Tightrope Walking in Low-latency Live Streaming: optimal joint adaptation of video rate and playback speed

Liyang Sun, Tongyu Zong, Siquan Wang, Yong Liu and Yao Wang
New York University
Brooklyn, NY, USA
{ls3817,tz1178,sw4112,yongliu,yw523}@nyu.edu

ABSTRACT

It is highly challenging to simultaneously achieve high-rate and low-latency in live video streaming. Chunk-based streaming and playback speed adaptation are two promising new trends to achieve high user Quality-of-Experience (QoE). To thoroughly understand their potentials, we develop a detailed chunk-level dynamic model that characterizes how video rate and playback speed jointly control the evolution of a live streaming session. Leveraging on the model, we first study the optimal joint video rate-playback speed adaptation as a non-linear optimal control problem. We further develop model-free joint adaptation strategies using deep reinforcement learning. Through extensive experiments, we demonstrate that our proposed joint adaptation algorithms significantly outperform rate-only adaptation algorithms and the recently proposed low-latency video streaming algorithms that separately adapt video rate and playback speed without joint optimization. In a wide-range of network conditions, the model-based and model-free algorithms can achieve close-to-optimal trade-offs tailored for users with different QoE preferences.

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**; • **Computing methodologies** → *Reinforcement learning*.

KEYWORDS

Live streaming, Linear quadratic regulator, Reinforcement learning

ACM Reference Format:
Liyang Sun, Tongyu Zong, Siquan Wang, Yong Liu and Yao Wang. 2021. Tightrope Walking in Low-latency Live Streaming: optimal joint adaptation of video rate and playback speed. In *12th ACM Multimedia Systems Conference (MMSys '21) (MMSys 21), September 28-October 1, 2021, Istanbul, Turkey.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3458305.3463382

1 INTRODUCTION

While the next generation network infrastructures, such as 5G networks, are designed for high-throughput and low-latency, the application designs that can fully take advantage of the promised network capability to deliver a high level of user Quality-of-Experience (QoE) are the exciting new research challenges. In the space of multimedia streaming, the emerging content, such as ultra-high-definition video and 360 degree/volumetric video, are truly high-bandwidth. In the live streaming of such content, low-latency has become one of the most critical requirements. A survey report from WOWZA [3] covering 391 broadcasters around the world shows low end-to-end latency has become the second most important factor (preferred by 31% users) for live video streaming, with high quality still leading (preferred by 35% users). However, compared with cable broadcasters that can deliver live content with 5 seconds latency on average [2], most Over the Top (OTT) live streaming services are still lagging behind. The same WOWZA report [3] also shows that about 40% of the OTT video distributors are streaming with latency greater than 10 seconds, and only about 25% of them can deliver sub-3 seconds latency. *The main challenge for low-latency live streaming over the Internet is to adapt to dynamic network conditions with short video buffer. It has to achieve the right balance among various QoE metrics, e.g., perceptual quality, playback latency and streaming continuity, with small margin for error, which is a tightrope-walking challenge.*

There are two recent trends to address this challenge, namely *chunk-based streaming*, e.g., CMAF [12] and HTTP 1.1 chunked transfer encoding [42], and *playback speed adaptation*, e.g., [1, 44]. In chunk-based streaming, each video segment is further divided into chunks, which can be encoded, transmitted and decoded in a pipelined fashion. This not only reduces the encoding and decoding delays, but also refines the data granularity for better streaming control. Playback speed adaptation allows the client to playback video at a speed faster or slower than the normal speed. It serves as an important additional knob (on top of video rate adaptation) to control playback latency in live streaming: when the playback lags too far behind the live event, one can choose a faster playback rate to catch up; when the latency is too short, the video buffer is necessarily shallow, leading to the danger of buffer depletion/video freeze, one may choose a slower playback speed to gradually bring up the buffer to a safe level. Obviously, video rate adaptation directly determines the perceptual quality, while playback speed adaptation directly controls the playback latency. Less obviously, the two knobs jointly control the video buffer evolution, which determines the streaming continuity, video freeze and latency change. When the buffer length is measured in video time, the playback speed is simply the buffer outflow rate, while the inflow rate is inversely proportional to the selected video rate. *For the live streaming "tightrope walking", video rate and playback speed adaptations sit at the two ends of its "balancing pole", therefore have to be jointly adapted to deliver a high-level of user QoE.*

MMSys 21, September 28-October 1, 2021, Istanbul, Turkey

ACM ISBN 978-1-4503-8434-6/21/09…$15.00
https://doi.org/10.1145/3458305.3463382

In this paper, we present our work on optimal joint adaptation of video rate and playback speed to maximize user QoE in low-latency live streaming. We start with building a detailed chunk-level live streaming dynamic model that characterizes how video rate and playback speed jointly control the state evolution of the live streaming session, including buffer length, playback latency, video freeze, download idle time, etc. The developed model brings forth important insights on how video rate adaptation and playback speed adaptation, individually and jointly, impact various user QoE metrics. Leveraging on the dynamic model, we first develop a model-based joint rate-speed adaptation solution based on a nonlinear optimal control technique, namely iterative Linear Quadratic Regulator (iLQR). The quality of the model-based solution can be degraded by the fine system dynamics not captured by the streaming model and network condition prediction errors. Our second effort resorts to model-free optimal control techniques, namely Deep Reinforcement Learning (DRL). We train a Branching Dueling Q-network (BDQ) model to generate rate adaptation and speed adaptation policies from separate branching Q-networks so that more playback speed control levels can be supported. Through extensive simulations driven by real network traces, we demonstrate that the proposed joint adaptation algorithms significantly outperform rate-only adaptation and heuristic rate-speed adaptation algorithms. In a wide-range of network conditions, the model-based and model-free algorithms can dynamically adjust video rate and playback speed to achieve the best trade-offs tailored for different user QoE preferences. We make the following contributions:

- We develop a detailed chunk-level dynamic model for low-latency live streaming system that characterizes the interplay of rate adaptation and playback speed control, and their impacts on various user QoE metrics.
- Based on the live streaming model, we study the joint adaptation of video rate and playback speed as a nonlinear optimal control problem. iLQR-type streaming algorithms are developed to generate real-time rate-speed control.
- We further develop a DRL-based joint adaptation algorithm to support a wide range of speed control. We show that the branching of BDQ can effectively explore the expanded action space and find close-to-optimal adaptation policies.
- We conduct extensive experiments using real LTE bandwidth traces. The results demonstrate the gain of our joint adaptation algorithms over simple heuristics and rate-only adaptation. We also show that the proposed algorithms can be flexibly adjusted to satisfy diverse user QoE preferences.
- Finally, we discuss the pros and cons of model-based vs. model-free joint adaptation algorithms for adoption in practical low-latency live streaming systems.

In the following parts of this paper, Sec. 2 reviews the related work. Live streaming dynamic model is developed in Sec. 3. iLQR and BDQ-based joint adaptation algorithms are derived in Sec. 4 and 5, respectively. Sec. 6 presents the performance evaluation results. At last, Sec. 7 concludes the paper.

2 RELATED WORK

To deliver smooth and high quality video to the users, **video rate adaptation** algorithms are widely employed in Video-on-Demand (VoD) and live streaming systems. With the help of Dynamic Adaptive Streaming over HTTP (DASH) [29] and HTTP Live Streaming (HLS) [26], different video rates can be selected over time to adapt to dynamic network conditions. Rate-based adaptation algorithms [19, 20, 23] choose the video rate based on the predicted bandwidth. However, user QoE is also affected by video rate smoothness and video freezing. Therefore, some enhanced adaptation algorithms including FESTIVE [13] and PANDA [17] are proposed in which buffer status is also considered to optimize the QoE. In [37, 48], proportional-integral-derivative (PID) controller is proposed to control video rate to maintain the video buffer around a target level. BOLA is a Lyapunov optimization based algorithm proposed in [30]. It has already been embedded in DASH.js player. In addition, another buffer-based algorithm (BBA) [11] selects video rate based on the receiving buffer occupancy so that the rebuffering time can be reduced by $10-20\%$ without affecting the video quality when compared with benchmarks. In [45], video rate is selected by solving a QoE maximization problem with the predicted available network bandwidth. To deal with the prediction error accumulation at long prediction interval, model predictive control (MPC) is utilized in which the optimal rate selections for several future steps are generated, but only the first action is taken at the next step. The system iterates video rate selection for each video segment. In addition, deep reinforcement learning (DRL) approach is adopted in [21, 28] to maximize the long-term accumulated QoE where the optimal bitrate is generated by a RL agent based on the system state. In [4], parameters are pre-computed for different network conditions, and the system can adapt the parameters based on the current network condition to improve the performance. Distributed queuing theory is adopted in [6] to download video segments in parallel from multiple servers.

In a live streaming system, **latency** is another important metric affecting user QoE due to the "live" properties of the events. In [43], buffer-based rate control algorithm with dynamic threshold is proposed to reduce the rate fluctuation and guarantee smooth playback for low-latency live video streaming. The study in [35] shows the latency in live streaming system is tied to the segment duration which can be of several seconds. To reduce the latency lower bound, Common Media Application Format (CMAF) is proposed in [12] where one video segment is divided into multiple chunks. With HTTP 1.1 chunked transfer encoding [42], the delivery of a video segment can be pipelined with encoding. An MPC type of video rate adaptation algorithm was proposed in [34] for achieving low-latency in live streaming. To control playback latency, heuristic [27] and DRL [14, 40, 47] based algorithms are proposed to skip video frames so that video latency can be reduced. However, skipping frames might cause significant user QoE degradation. The content harvest network architecture is proposed in [25] to achieve both low latency and sustainable bandwidth for the first mile delivery of live streaming. In [32], an online algorithm is proposed for delay-aware fountain codes. To reduce the start-up delay, [8] proposes to use WebSocket over HTTP/2 and server-push mechanism. Similarly, server-push strategy is also utilized in [39], but with a shorter segment duration. Short latency also leads to more accurate field of view (FoV) prediction in 360 video streaming [33]. The overhead of chunked streaming is studied in [7].

To smoothly control playback latency, instead of skipping frames, **playback speed adaptation** is proposed in [15, 31]. In the MMSys 2020 Grand Challenge [1], several systems utilize playback speed adaptation to achieve low latency. The authors of [16] solve the low-latency video rate adaptation problem using online convex optimization. The playback speed adaptation is adopted directly from the default playback speed module in the dash.js player. As a result, the video rate and playback speed adaptations are not jointly optimized to maximize user's QoE. Another heuristic low-latency video rate adaptation algorithm is proposed in [10]. A sliding window is utilized to measure the means and standard deviations of both throughput and latency, which are used to drive video rate adaptation. The default playback speed module of dash.js is modified to avoid unnecessary video stalls. In [18], an MPC-based low-latency video rate adaptation algorithm is proposed. With bandwidth predictions, the algorithm exhaustively searches among all possible rate selection combinations in a look-ahead window to maximize user QoE. However, such a brute-force search will suffer from space explosion due to long look-ahead window and/or many video rate levels. In the same work, a learning-based video rate adaptation algorithm using Self Organizing Maps is also introduced. Similar to [10, 16], the playback speed is adapted by a heuristic algorithm independent of video rate adaptation. Indeed, in low-latency live video streaming, video rate and playback speed adaptations are tightly coupled. They jointly determine the evolution of the system state and user QoE. *Different from the previous studies, we study the optimal joint adaptation of video rate and playback speed. Leveraging on the insights obtained from our detailed live streaming model, we develop both model-based and model-free optimal* **joint adaptation** *algorithms. At each step, the video rate and playback speed are jointly adapted by evaluating their joint influence on the system state and user QoE.* To address the computation challenge of joint adaptation, we use model-based optimal control technique, namely iLQR, to significantly reduce the computation cost of exhaustive MPC search in [18, 45], and use action branching to develop our DRL-based model-free joint adaptation algorithm.

3 LOW-LATENCY LIVE STREAMING

Generally, for the legacy live video streaming systems, several to ten seconds video is buffered on users' devices to cope with network condition fluctuations. On the contrary, in order to deliver "live" experience, low playback latency is desired. As a result, the amount of video time that can be buffered is limited by the target playback latency, leading to higher risk of video buffer underflow. Therefore, in a low-latency live streaming system, the first design trade-off is *how to balance between low playback latency (short video buffer length) and the robustness against bandwidth fluctuations.*

3.1 Playback Speed Control

In the traditional live streaming, whenever a video freeze happens, the playback latency keeps increasing until the playback resumes. After several freezes, the latency might become too long for being "live". With the normal playback speed, the only way to catch up with the live event is to skip frames/segments. Video skipping leads to user QoE degradation. Totally missing critical moments in a live event, such as a game-changing goal, is simply not acceptable. Playback speed adaptation serves as a smoother alternative to gradually reduce playback latency and catch up. It has been shown through subjective user studies that playback speed changes within 10% (faster or slower) are not even noticeable by users [9, 46]. For video-on-demand, users can choose different playback speeds for different videos over a wider range (×0.25 to ×2) on popular platforms, such as YouTube and iQIYI. It is impossible/meaningless to fast-playback/slow-playback throughout a live streaming session. As will be shown later, only sporadic playback speed adaptations are sufficient to effectively control playback latency and improve the overall user QoE in live streaming.

Video rate and playback speed are the two knobs to control video buffer and have different user QoE implications. For example, if fast playback is chosen (to reduce latency), video buffer will drain faster, then lower video rate might have to be selected so that video freeze can be avoided. But the delivered perceptual video quality will suffer. Oppositely, if high video rate is selected, video buffer inflow rate decreases, then slow playback speed is helpful to slow down the buffer drain rate. However, as a negative consequence, playback latency will increase. Therefore, *video rate and playback speed adaptations should be judiciously coordinated to achieve the best trade-off tailored for different user QoE preferences.*

3.2 Chunk-level Dynamic Model

To systematically study those trade-offs, we start with building a detailed dynamic live streaming model to understand the interplay between video rate and playback speed.

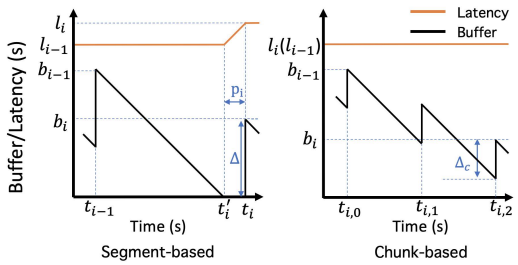


Figure 1: Comparison between segment-based and chunk-based delivery in terms of video buffer and latency.

One efficient way to reduce the latency is chunk-based streaming, which divides one video segment into multiple chunks and allows them to be streamed in a decoupled way. For example, if a video is streamed in segments (the left portion of Fig. 1), the download of segment i starts from time t_{i-1} and ends at t_i. During the downloading, video buffer (the black curve) is drained out at time t'_i. Therefore, video freeze p_i happens, leading to latency increase (the orange curve). However, if chunk-based delivery is adopted (the right portion of Fig. 1), the first and second chunk of segment i are received at time $t_{i,1}$ and $t_{i,2}$, respectively. In this case, even with the same initial buffer length b_{i-1} and total download time as the segment-based streaming, video freeze and latency increment can be avoided with chunk-based streaming.

To formulate the system evolution of a chunk-based streaming system, we assume one video segment is divided into J chunks

Table 1: Variables associated with segments

Notation	Definition (for segment i)
Δ	Video time in each segment (seconds)
J	Number of chunks per segment (scalar)
r_i	Video rate selected (Mbps)
s_i	Playback speed while downloading segment (scalar)
$u_i=\langle r_i, s_i\rangle$	Joint rate-speed selection (vector)
$\boldsymbol{q}(r_i)$	Perceptual video quality (scalar)

Table 2: Variables associated with chunks

Notation	Definition (for chunk j in segment i)
$\Delta_c = \Delta/J$	Video time in each chunk (seconds)
$\tau_{i,j}$	Chunk download duration (seconds)
$w_{i,j}$	Average download throughput (Mbps)
$rtt_{i,j}$	Round trip time (RTT, seconds)
$c_{i,j}$	Server-side coding completion time (seconds)
$t_{i,j}$	Client-side download completion time (seconds)
$z_{i,j}$	Idle time before downloading (seconds)
$b_{i,j}$	Buffer length after downloading (seconds)
$p_{i,j}$	Freeze time while downloading (seconds)
$l_{i,j}$	Playback latency (seconds)
$x_{i,j}$	System state $\langle b_{i,j}, l_{i,j}, r_i, s_i\rangle$ (vector)
$n_{i,j}$	Network condition $\langle w_{i,j}, rtt_{i,j}\rangle$ (vector)

and define the notations for segments and chunks in Table 1 and 2 respectively. Note that in chunk-based streaming system, even though video data is delivered in chunks, the video rate control is still operated at the segment level. The client sends out one request (through HTTP in DASH) for each segment, the server sequentially delivers chunks of the segment to the client immediately after the first chunk is completely encoded. The download duration of video chunk (i, j) can be written as:

$$\tau_{i,j} = \frac{r_i \Delta_c}{w_{i,j}}, \quad 1 \le j \le J, \tag{1}$$

in which $r_i\Delta_c$ is the data size of a chunk at video rate r_i [1] and $w_{i,j}$ is the average download bandwidth. Let $t_{i,j}$ be the time when chunk (i, j) is completely downloaded. We have

$$\begin{aligned} t_{i,1} &= \max\left(c_{i,1}, t_{i-1,J} + \frac{rtt_{i,1}}{2}\right) + \tau_{i,1} + \frac{rtt_{i,1}}{2}, \\ t_{i,j} &= \max\left(c_{i,j}, t_{i,j-1} - \frac{rtt_{i,j}}{2}\right) + \tau_{i,j} + \frac{rtt_{i,j}}{2}, 2 \le j \le J, \end{aligned} \tag{2}$$

where $c_{i,j}$ is the server coding completion time for chunk (i, j), and the first term in both equations is the earliest time when a chunk can be pushed out by the server. The max operation reflects the fact that a chunk can be pushed by the server only after it is completely encoded and the first chunk has to wait for the segment download request from the client. Then all the following chunks can be pushed out in a pipelined fashion without additional requests. To simplify the notation, we define a new variable $z_{i,j}$ as

$$z_{i,j} \triangleq t_{i,j} - t_{i,j-1} - \tau_{i,j}, \quad 1 \le j \le J, \tag{3}$$

[1]Given a selected rate r_i for segment i, the actual size for each chunk within the segment might be greater or less than $r_i\Delta_c$. In our implementation, we use the actual coded chunk size to calculate the download duration.

with the chunk index wrap-around as $(i, 0) \triangleq (i - 1, J)$, i.e. the chunk before the first chunk of a segment is the last chunk of the previous segment. $z_{i,j}$ is the potential idle time before the chunk download starts, due to either the segment request for the first chunk or the waiting for the server to complete the encoding.

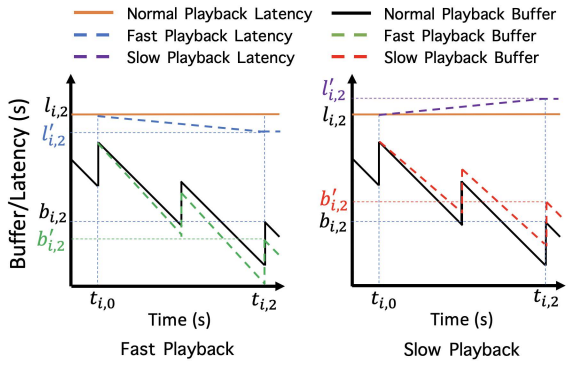


Figure 2: Impact of Fast and Slow playback speed. The y-axis represents buffer length or latency in time.

To achieve latency adaptation, the playback speed on the client device can be increased or decreased. We define s_i as the playback speed while downloading all chunks in segment i. With playback speed s_i, during the download, the buffer draining rate becomes s_i times the normal rate of 1. Then, the video receiving buffer length $b_{i,j}$ after downloading chunk (i, j) can be updated as:

$$\begin{aligned} b_{i,j} &= \max\left(b_{i,j-1} - s_i(t_{i,j} - t_{i,j-1}), 0\right) + \Delta_c \\ &= \max\left(b_{i,j-1} - \frac{s_i r_i \Delta_c}{w_{i,j}} - s_i z_{i,j}, 0\right) + \Delta_c, \end{aligned} \tag{4}$$

where the playback speed s_i shows up in two terms, while video rate r_i shows up only once in a product form with s_i in the second term. *This suggests that playback speed can be more effective than video rate in controlling video buffer length.* As will be shown next, video buffer evolution directly determines video freeze and playback latency. Therefore, playback speed is an important control knob for low-latency live streaming system.

In Eq. (4), the first term equals to zero if the video buffer depletes before chunk (i, j) download completes, triggering video freeze. The video freeze time can be calculated as:

$$p_{i,j} = \max\left(\frac{r_i \Delta_c}{w_{i,j}} + z_{i,j} - \frac{b_{i,j-1}}{s_i}, 0\right). \tag{5}$$

After each video freeze, the playback latency will increase by the amount of freeze time. Additionally, abnormal playback speed ($s_i \neq 1$) also directly changes the playback latency. Overall, the video playback latency will be updated as:

$$l_{i,j} = l_{i,j-1} - (s_i - 1)\left(\frac{r_i \Delta_c}{w_{i,j}} + z_{i,j}\right) + p_{i,j}. \tag{6}$$

Eq. (5) and (6) illustrate that latency can be reduced if video is played faster than the normal playback speed. At the same time, buffer is consumed faster, leading to higher risk of video freeze (shown as the green dotted curve in Fig. 2). On the contrary, if slow playback speed is chosen, video buffer is consumed slower and latency would increase (shown as the red dotted curve in Fig. 2).

The chunk-based system dynamic can be summarized as:

$$x_{i,j} = \mathcal{F}(x_{i,j-1}, u_i, n_{i,j}) \quad \text{with } i \in [1, I], j \in [1, J], \tag{7}$$

where $\mathcal{F}(\cdot)$ is the system dynamic function defined by Eq. (1), (2), (4), (5) and (6). After taking action $u_i := \langle r_i, s_i \rangle$ under network condition $n_{i,j} := \langle w_{i,j}, rtt_{i,j} \rangle$, the state evolves to $x_{i,j}$ which is defined as $\langle b_{i,j}, l_{i,j}, r_i, s_i \rangle$. Note that state variables with index $(i, 0)$ represent the state before downloading the first chunk of segment i, i.e., the final state of the previous segment. For example, $b_{i,0}$ is the buffer length before downloading chunk $(i, 1)$, i.e., the buffer length after downloading the last chunk of segment $(i-1)$.

3.3 Quality of Experience Model

In addition to the traditional QoE metrics, such as higher video rate, less video rate fluctuation, and less freeze, a latency-adapted live streaming system further attempts to control real-time latency through playback speed adaptation. So, the overall design goal can be formulated as *Joint Optimization of Video Rate and Playback Speed*:

$$\max_{\{r_i \in \mathcal{R}, s_i \in \mathcal{S}\}} \quad \sum_{i=1}^{I} QoE_i \tag{8}$$

$$\begin{aligned} \text{with} \quad QoE_i = {} & \alpha_1 \boldsymbol{q}(r_i) - \alpha_2 |\boldsymbol{q}(r_i) - \boldsymbol{q}(r_{i-1})| \\ & - \alpha_3 |1 - s_i| - \alpha_4 |s_i - s_{i-1}| \\ & - \alpha_5 \sum_{j=1}^{J} l_{i,j} - \alpha_6 \sum_{j=1}^{J} p_{i,j} \end{aligned}$$

subject to: streaming system state evolution dynamics (7),

in which $\mathcal{R}$ and $\mathcal{S}$ are the available video rates and playback speeds respectively. We adopt a segment-based QoE model similar to the QoE model of MMSys 2020 low latency video streaming challenge [1, 10, 16, 18]. QoE_i is the QoE of video segment i consisting of J chunks. Each QoE metric is defined as the following:

a) The first two terms are video rate based. $\boldsymbol{q}(r_i)$ is the perceptual quality that is normally modeled by a log function, e.g. $\boldsymbol{q}(r_i) = \eta_1 \log(r_i) + \eta_2$ in which η_1 and η_2 are the coefficients to be fitted. The second term is the penalty of rate fluctuation between two adjacent segments.
b) The following two terms are related to playback speed. $|1 - s_i|$ calculates the penalty of faster or slower playback which affects QoE negatively. The fourth term is the penalty for playback speed fluctuation. Note that this term was not included in [1]. We introduce additional penalty for playback speed fluctuation as it has negative impact on user QoE.
c) The last two terms are chunk-level metrics. The fifth term is defined as the QoE degradation caused by playback latency $l_{i,j}$. The last term represents the penalty of freeze time $p_{i,j}$ while downloading chunk (i, j).

The detailed parameter and function settings are discussed in Sec. 6.1. It is well understood that different users have different QoE preferences, and there is no one-size-fits-all QoE model. It is not the focus of this paper to develop a universal QoE model for low-latency live streaming. *Our model-based and model-free joint adaptation solutions to be introduced in the next two sections are designed to work with arbitrary QoE model with arbitrary weight settings.*

4 MODEL-BASED SOLUTION: ITERATIVE LINEAR QUADRATIC REGULATOR

The QoE maximization problem defined in (8) is to maximize the total reward for the entire live streaming session by jointly adapting video rate and playback speed of all the segments. The rate-speed choice $\langle r_i, s_i \rangle$ not only immediately determines the rendered quality and latency of segment i, but also changes the state of the live streaming session, in particular the client buffer length, for the following segments to work with. Additionally, since the QoE function consists of video rate and playback speed variations between the adjacent segments, the rate-speed cannot be optimized for each segment marginally. In this section, we adopt a technique from nonlinear optimal control, namely iterative Linear Quadratic Regulator (iLQR) [38], for the joint optimization of video rate and playback speed of all the segments. For presentation clarity, we discuss iLQR-based live streaming at the segment-level, our actual iLQR implementation is at the chunk-level with similar procedures.

4.1 Iterative Linear Quadratic Regulator

4.1.1 iLQR Prerequisite. In the classical iLQR framework, ① the control objective is to minimize some cost function, and ② both the cost function and system dynamic models are differentiable. To satisfy prerequisite ①, we reformulate the QoE maximization problem defined in (8) into a cost minimization problem with the cost of segment i as $c_i = -QoE_i$. The prerequisite ② is not strictly satisfied by the live streaming system dynamic models defined in Sec. 3. For example, in Eq. (4), after each video freeze, buffer length jumps from zero to a chunk duration. Such jumps are not differentiable. Similarly, the video freeze time in Eq. (5) and consequently, the latency in Eq. (6) are not differentiable due to the $\max(\cdot)$ function. We approximate the system dynamic models and the cost functions with differentiable functions. Due to the space limit, we skip the detail here. Given the "oracle" of network condition $\{n_i, i = 1, \cdots, I\}$, we can approximate the system dynamics and cost function as:

$$x_i \approx \boldsymbol{f}(x_{i-1}, u_i), \quad c_i \approx \boldsymbol{c}(x_{i-1}, u_i), \tag{9}$$

in which both $\boldsymbol{f}(\cdot)$ and $\boldsymbol{c}(\cdot)$ are differentiable.

4.1.2 iLQR Perturbation Optimization. With the approximated system evolution and control cost, starting from an initial control sequence $\mathcal{U}^{(0)} = \{u_i^{(0)}, i = 1, \cdots, I\}$, iLQR iteratively improves the current control sequence $\mathcal{U}^{(m)}$ by solving for the optimal control perturbation $\delta\mathcal{U}^{(m)}$ within the neighborhood of the current system state trajectory $\mathcal{X}^{(m)} = \{x_i^{(m)}, i = 1, \cdots, I\}$. Specifically, for step i, the system dynamics $\boldsymbol{f}(\cdot)$ around state point $\hat{x}_{i-1}$ after taking control $\hat{u}_i$ can be approximated using the 1st-order Taylor series expansion:

$$\boldsymbol{f}(x_{i-1}, u_i) \approx \boldsymbol{f}(\hat{x}_{i-1}, \hat{u}_i) + \nabla_{x_{i-1}, u_i} \boldsymbol{f}(\hat{x}_{i-1}, \hat{u}_i) \begin{bmatrix} x_{i-1} - \hat{x}_{i-1} \\ u_i - \hat{u}_i \end{bmatrix}. \tag{10}$$

If we define the state and control perturbations as $\delta x_i \triangleq x_i - \hat{x}_i$ and $\delta u_i \triangleq u_i - \hat{u}_i$. It can be shown that:

$$\begin{aligned} \delta x_i &= \boldsymbol{f}(x_{i-1}, u_i) - \boldsymbol{f}(\hat{x}_{i-1}, \hat{u}_i) \\ &\approx \nabla_{x_{i-1}, u_i} \boldsymbol{f}(\hat{x}_{i-1}, \hat{u}_i) \begin{bmatrix} \delta x_{i-1} \\ \delta u_i \end{bmatrix} = \mathbf{F}_i \begin{bmatrix} \delta x_{i-1} \\ \delta u_i \end{bmatrix}, \end{aligned} \tag{11}$$

where $\mathbf{F}_i$ is the state transition matrix of the linearized system around $\langle \hat{x}_{i-1}, \hat{u}_i \rangle$. Using the 2nd-order Taylor series expansion, the cost perturbation is approximated by:

$$\begin{aligned}\delta c_i &= c(x_{i-1}, u_i) - c(\hat{x}_{i-1}, \hat{u}_i) \\ &\approx \nabla_{x_{i-1}, u_i} c(\hat{x}_{i-1}, \hat{u}_i) \begin{bmatrix} \delta x_{i-1} \\ \delta u_i \end{bmatrix} \\ &\quad + \frac{1}{2} \begin{bmatrix} \delta x_{i-1} \\ \delta u_i \end{bmatrix}^T \nabla^2_{x_{i-1}, u_i} c(\hat{x}_{i-1}, \hat{u}_i) \begin{bmatrix} \delta x_{i-1} \\ \delta u_i \end{bmatrix} \\ &= \mathbf{c}_i \begin{bmatrix} \delta x_{i-1} \\ \delta u_i \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \delta x_{i-1} \\ \delta u_i \end{bmatrix}^T \mathbf{C}_i \begin{bmatrix} \delta x_{i-1} \\ \delta u_i \end{bmatrix},\end{aligned} \tag{12}$$

where $\mathbf{C}_i$ and $\mathbf{c}_i$ are the quadratic control cost matrixes.

We can find that Eq. (11) and (12) are in linear and quadratic forms in terms of δx_{i-1} and δu_i. Then, for the given state $\hat{x}_{i-1}$ and control $\hat{u}_i$, the optimal perturbation δu_i can be solved analytically using Linear Quadratic Regular (LQR) [5] shown as Algorithm 1. More specifically, the cost minimization problem can be solved analytically in two passes. The first pass is *backward propagation*, which calculates the closed-form solution of the optimal perturbation δu_i in terms of the given state change δx_{i-1} by setting the derivative of the quadratic cost term to be zero. It starts from the last step $i = I$, going backward until $i = 1$. The actual optimal perturbations $\langle \delta u_i, \delta x_i \rangle$ are calculated in the *forward pass*. From line 2 to 7 in Algorithm 1, $\mathbf{Q}_i, \mathbf{q}_i, \mathbf{K}_i, \mathbf{k}_i, \mathbf{V}_i$ and $\mathbf{v}_i$ are all intermediate variables. By substituting the actual initial state δx_0 in the closed-form optimal control strategy generated by *backward propagation*, the optimal perturbation δu_1^* can be solved. The next state change δx_1 can also be derived based on δx_0 and δu_1^*.

Algorithm 1 Linear Quadratic Regulator (LQR)

Input: $\mathcal{X}^{(0)}, \mathcal{U}^{(0)}$: the initial state and control sequence; I: number of segments to be optimized; $\{n_i, i \in [1, I]\}$: future network condition.
Output: $\delta\mathcal{U}^*$: the optimal control perturbation sequence.
Initialization: Set $\mathbf{V}_I$ and $\mathbf{v}_I$ to be zero matrix.

Backward Propagation
1: **for** each segment $i = I : 1$ **do**
2: $\quad \mathbf{Q}_i = \mathbf{C}_i + \mathbf{F}_i^T \mathbf{V}_i \mathbf{F}_i$
3: $\quad \mathbf{q}_i = \mathbf{c}_i + \mathbf{F}_i^T \mathbf{V}_i \mathbf{f}_i + \mathbf{F}_i^T \mathbf{v}_i$
4: $\quad \mathbf{K}_i = -\mathbf{Q}^{-1}_{u_i,u_i} \mathbf{Q}_{u_i,x_i}$
5: $\quad \mathbf{k}_i = -\mathbf{Q}^{-1}_{u_i,u_i} \mathbf{q}_{u_i}$
6: $\quad \mathbf{V}_{i-1} = \mathbf{Q}_{x_i,x_i} + \mathbf{Q}_{x_i,u_i} \mathbf{K}_i + \mathbf{K}_i^T \mathbf{Q}_{u_i,x_i} + \mathbf{K}_i^T \mathbf{Q}_{u_i,u_i} \mathbf{K}_i$
7: $\quad \mathbf{v}_{i-1} = \mathbf{q}_{x_i} + \mathbf{Q}_{x_i,u_i} \mathbf{k}_i + \mathbf{K}_i^T \mathbf{q}_{u_i} + \mathbf{K}_i^T \mathbf{Q}_{u_i,u_i} \mathbf{k}_i$
8: **end for**

Forward Pass
9: **for** each segment $i = 1 : I$ **do**
10: $\quad \delta u_i = \mathbf{K}_i \delta x_{i-1} + \mathbf{k}_i$
11: $\quad \delta x_i \leftarrow \boldsymbol{f}(x_{i-1}, u_i + \delta u_i, n_i) - x_i$
12: **end for**
13: **return** $\delta\mathcal{U}^*$

However, after each update, the state of the next step x_i might change, leading to further control and state changes of the following steps. Therefore, the state and control should be updated iteratively until convergence, as shown in Algorithm 2. Overall, through iteratively updating the state and improving control sequence, iLQR can eventually converge to a fixed-point solution $\{\langle x_i^*, u_i^* \rangle, i \in [1, I]\}$.

Algorithm 2 Iterative LQR (iLQR)

Input: $\mathcal{X}^{(0)}, \mathcal{U}^{(0)}$: the initial state and control sequence; I: number of segments to be optimized; $\{n_i, i \in [1, I]\}$: future network condition; η: step size; M: total number of iterations.
Output: $\{u_i^*, i \in [1, I]\}$: rate and playback speed sequence.
Initialization: $m = 0$
1: **while** $\{u_i\}$ is not converged **and** $m < M$ **do**
2: $\quad \hat{x}_i = x_i^{(m)}$, $\hat{u}_i = u_i^{(m)}$ for all $i \in [1, I]$
3: $\quad \mathbf{F}_i = \nabla_{\mathrm{x}_{i-1}, u_i} \boldsymbol{f}(\hat{\mathrm{x}}_{i-1}, \hat{u}_i)$.
4: $\quad \mathbf{c}_i = \nabla_{\mathrm{x}_{i-1}, u_i} c(\hat{\mathrm{x}}_{i-1}, \hat{u}_i)$.
5: $\quad \mathbf{C}_i = \nabla^2_{\mathrm{x}_{i-1}, u_i} c(\hat{\mathrm{x}}_{i-1}, \hat{u}_i)$.
6: $\quad$ Conduct LQR *Backward Propagation* on state $\delta \mathrm{x}_i$ and control δu_i for all $i \in [1, I]$.
7: $\quad$ Conduct LQR *Forward Pass* with dynamics $\mathrm{x}_i = \boldsymbol{f}(\mathrm{x}_{i-1}, u_i)$ and $u_i = \hat{u}_i + \eta(\mathbf{K}_i(\mathrm{x}_{i-1} - \hat{\mathrm{x}}_{i-1}) + \mathbf{k}_i)$ for all $i \in [1, I]$.
8: $\quad$ Check if u_i for all $i \in [1, I]$ are converged.
9: $\quad m = m + 1$
10: **end while**
11: $u_i^* = u_i^{(m)}$ for all $i \in [1, I]$.
12: **return** $\{u_i^*, i \in [1, I]\}$

4.2 Practical iLQR Implementation

In order to apply iLQR to a practical streaming system, there are still several missing parts to be filled in. ① The "oracle" of future network conditions is not available in practice. In particular, network bandwidth and RTT prediction are necessary. In this paper, we use harmonic mean of the past samples as the predicted value. ② Illustrated by Algorithm 2, to jointly optimize the control sequence for all the I steps, the network condition of each step is required by iLQR. However, the network prediction error will accumulate for a long horizon. To cope with it, we adopt model predictive control (MPC) in which, at step i, the control sequence is optimized for a future horizon $[i+1, i+H]$ using iLQR. Only the control u_{i+1} of step $i+1$ is applied, the system evolves from state x_i to x_{i+1}. At step $i+1$, iLQR calculates a new control sequence for horizon $[i+2, i+H+1]$, and u_{i+2} is applied, so on and so forth. Through taking control in this sliding-window fashion, error accumulation can be controlled. We use horizon length of 10 in our experiments. ③ As described in Sec. 4.1.1, both system dynamic and cost functions should be differentiable. For example, we adopt high order sigmoid function to approximate the $\max(\cdot)$. In addition, the vanilla iLQR solves the optimal control problem without considering upper/lower bounds of control values. However, both video rate and speed control in a practical system should be bounded. To solve this problem, we introduce exponential barrier functions on the control values. ④ iLQR generates continuous-valued optimal control sequence. But the actual controls are discrete-valued. Therefore, at each step, the optimal video rate of the current step and the average speed of the current and the next two steps are quantized to one of the available video rates and playback speeds as the actual action. ⑤ iLQR generates control at the segment level, while the state evolution is at

the chunk level as detailed in Section 3. We convert chunk-based state variables to segment-based ones, then feed them into iLQR.

5 MODEL-FREE SOLUTION: DRL WITH BRANCHING DUELING Q-NETWORK

The iLQR-based solution relies on accurate streaming system model and network condition prediction to obtain the control strategy. Another popular model-free approach to optimal control is reinforcement learning (RL), where an RL agent is trained with sample runs and reinforced by the obtained rewards to converge to the optimal control policy. The RL agent does not need explicit system dynamic model. Instead, it treats the system under control as a black-box, and learns how to interact with it through "trial-and-error" in the training phase to improve its control policy. Indeed, RL, in particular Deep-RL (DRL), has found its applications in video streaming recently [21, 28]. In our context, we would like to train a DRL agent that learns how to simultaneously adjust video rate and playback speed to maximize the user QoE in low-latency live streaming. The immediate new challenge is the expanded action space due to the added playback speed control. Given $|\mathcal{R}|$ available video rates, and $|\mathcal{S}|$ possible playback speeds, the control action for each segment is to choose one rate-speed tuple out of $|\mathcal{R}| \times |\mathcal{S}|$ candidates. For I consecutive segments, the number of possible joint rate-speed control sequences $(|\mathcal{R}| \times |\mathcal{S}|)^I$ will be extremely large when compared with $|\mathcal{R}|^I$ possible control sequences in rate-only adaptation. The conventional DRL algorithms, such as A3C used in [21], cannot efficiently explore the action space when $|\mathcal{R}|$ and $|\mathcal{S}|$ are reasonably large. In this paper, we adopt Branching Dueling Q-network (BDQ) [36], a recently proposed DRL algorithm for large action space, to learn joint rate-speed adaptation policy.

5.1 Overall Architecture and Feature Design

The overall architecture of the BDQ agent is illustrated in Fig. 3. It takes ten features to represent the state of live streaming system. The features are further encoded into shared representation to feed into BDQ model to simultaneously generate video rate and playback speed adaptation polices.

5.1.1 System State Features. The BDQ agent directly takes past system information as input. There is no explicit bandwidth prediction involved. To represent the network dynamic pattern, chunk size and download time are fed into the agent so that bandwidth can be predicted *implicitly*. Buffer length, idle time and video freeze are also used to show the past system evolution. To capture the temporal pattern from the feature sequences, LSTM is adopted. As QoE is also affected by playback latency and fluctuation of video rate and playback speed, the current latency, video rate and playback speed of the last segment are also selected as features. In addition, the indicator feature "player status" can help the agent to accumulate buffer fast during the startup phase.

In total, there are 10 features to define the state of the live streaming system. The first 5 features are temporal sequences of length 15 (corresponding to 15 chunks, each segment consists of 5 chunks), including ***chunk size*** ($r_i\Delta_c$, the data size of chunks), ***download time*** ($\tau_{i,j}$, download duration of chunks), ***buffer length*** ($b_{i,j}$, the buffer length after download each chunk), ***download idle time*** ($z_{i,j}$, the time spent on waiting for the server to prepare the next chunk) and ***video freeze*** ($p_{i,j}$, the duration of freeze during the downloading). The other 5 features are scalars, including ***last video rate*** (r_{i-1}, the video rate of the previous chunk), ***latency*** ($l_{i,j-1}$, the current playback latency), ***player status*** (if the player is in startup or normal playback status) and ***last playback speed*** (s_{i-1}, the previous playback speed). In addition, ***skip/repeat indicator*** is used to represent whether "skip" or "repeat" is chosen in the previous action.

5.1.2 Shared representation. Shown in the middle part of Fig. 3, to capture the state evolution of the system, the temporal sequence features are fed into a two-layer bidirectional long short-term memory (LSTM) to mine the temporal patterns and all the other scalar features are fed into a fully connected (FC) layer. The hidden state of both the LSTM and the FC are flatten/concatenated and taken as the input of the last FC layer. All the LSTM/FC layers are grouped together and named as the shared representation which extracts the hidden representation from the system state.

5.2 Policy Learning with Action Branching

Value-based reinforcement learning algorithms learn the optimal policy through estimating the Q values, i.e., the expected long-term reward $Q(S, a)$ for taking action a under state S. Then the action policy can be obtained as $\pi(S) = \arg\max_a Q(S, a)$. In Deep Q-Network (DQN), the Q function is approximated by a deep neural network (Q-network) parameterized by θ. At the convergence, the Q values should satisfy the Bellman equation [24]:

$$Q^*(S, a; \theta) = r + \gamma \max_{a'} Q^*(S', a'; \theta), \tag{13}$$

which states that the maximum total reward of the current state S is the immediate reward r after taking action a plus the maximum future reward starting from next state S'. To converge to $Q^*(S, a; \theta)$, for a given sample transition from state S to S' after action a, the parameter θ of the Q-network should be updated as:

$$\theta' = \theta + \eta(y - Q(S, a; \theta))\nabla_\theta Q(S, a; \theta), \tag{14}$$

where η is the step size and $y = r + \gamma \max_{a'} Q(S', a'; \theta)$ is the target value. The vanilla DQN has the overestimate problem due to the max function taken in the target value. Double DQN (DDQN) solves the overestimate problem by using two Q-networks (with parameters θ and θ^-) and the target value is

$$y = r + \gamma Q(S', \arg\max_a Q(S', a; \theta), \theta^-). \tag{15}$$

For each update, the action taken at S' is based on the Q network parameterized by θ, while the Q value is returned by the Q network parameterized by θ^-. The parameter θ^- is updated to θ at certain frequency (every 50 episodes in our experiments).

For joint rate-speed adaptation, the total number of candidate rate-speed tuples can be too large for a normal DQN model to handle. We adopt Branching Dueling Q-network (BDQ) [36] to learn video rate and playback speed adaptation policies using different branches. The detailed architecture of BDQ is shown in Fig. 3. The basic idea of BDQ is to train separate Q-networks to learn policies along different action dimensions so that each Q-network only needs to explore the action subspace along one dimension.

More specifically, given the hidden representation, a branching architecture is utilized in BDQ to generate the multi-dimensional

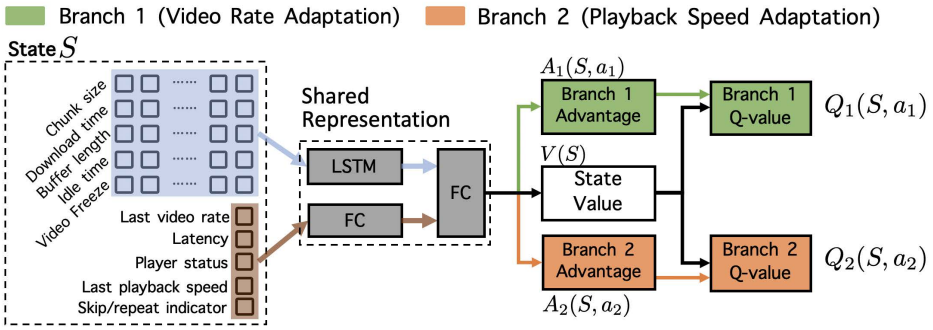


Figure 3: Architecture of BDQ for joint video rate (branch 1) and playback speed (branch 2) adaptation.

output. First of all, the state value $V(S)$ is estimated by a common state value estimator for all the branches. This approach is similar to the dueling network architecture [41] so that a general state value can be generated efficiently. With the help of the common state value estimator, the advantage of actions of branches can also be identified. Specifically, shown in the right part of Fig. 3, the shared hidden representation is distributed among both the video rate and playback speed control branches to generate the state-action based advantage value $A_d(S, a_d)$, $d = 1, 2$. Eventually, we can obtain the Q-value of dimension d as:

$$Q_d(S, a_d) = V(S) + \left(A_d(S, a_d) - \frac{1}{n} \sum_{a'_d \in \mathcal{A}_d} A_d(s, a'_d) \right), \quad (16)$$

where the size of the action space to be explored at each step is $|\mathcal{A}_d|$. If one were to train one DQN for both branches together, the action space size at each step would be $|\mathcal{A}_1| \times |\mathcal{A}_2|$, which significantly increases the complexity of DRL policy training.

According to [36, 41], the average reduction can lead to better performance. With Q-value $Q_d(S, a_d)$ of each branch dimension, the temporal difference target to update the BDQ model is:

$$y = r + \gamma \frac{1}{N} \sum_d Q_d \left(S', \arg\max_{a'_d \in \mathcal{A}_d} Q_d(S', a'_d; \theta_d), \theta_d^- \right), \quad (17)$$

where the number of action dimensions is $N = 2$ in this paper. To train the BDQ model, the state-action reward is defined as the QoE of each video segment. With the trained Q_d networks, the video rate selection and playback speed adaptation at the current state S can be obtained as $\arg\max_{a_d} Q_d(S, a_d)$, $d = 1, 2$ respectively.

6 EVALUATION

6.1 Experiment Configuration

6.1.1 Algorithm Implementation. We implement the proposed solutions and several benchmarks as following:

Pensieve: Serves as a benchmark for DRL based rate-only adaptation. As the algorithm proposed in [21] was designed for video on demand, its latency will be too long for live streaming if we test it as-is. For a fair comparison, we retrain an A3C-based DRL agent according to the settings in [21] with the same latency-aware QoE model (8) used by our algorithms.

STALLION: A heuristic video rate and playback speed adaptation algorithm proposed in [10]. Playback speed is adapted when latency is higher than the target latency, and video rate is selected based on the predicted throughput, considering the past buffer length and latency.

L2A-LL: A low-latency video rate adaptation algorithm based on online convex optimization (OCO) proposed in [16]. The playback speed is controlled by the default algorithm of dash.js. The parameters are kept consistent with the original paper.

LOL: The video rate adaptation algorithm proposed in [18]. All possible video rate combinations in a look-ahead window are exhaustively searched to maximize user QoE defined in (8). Playback speed is calculated based on the current and future buffer/latency heuristically. Compared with settings in [18], we have more video rate levels. To make the exhaustive MPC search run in real-time, we have to reduce the look-ahead window from 5 segments to 3 segments.

iLQR: The implementation of Algorithm 2 with horizon of 10 segments. Harmonic mean of history bandwidth is used to predict bandwidth in future.

iLQR*: Serves as the optimal performance benchmark by using "network oracle". Rate and speed adaptations are calculated offline using Algorithm 2 with the complete network bandwidth trace. The other configuration is the same as iLQR.

BDQ: Our DRL agent described in Sec. 5. The numbers of the video rates and playback speeds are 6 and 7, respectively.

To be compatible with our experiment configurations, e.g., segment duration, video rate levels, and playback speed range, we re-implement STALLION, L2A-LL and LOL by maximally following the original papers. All the algorithms are implemented in Python3. To evaluate the real-time processing performance, iLQR and BDQ are also implemented in a customized web based streaming platform. The results show that 3.91 ms and 55.69 ms are consumed by iLQR and BDQ, respectively, to generate each rate-speed action.

6.1.2 Rate and speed control. All the algorithms share the same set of available video rates $\mathcal{R} = \{0.3, 0.5, 1.0, 2.0, 3.0, 6.0\}$ Mbps. For playback speed control, the available speed set can have 3 choices with $\mathcal{S}_{(3)} = \{0.9, 1.0, 1.1\}$ for iLQR. However, if the playback speed is bounded with ±10% of the normal playback speed, the latency can only be changed by 1 second after displaying 10 seconds of video. Small adjustment might lead to sluggish response to the fast changing streaming environment, leading to QoE degradation. To cope with this, we extend the playback speed range in BDQ so that seven actions are supported, including five possible playback speeds:

Algorithms	# Rate Adaptations	# Speed Adaptations	Bandwidth Prediction
Pensieve	6	N/A	N/A
STALLION	6	3	Sliding Window
L2A-LL	6	3	N/A
LOL	6	3	Harmonic Mean
iLQR	6	3	Harmonic Mean
BDQ	6	7	N/A
iLQR*	6	3	Ground-truth

Table 3: Algorithm Settings and Control Spaces.

{0.75, 0.9, 1.0, 1.1, 1.25} and two other actions: *skip* and *repeat*. "Skip" will skip requesting several video segments (2 segments are used in experiments) therefore latency can be reduced immediately; On the contrary, "repeat" will replay the previous segments (1 segment is used) to increase the buffer/latency to avoid video rebuffering. Both "skip" and "repeat" will introduce additional QoE penalties which is discussed in Sec. 6.1.3. The detailed rate/speed control levels and bandwidth prediction of each algorithm are shown in Table 3.

6.1.3 QoE setting. We use three settings of QoE weights (the coefficients α_1 to α_6 in the QoE model in (8): {1, 1, 2, 2, 0.25, 6}, {1.5, 1, 2, 2, 0.1, 6} and {1, 1, 2, 2, 0.1, 10} to represent "Low-latency Preferred", "High-rate Preferred" and "Freeze Sensitive" users, respectively[2]. We retrain different Pensieve and BDQ models for different QoE settings. For iLQR/iLQR*, we modify the corresponding coefficients of cost terms. For STALLION, the target latency is set to 1.5s for "Latency Preferred" users and 2s for "Rate Preferred" and "Freeze Sensitive" users. The quality $\boldsymbol{q}(r_i)$ is set as $\log(r_i/\mathcal{R}_0)$ where $\mathcal{R}_0$ is the lowest available rate (0.3 Mbps). The penalty of each repeated/skipped segment is 3.

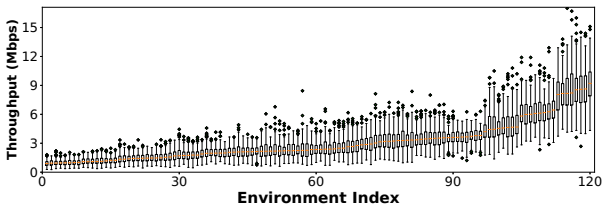


Figure 4: Throughput Statistics of 120 LTE traces.

6.1.4 Simulation setting and network traces. In our system, each segment contains one second of video and each chunk is 200 ms. Each simulated live session is 300 seconds. The LTE throughput dataset provided by [22] is utilized to simulate the network environment. The average and variation of the 120 testing bandwidth traces are illustrated by box plot in Fig. 4. The initial latency for each streaming session is randomly chosen between 3s and 6s. For each segment request, RTT is randomly chosen from 20 to 30 ms. Note that to make fair comparison, for the experiments using the same network trace, all the random values and initial state, e.g. initial latency, are kept the same for all the algorithms.

[2]The weights can be changed based on users' preference to different terms. Our solutions can work with any weight setting and any QoE function.

6.1.5 BDQ Model training. The bandwidth traces from [22] are divided into training (150 traces) and testing (120 traces) datasets. BDQ DRL Agent is trained for 50,000 episodes and 300 actions are taken in each episode. To explore the state-action space comprehensively, we use exponential decay function to control the exploration ratio for the first 20,000 episodes. We also adopt double Q-network approach to handle overestimate. Through trial and error, the DRL agent is expected to "learn" the optimal control policies in the face of uncertain future network conditions.

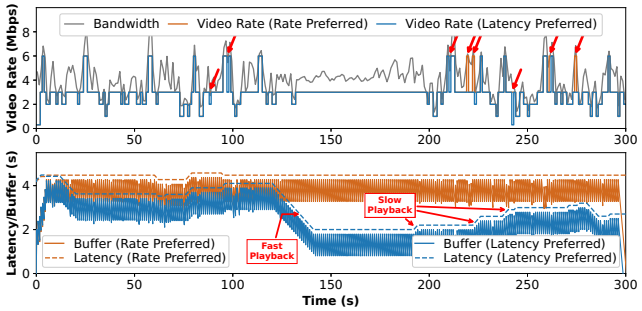


Figure 5: Rate vs. latency for different QoE preferences.

6.2 Optimal Rate-Speed Adaptation with Network Oracle

To gain insights about the trade-offs in live streaming mentioned in Sec. 3.1, we conduct offline experiments using iLQR* for "Latency Preferred" and "Rate Preferred" users. Fig. 5 illustrates the detailed buffer/latency evolution and rate-speed control selection. As iLQR* assumes the future network condition is known, for the period between 120 and 190 seconds when network bandwidth (gray curve) is stable, fast playback is chosen for "latency-preferred" users (shown in blue curves) so that latency is reduced from 4 seconds to below 2 seconds. When network becomes fluctuating again after 200 seconds, by choosing slow playback, buffer is accumulated to increase robustness against bandwidth variation. However, if high video rate is preferred (shown in orange curves), rate selections are more aggressive than "latency-preferred" users, as marked by the red arrows. At the same time, the latency remains constant around the initial latency for most of the session. *This demonstrates that iLQR can find the optimal rate-speed control sequence to strike the best QoE trade-off customized for individual users, as long as bandwidth prediction is accurate.*

6.3 Effectiveness of More Speed Control Levels

To evaluate how much benefit can be gained from increasing speed control levels, we compare BDQ with iLQR under the same network environment with the same QoE weight setting ("Latency Preferred"). Illustrated by the top part of Fig. 6, iLQR's video rate (shown in blue curves) is more stable than BDQ (shown in orange curves). The bottom part of Fig. 6 shows both iLQR and BDQ choose fast playback speeds to reduce the latency but in different fashions. For iLQR, from the beginning to 50 second, fast playback is chosen so that the latency can be reduced from the initial value of 5 seconds to 2 seconds. On the other side, BDQ can achieve similar latency

Table 4: QoE and performance metrics for "Latency Preferred" users (Fr: video freeze; Lat: average latency).

Scenario (μ, σ, l_0)	Pensieve			STALLION			L2A-LL			LOL			iLQR			BDQ			iLQR*		
	QoE	Rate	Lat	QoE	Rate	Lat	QoE	Rate	Lat	QoE	Rate	Lat	QoE	Rate	Lat	QoE	Rate	Lat	QoE	Rate	Lat
(1.2,0.3,5.0)	-53	**1.0**	5.4	32	0.55	**1.4**	21	0.52	1.6	73	0.74	1.6	109	0.79	1.7	**154**	0.99	1.8	131	0.84	1.6
(1.6,0.5,4.4)	-21	1.2	5.0	139	0.8	**1.3**	163	0.86	1.5	192	0.96	1.5	205	1.0	1.7	**206**	**1.3**	1.6	**206**	1.0	1.5
(2.3,0.6,4.5)	117	**2.1**	4.8	175	0.99	**1.3**	235	1.0	1.5	261	1.3	1.4	318	1.7	1.7	**354**	**2.1**	1.6	345	1.8	1.5
(2.6,0.8,5.9)	87	**2.4**	6.3	188	1.1	**1.5**	216	1.0	1.7	284	1.5	1.6	335	2.0	2.0	**376**	**2.4**	2.2	356	2.1	1.8
(3.0,1.0,5.6)	140	**2.6**	6.0	230	1.4	**1.5**	331	1.6	1.6	332	2.0	1.6	358	2.4	2.0	392	**2.6**	2.2	**414**	2.3	1.6
(3.6,1.0,5.9)	200	**3.2**	6.2	351	2.0	**1.5**	419	2.0	1.7	447	2.6	1.7	460	2.8	1.9	458	**3.2**	2.4	**491**	2.8	1.7
(6.1,1.4,4.9)	472	**5.6**	5.1	473	2.7	**1.3**	564	3.0	1.4	584	3.4	1.4	608	4.5	1.8	608	4.2	1.4	**622**	4.4	1.5
(6.4,1.5,3.4)	569	**5.4**	3.6	531	2.7	**1.0**	578	3.0	1.3	587	3.5	1.3	645	5.1	1.6	**673**	5.3	1.6	**673**	5.1	1.4
Overall	189	**3.0**	5.3	265	1.5	**1.3**	316	1.6	1.5	345	2.0	1.5	380	2.5	1.8	402	2.8	1.8	**405**	2.5	1.6

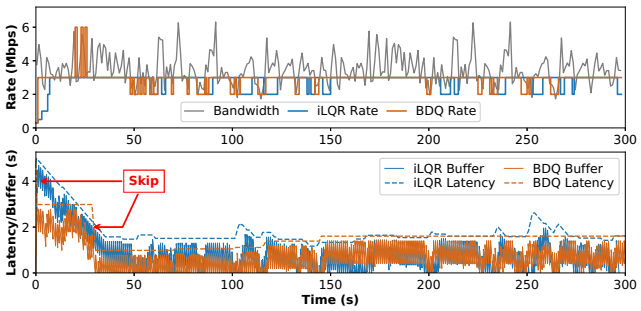


Figure 6: Effect of Increasing Speed Control Ranges

reduction within 28 second by taking the "skip" action twice at 1 and 28 second, respectively. Both one "skip" action (skipping 2 segments) and 20 consecutive ×1.1 playbacks can reduce latency by 2 seconds. Even though the "skip" action introduces more playback related penalty than fast playback action, BDQ still chooses the "skip" action at the beginning of the live streaming session so that the latency penalty for all the following segments can be reduced. *In other word, "skip" incurs one-time penalty to achieve higher accumulated reward for the entire streaming session.*

6.4 Algorithm Comparison

The detailed QoE metrics comparisons for "Latency Preferred" users under 8 network scenarios are shown in Table 4. Each scenario uses a different bandwidth trace and a randomly chosen initial latency l_0 between 3 and 6 seconds. The bandwidth trace is characterized by the average μ and standard derivation σ. Overall, iLQR* with future network oracle performs the best. It achieves the highest QoE for 5 of 8 scenarios. BDQ, even without network oracle, can still achieve the highest QoE for 5 scenarios (thanks to its unique skip and repeat control) with 2 are tied for first place with iLQR*. For iLQR, close-to-the-optimal performance can be achieved in most cases. The performance gap between iLQR and iLQR* is mainly due to the errors of harmonic mean based bandwidth prediction: under-estimate leads to lower video rate; over-estimate leads to higher freeze/latency. We expect more accurate real-time bandwidth estimation models can further improve the performance of iLQR.

Without playback speed adaptation, the performance of Pensieve highly depends on the initial latency. If the initial latency is high, it is impossible for Pensieve to reduce the latency to achieve high QoE. On the other hand, if the initial latency is low, for the example in scenario 7, the initial latency is 4.9s, which is long enough for the low relative bandwidth variation (σ/μ is small), Pensieve's average latency for the whole session is 5.1s, while all other rate-speed adaptation algorithms can reduce the average latency to below 2s. *This demonstrates that speed-adaptation algorithms can work with "bad" initial latency and adjust playback speed based on bandwidth condition to balance various QoE metrics.*

STALLION tunes the video rate control parameters based on the past observations and presets a target latency for playback speed adaptation. Even though the lowest latency is achieved by STALLION in all eight scenarios, its overall QoE is far from the optimum. For instance, STALLION always reduces the latency below the 1.5s target latency and selects video rate conservatively, which leads to lower QoE than the other algorithms with playback adaptation. Through selecting video rate by solving an online convex optimization (OCO) problem, L2A-LL achieves higher video rate and QoE than STALLION. Since L2A-LL does not use user QoE as its objective function in OCO, the overall QoE of L2A-LL is still not high enough. In LOL, video rate is selected more aggressively than STALLION and L2A-LL by exhaustively searching through all video rate combinations to maximize user QoE. It achieves higher QoE than STALLION and L2A-LL. However, since LOL only optimizes video rate marginally, its achieved QoE is still lower than our algorithms that jointly optimize video rate and playback speed. The overall QoE achieved by iLQR and BDQ are 10.1% and 16.5% higher than LOL respectively.

While comparing BDQ and iLQR*, higher rate is chosen by BDQ; on the contrary, iLQR* plans rate selection using future network oracle and avoids video freeze by selecting lower video rate. Detailed speed selection is shown in Fig. 7. In some cases, iLQR* chooses ×0.9 playback speed to slowdown the buffer consumption with the help of network "oracle". However, for STALLION, L2A-LL, LOL and iLQR, lower rate, instead of slow playback speed, might be chosen to maintain the buffer length. When the initial latency is high, all other algorithms except BDQ and Pensieve reduces latency by displaying ×1.1 faster. With "skip" action, BDQ can reduce latency

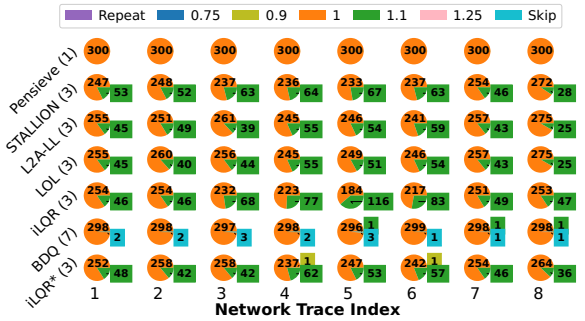
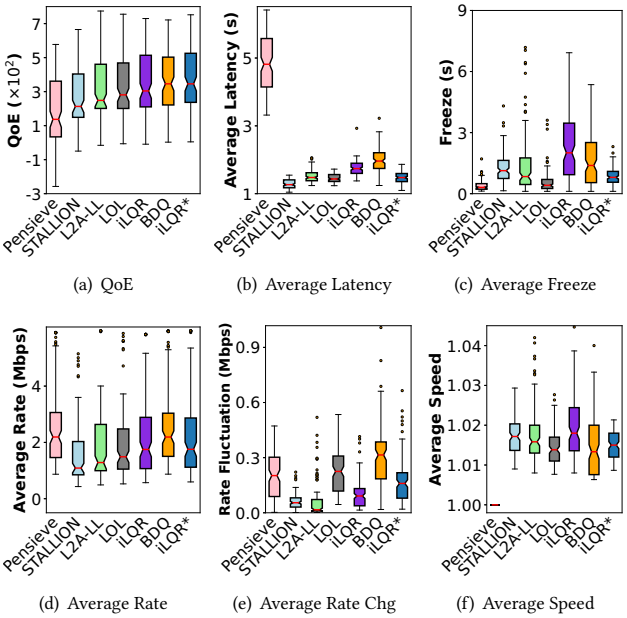


Figure 7: Distribution of playback speed.

more aggressively by skipping video segments. But if users are sensitive to missing content, BDQ should adopt a different strategy to reduce the latency by using a higher weight for the "skip" penalty.



(a) QoE (b) Average Latency (c) Average Freeze

(d) Average Rate (e) Average Rate Chg (f) Average Speed

Figure 8: QoE metrics comparison over 120 LTE traces.

We also conduct extensive experiments for "Latency Preferred" users on all the 120 testing traces. A summary of performance metrics is shown in Fig. 8. As expected, iLQR* outperforms all the others. With enhanced speed control, BDQ can achieve similar QoE as iLQR*, which is higher than iLQR. STALLION's QoE is below the average due to the limitation of the heuristic algorithm. By taking advantage of the OCO, L2A-LL obtains better performance. Exhaustive search helps LOL get an even higher QoE. However, it's not scalable with more video rate options. Pensieve performs the worst due to the lack of speed adaptation. Meanwhile, different algorithms balance QoE trade-offs in different ways. For example, Fig. 8(b), 8(d) and 8(e) show that BDQ can achieve higher video rate but incur higher latency and rate fluctuation. Oppositely, lower latency and rate fluctuation are obtained by iLQR and iLQR* but with lower video rate. The average speeds for all the algorithms

Table 5: Detailed QoE metrics comparison among different QoE preferences for the 7th scenario in Table 4.

Preference	Algorithms	Rate	Freeze	Latency	QoE
Latency Preferred	Pensieve	**5.58**	**0.17**	5.07	472
	STALLION	2.7	0.39	**1.3**	473
	L2A-LL	2.98	0.22	1.45	564
	LOL	3.42	0.22	1.43	584
	iLQR	4.53	1.11	1.84	608
	BDQ	4.2	0.41	1.36	608
	iLQR*	4.42	0.59	1.5	**622**
Rate Preferred	Pensieve	5.63	**0.18**	5.09	1141
	STALLION	2.83	0.22	2.14	843
	L2A-LL	2.98	0.22	**1.45**	972
	LOL	4.78	0.22	1.93	1085
	iLQR	5.57	0.19	4.8	1119
	BDQ	**5.66**	0.34	5.35	1126
	iLQR*	5.62	0.25	3.54	**1155**
Freeze Sensitive	Pensieve	5.64	**0.2**	5.1	705
	STALLION	2.83	0.22	2.14	534
	L2A-LL	2.98	0.22	**1.45**	628
	LOL	4.6	0.22	1.93	676
	iLQR	5.47	0.59	3.52	711
	BDQ	**5.67**	0.23	5.32	719
	iLQR*	5.45	0.24	2.69	**722**

are similar. With future network information available, iLQR* can avoid video freeze efficiently. STALLION, L2A-LL and LOL can achieve low latency and low freeze by sacrificing video rate. *Overall, joint video rate and playback speed adaptation can improve QoE dramatically.*

6.5 Adapting to Diverse User QoE Preferences

Users have different preferences for different QoE metrics. Our proposed algorithms can be flexibly tuned to satisfy diverse user QoE preferences by adjusting the weights in the QoE model. We compare all the algorithms under different QoE weight settings, including "Latency Preferred", "Rate Preferred" and "Freeze Sensitive" discussed in Sec. 6.1.3. The numerical comparison among different QoE preferences is shown in Table 5. For "Latency Preferred" users, similar with the observation in Sec. 6.4, all the model-based (iLQR) and model-free (BDQ) algorithms can achieve close-to-optimal QoE (compared with iLQR*) with the help of joint rate-speed adaptation. Without playback speed adjustment, Pensieve performs the worst. The performance of STALLION, L2A-LL and LOL is worse than the proposed algorithms due to their conservative rate selections and separated video rate and playback speed adaptations. When rate is preferred over latency, Pensieve's performance is dramatically improved to be similar to iLQR*. In the "Freeze Sensitive" case, in which rate and latency are not as important as freeze, Pensieve's performance is very close to iLQR/BDQ by accumulating long buffer to reduce the risk of freeze. When trained with the Freeze-sensitive QoE, BDQ also learns a strategy similar to Pensieve that sacrifices latency for low freeze and high video rate.

With different QoE weight settings, all of our proposed algorithms can adjust their adaptation strategies correspondingly. For example, when the latency is preferred, the average latency is reduced to around 1.5s. If rate is more important, the algorithms stream with higher video rate and higher latency. With "Freeze Sensitive" setting, the freeze time is reduced with slight sacrifice of video rate and latency. However, Pensieve, STALLION and L2A-LL

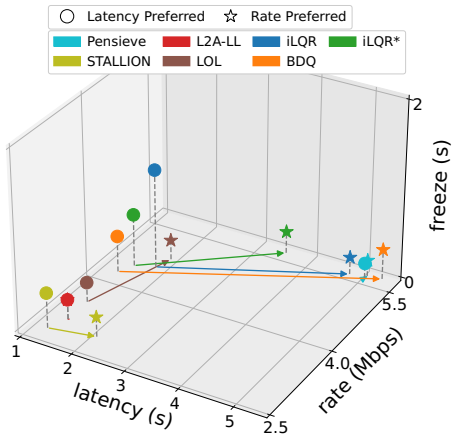


Figure 9: Metrics for different QoE preferences.

are limited by their designs and cannot flexibly trade-off among these three important QoE metrics. LOL can explore the best video rate selection for a look-ahead window. But without joint adaptation with playback speed, the QoE performance is still lower than iLQR/BDQ. Fig. 9 also demonstrates the capability of our proposed algorithms to tailor their streaming strategies to user QoE preferences. By changing preference from low-latency to high-rate, all of our proposed algorithms, including iLQR, BDQ and iLQR*, choose to use longer latency so that buffer can be accumulated and higher video rate can be delivered (all the arrows point from bottom left to upper right in Fig. 9). In addition, video freeze can also be avoided when latency/buffer is long (the heights of the star marks are lower than the circle marks). While comparing BDQ with iLQR/iLQR*, we again find BDQ is more aggressive by choosing higher video rate in both cases. Especially when rate is preferred, BDQ chooses to deliver much higher video rate with longer latency than iLQR/iLQR*. However, conservative rate selection is preferred by iLQR/iLQR* even when rate is preferred. This is consistent with the observation in the comparison between them in Sec. 6.4.

6.6 Model-based or Model-free Joint Adaptation?

Having seen the performance of iLQR (including iLQR*) and BDQ, there is no clear winner. They both solve the QoE maximization problem defined in Sec. 3.3, following model-based and model-free optimal control approaches respectively. Given the QoE model, iLQR (as described in Algorithm 1 and 2) obtains the optimal adaptation strategy semi-analytically based on the live streaming system model developed in Sec. 3 and network bandwidth prediction. On the other hand, BDQ does not need the live streaming model, nor explicit network bandwidth prediction. It learns the adaptation policy through training deep Q networks using live streaming session samples. The bandwidth prediction is implicitly done when the bandwidth history is processed by the LSTM layer in our BDQ architecture in Fig. 3. For adoption in practical streaming systems, iLQR and BDQ have their own pros and cons.

(1) iLQR does not need any training, and iLQR based algorithms can be quickly adjusted to work with new system dynamic model and new user QoE preferences. On the other hand, BDQ has to be trained with a large number of sample traces, and it has to be retrained whenever the system dynamics or user QoE preferences change.
(2) With an accurate system model and network bandwidth prediction, iLQR obtains the optimal strategy for an individual system in a specific network environment. The actual quality of the obtained solution depends on the fidelity of the system model and the accuracy of the predicted network bandwidth. In our experiments, the performance gap between iLQR and iLQR* is due to the error of harmonic mean based bandwidth prediction.
(3) When trained with a large set of representative sample traces, BDQ learns adaptation policies good for a class of systems in a range of network environments. The learned policies can generalize well to systems and environments similar to the training samples. On the other hand, a general policy optimized for multiple scenarios, when applied to an individual scenario, cannot beat the policy optimized just for that scenario, the so-called *price of generalization*. In our experiments, BDQ with 7 speed control levels cannot always beat iLQR* with only 3 speed control levels.
(4) iLQR exploits the domain knowledge of how live streaming system works to explicitly calculate control solutions which are *understandable* and *debuggable*. On the other hand, BDQ treats live streaming system as a black box, and the solution derived from Q networks cannot be intuitively interpreted and logically debugged.

Based on the previous discussion, our general recommendation is that if a live streaming system can be well modelled and network bandwidth can be accurately estimated, iLQR is the natural choice for joint rate-speed adaptation. On the other hand, for special live streaming systems that are hard to be modelled, and/or network environments for which accurate bandwidth predictions are not possible, BDQ can be used to train adaptation policies with representative sample traces in end-to-end fashion. Meanwhile, DRL is an active research field. Low-latency live streaming can certainly benefit from emerging DRL frameworks and techniques. In particular, we will explore *model-based reinforcement learning* to combine the merits of model-based and model-free adaptation.

7 CONCLUSIONS

Low-latency live streaming requires joint adaptation of video rate and playback speed to balance various QoE metrics. We developed a detailed dynamic model to understand the interplay between rate and speed adaptations. We further built model-based and model-free joint rate-speed adaptation algorithms to maximize the aggregate QoE. Through extensive simulations driven by real network traces, we demonstrated that our proposed joint adaptation algorithms significantly outperform rate-only adaptation algorithms and the recently proposed low-latency video streaming algorithms that separately adapt video rate and playback speed without joint optimization. Our proposed algorithms can achieve close-to-optimal performance in a wide-range of network conditions.

REFERENCES

[1] [n.d.]. *ACM MMSys 2020 grand challenge.* https://github.com/twitchtv/acm-mmsys-2020-grand-challenge

[2] [n.d.]. *Live Streaming vs. Traditional Live Broadcasting: What's the Difference.* https://www.wowza.com/blog/streaming-vs-cable-satellite-broadcasting

[3] [n.d.]. *Video Streaming Latency Report.* https://www.wowza.com/wp-content/uploads/Streaming-Video-Latency-Report-Interactive-2020.pdf

[4] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. 2018. Oboe: auto-tuning video ABR algorithms to network conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 44–58.

[5] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N Pistikopoulos. 2002. The explicit linear quadratic regulator for constrained systems. *Automatica* 38, 1 (2002), 3–20.

[6] Abdelhak Bentaleb, Praveen Kumar Yadav, Wei Tsang Ooi, and Roger Zimmermann. 2020. DQ-DASH: A Queuing Theory Approach to Distributed Adaptive Video Streaming. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 16, 1 (2020), 1–24.

[7] Nassima Bouzakaria, Cyril Concolato, and Jean Le Feuvre. 2014. Overhead and performance of low latency live streaming using MPEG-DASH. In *IISA 2014, The 5th International Conference on Information, Intelligence, Systems and Applications*. IEEE, 92–97.

[8] Wael Cherif, Youenn Fablet, Eric Nassor, Jonathan Taquet, and Yuki Fujimori. 2015. DASH fast start using HTTP/2. In *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. 25–30.

[9] Leana Golubchik, John CS Lui, and Richard Muntz. 1995. Reducing I/O demand in video-on-demand storage servers. In *Proceedings of the 1995 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*. 25–36.

[10] Craig Gutterman, Brayn Fridman, Trey Gilliland, Yusheng Hu, and Gil Zussman. 2020. Stallion: video adaptation algorithm for low-latency video streaming. In *Proceedings of the 11th ACM Multimedia Systems Conference*. 327–332.

[11] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2015. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. *ACM SIGCOMM Computer Communication Review* 44, 4 (2015), 187–198.

[12] ISO. 2013. *Common media application format (CMAF) for segmented media.* https://www.iso.org/standard/71975.html

[13] Junchen Jiang, Vyas Sekar, and Hui Zhang. 2012. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. 97–108.

[14] Xiaolan Jiang and Yusheng Ji. 2019. HD3: Distributed Dueling DQN with Discrete-Continuous Hybrid Action Spaces for Live Video Streaming. In *Proceedings of the 27th ACM International Conference on Multimedia*. 2632–2636.

[15] Mark Kalman, Eckehard Steinbach, and Bernd Girod. 2004. Adaptive media playout for low-delay video streaming over error-prone channels. *IEEE Transactions on Circuits and Systems for Video Technology* 14, 6 (2004), 841–851.

[16] Theo Karagkioules, Rufael Mekuria, Dirk Griffioen, and Arjen Wagenaar. 2020. Online learning for low-latency adaptive streaming. In *Proceedings of the 11th ACM Multimedia Systems Conference*. 315–320.

[17] Zhi Li, Xiaoqing Zhu, Joshua Gahm, Rong Pan, Hao Hu, Ali C Begen, and David Oran. 2014. Probe and adapt: Rate adaptation for HTTP video streaming at scale. *IEEE Journal on Selected Areas in Communications* 32, 4 (2014), 719–733.

[18] May Lim, Mehmet N Akcay, Abdelhak Bentaleb, Ali C Begen, and Roger Zimmermann. 2020. When they go high, we go low: low-latency live streaming in dash. js with LoL. In *Proceedings of the 11th ACM Multimedia Systems Conference*. 321–326.

[19] Chenghao Liu, Imed Bouazizi, and Moncef Gabbouj. 2011. Rate adaptation for adaptive HTTP streaming. In *Proceedings of the second annual ACM conference on Multimedia systems*. 169–174.

[20] Chenghao Liu, Imed Bouazizi, Miska M Hannuksela, and Moncef Gabbouj. 2012. Rate adaptation for dynamic adaptive streaming over HTTP in content distribution network. *Signal Processing: Image Communication* 27, 4 (2012), 288–311.

[21] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 197–210.

[22] Lifan Mei, Runchen Hu, Houwei Cao, Yong Liu, Zifa Han, Feng Li, and Jin Li. 2019. Realtime Mobile Bandwidth Prediction Using LSTM Neural Network. In *International Conference on Passive and Active Network Measurement*. Springer, 34–47.

[23] Konstantin Miller, Emanuele Quacchio, Gianluca Gennari, and Adam Wolisz. 2012. Adaptation algorithm for adaptive streaming over HTTP. In *2012 19th international packet video workshop (PV)*. IEEE, 173–178.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[25] Haitian Pang, Zhi Wang, Chen Yan, Qinghua Ding, Kun Yi, Jiangchuan Liu, and Lifeng Sun. 2018. Content Harvest Network: Optimizing First Mile for Crowdsourced Live Streaming. *IEEE Transactions on Circuits and Systems for Video Technology* 29, 7 (2018), 2112–2125.

[26] Roger Pantos and William May. 2017. *HTTP live streaming.* Technical Report.

[27] Huan Peng, Yuan Zhang, Yongbei Yang, and Jinyao Yan. 2019. A Hybrid Control Scheme for Adaptive Live Streaming. In *Proceedings of the 27th ACM International Conference on Multimedia*. 2627–2631.

[28] Satadal Sengupta, Niloy Ganguly, Sandip Chakraborty, and Pradipta De. 2018. HotDASH: Hotspot Aware Adaptive Video Streaming Using Deep Reinforcement Learning. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE, 165–175.

[29] Iraj Sodagar. 2011. The mpeg-dash standard for multimedia streaming over the internet. *IEEE MultiMedia* 18, 4 (2011), 62–67.

[30] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. 2016. BOLA: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 1–9.

[31] Eckehard Steinbach, Niko Farber, and Bernd Girod. 2001. Adaptive playout for low latency video streaming. In *Proceedings 2001 International Conference on Image Processing (Cat. No. 01CH37205)*, Vol. 1. IEEE, 962–965.

[32] Kairan Sun and Dapeng Wu. 2016. MPC-based delay-aware fountain codes for live video streaming. In *2016 IEEE international conference on communications (ICC)*. IEEE, 1–6.

[33] Liyang Sun, Yixiang Mao, Tongyu Zong, Yong Liu, and Yao Wang. 2020. Flocking-based live streaming of 360-degree video. In *Proceedings of the 11th ACM Multimedia Systems Conference*. 26–37.

[34] Liyang Sun, Tongyu Zong, Yong Liu, Yao Wang, and Haihong Zhu. 2019. Optimal Strategies for Live Video Streaming in the Low-latency Regime. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. IEEE, 1–4.

[35] Viswanathan Swaminathan and Sheng Wei. 2011. Low latency live video streaming using HTTP chunked encoding. In *2011 IEEE 13th International Workshop on Multimedia Signal Processing*. IEEE, 1–6.

[36] Arash Tavakoli, Fabio Pardo, and Petar Kormushev. 2018. Action branching architectures for deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[37] Guibin Tian and Yong Liu. 2012. Towards agile and smooth video adaptation in dynamic HTTP streaming. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. 109–120.

[38] Emanuel Todorov and Weiwei Li. 2005. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, American Control Conference, 2005.* IEEE, 300–306.

[39] Jeroen Van Der Hooft, Stefano Petrangeli, Tim Wauters, Rafael Huysegems, Tom Bostoen, and Filip De Turck. 2018. An HTTP/2 push-based approach for low-latency live streaming with super-short segments. *Journal of Network and Systems Management* 26, 1 (2018), 51–78.

[40] Chen Wang, Jianfeng Guan, Tongtong Feng, Neng Zhang, and Tengfei Cao. 2019. BitLat: Bitrate-adaptivity and Latency-awareness Algorithm for Live Video Streaming. In *Proceedings of the 27th ACM International Conference on Multimedia*. 2642–2646.

[41] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. 2016. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*. PMLR, 1995–2003.

[42] Wikipedia contributors. 2021. Chunked transfer encoding — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Chunked_transfer_encoding&oldid=997959340 [Online; accessed 5-March-2021].

[43] Lan Xie, Chao Zhou, Xinggong Zhang, and Zongming Guo. 2017. Dynamic threshold based rate adaptation for HTTP live streaming. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–4.

[44] Gang Yi, Dan Yang, Abdelhak Bentaleb, Weihua Li, Yi Li, Kai Zheng, Jiangchuan Liu, Wei Tsang Ooi, and Yong Cui. 2019. The ACM Multimedia 2019 Live Video Streaming Grand Challenge. In *Proceedings of the 27th ACM International Conference on Multimedia*. 2622–2626.

[45] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *ACM SIGCOMM Computer Communication Review*, Vol. 45. ACM, 325–338.

[46] Guanghui Zhang and Jack YB Lee. 2019. LAPAS: Latency-aware playback-adaptive streaming. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 1–6.

[47] Yin Zhao, Qi-Wei Shen, Wei Li, Tong Xu, Wei-Hua Niu, and Si-Ran Xu. 2019. Latency Aware Adaptive Video Streaming using Ensemble Deep Reinforcement Learning. In *Proceedings of the 27th ACM International Conference on Multimedia*. 2647–2651.

[48] Chao Zhou, Xinggong Zhang, Longshe Huo, and Zongming Guo. 2012. A control-theoretic approach to rate adaptation for dynamic HTTP streaming. In *2012 Visual Communications and Image Processing*. IEEE, 1–6.