Real-Time MPC for Adaptive Video Streaming

Vito Andrea Racanelli
Member, IEEE
Politecnico di Bari
vitoandrea.racanelli@poliba.it

Gioacchino Manfredi
Member, IEEE
Politecnico di Bari
gioacchino.manfredi@poliba.it

Luca De Cicco
Member, IEEE
Politecnico di Bari
luca.decicco@poliba.it

Saverio Mascolo
Fellow, IEEE
Politecnico di Bari
saverio.mascolo@poliba.it

***Abstract*—Dynamic Adaptive Streaming over HTTP (DASH) is the standard for video streaming applications such as YouTube and Netflix. According to DASH, the video player must include an Adaptive Bit-Rate (ABR) controller designed to maximize users' Quality of Experience (QoE). The controller must avoid playback interruptions, due to buffer underflow, while at the same time selecting the maximum video encoding quality compatible with the available time-varying bandwidth. This paper proposes a controller designed using a nicely constrained Model Predictive Control (MPC) which employs Bellman Dynamic Programming (DP) to reduce the computational cost of the algorithm from exponential to polynomial. Compared with state-of-the-art ABR algorithms, the proposed Real-Time MPC (RT-MPC) improves QoE while remarkably reducing the computational time, so that the algorithm can be used for both live-video streaming and real-time conferencing.**

I. Introduction

Nowadays, video content constitutes more than half of global Internet traffic [1]. Dynamic Adaptive Streaming over HTTP (DASH) is the standard adopted for classical 2D video streaming by platforms such as YouTube and Netflix and it is employed to stream 360-degree and volumetric videos as well. Online video providers aim at maximizing user engagement to ultimately minimize service abandonment by delivering the best possible Quality of Experience (QoE) under the constraints of user devices and network bandwidth.

DASH-based streaming systems encode video at different bitrate levels, forming the *video level set* $\mathcal{L} = l_1, l_2, \ldots, l_L$. Each level is divided into chunks, and the Adaptive BitRate (ABR) controller selects the appropriate level for each chunk based on the available bandwidth to avoid buffer underflow, thereby avoiding QoE degradation.

Defining a QoE function is challenging due to user-specific factors, but the literature offers metrics for reliable QoE estimation [2], [3]. Most ABR algorithms use static bitrate selection methods based on either estimated network bandwidth (*rate-based* algorithms) [4] or playback buffer size (*buffer-based* algorithms) [5]. These methods often lack efficiency. Alternatives such as the Model Predictive Control (MPC) framework [6] and hybrid approaches combining buffer-based and learning-based techniques [7], [8] have been proposed, but are computationally intensive and less suitable for real-time streaming. This paper introduces an ABR algorithm named *Real-Time MPC* (RT-MPC), which uses MPC and Dynamic Programming (DP) for real-time execution. By precomputing certain terms and applying DP, the algorithm achieves lower computational overhead compared to state-of-the-art approaches, while performing equally or surpassing them in terms of QoE.

This work has been partially funded by the "ELENA" Project n. 13910 under the Next Generation EU Italian National Recovery and Resilience Plan (NRRP) - Bando MedITech 2023 n.3, and partially funded by the "LOREN" Project n. 20223Y85JN under the PRIN (Progetti di Ricerca d'Interesse Nazionale) 2022 of the Italian Ministry of Research.

II. Related work

This section provides a brief overview of existing QoE-based adaptive streaming techniques.

In [7], a Buffer-Based (BB) approach selects the appropriate bitrate to maintain smooth video playback by keeping buffer occupancy above a threshold. BOLA [5] is another BB technique that uses Lyapunov optimization to choose the best bitrate based solely on buffer occupancy.

RobustMPC [6] goes beyond buffer occupancy by also considering throughput predictions to select the bitrate that maximizes QoE, accounting for errors between predicted and observed throughputs.

Pensieve [8] was the first to use Deep Reinforcement Learning (DRL) for training a neural network to select bitrates based on data from client video players. Similarly, [9] employs Bayesian Neural Networks (BNN) combined with an MPC-based ABR algorithm to predict future throughput distributions and adapt to network dynamics.

To integrate buffer-based and learning-based approaches, Stick [10] uses DRL to compute the buffer-bound necessary for maximizing QoE, offering computational improvements over previous methods. Another approach, presented in [11], uses Meta Reinforcement Learning (Meta-RL) to quickly adapt ABR policies to changing network conditions by separating throughput inference from the control mechanism.

All these techniques often have high computational overhead, making them impractical for real-time streaming. The proposed RT-MPC addresses this issue by offering a more realistic implementation with lower processing time.

III. The Video Streaming Model

According to the DASH standard, a video denoted by v is encoded into different representations or *levels* $l \in \mathcal{L}$. Each video level is divided into K segments of fixed duration τ. Video segments are identified by an index $s \in \mathcal{S}$, where $\mathcal{S} = \{1, 2, \ldots, K\}$ is the set of video segment indices. Let $c : \mathcal{L} \to \mathbb{R}$ denote a non-decreasing function that maps a selected bitrate

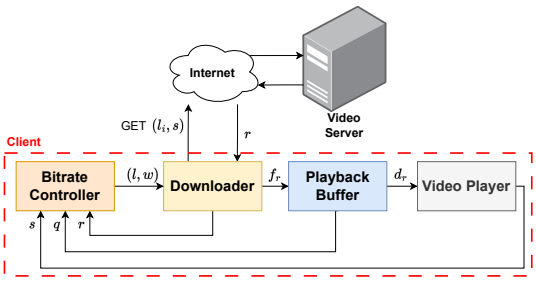

Fig. 1: Block diagram of an adaptive video streaming system.

video level l_s with the video quality perceived by the user while playing the s-th segment [12].

As shown in Figure 1, when a streaming session starts, the controller instructs the downloader to fetch segments at the appropriate quality level to optimize the user's QoE. The control law is computed after each segment download, determining the bitrate $u_s \in \mathcal{L}$ for the next segment.

The downloader starts fetching segment s at time t_s, completing the download after time t_{D_s}. It then waits w_s seconds before requesting segment $s+1$, at quality based on the playback buffer's queue length and available bandwidth. It stops downloading if the buffer exceeds the threshold Q_H.

The download time t_{D_s} for segment s is given by:

$$t_{D_s} = \frac{C_s(u_s)}{B_s} \tag{1}$$

Where B_s is the average available bandwidth obtained during the segment download and $C_s(u_s)$ is the video segment size coded at bitrate u_s.

The video player drains the playback buffer and renders the video. The player's state $\alpha(t)$ is 1 when playing (buffer is being drained) and 0 when paused (buffer is empty or at session start). The player's drain rate $d_r(t)$ is:

$$d_r(t) = P_R \cdot \alpha(t) \tag{2}$$

Where P_R is the video playback rate, generally constant and equal to 1 second per second.

The playback buffer length $q(t)$ impulsively increases by τ seconds when a segment is downloaded and decreases linearly as the video is played. The buffer length after downloading segment s is:

$$q(t_{s+1}) = q(t_s) + \tau - \int_{t_s}^{t_{s+1}} d_r(\xi)\, d\xi \tag{3}$$

Before a session starts, the server provides necessary information such as the set $\mathcal{L}$, video duration T, and chunk length τ through a manifest file.

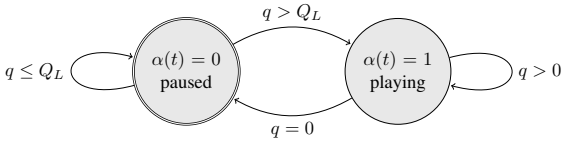

Fig. 2: Depiction of player state as finite state machine.

IV. Real-Time MPC Controller for ABR

This section presents the design and implementation of the proposed Real-time Adaptive Bitrate (ABR) algorithm, which is based on Model Predictive Control (MPC) and Dynamic Programming (DP).

A. The QoE Reward

The main goal of dynamic bitrate adaptation is to maximize users' quality of experience (QoE), which directly impacts user engagement. The QoE is often used as the reward function to evaluate the performance of an ABR algorithm. A commonly used metric to approximate QoE is defined in [6] as:

$$\begin{aligned} QoE_{lin}\,{}_1^K = & \underbrace{\sum_{k=1}^{K} c(u_k)}_{\text{Quality reward}} - \lambda \underbrace{\sum_{k=1}^{K-1} |c(u_{k+1}) - c(u_k)|}_{\text{Smoothness penalty}} \\ & - \mu \underbrace{\sum_{k=1}^{K} r_s}_{\text{Rebuffering penalty}} - \mu_s \underbrace{T_s}_{\text{Startup delay}} \end{aligned} \tag{4}$$

where λ, μ, and μ_s are non-negative weighting parameters. The *Quality reward* term reflects the visual quality of each level. The *Smoothness penalty* term penalizes level switches, while the *Rebuffering penalty* reflects the negative impact of rebuffering on QoE, and the *Startup delay* penalizes the initial delay before video playback begins.

Although [6] defines other metrics such as QoE_{log} and QoE_{hd}, we use QoE_{lin} to compare the proposed RT-MPC approach with other state-of-the-art techniques since they provide very similar results.

It is important to note that the reward function QoE used in the MPC optimization differs from QoE_{lin} (4), which is indeed used for fair comparisons of different algorithms.

As already known, frequent bitrate changes can negatively impact QoE (as captured by the *Smoothness penalty* in (4)). However, if a level change significantly improves visual quality, and substantial time has passed since the last change, it may be advantageous to switch levels, even with a slight smoothness penalty. This insight suggests that isolated level switches can enhance QoE at the cost of reduced smoothness.

To exploit this observation, we introduce the *Smoothness Window* which is a vector of size w defined as follows:

$$\mathbf{W} = [u_{s-w+1}, u_{s-w+1}, u_{s-w+2}, ...u_s]. \tag{5}$$

containing the last values of coding levels u belonging to the window. At the beginning, when filling the window vector $\mathbf{W}$, if the index $s - w + i < 0$ then $u_{s-w+i} = u_0$.

The idea is to modify the QoE (4) by adding a new component, named the *Smoothness window penalty*, that takes into account how the video levels have changed in the past window. This new added component penalizes frequent video level transitions over time, which is known to be detrimental to the user's experience [13].

Thus, the QoE to be maximised by the RT-MPC controller over the prediction window H_P is defined as follows:

$$QoE_s^{s+H_P} = \underbrace{\sum_{k=s}^{s+H_P} c(u_k)}_{\text{Quality reward}} - \lambda_1 \underbrace{\sum_{k=s}^{s+H_P-1} |c(u_{k+1}) - c(u_k)|}_{\text{Smoothness penalty}} - \mu_1 \underbrace{\sum_{k=s}^{s+H_P} r_s}_{\text{Rebuffering penalty}} - \beta \underbrace{\sum_{k=0}^{W-1} |u_{s-k} - u_{s-k-1}|}_{\text{Smoothness window penalty}} \quad (6)$$

where λ_1, μ_1, and β are non-negative weighting parameters.

B. Bandwidth Estimation

We employ a simple estimator $\hat{B}_s^{s+1} = \gamma B_s$, which estimates the next step's bandwidth as a fraction $\gamma \leq 1$ of the current bandwidth. This is a worst-case assumption for optimization.

C. Reducing Computational Overhead

For real-time implementation, the ABR algorithm must minimize the computational delay w_s incurred while solving the MPC optimization problem.

We observed that the optimal bitrate selection requires traversing a tree structure to find the maximum solution. Exhaustively exploring the tree along the prediction horizon involves $\mathcal{O}(|\mathcal{L}|^{H_P})$ states, with each state requiring an $\mathcal{O}(H_P)$ search plus constant time, leading to a total computational cost of $\mathcal{O}(|\mathcal{L}|^{H_P} H_P)$. Given that the proposed optimization is a multi-stage decision process, we exploit the Bellman's optimality principle [14] to reformulate the optimization problem as:

$$J^* = \max_{u_s} \left[QoE_s^{s+1} + \cdots + \max_{u_{s+Hp-1}} \left[QoE_{s+Hp-1}^{s+Hp} \right] \right] \quad (7)$$

By using Dynamic Programming to solve (7), the exponential cost is reduced to polynomial one $\mathcal{O}(|\mathcal{L}|{H_P}^2)$. Examining the reward function (6), it is worth noting that the *Quality Reward*, *Smoothness Penalty*, and *Smoothness Window Penalty* terms depend only on the input sequence from s to $s + H_p$. Since these terms are independent of the system's state, they can be precomputed. During initialization, the RT-MPC algorithm generates a table of potential rewards for all possible paths in the tree, further reducing computational cost to $\mathcal{O}(|\mathcal{L}|H_P)$. Another important consideration is that the smoothness penalty increases with frequent video level changes. A broader set $\mathcal{L}$ reachable within H_p steps requires exploring more graph branches, potentially leading to higher smoothness penalties, especially in highly variable bandwidth scenarios. To address this issue, we introduce Δ_u as the maximum level variation allowed at each control step. By imposing this constraint, we define $\mathbb{X}_R \subseteq \mathcal{L}$ as the reachability set within the horizon H_P, i.e., the subset of $\mathcal{L}$ reachable from the current chunk level by adding or subtracting Δ_u. This furtherly reduces the computational cost to $\mathcal{O}(|\mathbb{X}_R|H_P)$.

The formulation of the QoE maximization problem is shown in Figure 3, where QoE is defined in (6).

$$J^* = \max_{u_s,\ldots,u_{s+H_P}} QoE_s^{s+H_P} \quad (8)$$

$$\text{s.t.} \quad t_{s+1} = t_s + \frac{C_s(u_s)}{\hat{B}_s^{s+1}} + \omega_s \quad (9)$$

$$\hat{B}_s^{s+1} = \gamma B_s \quad (10)$$

$$q(t_{s+1}) = q(t_s) + \tau - \int_{t_s}^{t_{s+1}} d_r(\xi)\, d\xi \quad (11)$$

$$r_s(u_s) = \frac{C_s(u_s)}{\hat{B}_s^{s+1}} + \omega_s - \int_{t_s}^{t_{s+1}} \frac{d_r(\xi)}{P_R}\, d\xi \quad (12)$$

$$d_r(t) = P_R \alpha(t) \quad (13)$$

$$\alpha(t) = \begin{cases} 0 \\ 1 \end{cases} \quad \text{According to Fig. 2} \quad (14)$$

$$u_s \in \mathbb{X}_R, q \geq 0, t_s \geq 0, \omega_s \geq 0 \quad \forall s \in \mathcal{S} \quad (15)$$

Fig. 3: Formulation for QoE maximization subject to buffer and throughput dynamics.

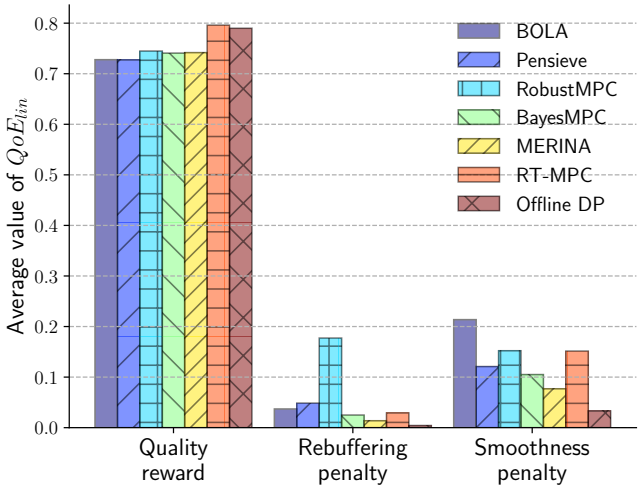

Fig. 4: Comparison of the single terms in the QoE_{lin} metric

V. Experimental Results

The experimental results are obtained using the implementation of the virtual player described in [8]. Such a virtual player simulates the transmission of video data from a server to a client, allowing us to vary the network conditions through the use of throughput traces gathered from real user sessions. To perform a fair comparison, we have employed the same environment settings as in [8], [15].

In this section, we describe the testbed, the scenarios and the used evaluation metrics. A comparison with other ABR algorithms is also provided.

A. The testbed setup

$\mathcal{L} = \{300, 750, 1200, 1850, 2850, 4300\}$ *kbps* is the set of available bitrate levels. The duration of each video chunk is set to $\tau = 4$ seconds, and the total number of video chunks is 48, which corresponds to a video duration of 192 seconds. The calibration of the controller parameters has been carried out utilising Optuna [16], which is a tool for parameter tuning

based on the maximization of a reward function. The resulting values for the parameters are as follows: $H_p = 4$, $H_u = 4$, $W = 4$, $c(l_s) = l_s/1000$, $\gamma = 1.45$, $\beta = 7.21$, $\Delta_u = 3$, $\lambda_1 = 2.48$ and $\mu_1 = 0.33$.

Regarding the QoE_{lin} (4) used for performance comparison, we set $\lambda = 4.3$ and $\mu = 1$ as in [6]. Finally, the buffer capacity is limited to 1 minute. All experiments were performed on a machine running Ubuntu 22.04 with 32GB of RAM and an Intel i7 8th generation CPU.

B. Scenarios

The goal is to investigate different bitrate adaptation techniques under realistic network conditions. To the purpose, the Federal Communications Commission (FCC) [17] dataset has been used. This dataset consists of more than 1 million sets of throughput measurements during a 5 s interval. We extract throughput traces of the same server and client IP address and concatenate these to match the length of the video. For experiments, we randomly pick 1000 concatenated traces.

C. Adaption algorithms

We compare RT-MPC against the following algorithms, which represent the state-of-the-art in bitrate adaptation techniques: Buffer-Based (BB) [7], Rate-Based (RB), BOLA [5], RobustMPC [6], Pensieve [8], BayesMPC [9], and Merina [11]. In order to maintain a consistent and fair basis for the comparison, we have evaluated all the algorithms using the Eq. 4 of the QoE defined in [6]. Furthermore, to provide an absolute reference for comparison, we also include the results obtained using the offline optimal strategy, which gives the best achievable solution. The offline optimal strategy represents the maximum QoE that an oracle policy with complete and perfect knowledge of future network throughput could achieve. This strategy is obtained through dynamic programming with complete information about future throughput, which is never available in practice. It is therefore to be considered as a reference target.

D. Discussion

Figure 4 shows the average values of the first three components of QoE_{lin} in (4) for each of the algorithms. In particular, RT-MPC shows the highest *Quality reward* while keeping a low *Rebuffering penalty*. This comes at the cost of a *Smoothness penalty* slightly larger than the average due to the worst case bandwidth assumption.

RT-MPC improves average QoE values across the dataset in an interval from 3% to 6%.

Comparison of computational times are shown in Table I. RT-MPC provides a significant improvement of the computational time average and standard deviation (one order of magnitude smaller *wrt* the best second algorithm, and two orders of magnitude improvement of the maximum computational time *wrt* the second best algorithm). This shows that the algorithm can be used in real-time control for both live-video streaming and conferencing.

TABLE I: Computational time of different algorithms. Values are expressed in ms.

	Pensieve	**RobustMPC**	**BayesMPC**	**Merina**	**RT-MPC**
μ	1.79	3.58	39.80	1.04	0.10
σ^2	1.32	5.53	3.18	1.84	0.02
max	108.84	35.40	211.74	157.68	0.21

VI. Conclusions

A Real-Time Model Predictive Control algorithm for video streaming applications to maximize the user's QoE has been proposed. Bellman Dynamic Programming along with a specific constraint on the number of encoding level changes have been exploited to reduce the computational complexity of the optimization problem and enable an effective real-time implementation. A careful evaluation of RT-MPC against the leading state-of-the-art ABR algorithms shows significant improvements of both QoE and computational time, which is essential for real-time execution. In particular, RT-MPC reduces average, standard deviation, and maximum values of the computation time of several orders of magnitude.

References

[1] Sandvine, "2024 global internet phenomena report," *Report*, 2024.

[2] K. Mitra, A. Zaslavsky, and C. Åhlund, "QoE modelling, measurement and prediction: A review," *arXiv preprint arXiv:1410.6952*, 2014.

[3] D. Tsolkas, E. Liotou, N. Passas, and L. Merakos, "A survey on parametric QoE estimation for popular services," *Journal of network and computer applications*, vol. 77, pp. 1–17, 2017.

[4] 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 *Proc. of ACM SIGCOMM*, 2016, pp. 272–285.

[5] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal bitrate adaptation for online videos," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1698–1711, 2020.

[6] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," in *Proc. of ACM SIGCOMM '15*, 2015, pp. 325–338.

[7] 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," in *Proc. of ACM SIGCOMM '14*, 2014, pp. 187–198.

[8] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. of ACM SIGCOMM '17*, 2017, pp. 197–210.

[9] N. Kan, C. Li, C. Yang, W. Dai, J. Zou, and H. Xiong, "Uncertainty-aware robust adaptive video streaming with bayesian neural network and model predictive control," in *Proc. of NOSSDAV workshop '21*, 2021.

[10] T. Huang, C. Zhou, R.-X. Zhang, C. Wu, X. Yao, and L. Sun, "Stick: A harmonious fusion of buffer-based and learning-based approach for adaptive streaming," in *Proc. of IEEE INFOCOM*, 2020, pp. 1967–1976.

[11] N. Kan, Y. Jiang, C. Li, W. Dai, J. Zou, and H. Xiong, "Improving generalization for neural adaptive video streaming via meta reinforcement learning," in *Proc. of ACM Multimedia '22*, 2022, pp. 3006–3016.

[12] G. Cofano, L. De Cicco, and S. Mascolo, "A hybrid model of adaptive video streaming control systems," in *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, 2016, pp. 6601–6606.

[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, 2015.

[14] R. E. Bellman and S. E. Dreyfus, *Applied dynamic programming*. Princeton university press, 2015, vol. 2050.

[15] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, "Mahimahi: accurate Record-and-Replay for HTTP," in *Proc. of USENIX ATC '15*, 2015, pp. 417–429.

[16] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proc. of ACM SIGKDD '19*, 2019.

[17] F. C. Commission, "Measuring broadband raw data releases," https://www.fcc.gov/oet/mba/raw-data-releases, 2023.