Towards Optimal Low-latency Live Video Streaming

Liyang Sun, Tongyu Zong, Siquan Wang, Yong Liu, *Fellow, IEEE* and Yao Wang, *Fellow, IEEE*

***Abstract*—Low-latency is a critical user Quality-of-Experience (QoE) metric for live video streaming. It poses significant challenges for streaming over the Internet. In this paper, we explore the design space of low-latency live streaming by developing dynamic models and optimal adaptation strategies to establish QoE upper bounds as a function of the allowable end-to-end latency. We further develop practical live streaming algorithms within the iterative Linear Quadratic Regulator (iLQR) based Model Predictive Control and Deep Reinforcement Learning frameworks, namely MPC-Live and DRL-Live, to maximize user live streaming QoE by adapting the video bitrate while maintaining low end-to-end video latency in dynamic network environment. Through extensive experiments driven by real network traces, we demonstrate that our live streaming algorithms can achieve close-to-optimal performance within the latency range of two to five seconds.** [1]

***Index Terms*—live streaming, chunk-base encoding**

I. Introduction

VIDEO currently accounts for more than 70% of the Internet traffic. It is projected that 82% of the Internet traffic will be made up of video in 2022, and live video streaming will contribute 17% of the Internet traffic [1]. To deliver a high level of user Quality-of-Experience (QoE), video needs to be streamed at high rate while avoiding video freeze and minimizing rate fluctuations. To achieve these goals in the face of dynamic network conditions, Video-on-Demand (VoD) streaming solutions, such as Dynamic Adaptive Streaming over HTTP (MPEG-DASH) [2] and HTTP Live Streaming (HLS) [3], prefetch video segments into a video buffer of 10 to 30 seconds or longer to maintain continuous and smooth video playback. For live streaming, users are additionally sensitive to the end-to-end (or screen-to-screen) video latency, namely the time lag from the moment when a video scene occurs till a user sees it on her screen. In the traditional TV broadcast system, the video latency with a mean of 6 seconds could be achieved [4]. By contrast, the current live streaming latencies on Over-the-Top (OTT) devices range from 10 to 30 seconds [5]. This long video latency could be detrimental for user QoE. For example, OTT users watching the World Cup Football final may have to wait for more than ten seconds to see a goal after their neighbors watching cable TV cheer for it. Long video latency will simply ruin the "live" experience.

As illustrated in Fig. 1, the end-to-end latency consists of several delay components incurred at video uploading, encoding, packaging, downloading, decoding and rendering. To squeeze for low end-to-end latency, all components have to be jointly optimized to streamline the entire live video reproduction process. On the encoding/packaging side, segment-based encoding/packaging can be pipelined to shorten the latency for getting a video frame ready for transmission. To achieve low-latency downloading/rendering, each segment has to be downloaded/rendered within a short time window after it is generated, and the client-side video buffer has to be short, which leads to a very tight error margin for video rate adaptation: if the selected video rates for segments are higher than the available bandwidth, the video buffer will drain quickly, running at high risk of video freezes. Each freeze will add to the playback latency of the subsequent video segments. To catch up with the non-stop live event, the streaming session has to be resynchronized by skipping video segments falling behind their playback deadlines.

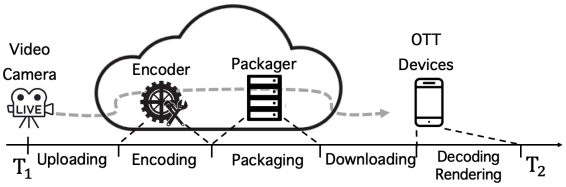


Fig. 1: End-to-End Video Latency in Live Streaming

In this paper, we explore the design space of low-latency live video streaming by developing dynamic models and optimal rate adaptation strategies to establish QoE upper bounds as a function of the allowable end-to-end latency. We further develop practical streaming algorithms within the iterative Linear Quadratic Regulator (iLQR) based Model Predictive Control (MPC) and Deep Reinforcement Learning (DRL) frameworks, namely MPC-Live and DRL-Live, to maximize user live streaming QoE by adapting the video bitrate while maintaining low end-to-end video latency in dynamic network environment. To minimize the end-to-end latency, chunk-based video packaging and streaming are adopted in our system. Each video segment ($\sim$ 1 second) is divided into multiple shorter video chunks ($\sim$ 200 ms), and the processes of encoding, streaming and decoding are operated in a pipelined fashion. MPC and DRL algorithms are developed for online video rate adaption based on the current system state and the predicted network conditions to strike the desirable balance between *video quality, playback latency, video freeze and skip*. Our main contributions are as follows:

1) We characterize the feasible video rate region and video quality upper bound as a function of the allowable video latency. We build detailed dynamic models for live streaming that capture the interplay between video rate adaption, buffer evolution, playback latency, video freeze/skip.

[1]L. Sun, T. Zong, S. Wang, Y. Liu, and Y. Wang are with the Department of Electrical and Computer Engineering, Tandon School of Engineering, New York University, Brooklyn, NY 11201 USA (e-mail: ls3817@nyu.edu; tz1178@nyu.edu; sw4112@nyu.edu; yongliu@nyu.edu; yw523@nyu.edu).

2) We design MPC type of streaming algorithms to explore the optimal video rate selection with network bandwidth estimation in finite future time horizon. We also develop a DRL-based algorithm that adapts video rate to maximize the long-term QoE.
3) We investigate the benefit of chunk-based video packaging and streaming, and demonstrate that it is superior than the segment-based design in achieving low latency.
4) We conduct performance evaluation of the proposed models and streaming algorithms through extensive numerical case studies and live streaming experiments driven by real network traces. We demonstrate that the proposed MPC and DRL based live streaming algorithms can achieve close-to-optimal performance within the latency range of two to five seconds.

II. Related Work

The traditional realtime multimedia transmission protocols include RTP [6], RTSP [7] and RTCP [8]. Real-Time Messaging Protocol (RTMP) [9] by Adobe became popular on Flash based platforms. Most of the current video streaming systems are running over HTTP, including Microsoft's Smooth Streaming (MSS) [10], Adobe's HTTP Dynamic Streaming (HDS) [11], Apple's HTTP Live Streaming (HLS) [3], and Dynamic Adaptive Streaming over HTTP (MPEG-DASH) [2]. Various adaptive streaming algorithms have been proposed to improve the performance of streaming systems [12]. Rate-based algorithm PANDA [13] was proposed to select video rate through bandwidth prediction. Buffer-based algorithms adapt the video rate based on video buffer evolution, e.g., PID [14], [15] and BBA [16]. An online control algorithm named BOLA [17] uses the Lyapunov optimization technique to maximize QoE. A video adaption algorithm based on the optimal control technique, MPC, was proposed in [18]. More recently, Deep Reinforcement Learning (DRL) techniques were applied to video rate adaption [19], [20].

However, all the previous algorithms followed the segment-based design, and none of them was optimized for low-latency live streaming. There were some related efforts to achieve low-latency in live streaming. Buffer-based low-latency HTTP live streaming algorithm was proposed in [21]. PID controller was used for live streaming over mobile network [22]. In order to achieve low latency, the benefit of HTTP chunked encoding was studied in [23]. However, in [24], the authors discussed the overhead of chunked-transfer encoding. Different from the related studies, we studied the optimal rate adaption strategy in the low latency regime under the MPC framework in [25]. Furthermore, in this paper, we thoroughly explore the design space of low-latency live streaming and develop chunk-based streaming strategies including MPC with iLQR and DRL based solutions to optimally trade-off various QoE metrics of live streaming.

III. Model of Low-latency Live Video Streaming

To formally explore the design space of low-latency live streaming, we start with characterizing the feasible video rate region with the complete future network condition oracle. We then study the online optimal control strategy given network information in a short future time horizon.

A. Playback Latency vs. Feasible Video Rate Region

Given the time-varying available bandwidth $w(t)$ for a video streaming session over $[0, T]$, assuming the video streaming rate $r(t)$ can vary continuously at infinitesimal granularity, for any streaming strategy, the total video playback rate is bounded by the total available bandwidth, i.e.,

$$\int_0^T r(\tau)d\tau \le \int_0^T w(\tau)d\tau. \tag{1}$$

The user perceived video quality is normally modeled as a concave function $Q(r(t))$ of video rate. Due to the Jensen's inequality, the average video quality satisfies:

$$\frac{1}{T}\int_0^T Q(r(\tau))d\tau \le Q\left(\frac{1}{T}\int_0^T r(\tau)d\tau\right) \le Q(\bar{w}), \tag{2}$$

where $\bar{w} = \frac{1}{T}\int_0^T w(\tau)d\tau$ is the average available bandwidth. *The highest video quality can be achieved when the video rate equals to the average bandwidth.* In practice, if the video is pre-recorded, the optimal rate of $r^*(t) = \bar{w}$ can be always achieved with the *download-then-play* strategy, but a user has to wait until the whole video file is completely downloaded, i.e., the playback latency is T.[2] To shorten the playback latency, one can also do VoD streaming by waiting for an initial latency of l_0, then playback at the constant rate,

$$r(t) = \bar{w}\mathbb{1}\{t \in [l_0, T + l_0]\},$$

where $\mathbb{1}(\cdot)$ is the indicator function. To avoid video freeze, l_0 should be chosen such that

$$\int_0^{\min(t,T)} w(\tau)d\tau \ge \bar{w}(t - l_0), \quad \forall t \in [l_0, T + l_0],$$

where the left-hand side is the accumulative video data that can be downloaded up to time t, and the right-hand side is the accumulative video rate that needs to be played up to t given the initial playback latency of l_0.

Both *download-then-play* and VoD streaming enjoy the freedom of downloading any part of the video with bandwidth $w(t)$ at any time $t \in [0, T]$ to achieve smooth and high-quality video playback. However, for live video streaming, a video segment generated at time t cannot benefit from any bandwidth before t. Additionally, live content has stringent playback latency requirement, which means a segment downloaded after its playback deadline will be useless. As a result, the time window for downloading a live video segment is limited between its generation time and playback deadline.

In DASH, video is divided into multiple temporal segments that are encoded and delivered independently. As shown in Fig. 2, assuming each video segment contains Δ seconds of video, and the i-th video segment is available for downloading at time $i\Delta$, then the download window for the i-th segment is $(i\Delta, (i + l - 1)\Delta]$, where $l\Delta$ is the target playback latency,

[2]In video download, the video length T_v can be different from the download session length, then the highest video rate achievable is $\frac{\bar{w}T}{T_v}$.

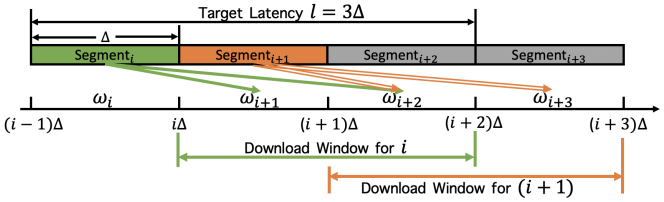


Fig. 2: Segment Download Windows with Latency of 3Δ

$(l-1)\Delta$ is the size of download window and l is greater than 2. Let r_i be the video rate for segment i, then for any consecutive sequence of segments from i_1 to i_2, the aggregate download window is $(i_1\Delta, (i_2+l-1)\Delta]$, and the bandwidth constraint to download this sequence of segments is:

$$\Delta \sum_{i=i_1}^{i_2} r_i \le \int_{i_1\Delta}^{(i_2+l-1)\Delta} w(\tau)d\tau, \quad \forall 1 \le i_1 \le i_2 \le N. \quad (3)$$

Indeed, it is easy to show that, given the available bandwidth $\{w(t), t \in [\Delta, (N+l-1)\Delta]\}$, any combination of $\left\{l, \{r_i\}_{i=1}^N\right\}$ satisfying (3) defines a feasible video streaming and playback strategy for N segments:

a) segment i is encoded, streamed and played at rate r_i;
b) all segments share the common playback latency of $l\Delta$ and download window size of $(l-1)\Delta$;
c) there is no video freeze as long as $r_i > 0$, $\forall i$;
d) the maximum buffered video time on client is $(l-1)\Delta$.

Remark 1: If $\left\{l, \{r_i\}_{i=1}^N\right\}$ is feasible under (3), then $\left\{l', \{r_i\}_{i=1}^N\right\}$ is feasible for all $l' > l$. In other words, *the feasible video rate region grows as playback latency increases.*

Remark 2: In (3), if we fix $i_1 = 1$, the reduced set of constraints define the feasible rate region for VoD[3] under the initial playback latency of $l\Delta$. In other words, with the same playback latency, *the feasible video rate region of VoD is a superset of that of live streaming.*

Given a target latency of $l\Delta$, any time slot $(i\Delta, (i+1)\Delta]$ falls into the download windows of $(l-1)$ active segments: $i-l+2, \cdots, i$. If we equally allocate the download bandwidth within any time slot to the $(l-1)$ active segments within it, we immediately get one feasible video rate solution for (3):

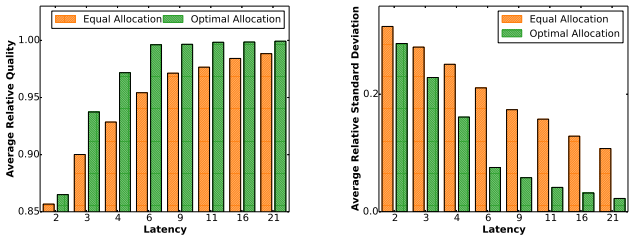
$$r_i^{(l)} = \bar{w}(i,l) \triangleq \frac{1}{(l-1)\Delta} \int_{i\Delta}^{(i+l-1)\Delta} w(\tau)d\tau. \quad (4)$$

In other words, we use the moving average download bandwidth of the next $(l-1)$ time slots as the video rate for the current segment. *The longer the target latency l, the smoother the video rate, the higher the aggregate video quality.*

In practice, a video segment can be encoded at a small set $\mathcal{R}$ of discrete video rates, and users are also sensitive to the perceptual quality variations over time. We normalize the time so that $\Delta = 1$, and assume that the i-th segment is ready to be downloaded at the beginning of the $(i+1)$-th time slot. To characterize the relation between playback latency and playback rate more precisely, we formulate a quality maximization problem to find the optimal rate r_i for segment i and the bandwidth allocated to download segment i in jth time slot, which is denoted as u_{ij}:

$$\begin{aligned} \max_{\{r_i, u_{ij}\}} \quad & \sum_{i=1}^{N} Q(r_i) - \sum_{i=2}^{N} |Q(r_i) - Q(r_{i-1})| \\ \text{subject to} \quad & \sum_{j=i+1}^{\min\{i+l-1, J\}} u_{ij} = r_i, \quad 1 \le i \le N, \\ & \sum_{i=\max\{j-l+1,1\}}^{j-1} u_{ij} \le W_j, \quad 2 \le j \le J, \\ & r_i \in \mathcal{R}, \quad 1 \le i \le N \end{aligned} \quad (5)$$

where $Q(\cdot)$ is the perceptual video quality function $Q(r_i) = \log \frac{r_i}{R_l}$, R_l is the lowest available video rate, W_j is the total bandwidth available in time slot j, $W_j = \int_{j-1}^{j} w(\tau)d\tau$, and J is the last time slot. The first set of constraints dictate that segment i can be downloaded in time slots of $[i+1, i+l-1]$, and the second set of constraints bound the total download speed of all active segments in one time slot with the total available bandwidth. Indeed, the two sets of constraints are the discrete version of (3). The optimal solution $\{r_i^*, u_{ij}^*\}$ is not unique in u_{ij}^*, since they only show up in the linear constraints. There are many feasible live streaming schedules $\{u_{ij}\}$ to realize the optimal video rate vector $\{r_i^*\}$.



(a) Average Relative Quality (b) Average Video Rate RSD

Fig. 3: Video Quality Increases and Video Rate Variation Decreases as Allowable End-to-End Latency Increases

Fig. 3 presents a case study of how the allowable playback latency impacts the achievable video quality. We take eight 100-second 4G Cellular bandwidth traces from [26]. For each trace, we calculate the video quality upper bound as $Q(\bar{w})$. We fix the segment duration at 1 second, and vary the playback latency from 2 to 21 seconds. For each latency, we compute the achieved quality (including the penalty of quality fluctuation) of the moving average video rate defined in (4), and that of the optimal solution of (5) obtained using nonlinear solver MINOS [27]. The relative quality is calculated as the ratio between the achieved quality over the upper bound. Fig. 3a plots the average relative quality for the eight traces. With low latency, the relative video qualities for both solutions are far from 1. The relative quality for the optimal allocation quickly approaches 1 as the latency allowance increases, while the simple moving average solution can only approach the upper bound when the latency is large. Fig. 3b plots the average video rate Relative Standard Deviation (RSD), which

[3]VoD can be treated as a special case of live streaming where all segments are generated at time 0, and playback deadline for the i-th segment is $(i+l-1)\Delta$.

is defined as the ratio of standard deviation to mean (σ/μ). It is clear that the optimal rate allocation can maximally take advantage of the additional latency allowance to quickly reduce the video rate variance among segments, and approach the video quality upper bound.

B. Discrete Time Model for Live Streaming

The optimal live streaming strategy calculated in (5) assumes the complete knowledge of network bandwidth. In practice, such bandwidth oracle is certainly not available. With random future bandwidth, a requested segment may not be delivered before its target playback deadline. Video may freeze, and video playback latency will be increased after each freeze. We develop a discrete-time dynamic model to study the interplay between *video rate selection*, *playback latency*, and *video freeze/skip* in live streaming. Table I summarizes the key variables in our model. A sample buffer evolution trajectory is illustrated with the defined variables in Fig. 4.

TABLE I: Key Variables of Discrete Live Streaming Model

Notation	Meaning
r_i	Video rate of segment i
Δ	Duration of video segment (1s)
Q_i	Video quality of segment i
w_i	Average throughput while downloading segment i
rtt_i	Round trip time (RTT) while downloading segment i
t_i	Time when downloading of segment i completes
y_i	Idle time before downloading segment i
b_i	Buffer length after downloading segment i
l_i	Playback latency for segment i
g_i	Video freeze time while downloading segment i
n_i	Number of segments skipped due to re-sync at i

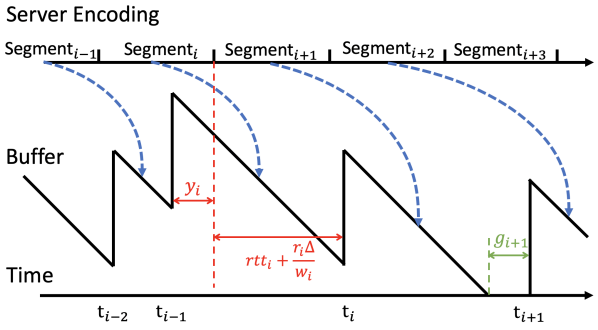


Fig. 4: A Sample Trajectory of Live Streaming Buffer

We assume the client sequentially downloads video segments generated in realtime. After segment $(i-1)$ is completely downloaded at t_{i-1}, the client requests segment i from the server if it has been encoded. The download completion time for i is updated as:

$$t_i = t_{i-1} + y_i + rtt_i + \frac{r_i\Delta}{w_i}, \tag{6}$$

where $y_i = (i\Delta - t_{i-1})^+$ is the potential download idle time if segment $(i-1)$ download completes before segment i is ready for download. For example, shown in Fig. 4, the downloading of segment $(i-1)$ finishes before the completion of the encoding of segment i, then the client has to wait for a time period of y_i until segment i is encoded. During the waiting, buffer is drained at rate 1 due to continuous playback. The buffered video time is updated as:

$$b_i = \left(b_{i-1} - y_i - rtt_i - \frac{r_i\Delta}{w_i}\right)^+ + \Delta, \tag{7}$$

where $y_i + rtt_i + \frac{r_i\Delta}{w_i}$ amount of video time is consumed from the buffer while segment i is being requested/downloaded. If video buffer goes down to zero before segment i is completely received, there will be video freeze, the freeze time is:

$$g_i = \left(y_i + rtt_i + \frac{r_i\Delta}{w_i} - b_{i-1}\right)^+. \tag{8}$$

In Fig. 4, video buffer depletes before segment $(i+1)$ is downloaded, therefore a freeze of time g_{i+1} occurs.

Now we model how segment playback latency evolves over time. The downloaded segments will be sequentially played as long as there is no video freeze. Therefore, we have

$$l_i = l_{i-1} + g_i. \tag{9}$$

The playback latency for the very first segment is a critical parameter that impacts the latencies for the following segments, risk of video freezes, and the feasible video rate region of all segments as discussed in Section III-A.

Assuming a user joins the live streaming service at time t^o, which falls into the encoding time period of segment o, $t^o \in [E_o, E_o + \Delta)$, where E_o is the time when the first frame of segment o is being encoded. The user can only request the previous segments that have already been encoded. Let α be the initial latency so that the user will request segment $i_0 = o - \alpha$ as the first segment to download, and completes the download at time:

$$t_{i_0} = t^o + rtt_{i_0} + \frac{r_{i_0}\Delta}{w_{i_0}}.$$

If the user plays the initial segment immediately after it is downloaded, the video buffer might be too shallow to maintain continuous streaming in future. Instead, most live streaming algorithms start the initial playback only after accumulating β segments in video buffer. The playback starts at time:

$$t_{i_1} = t^o + \sum_{i=i_0}^{i_1}(y_i + rtt_i + \frac{r_i\Delta}{w_i}),$$

where $i_1 = i_0 + \beta - 1$ is the segment triggering the playback. Therefore the startup delay is $t_{i_1} - t^o$, namely the lag between the user joins the live event and the video playback starts, which equals to the downloading time of the first β segments[4]. Since the video in the first segment i_0 was recorded/encoded starting from time $E_{i_0} = E_o - \alpha * \Delta$. The playback latency for the first segment is

$$l_{i_0} = t_{i_1} - E_{i_0} = \sum_{i=i_0}^{i_1}(y_i + rtt_i + \frac{r_i\Delta}{w_i}) + \alpha\Delta + (t^o - E_o),$$

where the first part is the startup delay, the second part is due to requesting a previously encoded segment, the last part is due to

[4]For the rest of the paper, we set β to 2.

the random arrival of client request within a segment encoding period, which we assume follows uniform distribution between $[0, \Delta]$. With this initial playback latency, we can update the playback latency for all the subsequent segments according to (9). Without adapting the playback speed, the playback latency l_i is non-decreasing over time, and each video freeze will increase the playback latencies for all the subsequent segments. To closely track the live event, we set up a **re-sync** mechanism: whenever there is a video freeze, say at t_i when segment i download completes, if $l_i > l^{max}$, the longest tolerable playback latency, we force the streaming session to restart following the (α, β) strategy for the initial playback. The number of segments skipped due to re-sync is:

$$n_i = (o^{(i)} - \alpha - i)^+, \tag{10}$$

where $o^{(i)}$ is the index of the segment being encoded at re-sync time instant of t_i. Before resuming video playback, β segments have to be accumulated.

C. Chunk-based Live Streaming

In the most existing DASH solutions, a server can only stream a segment to a client after the whole segment is completely encoded. This introduces a latency of Δ. To achieve low latency, the recent proposal in CMAF [28] is to break a video segment into multiple chunks, while the rate adaption is still done at the segment level, each chunk can be packaged and transmitted separately. As illustrated in Fig. 5, with the server-wait mechanism [23], after receiving the MPD of segment $(i-1)$, the client can send out request for segment i. The server can push chunk $c_{i,1}$ of segment i to the client whenever it is encoded and packaged. The chunk is ready to be rendered when it is completely received. In this case, the latency can be reduced to $\Delta_c + t_c$, where Δ_c is chunk duration and t_c is chunk transmission time, which can be significantly lower than the segment-based streaming latency of Δ plus segment transmission time. As will be shown in our evaluation, such latency reduction is crucial in low-latency live streaming.

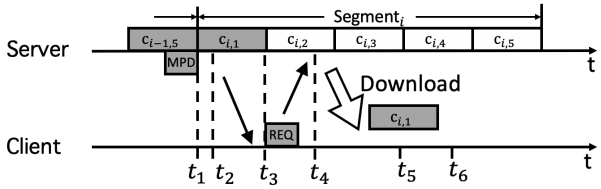


Fig. 5: Latency of Chunk-based Streaming

IV. Iterative Linear Quadratic Regulator based Rate Control

A. Maximizing Quality-of-Experience

Equations (7),(8),(9),(10) define a discrete-time dynamic system for live streaming, with system state before downloading segment i as $\mathcal{S}_i = \langle b_{i-1}, g_{i-1}, l_{i-1}, n_{i-1}, r_{i-1} \rangle$. Given the initial playback strategy of (α, β), the system evolution is determined by the video rate selection r_i for segment i and network condition. The system dynamics are summarized as:

$$\mathcal{S}_{i+1} = \boldsymbol{f}(\mathcal{S}_i, r_i, \{w_i, rtt_i\}). \tag{11}$$

We expand the perceptual quality model in (5) to take into account the negative impact of playback latency, freeze, and skip on user experience. The QoE of streaming segment i can be modeled as a weighted sum function:

$$\begin{aligned} QoE(\mathcal{S}_i, r_i) =& a_1 Q(r_i) - a_2 |Q(r_i) - Q(r_{i-1})| - a_3 g_i \\ & - a_4 \boldsymbol{h}(l_i) - a_5 n_i, \end{aligned} \tag{12}$$

where a_1, a_2, a_3, a_4 and a_5 are weights reflecting a user's relative sensitivity for different QoE components including reward of high video quality, penalty of video rate fluctuation, penalty of video freeze, penalty of playback latency and penalty of segment skip. $\boldsymbol{h}(\cdot)$ models latency penalty. In this paper, we adopt a logistic growth function $\frac{1}{1+e^{\phi - l_i}} - \frac{1}{1+e^{\phi}}$, where ϕ sets the latency sensitive range of users.

If we treat $\{w_i, rtt_i\}$ as exogenous inputs outside of a client's control, the optimal streaming strategy for a client is to select video rate r_i for each segment to maximize the aggregate QoE:

$$\begin{aligned} \max_{\{r_i\}} \quad & \sum_{i=1}^{m} QoE(\mathcal{S}_i, r_i) \\ \textbf{subject to} \quad & \mathcal{S}_{i+1} = \boldsymbol{f}(\mathcal{S}_i, r_i, \{w_i, rtt_i\}). \end{aligned} \tag{13}$$

Given the network condition $\{w_i, rtt_i\}$ with $i \in [1, m]$, the system dynamics are deterministic, and the optimal rate adaptation strategy $\{r_i^*\}$ can be obtained by solving the multi-stage optimal control problem in (13). As video is always encoded into several rates to be downloaded, the rate selection will be discrete and (13) can be considered as a nonlinear mixed integer programming problem. We will discuss several methods to solve this problem.

B. iLQR based Rate Selection

We solve the optimal rate adaption problem (13) under the framework of Linear quadratic regulator (LQR) [29] and iterative LQR (iLQR) [30]. LQR/iLQR solve the cost minimization problem for dynamic system:

$$\begin{aligned} \min_{\{u_i\}} \quad & \sum_{i=1}^{m} c(\mathrm{x}_i, u_i) \\ \textbf{subject to} \quad & \mathrm{x}_{i+1} = \boldsymbol{f}(\mathrm{x}_i, u_i). \end{aligned} \tag{14}$$

For live streaming, the control action at step i is the video rate selection, i.e., $u_i = r_i$, the state x_i can be simply $\mathcal{S}_i$ in (13). However, based on (7),(8),(9),(10), buffer evolution determines the evoluation of the other state variables, such as video freeze, skip and latency changes. It is sufficient to only take $\mathrm{x}_i = [b_{i-1}, u_{i-1}]^T$, where b_i is the buffer length. The cost function to be minimized can be simply the negation of the QoE function, $c(\mathrm{x}_i, u_i) = -QoE(\mathcal{S}_i, r_i)$.

In LQR/iLQR, both $\boldsymbol{f}(\cdot)$ and $c(\cdot)$ have to be differentiable. However, for the live streaming system model described in Section III, the system model and the cost function are not differentiable in some cases. For example, when video freeze

happens, video buffer length reduces to zero and jumps back to Δ when the segment is downloaded. We approximate the system dynamics and cost function using differentiable functions. More details are given in Appendix A.

When the system is linear, i.e.,

$$\mathrm{x}_{i+1} = \mathbf{F}_i \begin{bmatrix} b_{i-1} \\ u_{i-1} \\ u_i \end{bmatrix} + \mathbf{f}_i, \tag{15}$$

in which $\mathbf{F}_i$ is a time dependent matrix and $\mathbf{f}_i$ is constant, and the cost function is quadratic, i.e.,

$$c(\mathrm{x}_i, u_i) = \frac{1}{2} \begin{bmatrix} \mathrm{x}_i \\ u_i \end{bmatrix}^T \mathbf{C}_i \begin{bmatrix} \mathrm{x}_i \\ u_i \end{bmatrix} + \begin{bmatrix} \mathrm{x}_i \\ u_i \end{bmatrix}^T \mathbf{c}_i, \tag{16}$$

the optimal control strategy can be obtained analytically by a two-fold algorithm including backward propagation and forward pass. The backward propagation solves the minimizing action u_i in terms of state x_i. Then, with the given initial state x_1, minimizing action u_i and the following state x_{i+1} can be generated iteratively by the forward pass from step 1 to m. The detailed derivation is described in Appendix B and the algorithm is illustrated as Algorithm 1.

Algorithm 1 Linear Quadratic Regulator (LQR)

Input: x_1: the initial state; m: look-ahead horizon; $\{w_i, rtt_i, i \in [1, m]\}$: future available bandwidth and rtt.
Output: $\{\mathrm{u}_i^*, i \in [1, m]\}$: optimal rate sequence.
Initialization: Set $\mathbf{V}_{m+1}$ and $\mathbf{v}_{m+1}$ to be zero matrix.

Backward Propagation
for each segment $i = m : 1$ **do**
  $\mathbf{Q}_i = \mathbf{C}_i + \mathbf{F}_i^T \mathbf{V}_{i+1} \mathbf{F}_i$
  $\mathbf{q}_i = \mathbf{c}_i + \mathbf{F}_i^T \mathbf{V}_{i+1} \mathbf{f}_i + \mathbf{F}_i^T \mathbf{v}_{i+1}$
  $\mathrm{Q}(\mathrm{x}_i, u_i) = \frac{1}{2} \begin{bmatrix} \mathrm{x}_i \\ u_i \end{bmatrix}^T \mathbf{Q}_i \begin{bmatrix} \mathrm{x}_i \\ u_i \end{bmatrix} + \begin{bmatrix} \mathrm{x}_i \\ u_i \end{bmatrix}^T \mathbf{q}_i + \mathrm{const}$
  $\mathbf{K}_i = -\mathbf{Q}_{u_i,u_i}^{-1} \mathbf{Q}_{u_i,\mathrm{x}_i}$
  $\mathbf{k}_i = -\mathbf{Q}_{u_i,u_i}^{-1} \mathbf{q}_{u_i}$
  $u_i \leftarrow \operatorname{argmin} \mathrm{Q}(\mathrm{x}_i, u_i) = \mathbf{K}_i \mathrm{x}_i + \mathbf{k}_i$
  $\mathbf{V}_i = \mathbf{Q}_{\mathrm{x}_i,\mathrm{x}_i} + \mathbf{Q}_{\mathrm{x}_i,u_i} \mathbf{K}_i + \mathbf{K}_i^T \mathbf{Q}_{u_i,\mathrm{x}_i} + \mathbf{K}_i^T \mathbf{Q}_{u_i,u_i} \mathbf{K}_i$
  $\mathbf{v}_i = \mathbf{q}_{\mathrm{x}_i} + \mathbf{Q}_{\mathrm{x}_i,u_i} \mathbf{k}_i + \mathbf{K}_i^T \mathbf{q}_{u_i} + \mathbf{K}_i^T \mathbf{Q}_{u_i,u_i} \mathbf{k}_i$
  $\mathrm{V}(\mathrm{x}_i) = \frac{1}{2} \mathrm{x}_i^T \mathbf{V}_i \mathrm{x}_i + \mathrm{x}_i^T \mathbf{v}_i$
end for

Forward Pass
for each segment $i = 1 : m$ **do**
  $u_i = \mathbf{K}_i \mathrm{x}_i + \mathbf{k}_i$
  $\mathrm{x}_{i+1} = \boldsymbol{f}(\mathrm{x}_i, u_i)$
end for
return $u_{[1,\cdots,m]}^*$

For our problem, the system is nonlinear around video freeze, and the cost function is not quadratic. We obtain the optimal control strategy following the iterative LQR (iLQR) [30] framework. The main idea is to linearize the system around its current operation point, obtain the optimal improvement within the neighborhood of the current operation point using LQR, move to the new operation point, and repeat the process until the iterative update converges. More specifically, the first order Taylor series expansion of state evolution is:

$$\boldsymbol{f}(\mathrm{x}_i, u_i) \approx \boldsymbol{f}(\hat{\mathrm{x}}_i, \hat{u}_i) + \nabla_{\mathrm{x}_i,u_i} \boldsymbol{f}(\hat{\mathrm{x}}_i, \hat{u}_i) \begin{bmatrix} \mathrm{x}_i - \hat{\mathrm{x}}_i \\ u_i - \hat{u}_i \end{bmatrix}. \tag{17}$$

Let $\delta \mathrm{x}_i = \mathrm{x}_i - \hat{\mathrm{x}}_i$ and $\delta u_i = u_i - \hat{u}_i$ be the state and control perturbation from operation point $(\hat{\mathrm{x}}_i, \hat{u}_i)$ respectively. Then:

$$\begin{aligned} \delta \mathrm{x}_{i+1} = \bar{\boldsymbol{f}}(\delta \mathrm{x}_i, \delta u_i) &= \boldsymbol{f}(\mathrm{x}_i, u_i) - \boldsymbol{f}(\hat{\mathrm{x}}_i, \hat{u}_i) \\ &\approx \nabla_{\mathrm{x}_i,u_i} \boldsymbol{f}(\hat{\mathrm{x}}_i, \hat{u}_i) \begin{bmatrix} \delta \mathrm{x}_i \\ \delta u_i \end{bmatrix}. \end{aligned} \tag{18}$$

Similarly, the second order Taylor series expansion of cost can be written as:

$$\begin{aligned} c(\mathrm{x}_i, u_i) \approx\ & c(\hat{\mathrm{x}}_i, \hat{u}_i) + \nabla_{\mathrm{x}_i,u_i} c(\hat{\mathrm{x}}_i, \hat{u}_i) \begin{bmatrix} \mathrm{x}_i - \hat{\mathrm{x}}_i \\ u_i - \hat{u}_i \end{bmatrix} \\ & + \frac{1}{2} \begin{bmatrix} \mathrm{x}_i - \hat{\mathrm{x}}_i \\ u_i - \hat{u}_i \end{bmatrix}^T \nabla_{\mathrm{x}_i,u_i}^2 c(\hat{\mathrm{x}}_i, \hat{u}_i) \begin{bmatrix} \mathrm{x}_i - \hat{\mathrm{x}}_i \\ u_i - \hat{u}_i \end{bmatrix}, \end{aligned} \tag{19}$$

and the cost function of state and action change can be defined as follows:

$$\begin{aligned} \bar{c}(\delta \mathrm{x}_i, \delta u_i) &= c(\mathrm{x}_i, u_i) - c(\hat{\mathrm{x}}_i, \hat{u}_i) \\ &\approx \nabla_{\mathrm{x}_i,u_i} c(\hat{\mathrm{x}}_i, \hat{u}_i) \begin{bmatrix} \delta \mathrm{x}_i \\ \delta u_i \end{bmatrix} \\ &\quad + \frac{1}{2} \begin{bmatrix} \delta \mathrm{x}_i \\ \delta u_i \end{bmatrix}^T \nabla_{\mathrm{x}_i,u_i}^2 c(\hat{\mathrm{x}}_i, \hat{u}_i) \begin{bmatrix} \delta \mathrm{x}_i \\ \delta u_i \end{bmatrix}. \end{aligned} \tag{20}$$

Then the problem can be remodeled by system dynamics $\bar{\boldsymbol{f}}$, cost $\bar{c}$, state change $\delta \mathrm{x}_i$ and action change δu_i. The local optimal improvement can be obtained by LQR according to Algorithm 1. And the final control sequence can be obtained by applying iterative Algorithm 2 until it converges.

Algorithm 2 Iterative LQR (iLQR)

Input: $\{\hat{\mathrm{x}}_i, i \in [1, m]\}$: the initial state sequence; $\{\hat{u}_i, i \in [1, m]\}$: the initial rate control sequence; m: look-ahead horizon; $\{w_i, rtt_i, i \in [1, m]\}$: future available bandwidth and rtt; η: update step size; K: maximum iteration number.
Output: $\{\mathrm{u}_i^*, i \in [1, m]\}$: optimal rate sequence.
Initialization: $k = 0$
while $\{\mathrm{x}_i^*\}$ is not converged **and** $k \leq K$ **do**
  $k = k + 1$
  $\mathbf{F}_i = \nabla_{\mathrm{x}_i,u_i} \boldsymbol{f}(\hat{\mathrm{x}}_i, \hat{u}_i)\ \forall i$
  $\mathbf{c}_i = \nabla_{\mathrm{x}_i,u_i} c(\hat{\mathrm{x}}_i, \hat{u}_i)\ \forall i$
  $\mathbf{C}_i = \nabla_{\mathrm{x}_i,u_i}^2 c(\hat{\mathrm{x}}_i, \hat{u}_i)\ \forall i$
  Conduct LQR (Algorithm 1) *Backward Propagation* on $\delta \mathrm{x}_i$ and action $\delta u_i\ \forall i$
  Conduct LQR (Algorithm 1) *Forward Pass* with real system dynamics $\mathrm{x}_{i+1} = \boldsymbol{f}(\mathrm{x}_i, u_i)$ and $u_i = \hat{u}_i + \eta(\mathbf{K}_i(\mathrm{x}_i - \hat{\mathrm{x}}_i) + \mathbf{k}_i\ \forall i$
  Update $\hat{\mathrm{x}}_i$ and $\hat{u}_i$ based on the value generated in *Forward Pass* and check if $\{\mathrm{x}_i^*\}$ is converged.
end while
return $u_{[1,\cdots,N]}^*$

V. Practical Live Streaming Algorithms

In practice, the video is only encoded at a set of discrete values, denoted by $\mathcal{R}$. The continuous control solved in Section IV-B should be quantized to $\mathcal{R}$. In addition, the iLQR/LQR obtain optimal solution based on the input of future network condition $\{w_i, rtt_i\}$. However, due to the error of prediction, the deterministic optimal control problems studied in Section IV-B become stochastic optimal control problems. To cope with this, in this section, we develop three practical rate control algorithms: iLQR based Model Predictive Control (**iMPC**), exhaustive MPC (**eMPC**) and Deep Reinforcement Learning (**DRL**) based live streaming algorithms to approach the optimal solutions.

A. iLQR MPC

Algorithm 2 calculates the m-step optimal streaming strategy using network condition oracle for the future m steps. However, in practice, bandwidth oracle is not available and the bandwidth prediction error for large m can be high. To develop complete streaming solution for arbitrary number of stages with predicted network condition, we employ a sliding horizon framework described in Algorithm 3. To initialize the playback, the first β segments will be downloaded according to some pre-defined rate selection strategy. For each subsequent segment k, network condition for the next m segments is predicted (line 3). Then Algorithm 2 is called to obtain the optimal solution for the next m segments by using the predicted network condition (line 4). After quantizing the continuous solution, only the rate selection for segment k is adopted to drive the system to the next stage (line 5-6). The process repeats until all segments are downloaded. The entire streaming algorithm is illustrated as Fig. 6

Algorithm 3 iLQR based MPC Streaming (iMPC)

Input: α and β: startup parameters; m: look-ahead horizon; N: live streaming duration; $\mathcal{R}$: available rates.
Output: $\{u_i, i \in [1, N]\}$: rate sequence for all segments
1: Download the first β segments using predefined rate selection strategy $u_{[1,\cdots,\beta]}$, obtain $x_{\beta+1}$
2: **while** (live streaming session is on) **do**
3: Predict network condition $\{\widehat{w}_{[i,i+m-1]}, \widehat{rtt}_{[i,i+m-1]}\}$
4: Generate initial state sequence $\hat{\mathrm{x}}_{[i,i+m-1]}$ and control sequence $\hat{u}_{[i,i+m-1]}$
5: $\tilde{u}_i^{(m)}$ = **iLQR**$(\hat{\mathrm{x}}, \hat{u}, \{\widehat{w}, \widehat{rtt}\}, \mathcal{R})$
6: $u_i = \text{Quantize}(\tilde{u}_i^{(m)}[1])$
7: $\mathrm{x}_{i+1} = \boldsymbol{f}(\mathrm{x}_i, u_i, \{w_i, rtt_i\})$
8: **end while**

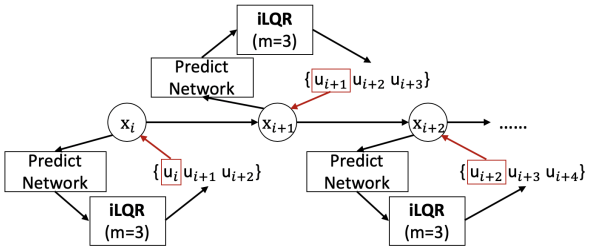


Fig. 6: iLQR based MPC (iMPC)

B. Exhaustive MPC based Live Streaming

In Section IV-B, at each step, the optimal video rate generated by iLQR needs to be quantized to one of the available video rates, and it might lead to suboptimal solution. Inspired by this, we also proposed another MPC live streaming algorithm that conducts exhaustive search for the optimal rate sequence in discrete video rate space directly. Illustrated as Fig. 7, from an initial state x_1, we get the next stage states of all the available actions. Then we operate state evaluation on them. A candidate state might be eliminated if the evaluation shows it is not likely to reach optimal from it. For example, if video freeze happens for a long period of time at state x_{22}, it can be eliminated and will not be considered in future. In addition, the accumulated rewards are also compared among the states at the same level. If with similar system status, the states with lower accumulated awards can also be eliminated. When we reach the final stage, we compare the accumulated cost and find the one with least cost. All the intermediate states and actions belonging to the path from initial state x_1 to it (x_{mk} in Fig. 7) are considered to be the optimal solution. The entire process is similar to branch-and-bound and it is named exhaustive MPC. If we replace iLQR (line 4) in Algorithm 3 by exhaustive MPC, we get a new streaming algorithm named eMPC. All the system evolution and cost (reward) used by eMPC are the original definitions described in Section III.

Note that even though iLQR generates the optimal solution for a given network condition, but the quantization might introduce additional loss. eMPC directly searches in the available rate space $\mathcal{R}$. With perfect knowledge of future network condition, the performance of eMPC is the upperbound of iMPC for discrete rate adaption. On the other hand, when the number of video rates is large, and/or the optimization horizon is long, eMPC quickly runs into the state explosion problem, the computation time will be too long for realtime operation. To the contrast, the performance loss of iMPC due to rate quantization decreases as the number of video rates increases, and LQR execution time only increases linearly with the horizon. This makes iLQR attractive when eMPC is too expensive for realtime operation. The detailed comparison is shown in Fig. 8. Both iMPC and eMPC are implemented and running in the same environment. With the lookahead horizon increasing, the runtime of iMPC increases slowly. However, for eMPC with 6 bitrates, the runtime grows exponentially and 510 ms is consumed when the lookahead equals 5. This limits the deployment of eMPC for realtime applications.

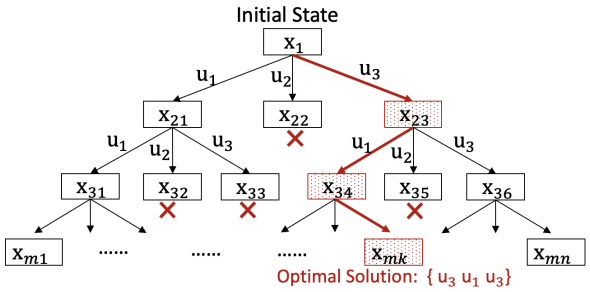


Fig. 7: Exhaustive MPC (eMPC)

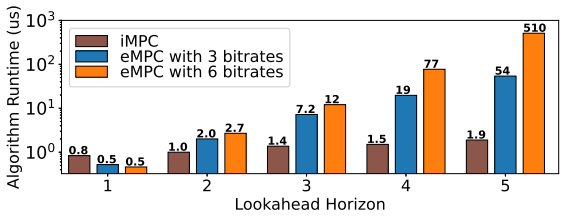
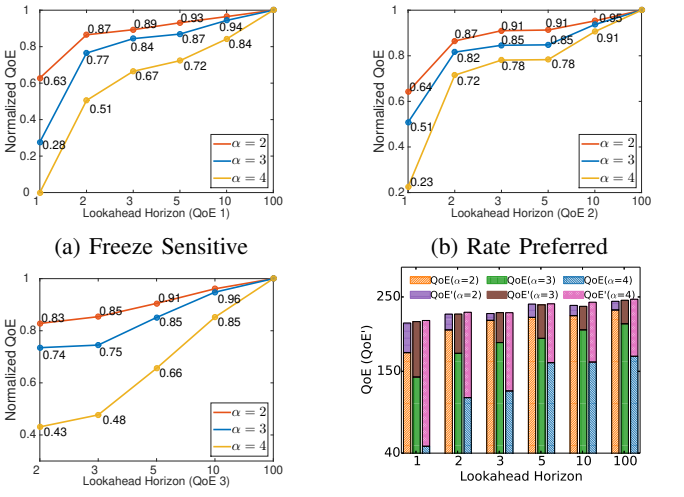


Fig. 8: Runtime comparison between eMPC and iMPC.

We conducted numerical case studies of eMPC streaming using eight different bandwidth traces from [26]. Each curve



(a) Freeze Sensitive (b) Rate Preferred

(c) Rate Fluctuation Sensitive (d) Absolute QoE

Fig. 9: eMPC Performance at Different Horizons

in Fig. 9a to 9c is the normalized QoE at different horizon window sizes for one QoE weight setting in (12) and one initial playback setting (α value). The weights for *Freeze Sensitive*, *Rate Preferred* and *Rate Fluctuation Sensitive* users are set as $\{1, 1, 6, 4, 6\}$, $\{2, 1, 6, 4, 6\}$ and $\{1, 1.5, 6, 4, 6\}$ respectively. The longer one can look into the future, the closer MPC can get to the optimal, and 10 steps horizon is long enough for close-to-optimal performance. When the initial latency is short, shorter look ahead horizon is needed to achieve the same normalized QoE. For example, in Fig. 9a, with initial latency of $\alpha = 2$, a horizon of two steps can already achieve 87% normalized QoE, while it takes a horizon of five steps for $\alpha = 3$ and more than ten steps for $\alpha = 4$ to achieve similar normalized QoE. This is good news for MPC type of live streaming: *if the target latency is short, only network conditions in a short horizon need to be estimated to achieve close-to-optimal performance.*

Finally, Fig. 9d compares the absolute QoE with and without latency penalty (noted as QoE') at different horizons for one network trace with freeze sensitive QoE weights setting. As expected, QoE and QoE' increase with horizon, and at each horizon, the longer the initial latency, the lower the overall QoE (due to large penalty weight for latency), but the higher the video quality (QoE').

TABLE II: eMPC Robustness to Estimation Errors

α	Estimation Rescale Factor (μ)					
	-50%	-30%	-10%	10%	30%	50%
2	0.73	0.90	0.98	0.99	0.83	0.49
3	0.79	0.95	0.98	0.97	0.76	0.42
4	0.76	0.93	0.97	0.83	0.41	0.17

Network QoS estimation errors are inevitable. To evaluate how MPC type of streaming algorithms' performance is affected by bandwidth prediction errors, we conduct a case study with harmonic mean based bandwidth prediction [18]. We take eight traces from [26]. For each trace, we first obtain harmonic mean bandwidth estimation $\widehat{w_k}$ and use it to drive eMPC algorithm to get the QoE results. Then we introduce additional errors to $\widehat{w_k}$ by rescaling it by a factor of $1 + e_k$, where e_k is a Gaussian random variable with mean μ varying from -0.5 to 0.5 and fixed σ of 0.01. We then rerun eMPC using garbled bandwidth estimate of $(1+e_k)\widehat{w_k}$ and calculate its relative QoE normalized by the QoE achieved by eMPC without additional error. In Table II, with $\pm 10\%$ scaling, the relative QoE is still close to 1. But as the scaling magnitude increases, the relative QoE decreases dramatically. Comparing performance with error -50% and 50%, we find that conservative prediction is more friendly to eMPC than aggressive prediction, because freezing is less acceptable than low video rate. Realtime network QoS prediction is a challenging problem. One can employ different approaches, e.g. [18], [26], [31]–[33], to achieve the desirable accuracy and complexity trade-off.

C. Deep Reinforcement Learning based Solution

Another approach to dealing with uncertain network conditions is to use the model-free optimal control framework known as Reinforcement Learning (RL) [34]. In principle, given a probabilistic model for network conditions, together with the live streaming models developed in Section III-B, we can cast the optimal live streaming problem as a Markov Decision Process (MDP): by expanding the system state $\mathcal{S}_i$ in (11) to Φ_i so that the streaming system can be modeled as a Markov process with state transition probability of $P(\Phi_{i+1}|\Phi_i, r_i)$. Due to the uncertainty of system state, the action at stage k is no longer a rate selection r_k, it is rather a policy $\pi(\cdot)$ that maps any possible state to a video rate selection, i.e., $r_k = \pi(\Phi_k)$. The goal of stochastic optimal control is to find the optimal streaming policy π^* to maximize the expected QoE:

$$\pi^* = \underset{\{\pi\}}{\textbf{argmax}}\ E_{\{\Phi_k\}}\left\{\sum_{k=1}^{m} \text{QoE}(\Phi_i, r_i)\right\}$$
$$\textbf{subject to}\ \Phi_1 = \phi_1, r_k = \pi(\Phi_k), \Phi_{k+1} \sim P(\Phi_{k+1}|\Phi_k, r_k),$$

where ϕ_1 is the given initial state. However, given the large state space, it is not practical to obtain accurate state transition model and solve the live streaming MDP exactly.

Deep Reinforcement Learning (DRL) [35], [36] is a powerful model-free optimal control framework that can work with large state space. As shown in Fig. 10, an agent interacts with the environment repeatedly through real or simulated experiences. At each step, the agent takes an action according to its policy and obtains certain immediate reward, depending on the state of the environment. The agent's goal is to find the optimal policy that maximizes the long-term reward. The agent dynamically improves its action policy based on its value function estimates. DRL trains a Deep Neural Networks (DNN), called the value network, to approximate the long-term reward, i.e. the value function $V(\Phi)$. In the actor-critic architecture [37], another DNN, called the actor network, is also trained to generate action policy $\pi(\Phi)$.

We develop a actor-critic-based live streaming algorithm. The state variables include the download history of the previous ten segments, including segment size, download duration,

and buffer length [5]. We also include video rate of the previous segments, freeze time and idle time (the time between the download completion of the current segment and the download start of the next segment) for the previous five segments, and the indicators of startup and re-sync phases. All inputs of DNN are summarized in Table III.

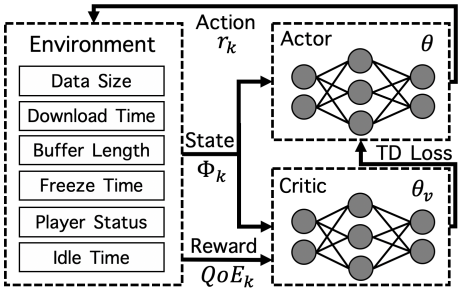


Fig. 10: Actor-Critic Architecture

TABLE III: Deep-RL State Definition

Inputs	Size	Input Layer
Data Size	10 (15)	LSTM
Download Duration	10 (15)	LSTM
Buffer Length	10 (15)	LSTM
Previous Video Rate	1	Fully Connected
Re-sync Indicator	1	Fully Connected
Player Status Indicator	1	Fully Connected
Idle Time	5	Fully Connected
Freeze Time	5	Fully Connected

Policy gradient method is used for training. Actor and critic networks are updated after taking several actions or the environment is terminated. The update is defined by the gradient which can be calculated by:

$$\nabla_\theta \log \pi(r_k|\Phi_k;\theta)A(\Phi_k, r_k;\theta,\theta_v) \tag{21}$$

where $A(\Phi_k, r_k;\theta,\theta_v)$ is the estimated advantage function defined by $\text{QoE}(r_k) + \gamma V(\Phi_{k+1};\theta_v) - V(\Phi_k;\theta_v)$. Therefore, the actor network parameter θ can be updated as:

$$\theta \leftarrow \theta + \alpha_\theta \sum_k \nabla_\theta \log \pi(r_k|\Phi_k;\theta)A(\Phi_k, r_k;\theta,\theta_v) \tag{22}$$

where α_θ is the actor network learning rate. In this way, the parameters θ is updated toward the direction to increase the advantage. The critic network is trained by reducing the MSE of Temporal Difference (TD Loss) which is defined by $\text{QoE}(r_k) + \gamma V(\Phi_{k+1};\theta_v) - V(\Phi_k;\theta_v)$. Entropy term is also introduced for actor network training to guarantee the action space is explored efficiently before getting converged.

[5]In the chunk-mode discussed next, each 1s segment is divided into 5 200ms chunks, we use the download history of the previous 15 chunks (3 seconds) instead.

VI. Performance Evaluation

A. Evaluated Algorithms

- **Naive** is a rate-based streaming algorithm. At each step, the estimated bandwidth $\widehat{w}$ for downloading the next video segment is set to the harmonic mean of download bandwidth of the past five segments. Video rate is chosen as $\eta\widehat{w}$, where $\eta = 80\%$.
- **PI-controller** is a buffer-based algorithm [14] that regulates the aggressiveness γ_p of rate selection, based on the difference between the actual buffer length and a reference q_{ref}. It also calculates a predicted bandwidth $\widehat{w}$, and the highest rate lower than the regulated bandwidth $\gamma_p\widehat{w}$ is chosen. q_{ref} is set to be the initial latency. [6]
- **eMPC-Live (eMPC$^{(s)}$)** is the segment-based implementation of the eMPC algorithm proposed in Section V-B. In experiments, each segment is 1s, the horizon is five segments. We use harmonic mean of past five bandwidth $\{w_k, k \in [i-5, i-1]\}$ as the prediction for $\widehat{w_i}$.
- **eMPC-Chunk (eMPC$^{(c)}$)** is a chunk-based implementation of eMPC-Live, with chunk size of 200 ms.
- **iMPC-Live (iMPC$^{(s)}$)** is the segment-based implementation of iMPC proposed in Section IV-B.
- **iMPC-Chunk (iMPC$^{(c)}$)** is a chunk-based implementation of iMPC.
- **DRL-Live (DRL$^{(s)}$)** is the segment-based implementation of DRL-based algorithm mentioned in Section V-C. We use general critic-actor framework to train separate DNN models for each specific initial latency α. The actor and critic networks take the states described in Section V-C as input and the output of the actor network is the probability distribution of actions (video rates).
- **DRL-Chunk (DRL$^{(c)}$)** is the chunk-based implementation of DRL-Live, with chunk size of 200 ms.
- **Optimal-Live** calculates the optimal rate sequence using the algorithm shown in Fig. 7 assuming the full knowledge of future bandwidth, and employs chunk-based streaming. It serves as a benchmark for all the algorithms.

B. Sample Optimal Trajectories Generated by Offline eMPC

With exhaustive search by **eMPC**, the optimal rate selection can be obtained offline if complete bandwidth trace is given. The available video rates are {0.3, 0.5, 1, 2, 3, 6} Mbps. The detailed optimal video rate and buffer length with different initial latencies are shown in Fig. 11. Given future bandwidth, the rate selection can be *aggressive and smooth*. For example, with 4s initial latency (shown by the green curve), 6 Mbps video rate is selected even if the available bandwidth (shown by the black curve in Fig. 11a) is only around 4 Mbps at 50 s. Correspondingly, the video buffer length drains fast. At time of 53 s, when buffer is close to 0, a video rate lower than the available bandwidth is selected to accumulate the video buffer. In addition, during the entire simulation, video rate rarely changes to avoid penalty of rate fluctuation. At the end of simulation, high video rate is selected and buffer becomes

[6]Video is streamed in chunk-mode for both Naive and PI-controller

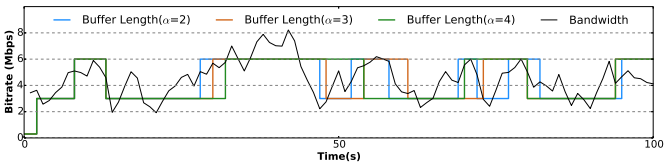
TABLE IV: Detailed QoE Metrics Comparison for 4G Cellular Trace with $\alpha = 2$.

Algorithms	Network 1 ($\mu = 6.5$, $\sigma = 1.25$)					Network 2 ($\mu = 1.77$, $\sigma = 0.51$)					Network 3 ($\mu = 2.35$, $\sigma = 0.72$)				
	QoE	Rate	Freeze	Change	Latency	QoE	Rate	Freeze	Change	Latency	QoE	Rate	Freeze	Change	Latency
Naive	209.6	3.0	0.28	0.09	2.62	90.5	1.0	0.51	0.04	2.85	110.7	1.3	0.51	0.13	2.85
PI	235.1	5.1	**0.17**	0.26	**2.51**	94.0	1.2	**0.40**	0.14	**2.74**	120.7	1.6	**0.40**	0.17	**2.74**
eMPC$^{(s)}$	246.4	5.0	0.24	0.42	2.52	108.6	1.3	0.44	0.17	**2.74**	117.6	2.0	1.91	0.27	3.67
eMPC$^{(c)}$	266.6	5.6	**0.17**	0.18	**2.51**	122.8	1.5	0.71	0.19	2.84	155.2	**2.1**	**0.40**	0.22	**2.74**
***i*MPC**$^{(s)}$	226.2	4.1	0.24	0.39	2.52	91.8	0.9	**0.40**	**0.01**	**2.74**	98.1	1.0	**0.40**	**0.01**	**2.74**
***i*MPC**$^{(c)}$	266.0	5.5	**0.17**	0.12	**2.51**	114.4	1.2	**0.40**	0.05	**2.74**	147.4	1.9	0.56	0.17	2.84
DRL$^{(s)}$	250.0	5.4	0.50	0.60	2.76	114.7	1.5	0.75	0.23	2.97	135.9	1.9	1.35	0.12	3.41
DRL$^{(c)}$	**276.2**	**5.8**	0.20	**0.06**	2.54	**131.4**	**1.6**	0.51	0.19	2.85	**160.1**	2.0	0.51	0.12	2.85
Optimal	**279.0**	**5.89**	**0.16**	**0.06**	**2.49**	**137.6**	**1.65**	**0.39**	**0.14**	**2.72**	**170.1**	**2.2**	**0.38**	**0.14**	**2.72**

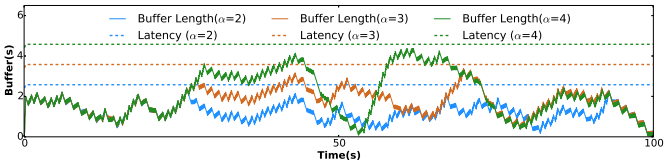
TABLE V: Detailed QoE Metrics Comparison for 4G Cellular Trace with $\alpha = 3$.

Algorithms	Network 1 ($\mu = 6.5$, $\sigma = 1.25$)					Network 2 ($\mu = 1.77$, $\sigma = 0.51$)					Network 3 ($\mu = 2.35$, $\sigma = 0.72$)				
	QoE	Rate	Freeze	Change	Latency	QoE	Rate	Freeze	Change	Latency	QoE	Rate	Freeze	Change	Latency
Naive	188.8	3.0	0.28	0.09	3.62	65.3	1.0	0.51	0.04	3.85	86.2	1.3	0.51	0.13	3.85
PI	214.0	5.0	**0.17**	0.26	**3.51**	72.9	1.2	**0.40**	0.12	**3.74**	87.1	1.6	**0.40**	0.23	**3.74**
eMPC$^{(s)}$	245.8	5.5	**0.17**	0.18	**3.51**	91.2	1.5	1.05	0.19	3.9	97.6	2.1	1.36	0.27	4.3
eMPC$^{(c)}$	**252.4**	**5.7**	**0.17**	0.12	**3.51**	99.4	**1.6**	0.60	0.26	3.81	122.0	**2.2**	0.87	0.20	4.05
***i*MPC**$^{(s)}$	236.3	5.1	**0.17**	0.15	**3.51**	75.0	1.0	**0.40**	**0.01**	**3.74**	99.4	1.4	**0.40**	**0.06**	**3.74**
***i*MPC**$^{(c)}$	251.4	**5.7**	**0.17**	**0.06**	**3.51**	**101.7**	1.4	**0.40**	0.10	**3.74**	132.4	2.1	0.63	0.18	3.85
DRL$^{(s)}$	249.8	**5.7**	0.62	0.12	3.59	91.3	1.4	0.45	0.25	3.78	123.2	2.1	0.44	0.36	3.78
DRL$^{(c)}$	221.7	4.5	**0.17**	0.33	**3.51**	92.1	**1.6**	**0.40**	0.45	**3.74**	**133.9**	**2.2**	**0.4**	0.31	**3.74**
Optimal	**260.5**	**5.89**	**0.16**	**0.06**	**3.49**	**115.0**	**1.65**	**0.39**	**0.14**	**3.72**	**147.4**	**2.2**	**0.38**	**0.14**	**3.72**

empty when the live streaming is terminated. This indicates that the resources can be fully utilized. The cases with 2s and 3s initial latency also show the similar performance.



(a) Optimal Bitrates Selection



(b) Buffer Length Evolution

Fig. 11: Optimal Bitrate Selection and Buffer Length Evolution with Different Initial Latencies (2-4s)

C. Trace-driven Simulations

We developed a detailed live streaming simulator and implemented all the above algorithms in Python. We used real 4G cellular bandwidth dataset collected in NYC Metro Area [26] to drive the simulations. Based on the raw bandwidth traces, we synthesized additional traces using Hidden Markov Models to generate diverse network scenarios for training and testing of the algorithms. As a result, we used 150 and 120 traces for training and testing the DRL models respectively. For all the experiments, the QoE weights in (12) are set as $\{1, 1, 6, 4, 6\}$. ϕ in latency penalty function is set to 6. RTT is uniformly distributed between 30 and 40 ms.

Table IV compares the performance of all the algorithms driven by three traces of 100 seconds with initial latency $\alpha = 2$. In general, eMPC, iMPC and DRL algorithms can achieve higher video rate than Naive and PI-controller. The highest video rate in each setting (marked in bold in the table) is close to the video rate of the optimal solution, and the lowest freeze and latency among them are also close to the optimal performance. But they do suffer more penalties from video bitrate change. This is expected because they don't have the future bandwidth oracle to optimally plan smooth video rate adaption. All the segment-based algorithms (MPC$^{(s)}$, iMPC$^{(s)}$ and DRL$^{(s)}$) get lower overall QoE due to the latency introduced by segment encoding. They suffer more video freezes than chunk-based algorithms. DRL$^{(s)}$ generates 1.35s video freeze under the network environment 3 with high relative standard deviation. DRL and eMPC based algorithms adapt video rate more aggressively than iMPC, resulting in higher average bitrates and larger rate changes. The results also show iMPC-based algorithms can achieve similar QoE to eMPC-based algorithm through choosing lower video rate and suffering less penalty of video rate fluctuation. As network becomes more fluctuating (higher relative standard deviation) from Network 1 to 3, the relative QoE gap between baselines and the proposed algorithms increases which demonstrates the proposed algorithms are more robust and adaptive to dynamic network.

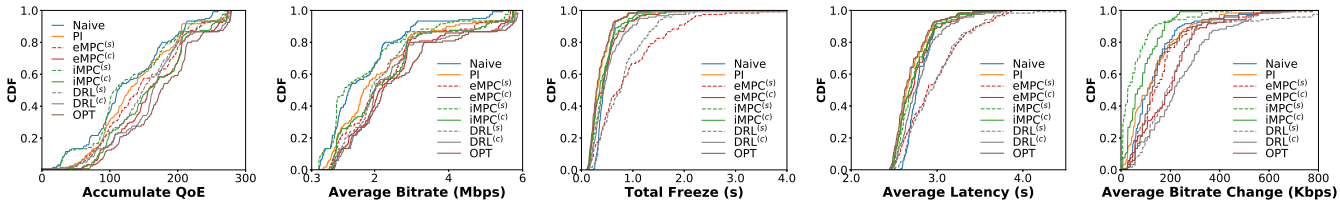


Fig. 12: CDF of QoE Metrics over 120 4G Traces with $\alpha = 2$

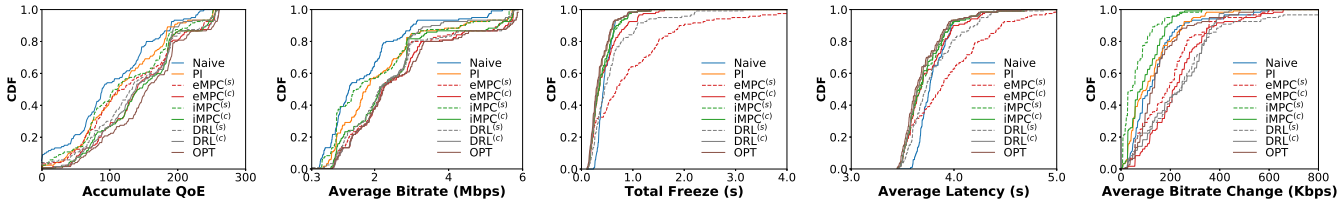
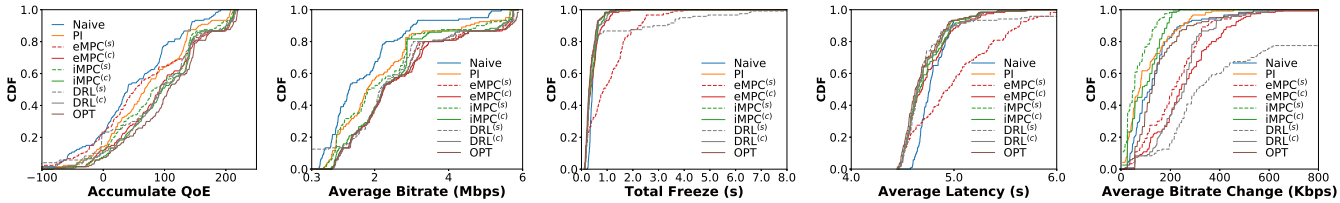


Fig. 13: CDF of QoE Metrics over 120 4G Traces with $\alpha = 3$



(a) Overall QoE (b) Average Bitrate (c) Average Freeze (d) Average Latency (e) Average Bitrate Change

Fig. 14: CDF of QoE Metrics over 120 4G Traces with $\alpha = 4$

Table V presents results with initial latency $\alpha = 3$ under the same three trances. The behaviors of different algorithms are similar to Table IV. With higher initial latency, overall, the average QoE for all algorithms decreases when compare with Table IV. eMPC, iMPC and DRL algorithms can still achieve close-to-optimal rate and latency. The performances of segment-based algorithms also improve because the buffer upper bound (latency) becomes longer. In addition, based on our observation, the QoE metric segment skip n_i which is not shown in the Table IVand V rarely happens.

We also test the algorithms over 120 testing bandwidth traces. The CDF of QoE metrics over all the test cases are shown in Fig. 12 to 14. On the whole, offline optimal algorithm can achieve the highest QoE for all the cases through suffering lowest video freeze while choosing highest rate. Among the other algorithms, chunk-based algorithms DRL$^{(c)}$, iMPC$^{(c)}$ and eMPC$^{(c)}$ achieve the highest QoE in most cases. eMPC$^{(c)}$ and DRL$^{(c)}$ are more aggressive than iMPC$^{(c)}$ and thereby introduces more video rate change. Overall, chunk based DRL$^{(c)}$ and eMPC$^{(c)}$ can achieve the highest performance but with different strategies. For iMPC$^{(c)}$, moderate video rate is selected with few video freeze and short latency. On the contrary, DRL$^{(c)}$ and eMPC$^{(c)}$ perform aggressively but higher video freeze and latency. In addition, iMPC which is solved with approximate system model and cost function can still achieve competitive performance with eMPC and DRL based algorithms. Segment-based eMPC, iMPC and DRL suffer higher penalty from freeze and rate change than their chunk-based counter-parts. This further proves chunk-based algorithms are more suitable for low-latency live streaming.

D. Run Time Analysis of System Implementation

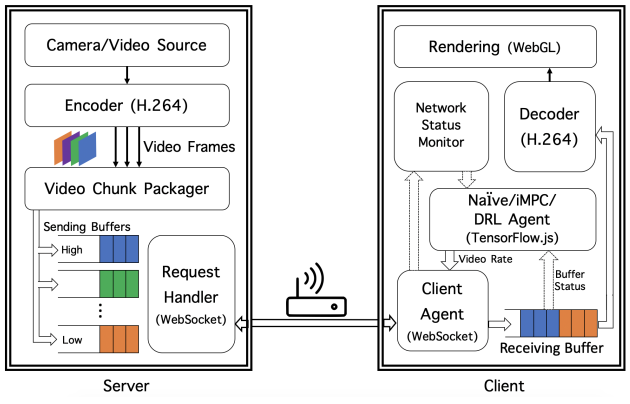


Fig. 15: Live Streaming System Architecture

We also implemented a chunk-based live streaming prototype system to evaluate the run time performance of algorithms. The system architecture is shown in Fig. 15. On the server side, video data from camera or a video file are fed into H.264 encoder and encoded into multiple rates in real-time. Websocket-based request handler is responsible for delivering the requested video chunks to the client. Received video frames are decoded by the web-based H.264 decoder

and rendered using WebGL to the users. The proposed live streaming algorithms are implemented on the client side. We use TensorFlow.js to support DRL-based algorithms. At each time of requesting, the rate adaption algorithms collect information from network monitor, buffer and player status to select video rate.

For the experiments, we use Alienware-15-R2 with Intel i7-6700HQ CPU@2.6 GHz×8 and 16 GB memory as the video server for encoding and packaging. We use Firefox 28.1 browser running on iPhone XR as the client to watch the video. The server and the client is connected by a home router. To emulate the mobile network environment, we use tc to introduce extra delay and throttle the downlink capacity. During the experiments, the measured rtt is with mean and standard derivation of 8.99 and 1.76 ms, and the measured download bandwidth is with average and standard derivation of 2.24 and 0.64 Mbps.

TABLE VI: Detailed Metrics and Run Time Comparison on Live Streaming System (RT: run time in microsecond).

Algorithms	QoE	Rate	Change	Freeze	Latency*	RT
Naive	91.5	**1.9**	60	5.2	6.6	**212**
iMPC$^{(c)}$	130.1	1.3	**50**	**0**	**3.7**	3,907
DRL$^{(c)}$	**134.1**	1.8	220	1.9	4.0	55,685

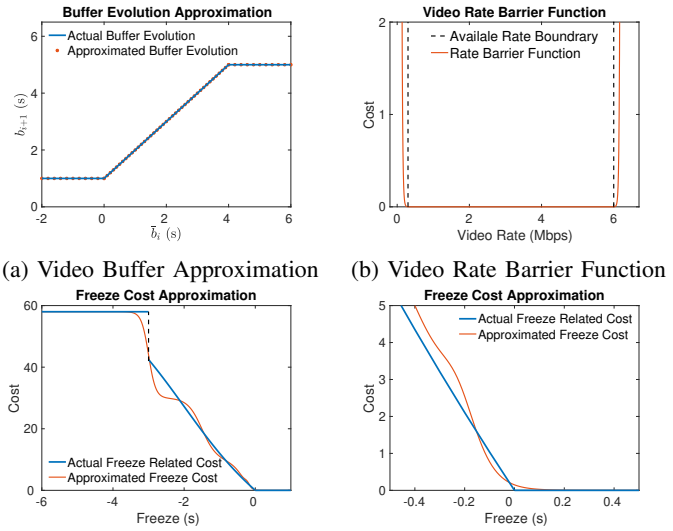
* Initial latency might not be consistent among different algorithms.

We compare performance of three chunk-based algorithms: Naive, iMPC(c) and DRL(c). eMPC(c) which is served as the upper bound of iMPC(c) is not implemented. Similar to the simulations, each segment is one second, and each chunk is 200 ms. Each algorithm ran for 100 seconds with initial latency randomly chosen between 3 and 4 seconds. The run time shows Naive can finish rate selection for each segment (once every second) within 210 us. iMPC$^{(c)}$ and DRL(c) consume 3.9 ms and 55.7 ms to complete rate selection. In addition, to load the DRL model, an average of 2.08s is consumed during the streaming initialization for 5 experiments. iMPC algorithm is integrated into the client side code, there is no extra loading time for it. This demonstrates that our proposed algorithms can be run in realtime with low computation overhead. DRL model has slightly better performance than iMPC, at the price of higher computation overhead and longer initial load latency.

VII. Conclusion

In this paper, we explored the design space of low-latency live video streaming by developing dynamic models and optimal control strategies. Our models capture the interplay between various important QoE metrics, including video quality, playback latency, video freeze and skip. We further developed live streaming algorithms under the framework of MPC and DRL. Through extensive experiments, we demonstrated that our proposed algorithms can achieve close-to-optimal performance in the latency range from two to five seconds. We also demonstrated that chunk-based packaging/streaming and playback pace adaption are two promising mechanisms to achieve a high level of user QoE in low-latency live streaming.

Appendix A
Differentiable Approximation Model



(a) Video Buffer Approximation

(b) Video Rate Barrier Function

(c) Video Freeze Cost Approximation

(d) Video Freeze Cost Approximation (around zero)

Fig. 16: System Dynamic and Cost Function Approximation.

We approximate the bounded piece-wise linear buffer evolution using an differentiable approximation:

$$b_i = z_i^1 \left(z_i^2 b^u + (1 - z_i^2)\bar{b}_i \right) + (1 - z_i^1)\Delta, \tag{23}$$

where z_i^1 and z_i^2 are the differentiable step functions to define the lower bound and upper bound of buffer length, respectively, and $\bar{b}_i = b_{i-1} - rtt_i - \frac{u_i \Delta}{w_i}$ which represents the difference between the buffer length before downloading and the time consumed by downloading. Note that the system state x_i is described discretely at the time of choosing video rate, thereby the buffer length b_i is limited to the range between the lower bound of Δ and upper bound of b^u. It is worth noting that the buffer upper bound b^u is not a parameter of buffer control on client device, but a value depending on the real time latency. As mentioned in (7), if a segment download completes before the next segment is ready for download, the client has to keep idle for time of y_i. And after idle, client can starts to request next video segment, and the buffer length at this time is noted by b^u which can be calculated based on latency l_i. Therefore z_i^1 and z_i^2 are defined as:

$$z_i^1 = \frac{e^{\gamma \bar{b}_i}}{1 + e^{\gamma \bar{b}_i}}, \quad z_i^2 = \frac{e^{\gamma(\bar{b}_i + \Delta - b^u)}}{1 + e^{\gamma(\bar{b}_i + \Delta - b^u)}}, \tag{24}$$

where γ is a large value (e.g. 100). Therefore the value of b_i approaches Δ when $\bar{b}_i$ is less than 0, and approaches b^u when the value of $\bar{b}_i$ exceeds $b^u - \Delta$. The actual buffer evolution and approximated function are illustrated in Fig. 16a.

In addition, the reward function in (12) is modified to be a differentiable cost function as below:

$$\begin{aligned} c(\mathrm{x}_i, u_i) = -\Big(& a_1' Q(u_i) - a_2' (Q(u_i) - Q(u_{i-1}))^2 \\ & - a_3' z_i^3 - a_4' z_i^4 - a_5' z_i^5 \Big), \end{aligned} \tag{25}$$

where z_i^3, z_i^4 are two barrier functions to ensure the selected rate is between the lowest video rate R_l and the highest available video rate R_h: $z_i^3 = e^{-50(u_i - R_l)}$ and $z_i^4 = e^{50(u_i - R_h)}$. z_i^5 represents penalty incurred if video freeze occurs while downloading segment i ($\bar{b}_i$ is less than zero). As video freeze will not only introduce one-time latency penalty (the third term in (12)), but also latency penalties for all the following segments. In addition, if video freeze time exceeds some threshold, e.g., 3s, the download of the current video segment will be terminated and several segments will be skipped. Therefore, the actual cost of video freeze can be formulated as the sum of all these related cost as: $-6\bar{b}_i + \frac{40}{1+e^{(2+\bar{b}_i)}} - \frac{40}{1+e^2}$ if $\bar{b}_i$ is greater than -3s. The first term represents the video freeze penalty and the last two terms represent the additional latency penalties for 10 following segments. If $\bar{b}_i$ is less than -3s (skip happens), the cost is constant, e.g. 58 in this case. In order to apply iLQR, we use a differentiable function $-\frac{e^{4\bar{b}_i+4}}{2\left(1+e^{15\bar{b}_i+3.2}\right)^2} + \frac{10}{1+e^{20(\bar{b}_i+0.2)}} + \frac{28}{1+e^{10(\bar{b}_i+3)}} + \frac{20}{1+e^{5(\bar{b}_i+1.5)}}$ to approximate the actual freeze related cost. The comparison between the actual and approximated cost functions are shown as Fig. 16c and 16d.

APPENDIX B
LINEAR QUADRATIC REGULATOR BACKWARD PROPAGATION

For the last action u_m, the state-action value $\mathrm{Q}(\mathrm{x}_m, u_m)$ is defined as:

$$\mathrm{Q}(\mathrm{x}_m, u_m) = \frac{1}{2}\begin{bmatrix}\mathrm{x}_m\\ u_m\end{bmatrix}^T \mathbf{C}_m \begin{bmatrix}\mathrm{x}_m\\ u_m\end{bmatrix} + \begin{bmatrix}\mathrm{x}_m\\ u_m\end{bmatrix}^T \mathbf{c}_m + \text{const} \tag{26}$$

where

$$\mathbf{C}_m = \begin{bmatrix}\mathbf{C}_{\mathrm{x}_m,\mathrm{x}_m} & \mathbf{C}_{\mathrm{x}_m,u_m}\\ \mathbf{C}_{u_m,\mathrm{x}_m} & \mathbf{C}_{u_m,u_m}\end{bmatrix} \text{ and } \mathbf{c}_m = \begin{bmatrix}\mathbf{c}_{\mathrm{x}_m}\\ \mathbf{c}_{u_m}\end{bmatrix}.$$

Through setting the derivative of $\mathrm{Q}(\mathrm{x}_m, u_m)$ to u_m to be zero as

$$\begin{aligned}\nabla_{u_m}\mathrm{Q}(\mathrm{x}_m, u_m) &= \mathbf{C}_{u_m,u_m}\mathrm{x}_m + \mathbf{C}_{u_m,u_m}u_m + \mathbf{c}_{u_m}^T\\ &= 0,\end{aligned} \tag{27}$$

we can get

$$u_m = -\mathbf{C}_{u_m,u_m}^{-1}(\mathbf{C}_{u_m,\mathrm{x}_m}\mathrm{x}_m + \mathbf{c}_{u_m}) = \mathbf{K}_m\mathrm{x}_m + \mathbf{k}_m \tag{28}$$

with

$$\mathbf{K}_m = -\mathbf{C}_{u_m,u_m}^{-1}\mathbf{C}_{u_m,\mathrm{x}_m} \text{ and } \mathbf{k}_m = -\mathbf{C}_{u_m,u_m}^{-1}\mathbf{c}_{u_m},$$

so that u_m minimizes the state-action cost $\mathrm{Q}(\mathrm{x}_m, u_m)$ [7]. As u_m in (28) only depends on x_m, through substituting u_m into $\mathbf{Q}(\mathrm{x}_m, u_m)$, we can obtain the state cost as:

$$\begin{aligned}\mathrm{V}(\mathrm{x}_m) &= \frac{1}{2}\begin{bmatrix}\mathrm{x}_m\\ \mathbf{K}_m\mathrm{x}_m + \mathbf{k}_m\end{bmatrix}^T \mathbf{C}_m \begin{bmatrix}\mathrm{x}_m\\ \mathbf{K}_m\mathrm{x}_m + \mathbf{k}_m\end{bmatrix} +\\ &\quad \begin{bmatrix}\mathrm{x}_m\\ \mathbf{K}_m\mathrm{x}_m + \mathbf{k}_m\end{bmatrix}^T \mathbf{c}_m + \text{const}\\ &= \frac{1}{2}\mathrm{x}_m^T\mathbf{V}_m\mathrm{x}_m + \mathrm{x}_m^T\mathbf{v}_m\end{aligned} \tag{29}$$

[7]Cost of terminate state $\mathrm{V}(\boldsymbol{f}(\mathrm{x}_m, u_m))$ can be minimized together with $\mathrm{Q}(\mathrm{x}_m, u_m)$.

with

$$\mathbf{V}_m = \mathbf{C}_{\mathrm{x}_m,\mathrm{x}_m} + \mathbf{C}_{\mathrm{x}_m,u_m}\mathbf{K}_m + \mathbf{K}_m^T\mathbf{C}_{u_m,\mathrm{x}_m} + \mathbf{K}_m^T\mathbf{C}_{u_m,u_m}\mathbf{K}_m$$

and

$$\mathbf{v}_m = \mathbf{c}_{\mathrm{x}_m} + \mathbf{C}_{\mathrm{x}_m,u_m}\mathbf{k}_m + \mathbf{K}_m^T\mathbf{c}_{u_m} + \mathbf{K}_m^T\mathbf{C}_{u_m,u_m}\mathbf{k}_m.$$

According to (15), $\mathbf{V}(\mathrm{x}_m)$ can be rewrite as:

$$\begin{aligned}\mathrm{V}(\mathrm{x}_m) =& \frac{1}{2}\begin{bmatrix}\mathrm{x}_{m-1}\\ u_{m-1}\end{bmatrix}^T \mathbf{F}_{m-1}^T\mathbf{V}_m\mathbf{F}_{m-1}\begin{bmatrix}\mathrm{x}_{m-1}\\ u_{m-1}\end{bmatrix} +\\ &\begin{bmatrix}\mathrm{x}_{m-1}\\ u_{m-1}\end{bmatrix}^T \mathbf{F}_{m-1}^T\mathbf{V}_m\mathbf{F}_{m-1} +\\ &\begin{bmatrix}\mathrm{x}_{m-1}\\ u_{m-1}\end{bmatrix}^T \mathbf{F}_{m-1}^T\mathbf{v}_m + \text{const}.\end{aligned} \tag{30}$$

Similarly, the cost of action u_{m-1} can be defined as:

$$\begin{aligned}\mathrm{Q}(\mathrm{x}_{m-1}, u_{m-1}) =& \frac{1}{2}\begin{bmatrix}\mathrm{x}_{m-1}\\ u_{m-1}\end{bmatrix}^T \mathbf{C}_{m-1}\begin{bmatrix}\mathrm{x}_{m-1}\\ u_{m-1}\end{bmatrix} +\\ &\begin{bmatrix}\mathrm{x}_{m-1}\\ u_{m-1}\end{bmatrix}^T \mathbf{c}_{m-1} + \mathbf{V}(\mathrm{x}_m) + \text{const}\\ =& \frac{1}{2}\begin{bmatrix}\mathrm{x}_{m-1}\\ u_{m-1}\end{bmatrix}^T \mathbf{Q}_{m-1}\begin{bmatrix}\mathrm{x}_{m-1}\\ u_{m-1}\end{bmatrix} +\\ &\begin{bmatrix}\mathrm{x}_{m-1}\\ u_{m-1}\end{bmatrix}^T \mathbf{q}_{m-1}\end{aligned} \tag{31}$$

with

$$\mathbf{Q}_{m-1} = \mathbf{C}_{m-1} + \mathbf{F}_{m-1}^T\mathbf{V}_m\mathbf{F}_{m-1}$$

and

$$\mathbf{q}_{m-1} = \mathbf{c}_{m-1} + \mathbf{F}_{m-1}^T\mathbf{V}_m\mathbf{f}_{m-1} + \mathbf{F}_{m-1}^T\mathbf{v}_m.$$

Now the state-action cost is written only in terms of x_{m-1} and u_{m-1}. Therefore the minimizing u_{m-1} can be calculated by setting the derivative to be zero as:

$$\begin{aligned}&\nabla_{u_{m-1}}\mathrm{Q}(\mathrm{x}_{m-1}, u_{m-1})\\ &= \mathbf{Q}_{u_{m-1},x_{m-1}}\mathrm{x}_{m-1} + \mathbf{Q}_{u_{m-1},u_{m-1}}u_{m-1} + \mathbf{q}_{u_{m-1}}\\ &= 0.\end{aligned} \tag{32}$$

Then the minimizing action u_{m-1} can be written as:

$$u_{m-1} = \mathbf{K}_{m-1}\mathrm{x}_{m-1} + \mathbf{k}_{m-1} \tag{33}$$

with

$$\mathbf{K}_{m-1} = -\mathbf{Q}_{u_{m-1},u_{m-1}}^{-1}\mathbf{Q}_{u_{m-1},\mathrm{x}_{m-1}}$$

and

$$\mathbf{k}_{m-1} = -\mathbf{Q}_{u_{m-1},u_{m-1}}^{-1}\mathbf{q}_{u_{m-1}}.$$

REFERENCES

[1] V. Cisco, "Cisco visual networking index: Forecast and trends, 2017–2022," *White Paper*, 2018.

[2] I. Sodagar, "The mpeg-dash standard for multimedia streaming over the internet," *IEEE MultiMedia*, vol. 18, no. 4, pp. 62–67, 2011.

[3] R. Pantos and W. May, "Http live streaming," Tech. Rep., 2017.

[4] AWS. (2019) Video latency in live streaming. [Online]. Available: https://aws.amazon.com/media/tech/video-latency-in-live-streaming/

[5] MUX. (2019) The low latency live streaming landscape in 2019. [Online]. Available: https://mux.com/blog/the-low-latency-live-streaming-landscape-in-2019/

[6] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "Rtp: A transport protocol for real-time applications," Tech. Rep., 2003.

[7] H. Schulzrinne, A. Rao, and R. Lanphier, "Real time streaming protocol (rtsp)," Tech. Rep., 1998.

[8] C. Huitema, "Real time control protocol (rtcp) attribute in session description protocol (sdp)," Tech. Rep., 2003.

[9] Adobe. (2012) Real-time messaging protocol (rtmp) specification. [Online]. Available: https://www.adobe.com/devnet/rtmp.html

[10] A. Zambelli, "Iis smooth streaming technical overview," *Microsoft Corporation*, vol. 3, p. 40, 2009.

[11] Adobe. (2013) Http dynamic streaming. [Online]. Available: https://www.adobe.com/products/hds-dynamic-streaming.html

[12] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia, "A survey on quality of experience of http adaptive streaming," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 469–492, 2014.

[13] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran, "Probe and adapt: Rate adaptation for http video streaming at scale," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 4, pp. 719–733, 2014.

[14] G. Tian and Y. Liu, "Towards agile and smooth video adaptation in dynamic http streaming," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM, 2012, pp. 109–120.

[15] 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 Transactions on Mobile Computing*, 2019.

[16] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 187–198, 2015.

[17] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "Bola: Near-optimal bitrate adaptation for online videos," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[18] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over http," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 325–338.

[19] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 197–210.

[20] S. Sengupta, N. Ganguly, S. Chakraborty, and P. De, "Hotdash: Hotspot aware adaptive video streaming using deep reinforcement learning," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE, 2018, pp. 165–175.

[21] L. Xie, C. Zhou, X. Zhang, and Z. Guo, "Dynamic threshold based rate adaptation for http live streaming," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[22] J. Wang, S. Meng, J. Sun, and Z. Quo, "A general pid-based rate adaptation approach for tcp-based live streaming over mobile networks," in *2016 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2016, pp. 1–6.

[23] V. Swaminathan and S. Wei, "Low latency live video streaming using http chunked encoding," in *2011 IEEE 13th International Workshop on Multimedia Signal Processing*. IEEE, 2011, pp. 1–6.

[24] N. Bouzakaria, C. Concolato, and J. Le Feuvre, "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, 2014, pp. 92–97.

[25] L. Sun, T. Zong, Y. Liu, Y. Wang, and H. Zhu, "Optimal strategies for live video streaming in the low-latency regime," in *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. IEEE, 2019, pp. 1–4.

[26] L. Mei, R. Hu, H. Cao, Y. Liu, Z. Han, F. Li, and J. Li, "Realtime mobile bandwidth prediction using lstm neural network," in *International Conference on Passive and Active Network Measurement*. Springer, 2019, pp. 34–47.

[27] AMPL. Minos. [Online]. Available: https://ampl.com/products/solvers/solvers-we-sell/minos/

[28] (2013) Common media application format (cmaf) for segmented media. [Online]. Available: https://www.iso.org/standard/71975.html

[29] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[30] E. Todorov and W. Li, "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, 2005, pp. 300–306.

[31] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli, "Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 2016, pp. 272–285.

[32] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, no. 3, pp. 199–222, 2004.

[33] E. Kurdoglu, Y. Liu, Y. Wang, Y. Shi, C. Gu, and J. Lyu, "Real-time bandwidth prediction and rate adaptation for video calls over cellular networks," in *Proceedings of the 7th International Conference on Multimedia Systems*. ACM, 2016, p. 12.

[34] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.

[35] V. Konda, "Actor-critic algorithms," Ph.D. dissertation, Cambridge, MA, USA, 2002, aAI0804543.

[36] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning*, 2016, pp. 1928–1937.

[37] ——, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.