{K-1} |q(R_{w+1}^{ij}) - q(R_w^{ij})|. \tag{6}$$

(3) Cumulative stall duration E_K^{ij}: When the buffer is drained out, a stall occurs and deteriorates the receiver's QoE. Therefore, the stall duration should also be a penalty in the QoE model. The cumulative stall duration of K consecutive frames sent from sender i to receiver j is:

$$E_K^{ij} = \sum_{w=1}^{K} \left(\frac{S_w^{ij}}{C_w^{down(ij)}} - B_w^{down(ij)} \right)_+ . \tag{7}$$

(4) Delay L_K^{ij}: Video-on-demand (VoD) streaming has a more relaxed requirement of delay and can use a large playback buffer, whereas live streaming cannot. To maintain interactivity, the most important requirement is low delay [20]. Therefore, the QoE model should also add delay as penalty. Let $T_w^{up(i)}$ denote the system time when sender i starts to generate the *w-th* frame and $T_w^{down(ij)}$ denote the system time when receiver j starts to receive the *w-th* frame of sender i. Then the delay of after sending K consecutive frames from sender i to receiver j is:

$$L_K^{ij} = T_K^{down(ij)} - T_K^{up(i)} + B_K^{down(ij)}. \tag{8}$$

Since receivers have different preferences for the above four aspects, we define the receiver j's QoE as the weighted sum of the above four aspects, namely:

$$QoE_j = \sum_i \left(\alpha_{ij} Q_K^{ij} - \beta_{ij} V_K^{ij} - \gamma_{ij} E_K^{ij} - \delta_{ij} L_K^{ij} \right), \tag{9}$$

where $\alpha_{ij}, \beta_{ij}, \gamma_{ij}$, and δ_{ij} are the weights of the different QoE terms between sender i and receiver j. Note that these weights are per sender-receiver pair, allowing each receiver to personalize its QoE preference to different senders depending on the amount of interaction needed and the context of the application. Finally, the overall session QoE is the weighted sum of all receivers' QoE:

$$QoE = \sum_j \eta_j QoE_j, \tag{10}$$

where η_j is the weight of the *j-th* receiver.

The server obtains global state information and calculates the bitrates that each sender should produce. Then it informs senders of these information. Senders generate the streams as required. So the problem is, to maximize the global QoE, how to design this adaptive bitrate control algorithm running on the server? That is, given the current buffer occupancy and uplink/downlink throughput prediction, how many streams are generated by each sender and what are their real bitrates R_w^{ij}? The problem can be formulated as:

$$\begin{aligned} &\max_{R_w^{ij}} \; QoE \\ &s.t. \; (1) \; (2) \; (3) \; (4) \end{aligned} \tag{11}$$

IV. Algorithm Design

In this section, we first give an overview of the algorithm, namely MultiLive. Then we elaborate the solution in three subsections, including non-linear programming solution, buffer feedback adjustment, and bitrate clustering.

A. Design Overview

The MultiLive algorithm workflow is shown in Fig. 2. To find the number of streams and the bitrate of each stream R_w^{ij} a sender i can transmit to the receiver j, we split the solution into two steps. First, we calculate the target bitrate $\hat{R}_w^{ij}$ for each pair of sender-receiver. Both the receivers and senders have to jointly decide which bitrate to produce and which bitrate to receive. Specifically, we build a non-linear

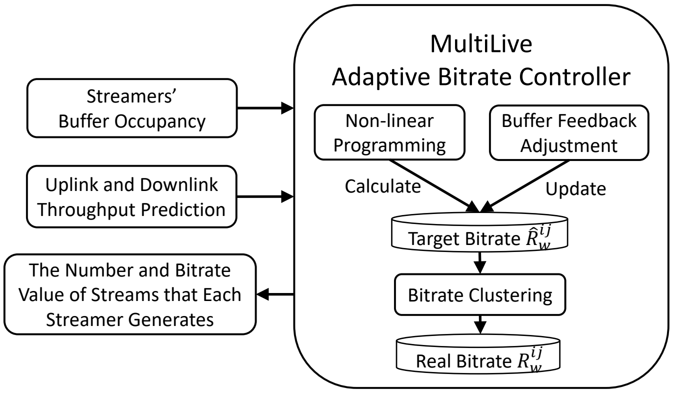
MultiLive
Adaptive Bitrate Controller
Streamers' Buffer Occupancy
Uplink and Downlink Throughput Prediction
The Number and Bitrate Value of Streams that Each Streamer Generates
Non-linear Programming
Buffer Feedback Adjustment
Calculate
Update
Target Bitrate $\hat{R}_w^{ij}$
Bitrate Clustering
Real Bitrate R_w^{ij}

Fig. 2. The algorithm workflow of adaptive bitrate controller.

programming solution to get the target bitrate $\hat{R}_w^{ij}$. It is also updated through the buffer feedback adjustment to alleviate system errors. Second, we cluster and aggregate the target bitrate $\hat{R}_w^{ij}$ to the real bitrate R_w^{ij} to transmit according to the SVC requirements.

B. Non-linear Programming (NLP)

In a multi-party scenario, finding the target bitrate is an optimization problem that takes into account various constraints from a global perspective, as we have formulated in Eq. (11). Previous studies [21], [22] show that as the bitrate increases, the rate of increase in video quality score decreases. In other words, there is no linear correlation between the bitrate of a video stream and its perceptual quality. Guo *et al.* [16] used logarithmic function to characterize the relationship between video quality and video bitrate. Based on these studies, the video quality in our QoE objective function is set to be logarithmic. Also, the target bitrate is calculated on a continuous domain rather than discrete.

For a constrained non-linear programming problem, the constraints can be converted to penalty to turn the problem into an unconstrained one, which is then solved by the gradient descent method. Using this approach, we build a non-linear programming (NLP) solution to solve Eq. (11) and get preliminary result (target bitrate $\hat{R}_w^{ij}$). Although the problem can be solved in polynomial time complexity, it is not fast enough for a real-time update. So we need a complementary approach to calculate the target bitrate faster. Furthermore, the input parameters are not always accurate due to factors, such as inaccurate throughput estimates and fluctuations in video encoding rate. Therefore, we introduce an additional step, buffer feedback adjustment, which updates the target bitrate based on the buffer state to alleviate the input errors. We will introduce it in the next section.

C. Buffer Feedback Adjustment (BFA)

Since the change of buffer occupancy reflects the change of throughput, we can make some feedback adjustments to the target bitrate $\hat{R}_w^{ij}$. In this way, we may reduce the throughput prediction errors and make target bitrate more accurate. In Eq. (3), given the buffer and throughput when the receiver starts to receive the w-*th* frame, we can estimate that the buffer occupancy of the (w+1)-*th* frame when the throughput is unchanged. However, at the moment when the receiver actually starts to receive the (w+1)-*th* frame, we can get the actual buffer occupancy. The difference between the target value and the real value reflects the change rate of the throughput. It determines the range of target bitrate adjustment.

To perform the adjustment, we refer to the PID controller [23], a widely used feedback control technique. It includes proportional controller, integral controller and derivative controller. It monitors the error value e_t, which is the difference between the target value and the real value. Then it can output the control signal u_t:

$$u_t = K_p e_t + K_i \int_0^t e_\tau d\tau + K_d \frac{de_t}{dt}, \tag{12}$$

where the three parameters K_p, K_i, and K_d represent the coefficients for the proportional, integral, and derivative terms respectively. The derivative term is sensitive to the measurement noise [24]. So we make some modifications to suit our specific scenario. In our control policy, the parameter for the derivative control K_d equals zero. So strictly speaking, our controller is a PI controller. The remaining two terms are as follows:

(1) Proportional controller calculates the difference between the real buffer and the target buffer to alleviate the prediction errors. We use Z_p to represent this term:

$$Z_p = K_p \left(B_{w+1}^{down(ij)} - \hat{B}_{w+1}^{down(ij)} \right). \tag{13}$$

(2) Integral controller integrates the difference between the real buffer and the target buffer to alleviate cumulative system errors. We use Z_i to represent this term:

$$Z_i = K_i \int_0^{w+1} \left(B_t^{down(ij)} - \hat{B}_t^{down(ij)} \right) dt. \tag{14}$$

Therefore, we obtain the updated target bitrate value based on the following buffer feedback adjustment (BFA):

$$\hat{R}_{w+1}^{ij} = \hat{R}_w^{ij} + Z_p + Z_i. \tag{15}$$

D. Bitrate Clustering

Through the non-linear programming solution and the buffer feedback adjustment, we can get the target bitrate $\hat{R}_w^{ij}$ for each pair of sender-receiver. The sender, however, may not have the encoding capacity or uplink bandwidth to encode and transmit at each of these target bitrates. To alleviate the problem, the server clusters the target bitrates $\hat{R}_w^{ij}$ to obtain the actual bitrates R_w^{ij}, according to which the sender produces the sub-streams and uses the SVC to aggregate them into one stream. This approach reduces the overhead of encoding and transmission.

In the process of clustering, we define *QoE loss* as the difference between the QoE value of the target bitrate and the QoE value of the cluster centroid bitrate. The ideal situation is that each sender produces an SVC stream that

Algorithm 1 Bitrate Clustering

```
Input: $N$: the number of senders;
    $\hat{R}_w^{ij}$: the target bitrate of the $w$-$th$ frame from $i$ to $j$;
    $m_i$: the number of streams generated by sender $i$
Output: $\mu_1^i, \mu_2^i \cdots \mu_{m_i}^i$: the cluster centroids of sender $i$
Initialize cluster centroids $\mu_1^i, \mu_2^i \cdots \mu_{m_i}^i$ randomly
for $i = 1$ to $N$ do
    repeat
        for $j = 1$ to $N$ and $j \neq i$ do
            $class(\hat{R}_w^{ij}) \leftarrow \arg\min_k |QoE(\hat{R}_w^{ij}) - QoE(\mu_k^i)|$
        end for
        for $k = 1$ to $m_i$ do
            $\mu_k^i \leftarrow \frac{\sum_j \mathbb{1}\left(class\left(\hat{R}_w^{ij}\right)=k\right)\hat{R}_w^{ij}}{\sum_j \mathbb{1}\left(class\left(\hat{R}_w^{ij}\right)=k\right)}$
        end for
    until convergence
end for
return $\mu_1^i, \mu_2^i \cdots \mu_{m_i}^i$
```

minimizes overall QoE loss of receivers. We use the K-means clustering algorithm for clustering. The details are shown in Algorithm 1. Ideally, if the cluster centroids in the last two consecutive iterations are the same, the algorithm is said to have converged. But in practice, we use a less strict criteria for convergence: Given a threshold σ, for the cluster centroid μ_k^i in an iterative process, if the $(\mu_k^i)'$ produced by the next iteration satisfies $\sigma < \frac{(\mu_k^i)'}{\mu_k^i} < \frac{1}{\sigma}$, then we consider the algorithm to be converged. The returned $\mu_1^i, \mu_2^i \cdots \mu_{m_i}^i$ (the cluster centroids of sender i) are the actual bitrates that sender i needs to encode the video into. The server notifies senders of these information. Then the senders generate streams as required.

V. Evaluation

We conducted extensive trace-driven simulations to evaluate our method. To obtain the throughput traces from an actual deployed live streaming service, we collected the uplink and downlink data from three geographically distributed live streaming servers for more than 72 hours[3]. We refer to this as the *Commercial Dataset*. We also use the Belgium 4G/LTE dataset [18] in the evaluation. This dataset consists of throughput measurements in 4G networks along several paths in and around the city of Ghent, Belgium. We distributed the two types of traces to five streamers in the simulator respectively and generated a frame sequence at a rate of 30 frames per second. Since developing a good throughput predictor is not the focus of this paper, we use the harmonic mean of the observed throughput of the last 100 frames to predict the next throughput value, following a previous study [24].

The parameters we used in our experiments are as follows. We set the buffer thresholds for faster and slower playback, $B_{fast}^{down(ij)}$ and $B_{slow}^{down(ij)}$ to 90 ms and 30 ms uniformly for each sender i and each receiver j. When adjusting the playback scaling factor P^{ij}, we either play at 2× (for faster) or 0.5× (for slower) the normal playback speed. In addition, we set the number of clusters used in K-means algorithms to 2, considering there are only five streamers in our settings. For the QoE model, we set $\alpha_{ij} = 1, \beta_{ij} = 1, \gamma_{ij} = 1$, and $\delta_{ij} = 20$ (strict requirement for delay term), for all i and j, except for $\alpha_{0,2} = 0.6$ and $\delta_{0,2} = 28$ (prefers lower quality but lower delay); $\alpha_{1,3} = 1.2$ and $\delta_{1,3} = 16$ (prefers higher quality but higher delay) to illustrate the different preferences for receivers 2 and 3.

A. Parameter Choice of PI Controller

In our first experiment, we study the sensitivity of the method to the parameters K_p and K_i of the PI controller. Since the PI controller is used to adjust the buffer feedback thus the target bitrate, if parameters K_p and K_i are sensitive to the network environment, tuning them will require more efforts. So we want to explore whether there exist a set of K_p and K_i values that work well in a wide range of network conditions. We consider the network throughput from the 72 hours traces separately. For the *k-th* hour network trace, we vary the values of K_p and K_i in a large range to obtain the corresponding QoE values. Then we consider all the network traces and accumulate the QoE values. Fig. 3 shows the heat map with the heat for each pair of K_p and K_i values as the average QoE value. A larger heat value means that it leads to good performance for more traces of network throughput. For different K_p and K_i pairs, the heat value varies. The white region represents the highest values, indicating that the corresponding K_p and K_i pairs provide good performance across almost all network traces. The recommended ranges for the proportional controller coefficient K_p and integral controller coefficient K_i are:

$$\begin{aligned} K_p &\in [1.2 \times 10^{-4},\ 2.0 \times 10^{-4}] \\ K_i &\in [0.2 \times 10^{-5},\ 1.0 \times 10^{-5}] \end{aligned} \tag{16}$$

Considering that the network traces are collected from a real live streaming server and that they exhibit different temporal and spatial characteristics, the results show that K_p and K_i can be tuned to accommodate the large variations among different traces. It means that we can find a range of K_p and K_i values to make the PI controller practical. We use $K_p = 1.2 \times 10^{-4}$ and $K_i = 1.0 \times 10^{-5}$ as our default settings.

B. Interval Choice of NLP and BFA

The two major steps in calculating the target bitrate are non-linear programming (*NLP*) and buffer feedback adjustment (*BFA*). The importance of *NLP* is to provide the consideration of global constraints and resource competition among streamers. It relies on the throughput estimation. In the meantime, *BFA* is used to alleviate system errors during the modeling and measurement process. They can be executed at different intervals (I_{NLP}, I_{BFA}), resulting in different QoE effects. We want to explore whether there exist a set of I_{NLP} and I_{BFA} values that work well in a wide range of network environment. We also consider 72 hours of network throughput traces

[3] https://github.com/wzywzy/dataset

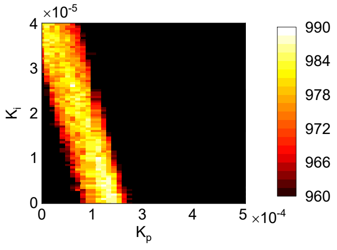

Fig. 3. Heat map for the proportional controller coefficient K_p and integral controller coefficient K_i.

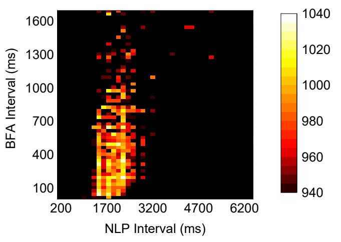

Fig. 4. Heat map for non-linear programming (*NLP*) and buffer feedback adjustment (*BFA*) interval.

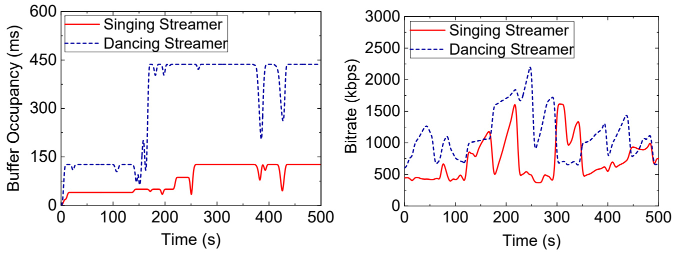

Fig. 5. Receiver buffer occupancy and bitrate from singing and dancing streamers respectively.

separately. For the k*-th* hour network trace, we vary the values of I_{NLP} and I_{BFA} in a large range to obtain the corresponding QoE values. Then we consider all the network traces and get the average QoE values. Fig. 4 shows the heat map. A larger heat value means that it leads to good performance for more traces of network throughput. The recommended ranges for *NLP* and *BFA* interval are:

$$\begin{aligned} I_{NLP} \in & [1400\,ms,\ 2300\,ms] \\ I_{BFA} \in & \ [50\,ms,\ 400\,ms] \end{aligned} \tag{17}$$

If the time interval of *NLP* is short, *NLP* will frequently generate target bitrates with measurement errors, leaving *BFA* little time to adjust the bitrate based on feedback, resulting in a low QoE value. On the other hand, if the time interval of *NLP* is long, the server cannot response to changes quick enough. While in live streaming, *BFA* is a relatively conservative strategy. It is hard to increase the bitrate once it is lack of global information, which leads to a low QoE value too. In the mean time, QoE value falls as the interval of *BFA* grows because a long interval of *BFA* leads to the insensitivity to buffer changes. We use $I_{NLP} = 2000$ ms and $I_{BFA} = 200$ ms as our default settings.

C. Preferences of Multiple Streamers

To illustrate how effective we can adjust to different preferences of steamers, we consider the scenario with three streamers i, j, and k. Streamer i and j are singing together in a chorus; Streamer k is dancing for them. Streamer i hopes the stream delay of j who sings with him is low. So i can increase the delay penalty weight in the QoE model. Streamer i also wants to see the dancing posture of k clearly. So i can increase the video quality weight of k in the QoE model. Dynamically setting the weights of different aspects in the QoE model for different streamers can effectively satisfy user preferences.

We simulate the above scenario in the simulator. Streamer i increases the delay penalty weight of j and the video quality weight of k. We measure i's receiver buffer occupancy, which stores frames sent from j and k respectively. We also measure bitrates from these two streamers. The result, shown in Fig. 5, shows that after the weight setting, the receiver buffer occupancy levels from two streamers have a great gap. The buffer storing j's data is obviously lower to maintain a lower delay while the buffer storing k's data is obviously higher to maintain a higher bitrate. Setting different weights for QoE model changes the receiver buffer occupancy. On the other hand, bitrates from two streamers also show differences. The bitrate from k is relatively high to ensure high video quality, and the bitrate from j is lower to ensure the smoothness and low delay. In fact, each streamer can set different priority for each stream he receives, including high bitrate, low bitrate switching, low stall duration, low delay and balanced. Personalized requirements can be well satisfied.

D. QoE Performance

We now evaluate the performance of MultiLive in terms of QoE and its four components, using the following methods as baselines:

- *NLP*: A simpler version of *MultiLive* where *BFA* is not performed.
- *BFA*: Another simpler version of *MultiLive* where *NLP* is not performed.
- *Single*: A simpler version of *MultiLive* where only a single bitrate is generated by each sender.
- *Fixed*: Computes the bitrates using initial network conditions, then continues to send at these bitrates without adapting to changing network conditions.
- *Janus*[4]: A bitrate control algorithm in an open-source video conferencing server based on WebRTC. It comprehensively considers sender-side bandwidth estimation and receiver-side RTCP REMB feedback message [25].

Fig. 6 and Fig. 7 present the performance of average bitrate, average bitrate variation, stall ratio (stall time/total time) and delay in the form of CDF for each dataset.

(i) *NLP*: With the estimation of global throughput and consideration of QoE weights, *NLP* takes more advantage of the network transmission capacity to achieve high bitrate. However, the adjacent two bitrate decisions use separate throughput data, resulting in large bitrate jitter. In addition, it is called at a larger interval because of its overhead on communication and computation. So it cannot provide a fine-grained adjustment. On the other hand, there exist system errors during the modeling and measurement process. For both reasons, stall happens frequently, resulting in high delay.

(ii) *BFA*: An essential difference between live streaming and VoD streaming is that the contents of live streaming is

[4] https://github.com/meetecho/janus-gateway

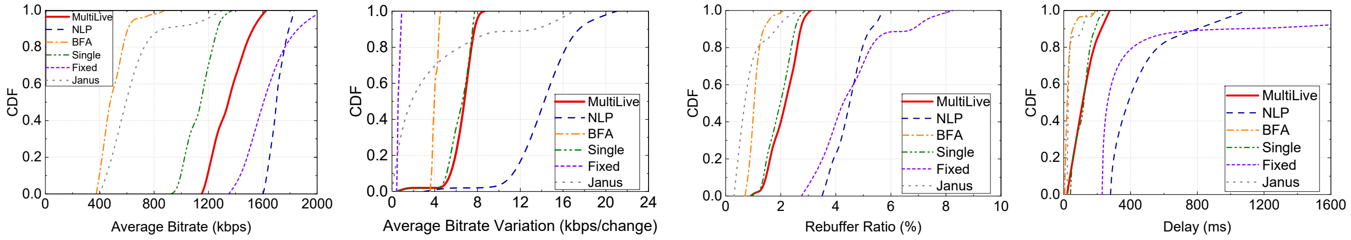

Fig. 6. Detailed performance using the Commercial Dataset.

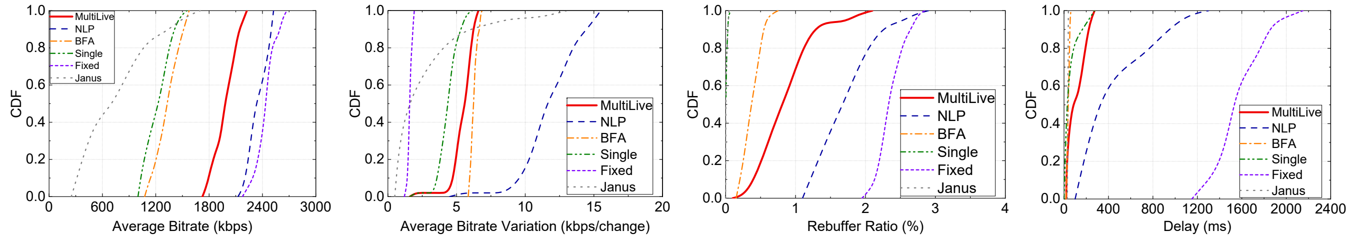

Fig. 7. Detailed performance using the Belgium 4G/LTE Dataset.

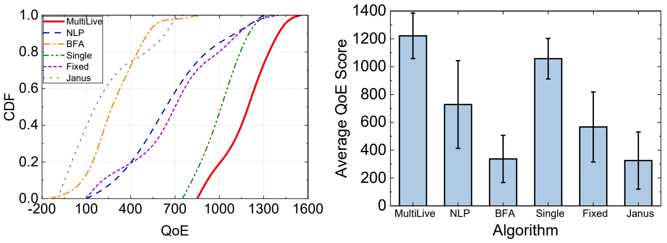

Fig. 8. QoE performance using the Commercial Dataset.

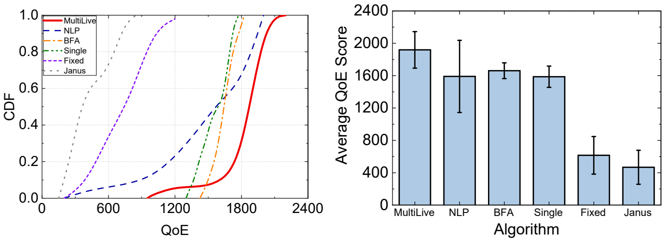

Fig. 9. QoE performance using the Belgium 4G/LTE Dataset.

produced just before it is played. So even if the stream bitrate in live streaming is far below the network throughput, the buffer occupancy cannot accumulate quickly as data that have not been generated cannot be buffered. The weak accumulation of buffer leads to a low bitrate. Due to this phenomenon, *BFA* results in low delay but poor video quality.

(iii) *Single*: It has to choose a single target bitrate that is the lowest common denominator among all the receivers, and thus the target bitrate is lower compared to other schemes, leading to lower quality. Due to the lower bitrate, however, the delay and the stall duration are smaller for receivers with higher uplink throughput.

(iv) *Fixed*: It ignores the receiver state and network fluctuation, which leads to high bitrate and severe stalling.

(v) *Janus*: It considers the uplink bandwidth of the sender and selects the highest bitrate allowed for the downlink bandwidth of all receivers. To ensure the smooth playback of users under low throughput condition, it conservatively chooses a lower bitrate, resulting in lower latency and fewer stalls.

(vi) Our algorithm *Multilive* combines the advantage of *NLP* and *BFA*. On the premise of guaranteeing fluency, it elevates users' video quality under the limited throughput to maximize the personalized QoE. As shown in the figure, it performs well on both datasets.

Fig. 8 and Fig. 9 present the QoE performance for both datasets. We can see that: (i) The overall QoE value of *NLP* is relatively low due to the low accuracy of prediction; (ii) Due to the lower bitrates and the hysteresis of feedback, *BFA* does not perform well either; (iii) Due to the worse overall network condition in the Commercial Dataset, compared to the Belgium 4G/LTE Dataset, the score of video quality falls quickly in the Commercial Dataset, according to the logarithmic characteristic of video quality function $q(\cdot)$, leading to a lower QoE for *BFA*; (iv) *Single* selects only one bitrate and the QoE effect is slightly worse; (v) *Fixed* selects the bitrate using initial network conditions, which change greatly during different time. So the QoE value is relatively unstable; (vi) Since *Janus* has lower bitrate, which causes poor video quality, it has a lower QoE score. Compared with *Fixed*, our proposed *MultiLive* algorithm improves QoE by 2-5×.

VI. Related Work

Previous ABR algorithms: The previous ABR algorithms can be primarily grouped into four classes: rate-based, buffer-based, hybrid, and learning-based. (i) Rate-based methods estimate the available network throughput and request the next chunk at the highest bitrate that the network is predicted to support. For example, Akhtar *et al.* [8] proposed a system for automatically tuning ABR algorithm configurations in real time to match the current network state. (ii) Buffer-based methods solely consider the client's buffer occupancy when deciding the bitrates for future chunks. For example, Spiteri *et al.* [11] devised a buffer-based online control algorithm using Lyapunov optimization techniques to minimize rebuffering and maximize video quality. (iii) Hybrid methods use both throughput prediction and buffer occupancy to select bitrates that are expected to maximize QoE over several future chunks. For example, Yin *et al.* [12] proposed a novel model predictive control algorithm that can optimally combine throughput and buffer occupancy information. (iv) Learning-based methods use reinforcement learning to adaptively select bitrate. For example, Mao *et al.* [9] trained a neural network model selecting bitrates for future video chunks based on observations collected by client video players. In addition, Yadav *et al.* [10] proposed a bitrate adaptation algorithm by modeling a client as an M/D/1/K queue. Lai *et al.* proposed a FoV-based bitrate adaptation algorithm for mobile virtual reality system to improve the QoE [26], [27]. However, all these methods only consider one streaming source and its delivery to a number of viewers in VoD streaming scenarios, rather than many-to-many in interactive live streaming scenarios. In addition, these methods select from discrete bitrate gears instead of adjusting on continuous bitrate domain.

Live streaming: Some previous studies provide architectures for live streaming delivery, mainly including two categories. The first category is the centralized architecture. For example, Mukerjee *et al.* [28] provided real-time control over individual streams from the CDN side and employed centralized quality optimization for responsiveness. The second category is distributed architecture. For example, Liu *et al.* [29] designed an open P2P live video streaming system which can accommodate a variety of video coding schemes. However, they mainly consider one media source and its delivery to a number of viewers, which is quite different from us. There are also some studies that focus on crowdsourced live streaming, which generalizes the single-source streaming. Chen *et al.* [30] explored the emerging crowdsourced live streaming systems and designed cloud leasing strategy to optimize the cloud site allocation. They, however, do not account for real-time interaction, where the delay requirement is very harsh. In addition, He *et al.* [31] proposed a novel framework for crowdsourced livecast systems offloading the transcoding assignment to the massive viewers. Pang *et al.* [32] observed unique characteristics related to viewers (proactive and passive) and designed a deep neural network model to capture the viewer interaction pattern. Huang *et al.* [21] proposed a deep-learning based rate control algorithm in the real-time video streaming.

Multi-party video conferencing: Some previous studies have put forward multi-party cloud video conferencing architecture. Hu *et al.* [14] studied the server selection problem to reduce end-to-end delay. Wu *et al.* [15] designed a fully decentralized algorithm to decide the best paths of streams and the most suitable surrogates along the paths. Hajiesmaili *et al.* [7] cast a joint problem of user-to-agent assignment and transcoding-agent selection, and proposed an adaptive parallel algorithm. Ooi *et al.* [33] divided up the in-network merging and transcoding process, and identified suitable cloud servers to run them, with the goal of minimizing the overall network cost. These studies, however, generally focus on the selection of transcoding server to minimize the cost of the service provider and the delay, and cannot meet the need of multi-party interactive live streaming where different online streamers may have different QoE preferences. Multi-party live streaming pays more attention to the user's experience and aims to improve the global QoE. Similar to our work, Amir *et al.* [34] proposed SCUBA, using scalable video coding and allowed the receiver to adjust the quality for different senders. But they do not consider global optimization that takes into consideration of QoE factors such as delay and smoothness. Small delay consideration is particular important for multi-party interactive live applications, where the tolerable delay is less than 100 ms [6], while for video conference, where participants take turns to talk, the tolerable delay is around 400 ms [35].

VII. Conclusion

In this paper, we propose an architecture for multi-party interactive live streaming. We build a QoE model and propose MultiLive, an adaptive bitrate control algorithm. Specifically, we apply non-linear programming to get the target bitrate for each pair of online streamers, and adjust the bitrate according to the buffer feedback to avoid the accumulation of system errors. To alleviate the problem of limited uplink transmission rate, we use bitrate clustering to reduce the number of streams to transmit from a streamer. Our results from extensive trace-driven simulations demonstrate that MultiLive outperforms the fixed bitrate algorithm, with 2-5× improvement of the average QoE. Furthermore, the end-to-end delay has been reduced to about 100 ms, which is much lower than 400 ms used as the delay threshold in existing schemes for video conferencing. As for future work, we will deploy the algorithm in a large-scale multi-party live streaming platform to verify performance. In addition, we plan to incorporate more accurate throughput prediction algorithm to improve the QoE.

Acknowledgment

This work was supported by NExT++ research established by the National Research Foundation, Prime Minister's Office, Singapore under its IRC@SG Funding Initiative.

References

[1] Z. Lu, H. Xia, S. Heo, and D. Wigdor, "You watch, you give, and you engage: a study of live streaming practices in china," in *ACM CHI*, 2018.

[2] O. L. Haimson and J. C. Tang, "What makes live events engaging on Facebook Live, Periscope, and Snapchat," in *ACM CHI*, 2017.

[3] J. C. Tang, G. Venolia, and K. M. Inkpen, "Meerkat and Periscope: I stream, you stream, apps stream for live streams," in *ACM CHI*, 2016.

[4] L. Provensi, A. Singh, F. Eliassen, and R. Vitenberg, "Maelstream: Self-organizing media streaming for many-to-many interaction," *IEEE TPDS*, 2018.

[5] F. Wang, J. Liu, M. Chen, and H. Wang, "Migration towards cloud-assisted live media streaming," *IEEE/ACM TON*, 2014.

[6] N. Schuett, "The effects of latency on ensemble performance," *Bachelor Thesis, CCRMA Department of Music, Stanford University*, 2002.

[7] M. H. Hajiesmaili, L. T. Mak, Z. Wang, C. Wu, M. Chen, and A. Khonsari, "Cost-effective low-delay design for multiparty cloud video conferencing," *IEEE TMM*, 2017.

[8] Z. Akhtar, Y. S. Nam, R. Govindan, S. Rao, J. Chen, E. Katz-Bassett, B. Ribeiro, J. Zhan, and H. Zhang, "Oboe: auto-tuning video ABR algorithms to network conditions," in *ACM SIGCOMM*, 2018.

[9] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with Pensieve," in *ACM SIGCOMM*, 2017.

[10] P. K. Yadav, A. Shafiei, and W. T. Ooi, "QUETRA: a queuing theory approach to dash rate adaptation," in *ACM MM*, 2017.

[11] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal bitrate adaptation for online videos," in *IEEE INFOCOM*, 2016.

[12] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," in *ACM SIGCOMM*, 2015.

[13] S. Sengupta, N. Ganguly, S. Chakraborty, and P. De, "HotDASH: Hotspot aware adaptive video streaming using deep reinforcement learning," in *IEEE ICNP*, 2018.

[14] Y. Hu, D. Niu, and Z. Li, "A geometric approach to server selection for interactive video streaming," *IEEE TMM*, 2016.

[15] Y. Wu, C. Wu, B. Li, and F. Lau, "vSkyConf: Cloud-assisted multi-party mobile video conferencing," in *ACM SIGCOMM workshop*, 2013.

[16] Y. Guo, Q. Yang, J. Liu, and K. S. Kwak, "Quality-aware streaming in heterogeneous wireless networks," *IEEE TWC*, 2017.

[17] C. Dong, W. Wen, T. Xu, and X. Yang, "Joint optimization of data-center selection and video-streaming distribution for crowdsourced live streaming in a geo-distributed cloud platform," *IEEE TNSM*, 2019.

[18] J. Van Der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck, "HTTP/2-based adaptive streaming of HEVC video over 4G/LTE networks," *IEEE Communications Letters*, 2016.

[19] A. Ahmed, Z. Shafiq, H. Bedi, and A. Khakpour, "Suffering from buffering? detecting QoE impairments in live video streams," in *IEEE ICNP*, 2017.

[20] X. Zuo, Y. Cui, M. Wang, T. Xiao, and X. Wang, "Low-latency networking: Architecture, techniques, and opportunities," *IEEE Internet Computing*, 2018.

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

[22] M. Mu, M. Broadbent, A. Farshad, N. Hart, D. Hutchison, Q. Ni, and N. Race, "A scalable user fairness model for adaptive video streaming over SDN-assisted future networks," *IEEE JSAC*, 2016.

[23] W. Huang, Y. Zhou, X. Xie, D. Wu, M. Chen, and E. Ngai, "Buffer state is enough: Simplifying the design of QoE-aware HTTP adaptive video streaming," *IEEE TBC*, 2018.

[24] Y. Qin, R. Jin, S. Hao, K. R. Pattipati, F. Qian, S. Sen, C. Yue, and B. Wang, "A control theoretic approach to ABR video streaming: A fresh look at PID-based rate adaptation," *IEEE TMC*, 2019.

[25] A. Amirante, T. Castaldi, L. Miniero, and S. P. Romano, "Janus: a general purpose WebRTC gateway," in *ACM IPTComm*, 2014.

[26] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, N. Dai, and H.-S. Lee, "Furion: Engineering high-quality immersive virtual reality on today's mobile devices," *IEEE TMC*, 2019.

[27] Z. Lai, Y. Cui, Z. Wang, and X. Hu, "Immersion on the edge: A cooperative framework for mobile immersive computing," in *ACM SIGCOMM Posters and Demos*, 2018.

[28] M. K. Mukerjee, D. Naylor, J. Jiang, D. Han, S. Seshan, and H. Zhang, "Practical, real-time centralized control for CDN-based live video delivery," in *ACM SIGCOMM*, 2015.

[29] Z. Liu, Y. Shen, K. W. Ross, S. S. Panwar, and Y. Wang, "Substream trading: Towards an open P2P live streaming system," in *IEEE ICNP*, 2008.

[30] F. Chen, C. Zhang, F. Wang, and J. Liu, "Crowdsourced live streaming over the cloud," in *IEEE INFOCOM*, 2015.

[31] Q. He, C. Zhang, and J. Liu, "Crowdtranscoding: Online video transcoding with massive viewers," *IEEE TMM*, 2017.

[32] H. Pang, C. Zhang, F. Wang, H. Hu, Z. Wang, J. Liu, and L. Sun, "Optimizing personalized interaction experience in crowd-interactive livecast: A cloud-edge approach," in *ACM MM*, 2018.

[33] W. T. Ooi and R. Van Renesse, "Distributing media transformation over multiple media gateways," in *ACM MM*, 2001.

[34] E. Amir, S. McCanne, and R. Katz, "Receiver-driven bandwidth adaptation for light-weight sessions," in *ACM MM*, 1997.

[35] ITU-T, "Recommendation G. 114, one-way transmission time," *Series G: Transmission Systems and Media, Digital Systems and Networks, Telecommunication Standardization Sector of ITU*, 2000.