Optimal Strategies for Live Video Streaming in the Low-latency Regime

***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 control 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 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 and playback pace 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 the close-to-optimal performance within the latency range of two to five seconds.**

***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 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 whole live video reproduction process. On the encoding/packaging side, segment-based encoding/packaging can be pipelined to shorten the time 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 margin for error: if the selected bitrates for segments are higher than the available bandwidth within their short download windows, 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.

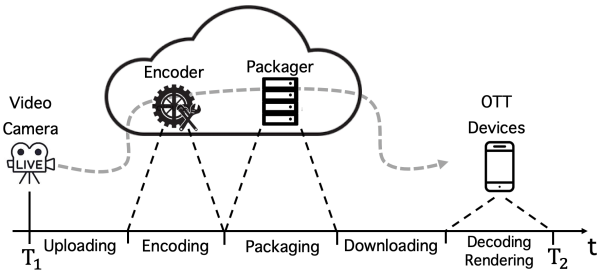


Fig. 1: End-to-End Video Latency in Live Streaming

In this paper, we explore the design space of low-latency live video streaming by developing dynamic models and optimal control strategies to establish QoE upper bounds as a function of the allowable end-to-end latency. We further develop practical streaming algorithms within the 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 and playback pace 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 duration) is divided into multiple shorter video chunks ($\sim$ 200 ms duration), and the processes of encoding, streaming and decoding is operated in a pipelined fashion. MPC and DRL algorithms are developed for online bitrate 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*. We also investigate playback pace adaption as an additional mechanism for QoE maximization in the low-latency regime. Our main contributions are as follows:

1) We characterize the feasible video rate region and the 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, and playback pace adjustment.

2) Based on the dynamic models, we design MPC type of streaming algorithms to find the optimal solutions of both video rate selection and playback pace adaption with network bandwidth estimation in finite horizon. We also develop DRL-based live streaming algorithm that adapts video rate to maximize the long-term QoE, in anticipation of the QoE impairments resulted from the unpredictable variability in future network conditions.
3) We analytically study the benefit of chunk-based video packaging and streaming and demonstrate that it is superior than the segment-based design in achieving low latency. We further demonstrate that playback pace adaption serves as a promising new control dimension to trade-off different QoE components in the tight design space of low-latency live streaming.
4) We conduct extensive performance evaluation of the proposed models and streaming algorithms through numerical case studies and live streaming simulations driven by real network traces. We demonstrated that the proposed MPC and DRL base live streaming algorithms can achieve close-to-optimal performance within the latency range of two to five seconds.

II. Background and Related Work

A. Segment-based DASH Live Streaming

In DASH, video is divided into multiple temporal segments, each of which is encoded and delivered independently. As illustrated in Fig. 2, at time t_1, the first frame of segment $(i-1)$ is received by the cloud server for encoding. The last frame of this segment arrives at the server at t_2, and the encoding finish time for the whole segment is t_3. Assuming video uploading and encoding can be done in realtime, the gap $(t_3 - t_1)$ is roughly the segment duration Δ plus the encoding time of the last frame. Immediately after t_3, a Media Presentation Description (MPD) file which includes the detailed descriptions about the encoded segment is generated and delivered to the users. The client selects a bitrate for the segment and sends out HTTP request for it at t_4. From t_5, the server streams the requested video segment to the client. When the segment is completely received by the end user, it can be added to video buffer for decoding and rendering at a future time t_7. In this case, the end-to-end video latency is larger than the summation of the segment duration Δ, the RTT between server and client (i.e., $t_5 - t_3$), and the segment transmission time. The bandwidth between t_2 and t_5 in Fig. 2 is only used to transmit the MPD file or segment request. To improve bandwidth utilization, after receiving the MPD file of the $(i-1)$-th segment, the client can treat it as the MPD for segment i and request a version for segment i. After the server finishes the encoding of segment i, it can stream the requested version to the client. This is called *server-wait* streaming [6]. This way the end-to-end video latency can be shortened to Δ plus segment transmission time.

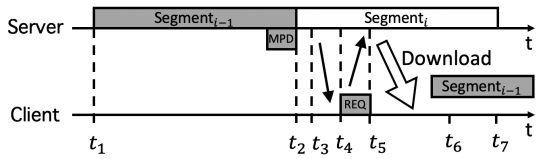


Fig. 2: Timing of Segment-based DASH Streaming

B. Related Work

The traditional realtime multimedia transmission protocols include RTP [7], RTSP [8] and RTCP [9]. Real-Time Messaging Protocol (RTMP) [10] 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) [11], Adobe's HTTP Dynamic Streaming (HDS) [12], 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 [13]. Rate-based algorithm PANDA [14] was proposed to select video rate through bandwidth prediction. Buffer-based algorithms adapt the video rate based on video buffer evolution, e.g., PID [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]. 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 [6]. However, in [23], the authors discussed the overhead of chunked-transfer encoding. With the help of HTTP/2, push-based approach was proposed to cope with high round-trip time over LTE networks [24]. In [25], it was demonstrated that low-latency live streaming system can benefit from multipath transmission. Different from the related studies, we thoroughly explore the design space of low-latency live streaming and develop chunk-based streaming strategies to optimally trade-off various QoE metrics of live streaming.

III. Optimal Live Streaming with Future Network Condition Oracle

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.,

$$\int_0^T r(\tau)d\tau \le \int_0^T w(\tau)d\tau. \quad (1)$$

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:

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

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 to watch the video, i.e., the playback latency is T.[1] 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 that need to be played up to t given the initial playback latency of l_0.

Both downloading and VoD streaming enjoy the freedom of downloading any part of the video using the 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 available 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.

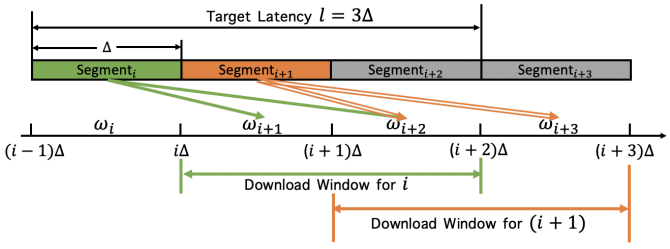


Fig. 3: Segment Download Windows with Latency of 3Δ

Assuming each video segment contains Δ seconds of video, and the i-th video segment is available for being download 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, $(l-1)\Delta$ 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:

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

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

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

The last property is because each segment is only ready for download Δ time after its first frame occurs, and have to be played $l\Delta$ time after the first frame, so the maximum sojourn time of a video segment in client streaming buffer is $(l-1)\Delta$.
Remark 1: If $\left\{l, \{r_k\}_{k=1}^N\right\}$ is feasible under (3), then $\left\{l', \{r_k\}_{k=1}^N\right\}$ 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[2] 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 $(k\Delta, (k+1)\Delta]$ falls into the download windows of $(l-1)$ active segments: $k-l+2, \cdots, k$. 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):

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

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 k-th segment is ready to be downloaded at the beginning of the $(k+1)$-th time slot. To more precisely characterize the relation between playback latency and playback rate, we formulate a quality maximization problem to find the optimal rate r_i for segment

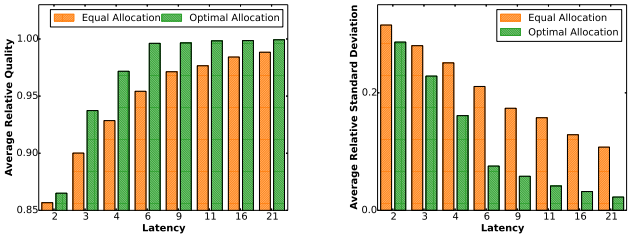
[1] In video download, the video length T_v can be different from the download session length, then the highest video rate achievable is $\frac{\bar{w}T}{T_v}$.

[2] 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_2-th segment is $(i_2 + l - 1)\Delta$.

i and the bandwidth allocated to download segment i in jth time slot, which is noted as u_{ij}:

$$\textbf{OPT-ALL:} \quad \max_{\{r_i,u_{ij}\}} \quad \sum_{i=1}^{N} Q(r_i) - \sum_{i=2}^{N} |Q(r_i) - Q(r_{i-1})|$$
$$\textbf{subject to} \quad \sum_{j=i+1}^{\min\{i+l-1,J\}} u_{ij} = r_i, 1 \le i \le N,$$
$$\sum_{i=\max\{j-l+1,1\}}^{j-1} u_{ij} \le W_j, 2 \le j \le J,$$
$$r_i \in \mathcal{R}, \quad 1 \le i \le N \tag{5}$$

where J is the last time slot, and W_j is the total bandwidth available in time slot j, $W_j = \int_{j-1}^{j} w(\tau)d\tau$. 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 bounds 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 $\{r_i^*\}$. To minimize the segment latency, one can give strict priority to earlier segments in accessing the download bandwidth in each time slot, i.e., downloading all segments sequentially.



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

Fig. 4: Video Quality Increases and Video Rate Variation Decreases as Allowable End-to-End Latency Increases

Fig. 4 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. 4a 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. 4b plots the average video rate Relative Standard Deviation (RSD), which 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. We assume the client sequentially download 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:

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
z_i	Idle time before downloading segment i
b_i	Buffer length after downloading segment i
l_i	Playback latency for segment i
x_i	Video freeze time while downloading segment i
n_i	Number of segments skipped due to re-sync at i

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

where $z_i = (i\Delta - t_{i-1})^+$ is the potential download idle time if segment $(i-1)$ download completes before segment i is ready for download. The buffered video time is updated as:

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

where $z_i + rtt_i + \frac{r_i\Delta}{w_i}$ 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:

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

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

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

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 previous segments that have already been encoded. Suppose the user requests segment $i_0 = o - \alpha$ as the initial segment, 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 chunk 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:

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

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[3]. 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

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

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 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 pace, 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.

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

C. Streaming with Bandwidth Oracle in Finite Horizon

Equations (7),(8),(9),(10) define a discrete-time dynamic system for live streaming, with system state before downloading segment i as $\mathcal{S}_i = \langle b_{i-1}, x_{i-1}, l_{i-1}, n_{i-1}, r_{i-1}\rangle$. 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:

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

Given the dynamic live streaming model, one approach is to estimate the future bandwidth and RTT in some finite horizon $[t, t + m]$, calculate the optimal streaming strategy for time t, then repeat this for the next time slot, following the *Model Predictive Control (MPC)* framework in stochastic optimization [28]. Now, we formally study how MPC type of live streaming can approach the optimal solution even with perfect bandwidth prediction in the finite horizon of m. The key question we want to investigate is that *how far into the future one should estimate network condition to achieve the optimal user QoE at different playback latencies?*

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 function:

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

where a_1, a_2, a_3, a_4 and a_5 are weights reflecting a user's relative sensitivity for different QoE components, $\boldsymbol{g}(\cdot)$ is arbitrary latency penalty function. In this paper, we adopt a logistic growth function $\frac{1}{1+e^{\phi - l_i}} - \frac{1}{1+e^{\phi}}$ to flexibly set the latency sensitive range of users by adjusting ϕ.

If we treat $\{w_i, rtt_i\}$ as exogenous inputs outside of 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:

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

With the oracle of network condition $\{w_i, rtt_i\}$, the system dynamics are deterministic, and the optimal control strategy $\{r_i^*\}$ can be obtained using dynamic programming for any finite time horizon $i \in [1, m]$, as illustrated in Algorithm 1. From any state at stage i, there are in principle $|\mathcal{R}|$ possible actions, so the maximum number of possible states at stage m is $|\mathcal{R}|^m$. The state explosion is problematic for large m. Fortunately, based on the accumulated QoE metric up to stage i, we can already eliminate some states that have no chance to be part of the final optimal solution, and therefore terminate further expansion from those states.[4] For example, if a high video rate r_i selection at state $\mathcal{S}_i$ triggers severe video freeze and skip, and incurs large penalty at the new resulting state $\mathcal{S}_{i+1}$, there is no need to consider further state expansion from

[4]Similar to the Branch-and-Bound approach for Integer Programming.

$\mathcal{S}_{i+1}$. After we obtain all the candidate states at all stages, we can find the optimal transition between those stages using dynamic programming with the following Bellman equation:

$$V^{(i)}(\mathcal{S}_i) = \max_{r_i \in \mathcal{R}} \left\{ QoE(\mathcal{S}_i, r_i) + V^{(i+1)}(\boldsymbol{f}(\mathcal{S}_i, r_i, \{w_i, rtt_i\})) \right\}$$

where $V^{(i)}(\mathcal{S}_i)$ is the optimal solution of the tail problem $\max \sum_{k=i}^{m} QoE(\mathcal{S}_k, r_k)$, i.e., the cumulative QoE from stage i to stage m if one starts with $\mathcal{S}_i$ and takes the optimal control at each stage from i to m. It is the counterpart of the *Value Function* in Reinforcement Learning. $V^{(1)}(\mathcal{S}_1)$ is simply the optimal aggregate QoE starting from the initial state of $\mathcal{S}_1$.

Algorithm 1 Optimal Streaming for Horizon-m

Input: $\mathcal{S}_1$: the initial state; m: look-ahead horizon; $\{w_i, rtt_i, i \in [1, m]\}$: future available bandwidth and rtt; $\mathcal{R}$: available rates;
Output: $\{r_i^*, i \in [1, m]\}$: optimal rate sequence.
Initialization: The possible states at stage 1: $\Omega_1 = \{\mathcal{S}_1\}$.

1: *Branch-and-Bound State Expansion*
2: **for** each segment $i \in [1, m]$ **do**
3: $\quad \Omega_i = \emptyset$
4: $\quad$ **for** each state $\mathcal{S}$ in Ω_{i-1} **do**
5: $\quad\quad$ **for** each $R_j \in \mathcal{R}$ **do**
6: $\quad\quad\quad \mathcal{S}_i' = \boldsymbol{f}(\mathcal{S}, R_j, \{w_i, rtt_i\})$
7: $\quad\quad\quad$ **if** $\mathcal{S}_i'$ could be part of the overall optimal solution **then**
8: $\quad\quad\quad\quad \Omega_i \leftarrow \Omega_i \bigcup \mathcal{S}_i'$
9: $\quad\quad\quad$ **end if**
10: $\quad\quad$ **end for**
11: $\quad$ **end for**
12: **end for**
13: Find Optimal Transition $\mathcal{S}_1 \xrightarrow{r_1^*} \mathcal{S}_2^* \in \Omega_2 \cdots \xrightarrow{r_{m+1}^*} \mathcal{S}_{m+1}^* \in \Omega_{m+1}$ to maximize accumulated QoE $\sum_{i=1}^{m} QoE(\mathcal{S}_i, r_i)$ through DP.
14: **return** $r_{[1,\cdots,m]}^*$

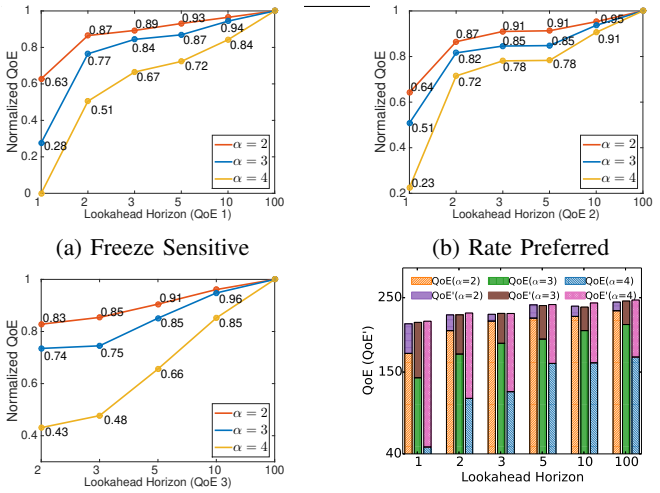
While Algorithm 1 calculates the m-step optimal streaming strategy using network condition oracle for the future m steps, due to fast state expansion, its complexity increases quickly with m. In practice, small value of m is preferred. To develop complete streaming solution for arbitrary number of stages, we employ a sliding horizon framework described in Algorithm 2. To initialize the playback, the first β segments will be downloaded according to some pre-defined rate selection strategy. For each of the following segment i, Algorithm 1 is called to obtain the optimal solution for the next m segments by using network oracle for the next m steps (line 3), but only the solution for segment i is adopted to drive the system to the next stage (line 4-5). The process repeats until all segments are downloaded.

We conducted numerical case studies of MPC streaming using eight different bandwidth traces from [26]. Each curve in Fig. 5a to 5c 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 $\{1, 4, 1, 6, 4\}$, $\{2, 4, 1, 6, 4\}$ and $\{1, 4, 1.5, 6, 4\}$ 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. 5a, with initial

Algorithm 2 Sliding Horizon-m Streaming

Input: $\mathcal{S}_1$: initial state; α and β: startup parameters; m: look-ahead horizon; N: live streaming duration; $\{w_i, rtt_i, i \in [1, N]\}$: available bandwidth and rtt; $\mathcal{R}$: available rates.
Output: $\{r_i, i \in [1, N]\}$: rate sequence for all segments

1: Download the first β segments using predefined rate selection strategy $r_{[1,\cdots,\beta]}$, obtain $\mathcal{S}_{\beta+1}$
2: **for** each segment $i \in [\beta + 1, N]$ **do**
3: $\quad rr_i^{(m)}$ =Horizon-m($\mathcal{S}_i, m, \{w_{[i,i+m-1]}, rtt_{[i,i+m-1]}\}, \mathcal{R}$)
4: $\quad r_i = rr_i^{(m)}[1]$
5: $\quad \mathcal{S}_{i+1} = \boldsymbol{f}(\mathcal{S}_i, r_i, \{w_i, rtt_i\})$
6: **end for**
7: **return** $r_{[1,\cdots,N]}$



(a) Freeze Sensitive (b) Rate Preferred

(c) Rate Fluctuation Sensitive (d) Absolute QoE

Fig. 5: Optimal MPC Performance at Different Horizons

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. 5d compare 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').

IV. Practical Live Streaming Algorithms

In practice, network condition oracles are not available for any future horizon. Essentially, the deterministic optimal control problems studied in Section III-C become stochastic optimal control problems, with exogenous random parameters $\{\mathbf{w_i}, \mathbf{rtt_i}\}$. There are two main directions for stochastic optimal control: *Model Predictive Control (MPC)* and model-free *Reinforcement Learning (RL) (or Approximate Dynamic Programming (ADP), Neuro Dynamic Programming (NDP) in control jargons)*. In this section, we develop practical MPC and

RL-based live streaming algorithms to approach the optimal solutions without network condition oracles.

A. MPC Live Streaming

One straightforward direction is to use the estimated bandwidth and RTT in the near future to drive the finite horizon optimal control problem in Section III-C. With a time horizon of m, at stage k, the rate of segment k is calculated as:

$$\textbf{MPC-Live:} \quad r_k^{mpc} = \underset{\{r_k,\cdots,r_{k+m-1}\}}{\textbf{argmax}} \sum_{i=k}^{k+m-1} QoE(\mathcal{S}_i, r_i)$$

$$\textbf{subject to } \mathcal{S}_{i+1} = \boldsymbol{f}(\mathcal{S}_i, r_i, \{\widehat{w_i}, \widehat{rtt_i}\}), i \in [k, k+m-1],$$

where $\boldsymbol{f}(\cdot)$ is again defined by equations (7),(8),(9),(10), and $\widehat{w_i}$ and $\widehat{rtt_i}$ are the estimated future bandwidth and RTT. The MPC solution can be obtained using the DP Algorithm 1, with w_i and rtt_i replaced by $\widehat{w_i}$ and $\widehat{rtt_i}$. Realtime network QoS prediction is a challenging problem. We use simple moving average for RTT estimation and leverage on previous bandwidth estimation models, including Harmonic Mean [18], Hidden Markov Model (HMM) [29], Support Vector Regress (SVR) model [30] , Recursive Least Squared (RLS) [31], and Long Short Term Memory (LSTM) recurrent neural networks [26], to generate bandwidth estimation.

TABLE II: MPC-Live 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-Live's performance is affected by bandwidth prediction errors, we conduct a case study with Harmony Mean based bandwidth prediction [18]. We take eight traces from [26]. For each trace, we first obtain Harmony Mean bandwidth estimation $\widehat{w_k}$ and use it to drive MPC-Live 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 MPC using garbled bandwidth estimate of $(1+e_k)\widehat{w_k}$ and calculate its relative QoE normalized by the QoE achieved by MPC 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 MPC-Live than aggressive prediction, because freezing is less acceptable than low video rate.

B. 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) [32]. 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:

$$\textbf{MDP-Live:} \quad \pi^* = \underset{\{\pi\}}{\textbf{argmax}} \quad E_{\{\Phi_k\}}\left\{\sum_{k=1}^{m} QoE(\Phi_i, r_i)\right\}$$

$$\textbf{subject to } \Phi_1 = \phi_1, r_k = \pi(\Phi_k), \Phi_{k+1} \sim P(\Phi_{k+1}|\Phi_k, r_k),$$

where ϕ_1 is the given initial state. Given the transition probability, the optimal streaming policy can be obtained by solving the MDP. The state transition matrix $P(\Phi_{i+1}|\Phi_i, r_i)$ encodes both the system model $\boldsymbol{f}(\cdot)$ in (11) and the probabilistic nature of network conditions $\{\mathbf{w_i}, \mathbf{rtt_i}\}$. 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) [33], [34] is a powerful model-free optimal control framework that can work with large state space. As shown in Fig. 6, 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 critic-actor architecture, another DNN, called the actor network, is also trained to generate action policy $\pi(\Phi)$.

We develop a DRL-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 bitrate 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. The shape of all inputs of DNN are summarized in Table III.

C. Chunk-based Live Streaming

As discussed in Section II-A, segment-based DASH 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 [35] is to break a video segment into multiple chunks, while the rate adaption is still done at the segment level, each chunk can be

[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.

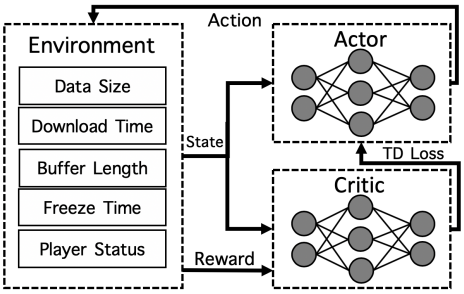


Fig. 6: Critic-Actor Architecture

TABLE III: Deep-RL State Definition

Inputs	Size	Input Layer
Data Size	10 (15)	1-D Convolution
Download Duration	10 (15)	1-D Convolution
Buffer Length	10 (15)	1-D Convolution
Previous Video Bitrate	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

packaged and transmitted separately. As illustrated in Fig. 7, with the server-wait mechanism, after receiving the MPD of segment $(i-1)$, the client can send out request for video segment i. The server can push chunk $c_{i,1}$ of segment i to the client whenever it has been encoded and packaged. The chunk is ready to be rendered when it is completely received. In this case, the end-to-end latency can be reduced to $\Delta_c + t_c$, where Δ_c is the chunk duration and t_c is the 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 can be crucial in low-latency live streaming.

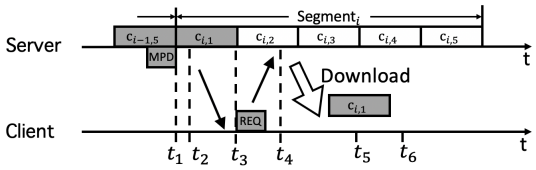


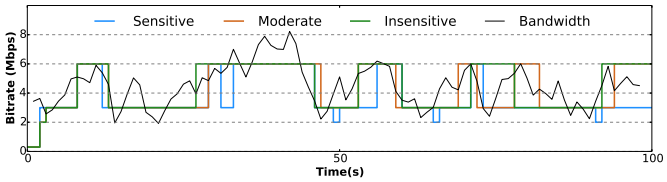
Fig. 7: Latency of Chunk-based Streaming

D. Playback Pace Adaption

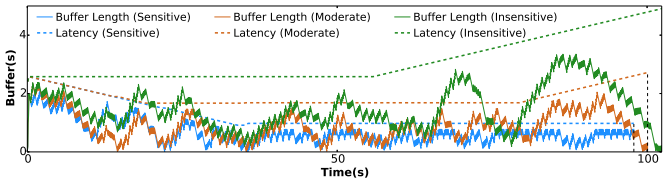
As demonstrated by our previous analysis and case studies, playback latency is a crucial parameter to control the trade-offs between different QoE components of live streaming. In the conventional live streaming, after the initial playback starts, the playback latency will remain constant or go up after each freeze. It can only be brought down through resync. This might be too rigid to adapt to dynamic network conditions. Ideally, one may tolerate longer playback latency when the network condition is unstable, and want low latency when the network condition is steadily good. And it is preferable to make the playback latency transition smooth. One solution is to change the video playback pace. Let v_i be the playback pace when downloading segment i. Based on the study of [36], additional $\pm 5\%$ extra playback pace is not noticeable by the users. We choose v_i from the set of $\{0.95, 1.0, 1.05\}$: 0.95 represents 5% slower playback pace which increases the end-to-end latency; 1.05 means 5% faster playback which reduces the end-to-end latency. Playback pace adaption will change the streaming system dynamics for buffer length, freeze time and end-to-end latency:

$$b_i = \left(b_{i-1} - v_i(z_i + rtt_i + \frac{r_i \Delta}{w_i}) \right)^+ + \Delta,$$
$$x_i = \left(z_i + rtt_i + \frac{r_i \Delta}{w_i} - \frac{b_{i-1}}{v_i} \right)^+,$$
$$l_i = l_{i-1} + (1 - v_i)(z_i + rtt_i + \frac{r_i \Delta}{w_i} - x_i) + x_i.$$

With the updated system dynamics, we can use MPC to find the optimal rate and playback pace of all segments that maximize the aggregated QoE of the whole streaming session. We extended Algorithm 1 to solve for the optimal $\{r_i^*, v_i^*\}$, and conducted case studies using three different latency penalty functions $\boldsymbol{g}(\cdot)$, representing latency sensitive, insensitive, and moderate sensitive user groups respectively. All groups share the same initial playback strategy of $\alpha = \beta = 2$, which means the playback latency of first segment l_{k_o} is around 2Δ. As shown in Fig. 8, starting from the beginning, for the latency sensitive users (the blue lines), the video is played with $1.05\times$ pace for almost 30s to reduce the latency from the value greater than 2s to around 1s. For latency insensitive users (the green lines), slower playback pace is chosen (from 55s) to accumulate buffer length so that highest rate can be requested without running into buffer underflow. For the moderate sensitive group (the orange lines), their behaviors lie in between the sensitive and insensitive: faster playback at the beginning to reduce latency, then normal pace in the middle, and slower playback at the end to choose higher rate.



(a) Bitrate and Bandwidth



(b) Buffer Length & Playback Latency

Fig. 8: Playback Pace Adaption vs. Latency Sensitivity

V. Performance Evaluation

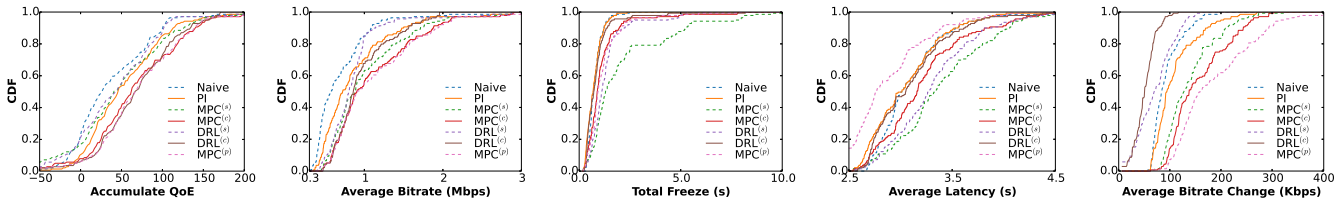
We conducted extensive trace-driven simulations to evaluate various live streaming algorithms derived within different theoretical frameworks in the low-latency regime.

TABLE IV: Detailed QoE Metrics Comparison for HSDPA Cellular Trace

Algorithms	$\alpha=2$					$\alpha=3$					$\alpha=4$				
	QoE	Bitrate	Freeze	Change	Latency	QoE	Bitrate	Freeze	Change	Latency	QoE	Bitrate	Freeze	Change	Latency
Naive	50.95	0.73	**0.76**	0.10	**3.09**	19.96	0.73	**0.76**	0.10	**4.09**	-42.6	0.73	**0.76**	**0.10**	**5.09**
PI-Controller	72.84	0.93	**0.76**	0.11	**3.09**	47.21	0.98	**0.76**	0.10	**4.09**	-18.2	0.96	**0.76**	**0.10**	**5.09**
MPC$^{(s)}$	33.24	1.23	4.84	0.20	3.55	36.69	1.29	2.41	0.21	4.39	-32.0	**1.33**	2.38	0.24	5.34
MPC$^{(c)}$	**99.96**	**1.29**	**0.76**	0.18	**3.09**	65.32	**1.32**	0.97	0.23	4.10	**6.95**	1.32	0.85	0.17	5.10
DRL$^{(s)}$	73.41	0.94	1.14	**0.04**	3.24	39.79	1.14	1.42	**0.07**	4.35	-35.3	1.11	0.84	0.16	5.17
DRL$^{(c)}$	89.04	1.04	**0.76**	0.05	**3.09**	**68.62**	1.26	**0.76**	0.10	**4.09**	-3.67	1.22	**0.76**	0.11	**5.09**
Optimal-Live	**106.8**	**1.32**	**0.76**	**0.01**	**3.09**	**81.46**	**1.33**	**0.76**	**0.01**	**4.09**	**19.77**	**1.32**	**0.76**	**0.01**	**5.09**
MPC$^{(p)}$	101.72	1.28	0.89	0.21	**2.52**	**95.77**	1.30	0.88	0.20	**3.02**	**71.80**	1.31	1.23	0.22	**3.71**

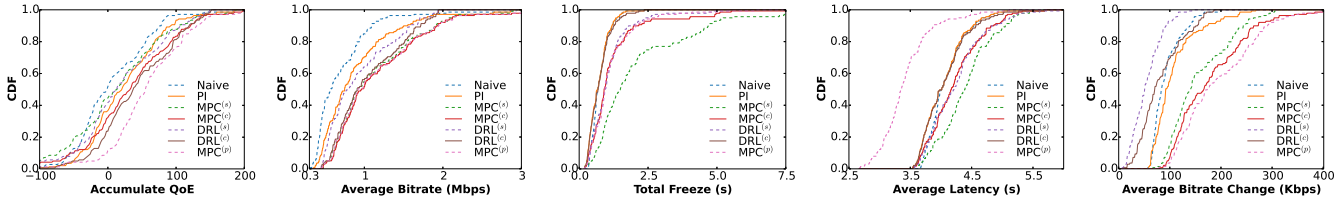
TABLE V: Detailed QoE Metrics Comparison for 4G Cellular Trace

Algorithms	$\alpha=2$					$\alpha=3$					$\alpha=4$				
	QoE	Bitrate	Freeze	Change	Latency	QoE	Bitrate	Freeze	Change	Latency	QoE	Bitrate	Freeze	Change	Latency
Naive	145.2	1.85	0.44	0.28	2.77	122.3	1.85	0.44	0.28	3.77	71.29	1.87	0.44	0.28	4.77
PI-Controller	124.2	2.05	**0.24**	0.39	**2.58**	114.4	2.21	**0.24**	0.38	**3.58**	86.27	2.40	**0.24**	0.33	**4.58**
MPC$^{(s)}$	174.0	2.59	1.14	0.21	3.12	174.4	2.71	**0.24**	**0.21**	**3.58**	62.33	2.83	1.18	0.25	5.37
MPC$^{(c)}$	191.5	2.68	**0.24**	0.23	**2.58**	**176.0**	2.81	**0.24**	0.25	**3.58**	**131.8**	2.86	**0.24**	0.25	**4.58**
DRL$^{(s)}$	174.3	2.20	0.33	**0.14**	2.66	170.9	2.69	0.26	**0.21**	3.60	122.1	2.52	0.26	**0.14**	4.60
DRL$^{(c)}$	**197.7**	**2.75**	**0.24**	0.15	**2.58**	169.3	**2.83**	**0.24**	0.44	**3.58**	130.6	**2.87**	**0.24**	0.32	**4.58**
Optimal-Live	**205.8**	**2.94**	**0.24**	**0.01**	**2.58**	**188.4**	**2.94**	**0.24**	**0.01**	**3.58**	**144.6**	**2.94**	**0.24**	**0.01**	**4.58**
MPC$^{(p)}$	192.9	2.72	0.24	0.27	**2.42**	186.6	2.78	0.56	0.27	**2.76**	**172.8**	2.82	0.40	0.27	**3.33**



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

Fig. 9: CDF of QoE Metrics over 139 HSDPA Traces with $\alpha = 2$

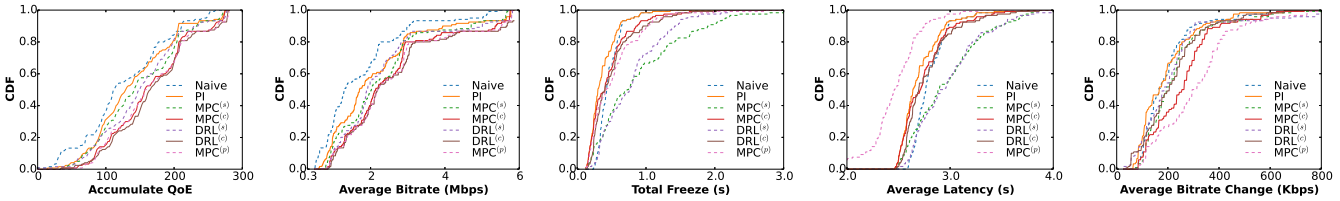


(a) Overall QoE (b) Average Bitrate (c) Average Freeze (d) Average Latency (e) Average Bitrate Change

Fig. 10: CDF of QoE Metrics over 139 HSDPA Traces with $\alpha = 3$

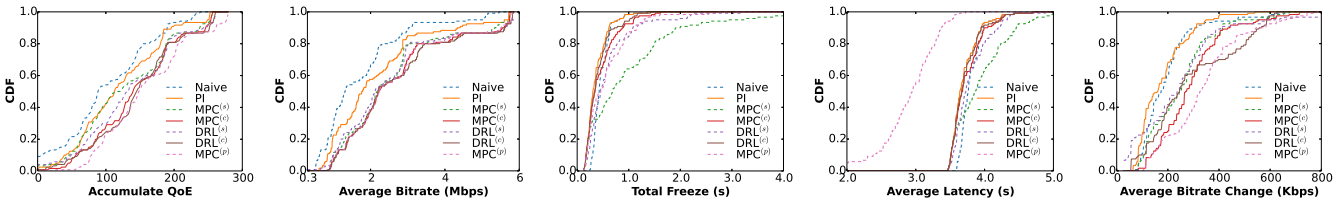
A. *Evaluated Algorithms*

- **Naive** is a simple 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. The highest rate lower than $\gamma\widehat{w}$ is requested, where γ is the utilization parameter. We set γ to 80% in all the experiments. To shorten the latency, video segments are encoded and download in the chunk-mode, each chunk is 200 ms.
- **PI-controller** is a buffer-based algorithm proposed in [15] that regulate the aggressiveness γ_p of rate selection based on the difference between the actual buffer length and a reference buffer length. It also calculates a predicted bandwidth $\widehat{w}$, and the highest rate lower than the regulated bandwidth $\gamma_p\widehat{w}$ is chosen. We set the reference buffer length of PI to be the initial latency target. We further implement it in the chunk-mode (200 ms).
- **MPC-Live (MPC$^{(s)}$)** is the segment-based implementation of MPC algorithm proposed in Section IV-A. In our experiments, each segment is one second, the horizon is five segments. We use harmonic mean of the bandwidth of the past five segments $\{w_k, k \in [i-5, i-1]\}$ as the



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

Fig. 11: CDF of QoE Metrics over 120 4G Traces with $\alpha = 2$



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

Fig. 12: CDF of QoE Metrics over 120 4G Traces with $\alpha = 3$

prediction for $\widehat{w_i}$.

- **MPC-Chunk (MPC$^{(c)}$)** is the chunk-based implementation of MPC-Live, with chunk size of 200 ms.
- **DRL-Live (DRL$^{(s)}$)** is the segment-based implementation of DRL-based live streaming algorithm in Section IV-B. 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 IV-B as input and have the hidden layer with size of 128. The output of the actor network is the probability distribution of actions and each action represents one available video rate.
- **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 Algorithm 1 assuming the full knowledge of future bandwidth, and employs chunk-based streaming. It serves as a benchmark for all the algorithms.
- **MPC-Pace (MPC$^{(p)}$)** is the MPC-based joint rate and pace adaption algorithm proposed in Section IV-D. The MPC horizon is five segments and the future bandwidth estimation is the same as MPC-Live.

B. Trace-driven Simulations

We developed a detailed live streaming simulator and implemented all the above algorithms in Python. We used two real bandwidth datasets to drive the simulations: 3G/HSDPA mobile bandwidth dataset from Norway [37] and a recent 4G cellular dataset collected in NYC Metro Area [26]. 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 139 and 120 individual testing traces for HSDPA and 4G cellular, respectively. In addition, 127 and 150 traces from each of the datasets are used to train the DRL models. For all the experiments, the QoE weights in (12) are set as $\{1, 6, 1, 4, 6\}$. RTT is uniformly distributed between 30 and 40 ms.

To make detailed comparisons, we collected the overall QoE as well as the individual QoE metrics of the algorithms driven by two network traces of 100 seconds. Trace 1 is chosen from HSDPA dataset with the average bandwidth of 1.46 Mbps and variance of 0.34, and trace 2 is from the 4G cellular dataset with average bandwidth of 3.12 Mbps and variance of 0.65. For all the experiments, the QoE weights in (12) are set as $\{1, 6, 1, 4, 6\}$. The RTT for each segment is uniformly distributed between 30 and 40 ms.

Table IV compares the performance of all the algorithms driven by the HSDPA trace 1. In general, MPC and DRL based streaming algorithms can achieve higher video rate than Naive and PI controller, the highest video rate (marked in bold in the table) is close to the video rate of the optimal solution, the lowest latency among them is also close to the optimal performance. But they do suffer more penalties from video freeze and video rate change. This is expected because they don't have the future bandwidth oracle to optimally plan smooth video rate adaption to avoid freeze. Both segment-based algorithms MPC$^{(s)}$ and DRL$^{(s)}$ get lower overall QoE due to the latency introduced by segment encoding. They suffer more video freezes than chunk-based algorithms. For example, MPC$^{(s)}$ generates 4.84s video freeze when $\alpha = 2$, and the freeze time reduces as α increases. For rate and rate change, the results show MPC-based algorithms are more aggressive than DRL-based algorithms. For most of the testing cases, both the average rate and rate change of MPC-based algorithms are higher than those of DRL algorithms.

Interestingly, if playback pace can be adapted by MPC$^{(p)}$, the overall QoE can be improved dramatically over MPC-Live, and can even beat the Optimal upper bound when the initial latency α is large. This is because MPC$^{(p)}$ can use faster playback to gradually reduce the long initial latency while suffering a little bit more video freeze. This demonstrates

that playback pace adaption is important for low-latency live streaming. Table V presents results for the 4G cellular trace. The behaviors of different algorithms are similar to Table IV. MPC and DRL algorithms can still achieve close-to-the-optimal rate and latency, MPC$^{(p)}$ can still beat the optimal upper bound when the initial playback latency is long. The relative performances of segment-based algorithms also improve because the network bandwidth becomes higher and more stable than in the HSDPA case.

We also test the algorithms over 259 bandwidth traces in the two datasets with different properties. The CDF of QoE metrics over all the test cases are shown as Fig. 9 to 12. On the whole, MPC$^{(p)}$ excels all the others algorithms when $\alpha = 3$. For algorithms without pace adaption, chunk-based streaming algorithms DRL$^{(c)}$ and MPC$^{(c)}$ achieve the highest QoE in most cases. Segment-based MPC and DRL algorithms suffer higher penalty from video freeze and rate change than their chunk-based counter-parts. This further proves that chunk-based algorithms are more suitable for supporting low-latency live streaming.

C. System Implementation

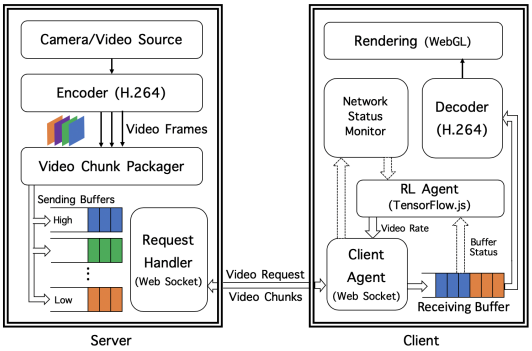


Fig. 13: Live Streaming System Architecture

We also implemented the chunk-based streaming system running over real networks. The system architecture is shown as Fig. 13. On the server side, video data from camera or a video file are fed into H.264 encoder and encoded into multiple rates in realtime. Then the accumulated frames will be packaged together into video chunks, and stored in the sending buffer temporarily. Websocket-based handler is responsible to deliver the requested video chunk to the client. The received video chunk is appended into the receiving buffer. Video frames are decoded by the web based H.264 decoder and rendered using WebGL to the users. The DRL model (in TensorFlow.js) running in RL agent is trained offline based on network bandwidth datasets. At each time of requesting, RL agent collects state information from network monitor, buffer status and the player to choose a proper video rate.

VI. 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 the tight low-latency design space. As future work, we will study low-latency streaming of 360 degree video from edge servers in 5G mobile networks.

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] 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.

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

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

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

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

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

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

[13] 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.

[14] 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.

[15] 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.

[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] 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.

[24] J. Van Der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck, "Http/2-based adaptive streaming of hevc video over 4g/lte networks," *IEEE Communications Letters*, vol. 20, no. 11, pp. 2177–2180, 2016.

[25] P. Houze, E. Mory, G. Texier, and G. Simon, "Applicative-layer multipath for low-latency adaptive live streaming," in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–7.

[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] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[29] 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.

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

[31] 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.

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

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

[34] 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.

[35] (2013) Common media application format (cmaf) for segmented media. [Online]. Available: https://www.iso.org/standard/71975.html

[36] L. Golubchik, J. Lui, and R. Muntz, *Reducing I/O demand in video-on-demand storage servers*. ACM, 1995, vol. 23, no. 1.

[37] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3g networks: analysis and applications," in *Proceedings of the 4th ACM Multimedia Systems Conference*. ACM, 2013, pp. 114–118.