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

Liyang Sun*, Tongyu Zong*, Yong Liu*, Yao Wang*, and Haihong Zhu**

*{ls3817, tz1178, yongliu, yw523}@nyu.edu, Tandon School of Engineering, New York University
**hzhu@futurewei.com, Futurewei Technologies

***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 video streaming by developing dynamic models and optimal control strategies. We further develop practical live video streaming algorithms within the Model Predictive Control (MPC) framework, namely MPC-Live, to maximize user 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 video streaming algorithms can improve the performance dramatically within 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 [2]. 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. For live streaming, users are additionally sensitive to the end-to-end 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 [3]. By contrast, the live streaming latencies on Over-the-Top (OTT) devices range from 10 to 30 seconds [4]. This long video latency could be detrimental for user QoE.

In this paper, we explore the design space of low-latency live video streaming by developing dynamic models and optimal control strategies. We further develop practical live video streaming algorithms within the Model Predictive Control (MPC) framework, namely MPC-Live, to maximize user 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 [5] are adopted in our system. Each video segment (1 second duration) is divided into multiple shorter video chunks (200 ms duration), and the processes of encoding, streaming and decoding are operated in a pipelined fashion. MPC algorithm [6] is 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*. Our main contributions are as follows:

1) We build detailed dynamic models that capture the interplay between video rate adaption, video buffer evolution, playback latency, and video freeze/skip.
2) Based on the dynamic models, we design MPC type of live streaming algorithms to find the optimal solution of video rate selection with future bandwidth estimation in finite horizon.
3) We conduct extensive performance evaluation of the proposed models and streaming algorithms through live streaming simulations driven by real network traces. We demonstrate that our live streaming algorithms can improve the performance dramatically within latency range of two to five seconds.

The full version of this paper is available at [1]. This work was partially supported by USA NSF under contract CNS-1816500.

II. Optimal and Practical Live Streaming

A. Discrete Time Model for Live Streaming

In live video streaming, video segments are encoded and streamed in realtime. Due to network dynamics, a requested video 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*, *playback latency*, and *video freeze/skip*. Table I summarizes the key variables in our model.

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

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:

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

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:

$$b_i = \left(b_{i-1} - z_i - rtt_i - \frac{r_i \Delta}{w_i}\right)^+ + \Delta, \tag{2}$$

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, video freeze will happen and the freeze time is:

$$x_i = \left(z_i + rtt_i + \frac{r_i \Delta}{w_i} - b_{i-1}\right)^+. \tag{3}$$

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

$$l_i = l_{i-1} + x_i. \tag{4}$$

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

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[1]. 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

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

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

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 (4). 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:

$$n_i = (o^{(i)} - \alpha - i)^+, \tag{5}$$

where $o^{(i)}$ is the index of the segment being encoded at re-sync time instant of t_i.

B. Streaming with Bandwidth Oracle in Finite Horizon

Equations (2),(3),(4),(5) 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:

$$\mathcal{S}_{i+1} = \boldsymbol{f}(\mathcal{S}_i, r_i, \{w_i, rtt_i\}). \tag{6}$$

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 [6].

The user experience of streaming segment i can be modeled as a weighted QoE function:

$$\begin{aligned} QoE(\mathcal{S}_i, r_i) =& a_1 Q(r_i) - a_2 x_i - a_3 |Q(r_i) - Q(r_{i-1})| \\ & - a_4 \boldsymbol{g}(l_i) - a_5 n_i, \end{aligned} \tag{7}$$

where $Q(r_i) = \log \frac{r_i}{R_0}$ is the perceptual video quality with R_0 representing the lowest rate in $\mathcal{R}$. And $\boldsymbol{g}(\cdot)$ is the playback latency penalty. 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 by adjusting ϕ. a_1, a_2, a_3, a_4 and a_5 are weights reflecting a user's relative sensitivity to different QoE components.

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:

$$\begin{aligned} \textbf{OPT-CTRL:} \quad & \max_{\{r_i\}} \quad \sum_{i=1}^{m} QoE(\mathcal{S}_i, r_i) \\ \textbf{subject to} \quad & \mathcal{S}_{i+1} = \boldsymbol{f}(\mathcal{S}_i, r_i, \{w_i, rtt_i\}). \end{aligned} \tag{8}$$

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.[2] 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.

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:     $\Omega_i = \emptyset$
 4:     for each state $\mathcal{S}$ in $\Omega_{i-1}$ do
 5:         for each $R_j \in \mathcal{R}$ do
 6:             $\mathcal{S}_i' = \boldsymbol{f}(\mathcal{S}, R_j, \{w_i, rtt_i\})$
 7:             if $\mathcal{S}_i'$ could be part of the overall optimal solution then
 8:                 $\Omega_i \leftarrow \Omega_i \bigcup \mathcal{S}_i'$
 9:             end if
10:         end for
11:     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, we want to limit m to small numbers. 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).

C. MPC-Live Streaming Algorithms

In practice, network condition oracles are not available for any future horizon. Essentially, the deterministic optimal control problems studied in Section II-B become stochastic

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

Algorithm 2 Sliding Horizon-m Streaming

```
    Input: $\mathcal{S}_1$: initial state; $\alpha$ and $\beta$: 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 $\beta$ 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:     $rr_i^{(m)}$ =Horizon-m($\mathcal{S}_i, m, \{w_{[i,i+m-1]}, rtt_{[i,i+m-1]}\}, \mathcal{R}$)
4:     $r_i = rr_i^{(m)}[1]$
5:     $\mathcal{S}_{i+1} = \boldsymbol{f}(\mathcal{S}_i, r_i, \{w_i, rtt_i\})$
6: end for
7: return $r_{[1,\cdots,N]}$
```

optimal control problems, with exogenous random parameters $\{\mathbf{w_i}, \mathbf{rtt_i}\}$. 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 II-B. With a time horizon of m, at stage k, the rate of segment i is selected as:

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

$$\textbf{subject to } \mathcal{S}_{k+1} = \boldsymbol{f}(\mathcal{S}_k, r_k, \{\widehat{w_k}, \widehat{rtt_k}\}), k \in [i, i+m-1],$$

where $\boldsymbol{f}(\cdot)$ is again defined by equations (2),(3),(4),(5), and $\widehat{w_k}$ and $\widehat{rtt_k}$ 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. One can employ different approaches, e.g. [7], to achieve the desirable prediction accuracy and complexity trade-off.

D. Chunk-based Live Streaming

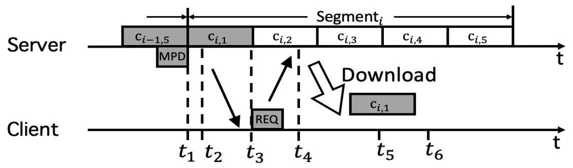
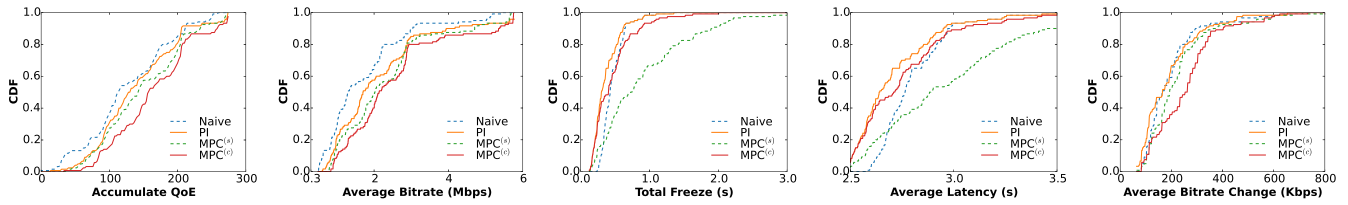


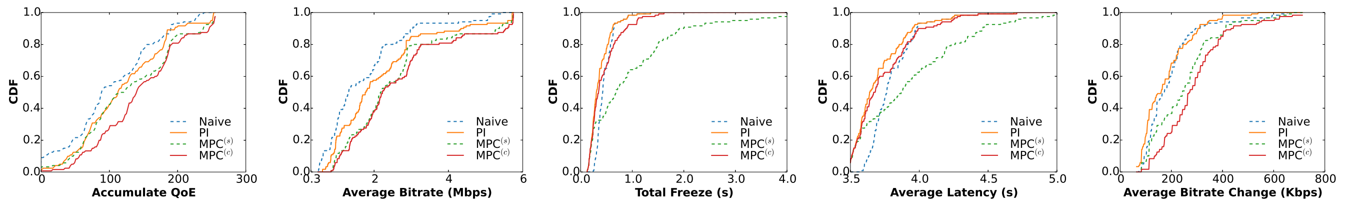
Fig. 1: Latency of Chunk-based Streaming

In most existing DASH solutions, a server can only stream a segment to a client after the entire segment is completely encoded. This introduces a latency of Δ. To achieve low latency, the recent proposal in CMAF [8] 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. 1, with the server-wait mechanism [5], 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. 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. Due to space limit, we refer interested readers to our project website [1] for more details.



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

Fig. 2: 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. 3: CDF of QoE Metrics over 120 4G Traces with $\alpha = 3$

III. Performance Evaluation

We conducted extensive trace-driven simulations to evaluate various live streaming algorithms in the low-latency regime.

- **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.[3]
- **PI-controller** is a buffer-based algorithm [9] that regulates the aggressiveness γ_p of rate selection, based on the difference between the actual buffer length and a reference q_{ref}. The highest rate lower than the regulated bandwidth $\gamma_p\widehat{w}$ is chosen where $\widehat{w}$ is the predicted bandwidth, and q_{ref} is set to be the initial latency. [4]
- **MPC-Live (MPC$^{(s)}$)** is the segment-based implementation of the MPC algorithm proposed in Section II-C. 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 $\widehat{w_i}$.
- **MPC-Chunk (MPC$^{(c)}$)** is the chunk-based implementation of MPC-Live, with chunk size of 200 ms.

We developed a detailed live streaming simulator and implemented all the above algorithms in Python. We used a recent 4G cellular bandwidth dataset collected in NYC Metro Area [7] with 120 individual testing traces to drive the simulations. The CDF of QoE metrics over all the test cases are shown as Fig. 2 and 3. Naive algorithm performs conservatively with lowest average video rate which leads to lowest QoE. Chunk-based streaming algorithms MPC$^{(c)}$ achieves the highest QoE in most cases even with more video rate fluctuation. The performance of PI-controller is in between of Naive and MPC$^{(c)}$. Segment-based MPC$^{(s)}$ algorithm has similar average video rate to its chunk-based MPC, but it suffers higher penalty from video freeze and latency. This further proves that chunk-based design is more suitable for supporting low-latency live streaming.

[3]Video rate is chosen as $\gamma\widehat{w}$, where $\gamma = 80\%$.

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

IV. 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. Through extensive experiments, we demonstrated that our proposed algorithms can improve the performance dramatically in the latency range from two to five seconds. We also demonstrated that chunk-based packaging/streaming is a promising mechanism 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.

References

[1] “Live demo,” https://github.com/ullstreaming2020/Live-Demo, 2019.

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

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

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

[5] 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.

[6] 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.

[7] 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.

[8] (2013) Common media application format (cmaf) for segmented media. [Online]. Available: https://www.iso.org/standard/71975.html

[9] 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.