This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2019.2929125, IEEE Transactions on Mobile Computing

A Control Theoretic Approach to ABR Video Streaming: A Fresh Look at PID-based Rate Adaptation

Yanyuan Qin, *Student Member, IEEE,* Ruofan Jin, *Member, IEEE,* Shuai Hao, *Member, IEEE,* Krishna R. Pattipati, *Fellow, IEEE*, Feng Qian, *Member, IEEE,* Subhabrata Sen, *Fellow, IEEE*, Chaoqun Yue, *Student Member, IEEE,* Bing Wang, *Senior Member, IEEE*

Abstract—Adaptive bitrate streaming (ABR) has become the *de facto* technique for video streaming over the Internet. Despite a flurry of techniques, achieving high quality ABR streaming over cellular networks remains a tremendous challenge. ABR streaming can be naturally modeled as a control problem. There has been some initial work on using PID, a widely used feedback control technique, for ABR streaming. Existing studies, however, either use PID control directly without fully considering the special requirements of ABR streaming, leading to suboptimal results, or conclude that PID is not a suitable approach. In this paper, we take a fresh look at PID-based control for ABR streaming. We design a framework called PIA (PID-control based ABR streaming) that strategically leverages PID control concepts and incorporates several novel strategies to account for the various requirements of ABR streaming. We evaluate PIA using simulation based on real LTE network traces, as well as using real DASH implementation. The results demonstrate that PIA outperforms state-of-the-art schemes in providing high average bitrate with significantly lower bitrate changes (reduction up to 40%) and stalls (reduction up to 85%), while incurring very small runtime overhead. We further design PIA-E (PIA Enhanced), which improves the performance of PIA in the important initial playback phase.

Index Terms—Adaptive video streaming, control theory, PID control, DASH

1 Introduction

Video streaming has come to dominate mobile data consumption today. As per Cisco's 2016 Visual Network Index report [4], mobile video traffic now accounts for more than half of all mobile data traffic. Ensuring good viewing experience for this important application class is critical to content providers, content distribution networks, and mobile operators. Despite much effort, achieving good quality video streaming over cellular networks remains a tremendous challenge [5].

Most video contents are currently streamed using Adaptive Bit-Rate (ABR) streaming over HTTP, the *de facto* technology adopted by industry. In ABR streaming, a video is encoded into multiple resolutions/quality levels (or tracks). The encoding at each resolution/quality level is divided into equal-duration chunks, each containing data for a short interval's worth of playback (e.g., several seconds). A chunk at a higher resolutions/quality level requires more bits to encode, and is therefore larger in size. During playback, to fetch the content for a particular playpoint in the video, the video player dynamically determines what bitrate/quality chunk to download based on time-varying network conditions. The resulting play-back involves showing different portions of the video using chunks selected from different tracks.

Various user engagement studies [17], [22], [24], [27] indicate that satisfactory ABR streaming needs to achieve three conflicting goals simultaneously: (1) maximize the playback bitrate; (2) minimize the likelihood of stalls or rebuffering; and (3) minimize the variability of the selected video bitrates for a smooth viewing experience. Reaching any of the three goals alone is relatively easy – for instance, the player can simply stream at the highest bitrate to maximize the video quality; or it can stream at the lowest bitrate to minimize the stalls. The challenge lies in achieving all three goals simultaneously, especially over highly varying network conditions, typical of the last-mile scenarios in cellular networks.

Rate adaptation for ABR streaming can be naturally modeled as a control problem: the video player monitors the past network bandwidth and the amount of content in the playback buffer to decide the bitrate level for the current chunk; the decision will then affect the buffer level, which can be treated as feedback to adjust the decision for the next chunk. PID (named after its three correcting terms, namely "proportional", "integral", and "derivative" terms) is one of the most widely used feedback control techniques in practice [12]. It is conceptually easy to understand and computationally simple. There has been initial work on using PID control theory for ABR streaming. Specifically, the studies in [15], [16] directly apply the standard PID controller to ABR streaming with no modifications, which was shown to lead to significantly suboptimal performance [33]. The

- *Manuscript received xx xx, 2018. A preliminary version of this paper [30] appeared in INFOCOM 2017.*
- *Y. Qin, C. Yue and B. Wang are with the Computer Science & Engineering Department at the University of Connecticut. The work of R. Jin was done when he was a PhD student with the Computer Science & Engineering Department at the University of Connecticut. S. Hao and S. Sen are with AT&T Labs - Research. K. Pattipati is with the Electrical & Computer Engineering Department at the University of Connecticut. The work of F. Qian was done while he was with the Computer Science Department at Indiana University.*

1536-1233 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2019.2929125, IEEE Transactions on Mobile Computing

studies [36], [39] conclude that PID control is not suitable for ABR streaming.

In this paper, we adopt a contrarian perspective and take a fresh look at the potential of PID control for ABR video streaming. We start by pointing out that a recent heuristic technique, BBA [20], can be shown to be, in effect, using a simplified form of PID control. We then conduct an in-depth study that explores using PID control for ABR streaming. Specifically, we design *PIA* (PID-control based ABR streaming), a novel control-theoretic video streaming scheme that strategically incorporates PID control concepts and domain knowledge of ABR streaming. Our main contributions include the following.

- We take a fresh look at PID-based control for ABR streaming, and strategically leverage PID control concepts as the base framework for PIA. Specifically, the *core controller* in PIA differs from those in [15], [16] in that we define a control policy that makes the closed-loop control system linear, and easy to control and analyze. The core controller maintains the playback buffer to a target level, so as to reduce rebuffering.
- We add domain-specific enhancements to further improve the robustness and adaptiveness of ABR streaming. Specifically, PIA addresses two key additional requirements of ABR streaming, i.e., maximizing playback bitrate and reducing frequent bitrate changes. It also incorporates strategies to accelerate initial ramp-up and protect the system from saturation. We further develop *PIA-E* (PIA Enhanced) that improves the performance of PIA in the important initial playback phase, by dynamically adjusting the parameters used in the control loop.
- We explore parameter tuning. Specifically, the proportional gain K_p and the integral gain K_i are the two fundamental and most critical parameters that guide the PIA controller's behavior. We develop a methodology that systematically examines a wide spectrum of network conditions and parameter settings to derive their (K_p, K_i) configurations that yield satisfactory quality of experience (QoE). Our results demonstrate that a common set of (K_p, K_i) values exist that have good performance across a wide range of network settings, indicating our schemes can be easily deployed in practice.

We conduct comprehensive evaluations of PIA using a large number of real cellular network traces with diverse network variability characteristics. The traces were collected from two commercial LTE networks at diverse geographic locations, with a range of mobility conditions. Our key findings include the following.

- PIA achieves comparable bitrates as two state-of-the-art schemes, BBA [20] and MPC [39], while substantially reducing bitrate changes (49% and 40% lower, respectively) and rebuffering time (68% and 85% lower, respectively). Overall, PIA achieves the best balance among the three QoE metrics. Compared to PIA, the enhanced version, PIA-E, achieves higher bitrate for the beginning part of the video; for the entire video, PIA-E has similar rebuffering, average bitrate and bitrate changes as PIA.
- PIA and PIA-E have low computation overhead (e.g., comparable to BBA and only 0.5% of MPC based on our simulations). Our emulation results also show that their execution time is less than 2 seconds for a 15-minute video.

C_t	Network bandwidth at time t
$\widehat{C}_t$	Estimated network bandwidth at time t
x_t	Buffer level at time t (in seconds)
x_r	Target buffer level (in seconds)
R_t	Selected video bitrate for time t
Δ	Video chunk duration (in seconds)
δ	Startup latency (in seconds)
u_t	PID controller output
K_p, K_i, K_d	PID controller parameters
ζ	Damping ratio
ω_n	Natural frequency
β	Setpoint weighting parameter

TABLE 1: Key notation.

The rest of this paper is organized as follows. Section 2 summarizes the background and motivations. Section 3 presents PIA, the PID based controller for ABR streaming. Section 4 evaluates the performance of PIA under a wide range of settings. Section 5 presents PIA-E, designed to improve the startup performance of PIA. Section 6 presents the implementation and evaluation of PIA and PIA-E using a real video player. Section 7 briefly describes the related work. Finally, Section 8 concludes the paper and presents future directions.

2 Motivation and Background

ABR streaming has been used in many commercial systems [8], [2], [1]. For a satisfactory user-perceived QoE, ABR streaming needs to optimize several conflicting goals, including maximizing the average playback rate, minimizing stalls (or rebuffering), and reducing sudden and frequent quality variations [17], [22], [24], [27]. We next formulate ABR streaming as a control problem and describe the motivation for our study. Table 1 summarizes the main notation used in this paper.

2.1 ABR streaming as a control problem

Deciding which level to choose in ABR streaming can be modeled as a control problem. Specifically, let x_t be the buffer level (in seconds) of the video player at time t, C_t the real-time network bandwidth at time t, and R_t the bitrate of the video chunk that is being downloaded at time t. Further, let Δ denote the video chunk duration (i.e., the duration of a chunk's playback time), δ denote the startup delay, i.e., how long it will take for the player to start playing. Then the player's buffer dynamics can be written as

$$\dot{x}_t = \begin{cases} \frac{C_t}{R_t}, & \text{if } t \le \delta \\ \frac{C_t}{R_t} - \mathbf{1}(x_t - \Delta), & \text{otherwise} \end{cases} \quad (1)$$

where $\mathbf{1}(x_t - \Delta) = 1$ if $x_t \ge \Delta$; otherwise, $\mathbf{1}(x_t - \Delta) = 0$ since in ABR streaming, a chunk has to be downloaded completely before any part of it can be played back.

1536-1233 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2019.2929125, IEEE Transactions on Mobile Computing

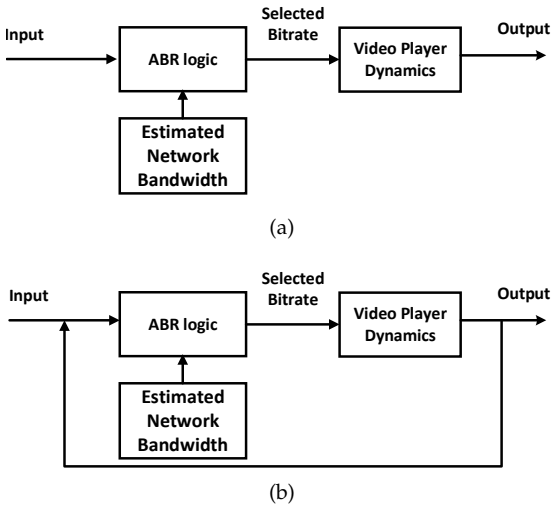


Fig. 1: ABR streaming based on open-loop control (a) and closed-loop control (b).

In Eq. (1), $\dot{x}_t$ is the rate of change of the buffer at time t. Here, C_t/R_t models the relative buffer filling rate. If $C_t > R_t$, i.e., the actual network bandwidth is larger than the bitrate of the video chunk being downloaded, the buffer level will increase. Otherwise, the buffer level will be at the same level (if $C_t = R_t$) or decrease (if $C_t < R_t$).

One simple control strategy is to select the video bitrate for each chunk based on the prediction of real-time link bandwidth, $\widehat{C}_t$. Specifically, it simply chooses the highest bitrate that is less than $\widehat{C}_t$. This is an open-loop control strategy (illustrated in Fig. 1(a)), since the output (e.g., the buffer level) is not fed back to the system to assist the decision making. It is not robust against network link bandwidth estimation errors. As an example, it may choose a high video bitrate if the estimated bandwidth, $\widehat{C}_t$, is high, even if the current playback buffer level is very low. If it turns out that $\widehat{C}_t$ is an overestimate of the actual network bandwidth, the buffer can be even further drained and become empty, causing stalls. Closed-loop or feedback control, as illustrated Fig. 1(b), is more effective in dealing with network link bandwidth estimation errors. We therefore focus on closed-loop/feedback control in this paper.

2.2 PID control

As mentioned earlier, PID control is by far the most common way of using feedback in engineering systems. A PID controller works by continuously monitoring an "error value", defined as the difference between the setpoint and measured process variable [12]. Specifically, let u_t represent the control signal, and e_t the error feedback at time t. Then

$$u_t = K_p e_t + K_i \int_0^t e_\tau d\tau + K_d \frac{de_t}{dt}, \tag{2}$$

where the three parameters K_p, K_i and K_d are all non-negative, and denote the coefficients for the proportional, integral and derivative terms, respectively. As defined above, a PID controller takes account of the present, past and future values of the errors through the three terms, respectively. Some applications may require using only one or two terms to provide the appropriate system control. This is achieved by setting the other parameters to zero. A PID controller is called a PI, PD, P or I controller in the absence of the respective control actions [12].

In the video streaming scenario, the real-time buffer level is the measured process variable, and the reference/target buffer level is the setpoint. We next show that a recent state-of-the-art buffer based scheme, BBA [20], can be mapped to a P-controller (though the paper does not claim any control-theoretic underpinnings). In BBA, the video player maintains a buffer level, and empirically sets two thresholds, $\theta_{\text{high}} > \theta_{\text{low}}$. If the buffer level is below θ_{low}, the video player always picks the lowest bitrate, R_{min}; if the buffer level is above θ_{high}, the video player picks the highest bitrate, R_{max}; otherwise, the video player picks the video bitrate proportionally to buffer level. The selected bitrate, R_t, can therefore be represented as

$$R_t = \begin{cases} R_{\text{min}}, & x_t < \theta_{\text{low}}, \\ \frac{R_{\text{max}}-R_{\text{min}}}{\theta_{\text{high}}-\theta_{\text{low}}}\,(x_t-\theta_{\text{low}}) + R_{\text{min}}, & \theta_{\text{low}} \le x_t \le \theta_{\text{high}} \\ R_{\text{max}}, & x_t > \theta_{\text{high}} \end{cases}$$

Comparing the above with Eq. (2), we see that it is equivalent to a P-controller when $x_t \in [\theta_{\text{low}}, \theta_{\text{high}}]$ with $K_p = \frac{R_{\text{max}}-R_{\text{min}}}{\theta_{\text{high}}-\theta_{\text{low}}}$, $K_i = 0$ and $K_d = 0$.

BBA has been tested successfully in a large-scale deployment [20], indicating that a PID-type control framework has the potential for ABR streaming. On the other hand, P-controller only considers the present error (i.e., the proportional term), and ignores the other two terms. It is well known that the absence of an integral term in a system may prevent the system from reaching its target value [12]. This is especially true for video streaming, where inaccurate network bandwidth estimation may cause the error to accumulate over time. Therefore, including the integral term can potentially further improve the performance of BBA. PID's ability to address accumulative errors is advantageous compared to model predictive control (MPC) based approach in [39], which does not consider accumulative errors (unless new state variables are added) and also requires accurate network bandwidth prediction. In addition, MPC is much more computation-intensive than PID (see Section 4.4).

We investigate PID-based control for ABR streaming in this paper, motivated by the widespread adoption of PID control in various domains beyond video streaming (e.g., industrial control, process control) and BBA. Using PID for ABR streaming, however, has several challenges. First, the goal of PID control is to maintain a target buffer level that is only indirectly related to QoE. Indeed, while maintaining the buffer at a target level can help in preventing rebuffering, it does not help with the other two metrics on playback quality and bitrate variation. Second, PID is often used in continuous time and state space, while video streaming is a discrete-time system, where the decisions are made at chunk boundaries and the video bitrate levels are discrete. Finally, while PID is conceptually simple, the parameters (K_p, K_i and K_d) need to be tuned carefully. Here important questions are how to choose these parameters, and to determine whether there exists a parameter set that is applicable to a wide range of network and video settings. Some of the above challenges have been pointed out in [36],

1536-1233 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2019.2929125, IEEE Transactions on Mobile Computing

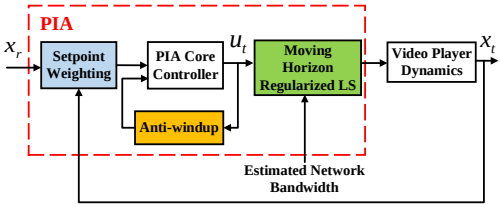


Fig. 2: PIA main components.

[39], which take the position that PID is not suitable for ABR streaming. As we shall show, none of the above challenges is a fundamental hurdle for using PID-based control for ABR streaming.

3 Adapting PID Control for ABR Streaming

We propose PIA, a PID based rate adaption algorithm for ABR streaming. As shown in Fig. 2, it contains a PI-based core control block as well as three mechanisms to address specific requirements for ABR streaming. We first describe the core component, and then the three performance enhancing mechanisms.

3.1 PIA core component

The core component of PIA adjusts the standard PID control policy in Eq. (2) so that the resultant closed-loop system is linear, and hence easier to control and analyze. We next define the controller output, analyze the system behavior, and provide insights into how to choose the various parameters.

Recall the dynamic video streaming model in Eq. (1), where x_t is the video player buffer level at t, C_t is the network bandwidth at time t, and R_t is the video bitrate chosen for time t. We define the controller output, u_t, as

$$u_t = \frac{C_t}{R_t}, \tag{3}$$

and set the control policy as

$$u_t = K_p(x_r - x_t) + K_i \int_0^t (x_r - x_\tau)d\tau + \mathbf{1}(x_t - \Delta) \tag{4}$$

where K_p and K_i denote, respectively, the parameters for proportional and integral control, x_r denotes target buffer level, and Δ is the chunk duration. The choice of x_r depends on system constraints, a point we will come back to in Section 4.

The above control policy differs from the standard PID control policy in Eq. (2) in the last term $\mathbf{1}(x_t - \Delta)$, which is a novel aspect of our design. As we shall see, it provides linearity, making the closed-loop control system easier to control and analyze. In our control policy, the parameter for derivative control $K_d = 0$ (hence strictly speaking, our controller is a PI controller). This is because derivative action is sensitive to measurement noise [12] and measuring network bandwidth in our context is prone to noise.

Intuitively, u_t defined in Eq. (3) is a unitless quantity representing the relative buffer filling rate. With u_t selected, based on Eq. (3), the player can select the corresponding bitrate as

$$R_t = \frac{\widehat{C}_t}{u_t}, \tag{5}$$

where $\widehat{C}_t$ is the estimated link bandwidth at time t. Since video bitrate levels are discrete, we can choose the bitrate to be the highest that is below $\widehat{C}_t/u_t$. This choice of R_t can increase, decrease or maintain the buffer level.

We next analyze the system to provide insights into its behavior as well as providing guidelines in choosing the controller parameters. Combining equations (1) and (4) yields

$$\dot{x}_t = u_t - \mathbf{1}(x_t - \Delta) = K_p(x_r - x_t) + K_i \int_0^t (x_r - x_\tau)d\tau, \tag{6}$$

when the video starts playback (i.e., when $t \geq \delta$). We see that it is a linear system. Taking Laplace transform on both sides of Eq. (6) yields

$$sx(s) = K_p(x_r(s) - x(s)) + \frac{K_i}{s}\left(x_r(s) - x(s)\right), \tag{7}$$

where s is a complex Laplace transform variable. Let $T(s)$ be the system transfer function, which describes the relationship of the input and output of a linear time-invariant system. From Eq. (7), we have

$$T(s) = \frac{x(s)}{x_r(s)} = \frac{K_p s + K_i}{s^2 + K_p s + K_i}, \tag{8}$$

which is a second-order system. In the above transfer function, since $T(0) = 1$, the system can track step changes in the target buffer level, x_r, with zero error in the steady state [34], indicating that our system can maintain the preset target buffer level. From Eq. (8), we have

$$2\zeta\omega_n = K_p, \quad \omega_n^2 = K_i\,, \tag{9}$$

where ζ and ω_n are the *damping ratio* and the *natural frequency*, respectively, two important properties of the system. Solving the above two equations, we have

$$\zeta = \frac{K_p}{2\sqrt{K_i}}, \quad \omega_n = \sqrt{K_i}\,. \tag{10}$$

Damping ratio represents the system's ability for reducing its oscillations. In our context, it measures how the buffer will oscillate around the target buffer level – small damping will cause the buffer to change rapidly, while large damping will cause the buffer to change slowly in a sluggish manner. Natural frequency represents the frequency at which a system tends to oscillate in the absence of any driving or damping force. Empirically it has been found that ζ in the range $[0.6, 0.8]$ yields a very good system performance [28]. As a result, K_p and K_i should be chosen so that $\zeta \in [0.6, 0.8]$. We will discuss how to tune K_p and K_i in Section 4.2.

3.2 PIA performance enhancing techniques

Based on the core component of PIA, we now add three domain-specific enhancements to PIA to further improve its robustness and adaptiveness for ABR streaming.

Accelerating initial ramp-up. At the beginning of the video playback, the buffer level x_t can be much smaller than the

1536-1233 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2019.2929125, IEEE Transactions on Mobile Computing

target buffer level x_r. In this case, observe from Eq. (4) that u_t will be large. A large u_t will result in low video bitrate, R_t, and therefore a low quality at the beginning, which can adveresley impact initial user experience. To address this issue, we include a setpoint weighting parameter [12], $\beta \in (0, 1]$, into the control policy as

$$u_t = K_p(\beta x_r - x_t) + K_i \int_0^t (x_r - x_\tau)d\tau + \mathbf{1}(x_t - \Delta). \quad (11)$$

Note that β is only included in the proportional term; it does not affect the steady-state behavior of the control system [12]. When $\beta = 1$, the above control policy reduces to (4). When $\beta < 1$, it can lead to smaller u_t, and hence faster initial ramp-up in video bitrate. However, very small β can lead to aggressive choice of video bitrate, and hence increase the chance of buffer emptying, causing rebuffering at the beginning of the playback. We explore how to set β in Section 4.2. The system transfer function corresponding to the above control policy is

$$T(s) = \frac{x(s)}{x_r(s)} = \frac{\beta K_p s + K_i}{s^2 + K_p s + K_i} = \frac{\beta K_p s + K_i}{s^2 + 2\zeta\omega_n s + \omega_n^2}. \quad (12)$$

Note that both damping ratio, ζ, and natural frequency, ω_n remain the same as those in Eq. (10).

Minimizing bitrate fluctuations. The simple choice of R_t in Eq. (5) mainly tracks the network bandwidth. It, however, may lead to frequent and/or abrupt bitrate (and hence quality) changes, adversely impacting viewing quality. To address the above issue, we develop a regularized least squares (LS) formulation that considers both video bitrate and the changes in video bitrate to achieve a balance between both of these metrics. Specifically, it minimizes the following objective function

$$J(\ell_t) = \sum_{k=t}^{t+N-1} \left(u_k R(\ell_t) - \widehat{C}_k\right)^2 + \eta \left(R(\ell_t) - R(\ell_{t-1})\right)^2, \quad (13)$$

where ℓ_t and ℓ_{t-1} represent the tracks selected for chunks t (the current chunk) and $t-1$ (i.e., the previous chunk), respectively[1], u_k is the controller output for the k-th chunk, $\widehat{C}_k$ is the estimated link bandwidth for the k-th chunk, η is the weight factor for bitrate changes, and $R(\ell)$ represents the bitrate corresponding to track ℓ.

Let $\mathcal{L}$ denote the set of all possible track levels. For every $\ell_t \in \mathcal{L}$, the formulation in Eq. (13) considers a moving horizon of N chunks in the future (represented as the sum of N terms, one for each of the N future chunks). The first term in the sum aims to minimize the difference between $u_k R(\ell_t)$ and the estimated network bandwidth $\widehat{C}_k$ so as to maximize $R(\ell_t)$ (and hence quality) under the bandwidth constraint and the selected u_k. The second term aims to minimize the variability in bitrate for two adjacent chunks (i.e., the current and previous chunks) for a smooth viewing experience. The weight factor η can be set to reflect the relative importance of these two terms. We use $\eta = 1$ (i.e., equal importance) in the rest of the paper since maximizing the video quality and reducing quality variation are both important for user QoE. To reduce the number of video bitrate changes, in Eq. (13), we assume that the same track is chosen for the next N chunks (this assumption is used only for determining the track/bitrate for chunk t; the actual track/bitrate for the future chunks will be decided at later times, independent of the assumption made for the decision of chunk t). For Constant Bitrate (CBR) videos (which is the focus of this paper), the chunks in the same track have the same bitrate. Therefore, in Eq. (13), the bitrate of the next N chunks are all equal to $R(\ell_t)$. For Variable Bitrate (VBR) videos, the formulation can be modified in a straightforward manner[2]. Last, in Eq. (13), the control output, u_k, is updated according to the control policy in Eq. (11), based on the estimated buffer size, x_k, over the moving horizon. The estimated buffer size, x_k, is updated through Eq. (1) using $R(\ell_t)$ as the video bitrate and the estimated network bandwidth $\widehat{C}_k$.

The optimal solution of Eq. (13) is

$$\ell_t^* = \arg\min_{\ell_t \in \mathcal{L}} J(\ell_t), \quad (14)$$

where $\mathcal{L}$ denotes the set of all possible track levels. We can find ℓ_t^* easily by plugging in all possible values of ℓ_t, $\ell_t \in \mathcal{L}$, into Eq. (13), and select the value that provides the minimum objective function value in Eq. (13).

The complexity of the above formulation is as follows. For every $\ell_t \in \mathcal{L}$, obtaining $J(\ell_t)$ requires computation of N steps. Therefore, the total computational overhead is $O(|\mathcal{L}|N)$, significantly lower than the complexity of $O(|\mathcal{L}|^N)$ in [39].

Dealing with bitrate saturation. Following the control policy, u_t may become negative (e.g., when the current buffer level exceeds the target buffer level). In this case, solving Eq. (13) will lead R_t to be the minimum bitrate. During this time period, if we continue using the integral term, $I_{\text{out}} = K_i \int_0^t (x_r - x_\tau)d\tau$, u_t may remain negative for an extended period of time, causing R_t to stay at the minimum bitrate level for an extended period of time (so called system saturation [12]), and causing the buffer level to continue to grow.

Such system saturation is undesirable since it will cause the client to select low bitrate (and hence low quality) chunks that adversely impact user QoE, even though the network bandwidth is able to support higher quality streaming. To deal with the above scenario, we incorporate an anti-windup technique (to deal with integral windup, i.e., integral term accumulates a significant error) for negative u_t, or more specifically, when $u_t \leq \epsilon$, $0 < \epsilon \ll 1$. Many anti-windup techniques have been proposed in the literature [12]. We adopt a simple technique, which sets u_t to ϵ, chooses R_t as the maximum bitrate, and does not change I_{out} when $u_t \leq \epsilon$. This corresponds to turning off the integral control when u_t is below ϵ. We set ϵ to a small positive value, 10^{-10}, in the rest of the paper.

1. Here we slightly abuse notation by using t to represent the index of the chunk chosen at time t, and using $t-1$ to represent the index of the previous chunk.

2. For VBR videos, even the chunks in the same track have variable bitrate. In that case, we can modify (13) as follows. In the first term of (13), we replace $R(\ell_t)$ with $R_k(\ell_t)$ to reflect that the bitrate is time varying; in the second term, we replace $R(\ell_t)$ and $R(\ell_{t-1})$ with the average bitrate of the corresponding tracks, denoted as $\bar{R}(\ell_t)$ and $\bar{R}(\ell_{t-1})$, respectively, so that the penalty for quality variation is zero as long as the two adjacent chunks are in the same track.

1536-1233 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2019.2929125, IEEE Transactions on Mobile Computing

3.3 PIA parameter tuning

Three important parameters in PIA are K_p, K_i and β, where K_p and K_i determine the system behavior and β is used for faster initial ramp-up. For a given network setting (e.g., cellular networks), since β does not affect the steady-state behavior [12], we can first assume a fixed β (e.g., $\beta = 1$) and tune K_p and K_i to achieve a desirable steady-state behavior (i.e., jointly maximize the three metrics in QoE). Once K_p and K_i are fixed, we then tune β for the initial stage of the video playback. The values of K_p and K_i need to be tuned so that the resultant system behavior is compatible with the network setting. Taking cellular networks as an example, since the bandwidth is highly dynamic, it is reasonable to tune the system so that the buffer level does not fluctuate drastically. Otherwise, the buffer can suddenly become very low, making the system vulnerable to stalls. We describe this approach using a set of network traces from commercial cellular networks in Section 4.2.

3.4 Putting it all together

We now summarize the workflow of PIA depicted in Fig. 2. PIA takes the target buffer level x_r, the current buffer level x_t, and the estimated network bandwidth as input, and computes the selected track level ℓ_t^*, which is then fed into the Video Player Dynamics block to update the buffer level. PIA considers both present and past estimation errors, as well as incorporates all the three QoE metrics in the control loop. PIA also includes an anti-windup mechanism to deal with bitrate saturation, and a setpoint weighting technique to provide faster initial ramp-up.

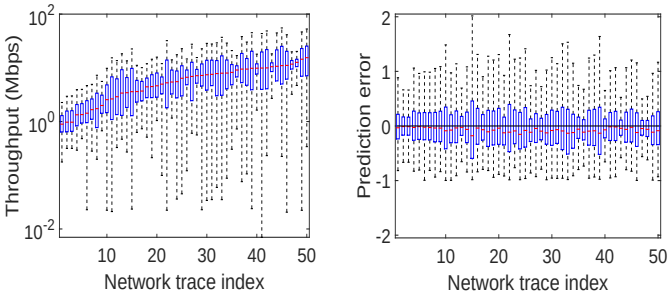
4 Performance Evaluation

In this section, we evaluate the performance of PIA using simulation; evaluation through real implementation on a video player is deferred to Section 6. Simulation allows us to evaluate a large set of parameters in a scalable manner, while real implementation provides insights under various system constraints. In both cases, the network conditions are driven by a set of traces captured from commercial LTE networks that allow reproducible runs, as well as apple-to-apple comparison of different schemes. We first describe the evaluation setup, choice of parameters for PIA, and then compare PIA against several state-of-the-art schemes.

4.1 Evaluation setup

Network bandwidth traces. We focus on LTE networks that dominate today's cellular access technology. For evaluation under realistic LTE network environments, we collected 50 network bandwidth traces from two large commercial LTE networks in the US. These traces were collected under a wide range of settings, including different times of day, different locations (in three U.S. states, CT, NJ and NY), and different movement speed (stationary, walking, local driving, and highway driving).

Each trace contains 30 minutes of one-second measurement of network bandwidth. The bandwidth was measured on a mobile device, as the throughput of a large file downloading from a well provisioned server to the device. Fig. 3(a) is a boxplot that shows the minimum, first quartile, median, third quartile, and maximum bandwidth of each trace, where the traces are sorted by the median bandwidth. We see that the network bandwidth is indeed highly dynamic. For some traces, the maximum bandwidth is tens of Mbps, while the minimum bandwidth is less than 10 Kbps.



(a) Throughput. (b) Prediction error.

Fig. 3: Characteristics of the network bandwidth traces that are used in performance evaluation.

These network traces, by capturing the bandwidth variability over time, accurately reflect the impact of lower level network characteristics (e.g., signal strength, loss, and RTT) on the network bandwidth perceived by an application. Using the network traces is sufficient when evaluating ABR schemes; there is no need to explicitly incorporate lower level network characteristics since ABR adaptation operates at the application level, using application-level estimation of the network bandwidth.

Video parameters. We use three video bitrate sets, all being Constant Bitrate (CBR) videos: $\mathcal{R}_1 = [0.35, 0.6, 1, 2, 3]$ Mbps, $\mathcal{R}_2 = [0.35, 0.6, 1, 2, 3, 5]$ Mbps and $\mathcal{R}_3 = [0.2, 0.4, 0.6, 1.2, 3.5, 5, 6.5, 8.5]$ Mbps. The first set is based on the reference for YouTube video bitrate levels (corresponding to 240p, 360p, 480p, 720p and 1080p respectively) [9]. The second set adds a higher bitrate level of 5 Mbps to the first set. The third set is based on Apple's HTTP Live Streaming standard [3]. For each bitrate set, we further consider three variants with chunk duration of 2, 4, and 8 s.

ABR Schemes. We compare PIA against four other schemes; in Section 6, we further compare the performance of PIA and BOLA [33], another state-of-the-art ABR scheme, using DASH implementation.

- RB: The bitrate is picked as the maximum possible bitrate that is below the predicted network bandwidth. This is a simple open-loop controller (see Section 2), serving as a baseline.
- BBA [20]: It is a state-of-the-art buffer based scheme. We use BBA-0, which is the BBA variant for CBR streaming, the focus of this paper. We set the lower and upper buffer thresholds as $\theta_{\text{low}} = 10$ s and $\theta_{\text{high}} = 60$ s, respectively, and empirically verified that the above thresholds work well on our dataset.
- MPC and RobustMPC [39]: Both are state-of-the-art ABR schemes based on model predictive control. RobustMPC is more conservative in estimating network bandwidth, and has been shown to outperform MPC [25] in more dynamic network settings (e.g.,

1536-1233 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2019.2929125, IEEE Transactions on Mobile Computing

cellular networks). Both schemes use a look-ahead horizon of 5 chunks (as suggested by the paper).

- PIA: Unless otherwise stated, the target buffer level $x_r = 60$ s that is compatible with the setting of BBA. In Section 4.3, we vary the target buffer level and explore its impact on performance. The look-ahead horizon is set to 5 chunks, i.e., $N = 5$ in Eq. (13).

For all the schemes, unless otherwise stated, the startup playback latency (i.e., the latency from when requesting the first chunk of the video to starting the playback of the video), δ, is set to 5, 10 or 15 s. For BBA and MPC, their parameters are either selected based on the original papers, or configured by us based on the properties of the videos (e.g., chunk duration and encoding rates) as justified above.

Network bandwidth prediction. For the schemes that require network bandwidth estimation, it is set as the harmonic mean of the network bandwidth of the past 20 s. Harmonic mean has been shown to be robust to measurement outliers [21]. Fig. 3(b) shows the boxplots of the bandwidth prediction errors (the difference of the predicted and actual bandwidths divided by the actual bandwidth) of the network bandwidth traces used in our evaluations. Each box in the plot corresponds to the distribution of all prediction instances within a particular trace. The prediction is at the beginning of every second. As shown, the median prediction error is 20% to 40%, highlighting the challenges of accurate bandwidth prediction in LTE networks. As we shall show later in Section 4.3, our PIA scheme is very robust to such levels of inaccuracy in network bandwidth estimation.

4.2 PIA: Choice of parameters

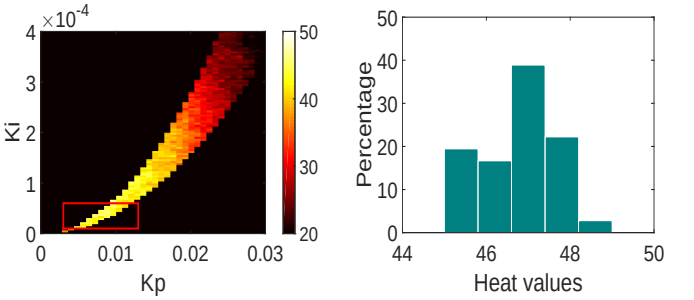
Following the methodology outlined in Section 3.3, we first tune K_p and K_i, and then tune β for PIA. One question we aim to answer is whether there exists a set of K_p and K_i values that works well in a wide range of settings. This is an important issue related to the practicality of PIA – because if the choice of K_p and K_i were too sensitive to the settings, then tuning and/or adapting them for different settings would require more effort.

4.2.1 *Tuning K_p and K_i*

We use a single combined performance metric when tuning K_p and K_i. This is because, while as described earlier, the QoE is affected by three metrics (average video bitrate, the amount of bitrate changes and rebuffering) jointly, comparing the QoE under different choices of K_p and K_i is much simpler when using a single combined metric. Currently there is no consensus in the field around the form of such a metric. One approach is using a weighted sum of the three metrics as in [39]. Specifically, for a video of M chunks,

$$\text{QoE} = \sum_{t=1}^{M} R_t - \mu \sum_{t=1}^{M-1} |R_{t+1} - R_t| - \lambda \sum_{t=1}^{M} S_t. \qquad (15)$$

where R_t is the bitrate of the t-th chunk and S_t is the amount of stalls for the t-th chunk, and μ and λ are weights that represent, respectively, the importance of the middle and last terms (i.e., bitrate changes and rebuffering) relative to the first term (i.e., average bitrate) in the sum. There is no well agreed-upon settings for μ and λ; we therefore vary μ and λ over multiple values to assess sensitivity.



(a) Heatmap. (b) Histogram of "heat" values.

Fig. 4: Region of K_p and K_i and the corresponding "heat" values in one setting (video bitrate set $\mathcal{R}_3$, chunk duration 2 s, video length 20 min, startup latency 10 s, $\mu = 1$, $\lambda = 8.5$ (which is the same as the maximum bitrate (in Mbps) in $\mathcal{R}_3$).

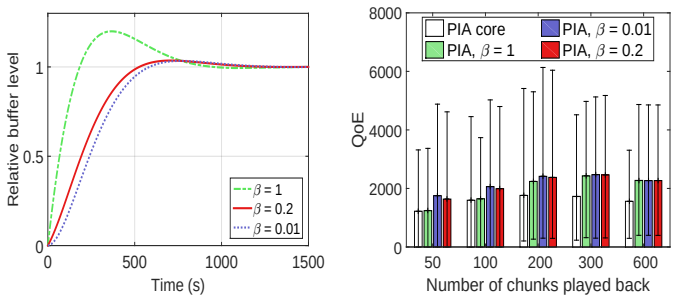
The trace-driven simulation allows us to consider a very wide range of settings by varying a number of parameters: the video bitrate set, video length, chunk size, startup latency, and μ and λ in (15). Specifically, the video bitrate set is either $\mathcal{R}_1$, $\mathcal{R}_2$, or $\mathcal{R}_3$ (see Section 4.1), video length is 5, 10 or 20 minutes, chunk duration is 2, 4 or 8 s, startup latency is 5, 10 or 15 s, and μ is 1 or 2, and λ is the maximum bitrate level of a video (e.g., 3 Mbps in $\mathcal{R}_1$) or twice as much. The choice of of μ and λ is based on the settings in [39]. In each setting (i.e., after fixing the above parameters), we consider each of the 50 network bandwidth traces individually. For the k-th network trace, we vary the values of K_p and K_i in a wide range to find a pair of K_p and K_i that maximizes the QoE (note that as described in Section 3, we only consider valid combinations of K_p and K_i values, i.e., those so that the damping ratio is in $[0.6, 0.8]$). Once the maximum QoE, denoted as Q_k^*, is determined, the QoE under each valid (K_p, K_i) pair is compared to Q_k^* to see whether it is within 90% of Q_k^*. Specifically, we define a binary function $f_k(K_p, K_i)$ for the k-th network bandwidth trace, where $f_k(K_p, K_i) = 1$ if the resulting QoE under K_p and K_i is within 90% of Q_k^*; otherwise, $f_k(K_p, K_i) = 0$. We then consider all the network bandwidth traces, and create a heat map with the "heat" for each valid pair of K_p and K_i values as $\sum_k f_k(K_p, K_i)$. Clearly, a larger "heat" value for a K_p and K_i pair means that it leads to good performance for more network bandwidth traces.

Fig. 4(a) shows an example heat map for one setting (details of the setting described in the caption of the figure). The black region represents invalid K_p and K_i pairs (i.e., those causing the damping ratio out of the desired range $[0.6, 0.8]$). For the valid K_p and K_i pairs, the "heat" value varies, with the highest values in the bottom left region, marked by the rectangle (brighter color represents higher "heat" values). Fig. 4(b) is the histogram of the "heat" values in the rectangle area (excluding those corresponding to invalid K_p and K_i pairs). It shows that majority of the values are close to 50 (i.e., the maximum "heat"), indicating that the valid K_p and K_i pairs marked by the rectangle provide good performance across almost all network traces.

We repeat the above procedure for all the settings, and

1536-1233 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2019.2929125, IEEE Transactions on Mobile Computing



(a) Step response. (b) QoE.

Fig. 5: Choosing β in PIA. In (b), the setting is the same as that used for Fig. 4.

find the following region of K_p and K_i values leads to good performance for all the settings

$$\begin{aligned} K_p &\in \left[1 \times 10^{-3}, 14 \times 10^{-3}\right] \\ K_i &\in \left[1 \times 10^{-5}, 6 \times 10^{-5}\right] \\ \text{s.t. } &\zeta\left(K_p, K_i\right) \in [0.6, 0.8]\,. \end{aligned} \tag{16}$$

Specifically, under the above range of values, the average "heat" for the different settings varies from 31 to 50, and the standard deviation varies from 0.25 to 3.45. The results in the rest of the paper use $K_p = 8.8 \times 10^{-3}$, approximately the middle of the range of K_p in (16), and $K_i = 3.6 \times 10^{-5}$ so that the damping ratio is $1/\sqrt{2}$, a widely recommended value for damping ratio [28], [26].

The finding that a set of K_p and K_i values works well under a wide range of settings is encouraging. Considering that the network traces were collected under a wide range of settings and that they exhibit significantly different characteristics (see Fig. 3), our results show that K_p and K_i can be tuned to accommodate the large variations among individual traces. The above results indicate that we can find a range of K_p and K_i values to make the system capable of dealing with the rapid bandwidth variations (one of the predominant characteristics of cellular networks), despite the differences across individual network conditions.

4.2.2 *Tuning β*

Once K_p and K_i are determined, we tune β for the initial stage of the video playback. Specifically, we set β to 0.01, 0.2, 0.4, 0.6, 0.8, and 1.0. Fig. 5(a) shows the step response of the control policy (only the results for $\beta = 0.01$, 0.2 and 1 are shown for better clarity). When $\beta = 1$, the buffer becomes full much more quickly than when $\beta = 0.2$ and 0.01. This is because, as explained in Section 3.2, lower bitrate tends to be selected when $\beta = 1$, causing the buffer to fill up more quickly.

To examine the three QoE metrics jointly, Fig. 5(b) plots the QoE when playing up to the i-th chunk of a video of 600 chunks when $\beta = 0.01$, 0.2 or 1. The results are averaged over all the network traces; the 95% confidence intervals are plotted in the figure as well. We see that β indeed affects the QoE for the initial playback, and $\beta = 1$ leads to lower QoE compared to $\beta = 0.01$ and 0.2. Further investigation reveals that $\beta = 0.01$ leads to more rebuffering than $\beta = 0.2$. The above results are for the setting used for Fig. 4. Results in other settings show similar trends. Since rebuffering has very detrimental effects on viewing quality, we use $\beta = 0.2$ in the rest of the paper.

Last, the results of PIA core (i.e., without the three enhancing techniques) are also shown in Fig. 5(b). We see that PIA core indeed leads to lower QoE compared to the full-fledged PIA, demonstrating the benefits of our three enhancing techniques.

4.3 Performance comparison

In the following, we first present the performance of PIA in the default setting, i.e., chunk duration of 2 s, video bitrate set $\mathcal{R}_2$, video length of 20 minutes, and startup latency of 10 s. After that, we evaluate the impact of the various parameters on the performance of PIA.

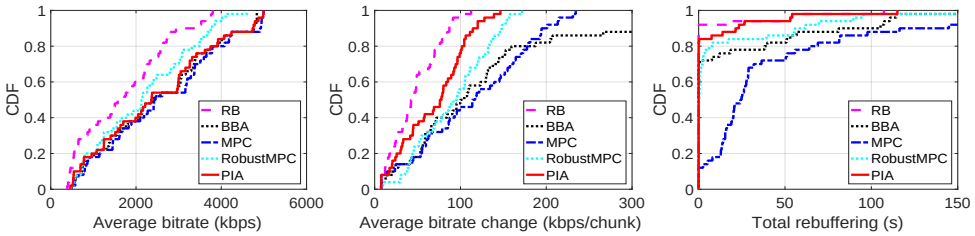
Fig. 6 plots the CDF of the three QoE metrics over all network bandwidth traces in the default setting. The performance of four schemes, RB, BBA, MPC, RobustMPC and PIA, are plotted. We see that, while the amount of bitrate change and rebuffering is low under RB, its average bitrate is significantly lower than those of the other schemes. PIA achieves comparable average bitrate as BBA and MPC, with significantly less bitrate changes and rebuffering. Specifically, the average bitrate of PIA is 98% and 96% of that of BBA and MPC, respectively, while the average amount of bitrate change is 49% and 40% lower, and the average amount of rebuffering is 68% and 85% lower than BBA and MPC, respectively. RobustMPC has significantly lower rebuffering than MPC, but its rebuffering is still higher than that under PIA. In addition, RobustMPC leads to significantly lower average bitrate than MPC, PIA and BBA.

Overall, *PIA achieves the best balance among the three conflicting QoE metrics*. As described earlier, the inferior performance of RB is because it uses an open-loop control without any feedback. The superior performance of PIA compared to BBA is because BBA implicitly uses one form of P-control (Section 2) that only takes the present error into account, while PIA considers both the present and past errors. PIA's approach of applying PID in an explicit and adaptive manner further facilitates the design and improves the performance. The performance of MPC is sensitive to network bandwidth estimation errors [39]: it solves a discrete optimization problem at each step; when network bandwidth estimation is inaccurate, the input to the optimization problem is correspondingly inaccurate, leading to suboptimal performance. RobustMPC leads to lower rebuffering than MPC due to its more conservative network bandwidth estimation. It, however, also leads to significantly lower bitrate choices.

To provide further insights, Fig. 7 plots the bitrate selection and the buffer level over time for BBA, MPC and PIA when using one network trace. For reference, it also plots the network bandwidth of the trace. We clearly see that BBA has significantly more bitrate changes; MPC tends to be more aggressive in choosing higher bitrates, which can lead to excessive rebuffering. The bitrate selection under PIA matches well with the network bandwidth without frequent bitrate changes. In terms of the buffer level/occupancy (i.e., the duration of the video content that has been brought in and has not yet been played back) shown in the bottom plot

1536-1233 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2019.2929125, IEEE Transactions on Mobile Computing



(a) (b) (c)

Fig. 6: Performance comparison in the default setting (chunk duration 2 s, video bitrate set $\mathcal{R}_2$, video length 20 minutes, startup latency 10 s).

in Fig. 7, the buffer level of MPC is lower than that of BBA and PIA due to its aggressive choice of bitrate; the buffer level of PIA reaches the target level of 60 s at around 300 s, and then stays around the target level; the buffer level of BBA is in between that of MPC and PIA.

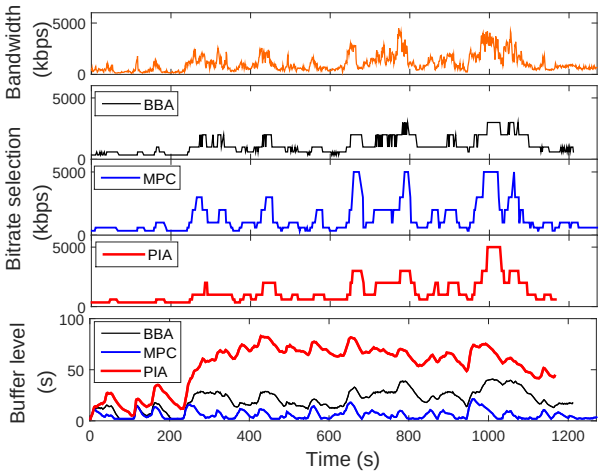
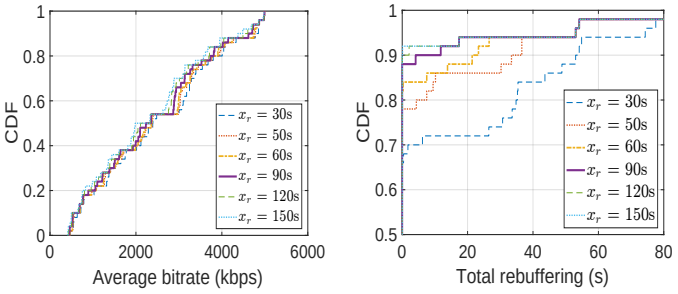


Fig. 7: Comparison of different schemes for one trace under the default setting (chunk duration 2 s, video bitrate set $\mathcal{R}_2$, video length 20 minutes, startup latency 10 s). The plot on buffer level shows the results until the end of the downloading.

Impact of target buffer level. We vary the target buffer level, x_r, from 30 to 200 seconds, and evaluate its impact on the various performance metrics. We observe that larger target buffer levels lead to lower bitrate choices (to reach the larger target buffer levels during buffer ramp-up periods, e.g., at the beginning of the playback or after stall events). A large target buffer level also has the drawback that it may lead to more waste of resources when a user abandons watching a video in the middle of the playback. Using a small target buffer level, however, can lead to higher rebuffering. Fig. 8 plots the average bitrate and the amount of rebuffering for different x_r values. We observe noticeably lower bitrate when $x_r = 150$ seconds, and noticeably higher rebuffering when $x_r = 30$ seconds. Setting x_r to 50 to 120 seconds leads to similarly good performance in all three performance metrics (the average bitrate change between two consecutive chunks across the x_r values is similar, and is not shown in the figure).



(a) Bitrate. (b) Rebuffering.

Fig. 8: Impact of the target buffer level on performance (video bitrate set $\mathcal{R}_2$, chunk duration 2 s, video length 20 minutes, startup latency 10 s).

Impact of video length. The above results are for video length of 20 mins. We vary the ending time of the video to investigate PIA's performance for shorter videos. When the ending time is larger than 5 mins (i.e., video length longer than 5 mins), we observe similar results as before; for much shorter videos, PIA has lower average bitrate compared to MPC (the average bitrate of PIA is 81% of MPC when the video length is 2 minutes, and 90% of MPC when the video length is 5 minutes), but still outperforms BBA and MPC on the other two metrics. We investigate how to improve the bitrate of PIA in the startup phase in Section 5.

Impact of video bitrate sets. Recall that the video bitrate set $\mathcal{R}_2$ has one higher bitrate level of 5 Mbps compared to $\mathcal{R}_1$. We further investigate two more video bitrate sets $\mathcal{R}_4 = [0.2, 0.35, 0.6, 1, 2, 3]$ Mbps, which has one lower bitrate of 0.2 Mbps compared to $\mathcal{R}_1$; and $\mathcal{R}_5 = [0.2, 0.35, 0.6, 1, 2, 3, 5]$ Mbps, which has one lower and one higher video bitrate levels (of 0.2 and 5 Mbps) compared to $\mathcal{R}_1$. We observe consistent trend for all the schemes under the above three video bitrate sets. Comparing the results under $\mathcal{R}_1$ and $\mathcal{R}_2$, we see that adding one higher bitrate level leads to higher average video bitrate, more bitrate changes, and more rebuffering; comparing the results under $\mathcal{R}_1$ and $\mathcal{R}_4$, we see that adding one lower bitrate level maintains the average video bitrate while reducing bitrate changes and rebuffering; comparing $\mathcal{R}_1$ and $\mathcal{R}_5$, we see that adding both one lower and higher bitrate levels increases the average video bitrate and bitrate changes, while reducing the rebuffering. In general, adding more bitrate levels helps improve at least one of the three metrics. Across the settings, PIA has the lowest bitrate

1536-1233 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2019.2929125, IEEE Transactions on Mobile Computing

switches, the lowest rebuffering, and similar average bitrate compared to BBA and MPC.

Impact of video chunk duration. We vary the video chunk duration by setting it to 2, 4, or 8 s. We found that, for all chunk durations, PIA consistently outperforms MPC and BBA in balancing the tradeoffs incurred by the three metrics. For example, for chunk duration of 8 s, PIA's average playback bitrate differs from BBA and MPC only by 0.1% and 3.2%, respectively, while PIA reduces the rebuffering duration by 67% and 68% compared to those of BBA and MPC, respectively.

Buffer occupancy and impact of maximum buffer size. In the evaluations so far, we assume that all data downloaded ahead of the current playback point is stored in the client buffer until it is played back. Fig. 9 plots the distribution of the buffer occupancy (i.e., all the chunks brought in that have not yet been played back) under PIA in the default setting across all the network traces, where we record the buffer occupancy after downloading each chunk. We observe that 52% of the time, the buffer occupancy is below the target buffer size (60 s), 85% of the time it is below 100 s, and 95% of the time it is below 200 s. The above results indicate that, while there is no explicit constraint on buffer size, the amount of video stored at the client under PIA is not large (200 s of video corresponds to at most 125 MB even if we consider buffering the maximum bitrate track in $\mathcal{R}_2$).

In practice, a player may impose a maximum buffer size, $B_{\max}$, so that the amount of video downloaded before its playback time does not exceed this limit. We explore a simple strategy for this case. Specifically, the client stops downloading video when the buffer is full, and resumes downloading when there is space in the buffer. We set $B_{\max} = 90$, 120, 150, 180 or 210 s, motivated by the practice of commercial players, which set the maximum buffer limit to tens to hundreds of seconds [6], [7], [20], [38]. In the following, in the interest of space, we only report the results when $B_{\max} = 90$ or 210 s.

When imposing the maximum buffer limit, not surprisingly, the average bitrate of all the schemes is reduced. On the other hand, the reduction under PIA when $B_{\max} = 90$ s is only slightly more than that $B_{\max} = 200$ s (consistent with the observation in Fig. 9 that 82% and 96% of the time, the buffer size is less than 90 s and 210 s, respectively). PIA still achieves the best balance among the three conflicting performance metrics. Fig. 10 plots the distribution of the tracks selected by PIA under the default setting across all the network traces (the video bitrate set $\mathcal{R}_2$ has six tracks with increasing bitrate). For comparison, the bitrate choices when there is no maximum buffer limit are also plotted in the figure. We observe that, when imposing a maximum buffer limit, the probability of choosing the highest track is reduced (particularly when $B_{\max} = 90$ s), while the probability of choosing the lower tracks is increased. This is because, with the maximum buffer limit, the amount of video in the buffer is limited (and hence tends to be less than the amount when there is no buffer limit); the player thus has less cushion for preventing buffer underruns, and is less likely to choose the highest track (which takes longer to download compared to lower tracks, causing more buffer drainage). Fig. 11 shows an example. We see from 150 to 260 seconds, the buffer level is significantly lower under $B_{\max} = 90$ s than that when $B_{\max} = 210$ s. As a result, the player chooses lower tracks for the former, while choosing the highest track for the latter.

In the above, we use a simple strategy that stops downloading when the buffer is full, which does not fully leverage the network bandwidth. This simple strategy can be improved in two directions. The first direction is segment replacement, i.e., the player can leverage the bandwidth to preempt some earlier downloaded chunks that are of lower bitrate (i.e., replace them with chunks with higher bitrate and hence quality). While segment replacement has been used in commercial systems, the performance of existing commercial implementation is not satisfactory [38]. The key decision of segment replacement is to select which chunks to preempt and which bitrate levels to replace them with, which we will investigate in future studies. The second direction is designing bitrate selection strategies to download higher bitrate chunks proactively, instead of filling in the buffer with lower bitrate chunks. Further exploration along this direction is also left as future work.

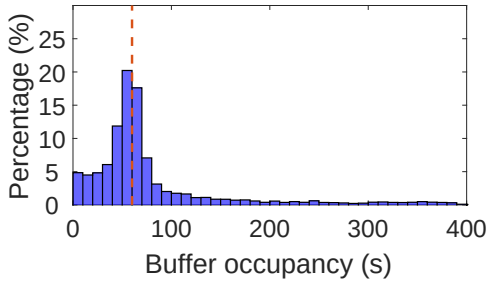


Fig. 9: The distribution of buffer occupancy of PIA (under the default setting, i.e., chunk duration 2 s, video bitrate set $\mathcal{R}_2$, video length 20 minutes, startup latency 10 s).

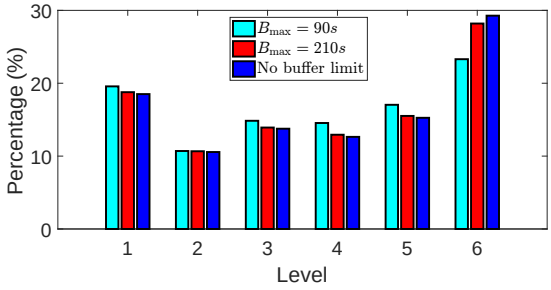


Fig. 10: Impact of the maximum buffer size on the bitrate adaptation of PIA (under the default setting, i.e., chunk duration 2 s, video bitrate set $\mathcal{R}_2$, video length 20 minutes, startup latency 10 s).

Impact of startup delay. So far we have set the startup delay to be 10 s, i.e., waiting for 10 s after requesting the first chunk of the video before starting the playback of the video. We next vary the startup delay. Specifically, we assume the chunk duration is 2 or 6 s and the startup delay is 6 or 12 s, equivalent to 3 or 6 chunks for chunk duration of 2 s, and 1 or 2 chunks for chunk duration of 6 s. Our results show that the amount of rebuffering depends more on the number of chunks that is downloaded during the startup delay, and

1536-1233 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2019.2929125, IEEE Transactions on Mobile Computing

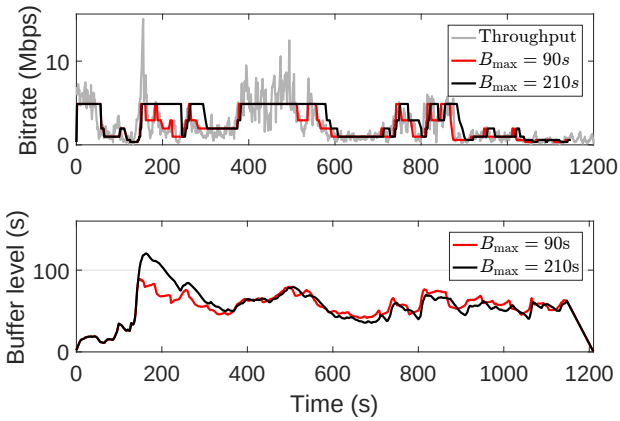


Fig. 11: An example that illustrates the impact of the maximum buffer size on the bitrate choice of PIA (under the default setting, i.e., chunk duration 2 s, video bitrate set $\mathcal{R}_2$, video length 20 minutes, startup latency 10 s).

is less sensitive to the length of the startup delay. For the same amount of startup delay (in seconds), using a smaller chunk duration leads to less rebuffering than using a larger chunk duration. In addition, having at least 2 or 3 chunks in buffer before playback starts leads to significantly less rebuffering than having a single chunk. The above findings are consistent with those in [38].

4.4 Computational overhead

As described earlier, the computational overhead of PIA is much lower than that of MPC: for $|\mathcal{L}|$ bitrate levels and horizon N, the complexity of MPC is $O(|\mathcal{L}|^N)$, while the complexity of PIA is $O(|\mathcal{L}|N)$. Table 2 compares the average execution time of running MPC, BBA and PIA for a 600-chunk video (2-second chunk with bitrate set $\mathcal{R}_2$) on a commodity laptop with Intel i5 2.6 GHz CPU and 16 GB RAM. The CPU time for PIA is 0.17 s, comparable to that of BBA. The CPU time for MPC is more than 200 times higher than that for PIA.

TABLE 2: Comparison of computational overhead.

	MPC	BBA	PIA
CPU time (s)	36.04	0.08	0.17

5 Improving Startup Performance

As shown in Section 4, the average bitrate under PIA in the startup phase (up to 5 minutes into the playback) can be 20% lower than that of MPC. We next present a variant of PIA, PIA-E (PIA Enhanced), that improves PIA's startup performance. The bitrate selection during the startup phase needs to balance two aspects: the quality of the video and the accumulation of the video content. The quality of the early part of the video (the chunks downloaded during the startup phase) is important, since low quality may prompt a user to stop watching the video. On the other hand, the startup phase also plays an important role in building up the video content in the buffer to reduce the likelihood of rebuffering in the future; choosing lower bitrate chunks helps to accumulate more content in the buffer. A good strategy for the startup phase thus needs to account for these two conflicting aspects.

Our goal of designing PIA-E is to increase the bitrate for the early part of the video, without increasing the amount of rebuffering during the later part of the playback. We achieve this goal by dynamically adjusting a selective set of parameters over time. Specifically, observe from Eq. (13) that the bitrate for the early part of the video can be increased by decreasing the controller output, u_t, which can be achieved by adjusting the parameters, K_p, β, x_r, and K_i, based on Eq. (11). Let $K_p(t)$, $\beta(t)$, $x_r(t)$, and $K_i(t)$ denote the values of these parameters at time t. We next describe one design of PIA-E that adjusts $K_p(t)$ and $x_r(t)$ overtime. At the end of this section, we discuss other design options.

Adjusting $K_p(t)$ and $x_r(t)$. This design keeps $\beta(t) = \beta$ and $K_i(t) = K_i$, and dynamically adjusts $K_p(t)$ and $x_r(t)$ over time, starting with initial values that are selected for the startup phase, and ending with the values that have been tuned for the steady state. Specifically, we set $K_p(t)$ as a decreasing function of t that decreases from an initial value αK_p, $\alpha > 1$, to K_p over a time interval, and set $x_r(t)$ as an increasing function of t that increases from a small value to x_r over a time interval. As a result, the first term in Eq. (11) is a non-negligible negative value at the beginning of the playback, leading to lower u_t and hence larger bitrate choices. Specifically, let τ denote the time interval. We set

$$K_p(t) = \begin{cases} \alpha K_p - \frac{(\alpha K_p - K_p)t}{\tau}, & 0 \le t \le \tau \\ K_p, & t > \tau \end{cases} \quad (17)$$

Similarly, we set $x_r(t)$ as a linear function with the minimum value of 2Δ initially (which is twice of the chunk duration; the target buffer level cannot be zero, and the value 2Δ is chosen empirically).

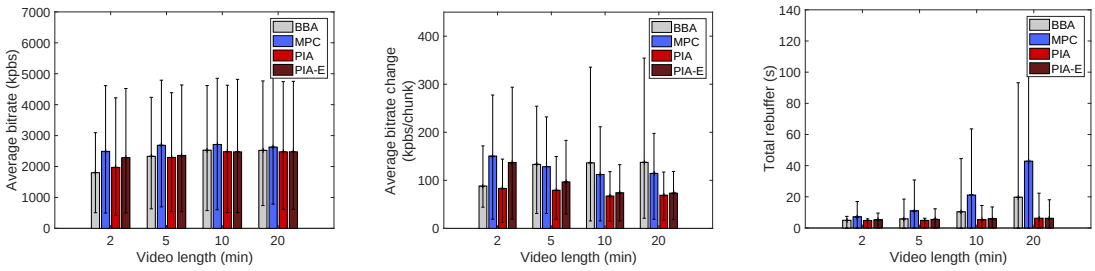
$$x_r(t) = \begin{cases} \max\left(2\Delta, \frac{x_r t}{\tau}\right), & 0 \le t \le \tau \\ x_r, & t > \tau \end{cases} \quad (18)$$

In the above design, $x_r(t)$ follows a ramp change. For such a ramp change, the system in Eq. (11) can track x_r with the error proportional to the inverse of the so-called velocity constant K_v in the steady state [34]. From the transfer function Eq. (12), we derive K_v as $1/K_v = K_p(1-\beta)/K_i$. Evidently, when $\beta = 1$, the steady sate error is zero, and is non-zero when $\beta < 1$. Therefore, we choose $\beta = 1$ for the above choice of $x_r(t)$.

We explored the choice of α (as 2 or 4) and τ (as 2, 5, or 10 minutes). Fig. 12 plots the performance of PIA-E versus other schemes, $\alpha = 4$ and $\tau = 5$ minutes (which achieves the best tradeoff). The results when playing up to 2, 5, 10 or 20 minutes of the video are shown in the figure. We observe that for the first 2 minutes, PIA-E indeed leads to significantly higher bitrate: the average bitrate is 14% higher than that of PIA, 27% higher than that of BBA and only 8% lower than that of MPC. On the other hand, the average bitrate change for PIA-E is higher than that of PIA, but still 9% less compared to MPC on average. The amount of rebuffering of PIA-E is similar to that of PIA. When the ending time is larger (i.e., 5, 10 or 20 minutes), the performance of PIA-E is very close to PIA, confirming that the more aggressive bitrate choice of PIA-E for the early

1536-1233 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2019.2929125, IEEE Transactions on Mobile Computing



(a) (b) (c)

Fig. 12: Performance of PIA-E versus other schemes (chunk duration 2 s, video bitrate set $\mathcal{R}_2$, startup latency 10 s).

part of the video does not adversely affect the performance for the later part of the video.

Other designs. In the above design, we set $K_p(t)$ and $x_r(t)$ as piece-wise linear functions. We also explored setting them as exponential functions, and observed that the resultant design can achieve similar performance. In addition, we have explored dynamically adjusting $K_p(t)$ and $\beta(t)$, and observed similar performance[3]. Last, we explored dynamically adjusting a single parameter (e.g., $x_r(t)$), and observed that the resultant performance is inferior to that when adjusting two parameters simultaneously.

6 Evaluation using DASH Implementation

We have implemented PIA and PIA-E using `dash.js` (version 2.2.0) [6], a production quality open source framework provided by DASH Industry Forum [31]. To evaluate the performance of PIA and PIA-E under realistic network settings, we create an emulation environment as follows. We use a Linux machine running Apache httpd as the video server and a Windows laptop (with i7-5700HQ 3.50 GHz CPU and 16 GB memory) as the client. The server and client are connected by a 100 Mbps Ethernet link. We apply the Linux `tc` tool at the server to emulate the downlink bandwidth using the LTE bandwidth traces that we collected. The latency between the server and client is set to 70 ms as it is the average latency reported by OpenSignal's latency report. The client uses Chrome browser to run `dash.js`.

Implementation of PIA and PIA-E in `dash.js`. We implemented two new ABR streaming rules (each about 400 LoC) in `dash.js` to realize PIA and PIA-E. The parameters used for PIA and PIA-E are as those used in Sections 4 and 5, respectively. The bandwidth estimation requires knowing the past throughput per second (we use the harmonic mean of the throughput of the past 20 seconds). We therefore further developed a bandwidth estimation module that uses progress events in `dash.js` to obtain the past throughput per second. Specifically, when receiving one progress event, we calculate the number of bytes downloaded since the last event.

Videos. We use three CBR videos encoded using `FFmpeg` [18]. The first video is a music show, around 15 minutes long, with five tracks of resolution 240p, 360p, 480p, 720p and 1080p, respectively (the bitrate of the tracks varies from 0.36 to 3.09 Mbps); the chunk duration is 2 s. The other two videos are Big Buck Bunny (BBB) and Tears of Steel (ToS), both around 10 minutes long, with chunk duration of 5 s. ToS has five tracks, with the same resolutions and slightly lower bitrate (0.32 to 2.84 Mbps across the tracks) compared to the music show. BBB has one additional lower track (144p resolution) and the bitrate of the six tracks varies from 0.11 to 2.20 Mbps.

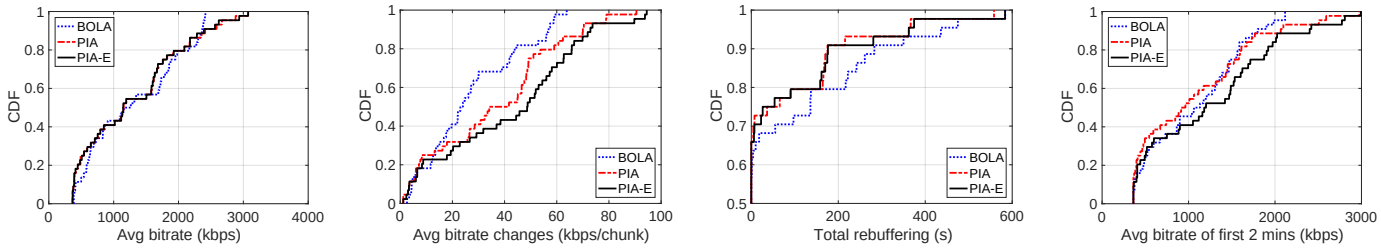
Comparing simulation and implementation results. We compare the results obtained from our `dash.js` implementation with those from the simulations, and confirm that the results are consistent. Specifically, for the music show, under PIA, 90% of the relative differences between implementation and simulation results are within 6.5% for average bitrate; for bitrate changes and rebuffering duration, 90% of the absolute differences are within 15 Kbps/chunk and 3 s, respectively. The results for PIA-E are similar. The performance differences between implementation and simulation results are due to multiple reasons. First, the simulation assumes a perfect CBR video where all the chunks in the same track have exactly the same bitrate; the video used in the implementation, while encoded as CBR, has bitrate variability across the chunks. Second, the implementation results are affected by various practical factors (e.g., server response time, client computational and response time, network RTT and TCP window size), which are not accounted for in the simulations.

Performance comparison. We compare the performance of PIA and PIA-E with a state-of-the-art scheme, BOLA [33], that was implemented in `dash.js` version 2.2.0. BOLA selects the bitrate to maximize a utility function considering both rebuffering and delivered bitrate. The upper threshold of BOLA is set to 60 s (the default value is 30 s) to be compatible with the target buffer level of 60 s in PIA. Fig. 13 plots the QoE metrics of PIA, PIA-E and BOLA for the music show video. We observe that PIA-E achieves higher average bitrate than PIA and BOLA for the first 2 minutes of the video. For the entire video, PIA-E leads to comparable performance as PIA: PIA-E leads to slightly more rebuffering, and similar average bitrate and bitrate changes. BOLA has higher rebuffering and lower bitrate changes than PIA and PIA-E. Fig. 14 plots the results for the BBB video. We observe similar results as those for the music show except that (i) the average bitrate change per chunk for all the three schemes is larger, which is due to the larger chunk duration in BBB (5

3. Note that in Eq. (11), since u_t is affected by the product of β and x_r, it is sufficient to vary either $\beta(t)$ or $x_r(t)$; there is no need to vary them simultaneously.

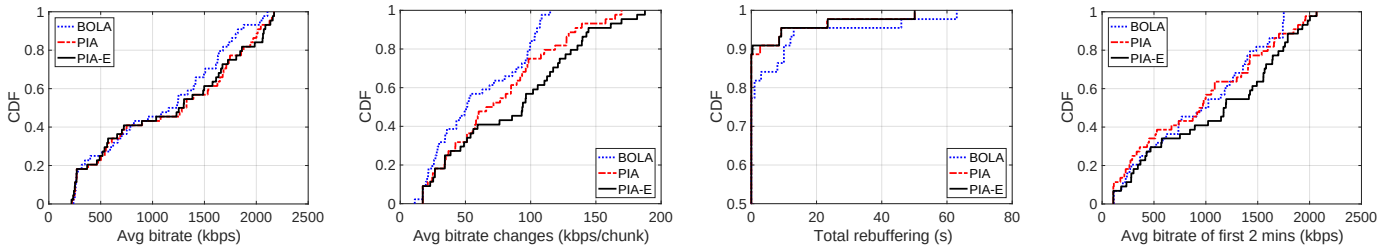
1536-1233 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2019.2929125, IEEE Transactions on Mobile Computing



(a) (b) (c) (d)

Fig. 13: Performance comparison in DASH (music show, chunk duration 2 s, video length 15 minutes, startup latency 10 s).



(a) (b) (c) (d)

Fig. 14: Performance comparison in DASH (BBB, chunk duration 5 s, video length 10 minutes, startup latency 10 s).

versus 2 s), and (ii) the amount of rebuffering is significantly lower due to the much lower bitrate of the lowest track in BBB (0.11 versus 0.36 Mbps). For ToS, the average bitrate change per chunk is similar to that for BBB due to the same chunk duration of 5 s of these two videos, and the amount of rebuffering is similar to that of the music show due to the similar bitrate of the lowest track of these two videos (the figures are omitted in the interest of space).

Runtime overhead. We record the CPU execution time of the ABR logic in the JavaScript code when a video is being played. The execution time of the default ABR logic in the `dash.js` player is 1.2 s for the entire 15-min music show video. The execution time of our PIA logic is 1.9 s, only slightly larger than that of the default ABR logic. For PIA-E, the execution time is 2.0 s (compared to PIA, it has additional calculation of $K_p(t)$ and $\beta(t)$). The execution time PIA-E and for the other two videos is similar. The above results indicate that PIA incurs very small runtime overhead, despite its non-trivial decision process shown in Fig. 2.

7 Related Work

The existing studies closest to ours are the several studies that use PID for adaptive video streaming. The studies [15], [16] directly use the standard PID controller, without adapting it to accommodate the special requirements of ABR streaming. The study in [33] shows that directly using PID control leads to worse performance than BOLA, while our proposed schemes (PIA and PIA-E) carefully adapt PID control for ABR streaming and outperform BOLA. The studies in [36], [39] conclude that PID control is not suitable for ABR streaming, and develop other control based approaches. Our study takes a fresh look at using PID control for ABR streaming and shows a somewhat surprising high-level finding: *by applying PID control in an explicit and adaptive manner, our ABR streaming algorithms substantially outperform the state-of-the-art video streaming schemes.* MPC [39] uses another branch of control theory, model predicative control, to solve a QoE optimization problem. It, however, requires accurate future network bandwidth estimation and incurs significant computation overhead. As we have shown in Section 4, our proposed PIA scheme incurs much lower computational overhead and achieves a significantly better balance among the three performance metrics than MPC; PIA also outperforms RobustMPC [39].

BBA [20] selects video bitrates purely based on buffer occupancy. However, as we have discussed, BBA essentially uses a P-controller and does not consider the integral part, which affects its performance. BOLA [33] and an improved version BOLA-E [32] select the bitrate based on maximizing a utility function, considering both rebuffering and video quality. We showed in Section 6 that BOLA leads to significantly more rebuffering than our approach. Another type of approach uses machine learning (e.g., reinforcement learning) to "learn" an ABR scheme from data. The study in [14] proposes a tabular Q-Learning based reinforcement learning approach for ABR streaming. This approach, however, does not scale to large state/action spaces. A more recent scheme, Pensieve [25], addresses the scalability issue using reinforcement learning based on neural networks. In [25], Pensieve is realized as a server-side ABR algorithm – the client feeds back information to the server and the server makes the decisions on bitrate choices. For a given ABR algorithm, Oboe [11] pre-computes the best possible parameters for different network conditions and dynamically adapts these parameters at run-time. The study in [29] characterizes the

1536-1233 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2019.2929125, IEEE Transactions on Mobile Computing

VBR encoding characteristics, proposes design principles for VBR-based ABR streaming and a concrete scheme, CAVA, that instantiates these design principles.

FESTIVE [21] and PANDA [23] both consider scenarios with multiple video flows. piStream [37] is designed specifically for LTE networks and uses physical layer information to improve the bandwidth prediction. CS2P [35] uses a data-driven approach to improve the bandwidth prediction for ABR streaming. The study in [13] proposes a network-based scheduling framework for adaptive video delivery over cellular networks. The work in [40] demonstrates the benefits of knowing the network bandwidth on the performance of ABR streaming. None of them focuses specifically on leveraging control theory for improving the video streaming QoE.

Last, several studies evaluate the performance of the rate adaptation schemes in commercial players [10], [19], which have motivated later schemes. A recent work [38] conducts a detailed measurement study of a wide range of popular HTTP Adaptive Streaming services over cellular networks to understand the design and performance of these services.

8 Conclusion and Future work

In this paper, we have explored using feedback control theory for creating the adaptation logic component critical to ABR video streaming. By strategically applying a PID controller in an explicit and adaptive manner, PIA considerably outperforms the state-of-the-art video streaming schemes in balancing the complex tradeoffs associated with the key QoE metrics, as demonstrated by extensive evaluations. PIA is also lightweight and easy to deploy. We believe the same high-level principle can be applied to other multimedia applications with content quality adaptation, such as live video conferencing. In our future work, we plan to port our implementation to mobile devices to better assess PIA's performance in the wild, and also conduct deeper exploration of other network and streaming settings (e.g., live streaming).

References

[1] Adobe HTTP Dynamic Streaming. http://goo.gl/fgPXdH.

[2] Apple's HTTP Live Streaming. https://developer.apple.com/streaming.

[3] Best Practices for Creating and Deploying HTTP Live Streaming Media for Apple Devices (Apple Technical Note TN2224). https://goo.gl/xTPwJa.

[4] Cisco VNI: Global Mobile Data Traffic Forecast Update, 2015-2020. https://goo.gl/TNfv8c.

[5] Ctrix Mobile Analytics Report, 2014. https://www.citrix.com/products/bytemobile-adaptive-traffic-management/tech-info.html#reports.

[6] Dash-Industry-Forum/dash.js. https://github.com/Dash-Industry-Forum/dash.js.

[7] ExoPlayer. https://github.com/google/ExoPlayer.

[8] Microsoft Smooth Streaming Protocol. https://goo.gl/Z8fDX6.

[9] YouTube live encoder settings, bitrates and resolutions. https://support.google.com/youtube/answer/2853702?hl=en.

[10] S. Akhshabi, A. C. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *Proc. of ACM MMSys*, 2011.

[11] Z. Akhtar, Y. S. Nam, R. Govindan, S. Rao, J. Chen, E. Katz-Bassett, B. Ribeiro, J. Zhan, and H. Zhang. Oboe: auto-tuning video abr algorithms to network conditions. In *Proc. of ACM SIGCOMM*, 2018.

[12] K. J. Åström and R. M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2008.

[13] J. Chen, R. Mahindra, M. A. Khojastepour, S. Rangarajan, and M. Chiang. A scheduling framework for adaptive video delivery over cellular networks. In *Proc. of ACM MobiCom*, 2013.

[14] F. Chiariotti, S. D'Aronco, L. Toni, and P. Frossard. Online learning adaptation strategy for dash clients. In *ACM MMSys*, 2016.

[15] L. D. Cicco, S. Mascolo, and V. Palmisano. Feedback control for adaptive live video streaming. In *Proc. of ACM MMSys*, 2011.

[16] L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo. ELASTIC: a client-side controller for dynamic adaptive streaming over HTTP (DASH). In *Proc. of Packet Video Workshop (PV)*. IEEE, 2013.

[17] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the impact of video quality on user engagement. In *Proc. of ACM SIGCOMM*, 2011.

[18] FFmpeg. FFmpeg Project. https://www.ffmpeg.org/, 2017.

[19] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari. Confused, timid, and unstable: picking a video streaming rate is hard. In *Proc. of IMC*. ACM, 2012.

[20] 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*, 2014.

[21] J. Jiang, V. Sekar, and H. Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with FESTIVE. In *Proc. of ACM CoNEXT*, pages 97–108, 2012.

[22] S. S. Krishnan and R. K. Sitaraman. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. *IEEE/ACM Transactions on Networking*, 21(6):2001–2014, 2013.

[23] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. Begen, and D. Oran. Probe and adapt: Rate adaptation for HTTP video streaming at scale. *IEEE JSAC*, 32(4):719–733, 2014.

[24] Y. Liu, S. Dey, F. Ulupinar, M. Luby, and Y. Mao. Deriving and validating user experience model for DASH video streaming. *IEEE Transactions on Broadcasting*, 61(4), December 2015.

[25] H. Mao, N. Ravi, and A. Mohammad. Neural adaptive video streaming with pensieve. In *Proc. of ACM SIGCOMM*, 2017.

[26] N. H. McClamroch. *State Models of Dynamic Systems: A Case Study Approach*. Springer, 1980.

[27] P. Ni, R. Eg, A. Eichhorn, C. Griwodz, and P. Halvorsen. Flicker effects in adaptive video streaming to handheld devices. In *Proc. of ACM Multimedia*, 2011.

[28] K. Ogata. *Modern Control Engineering*. Prentice Hall, 2010.

[29] Y. Qin, S. Hao, K. R. Pattipati, F. Qian, S. Sen, B. Wang, and C. Yue. Abr streaming of vbr-encoded videos: characterization, challenges, and solutions. In *Proc. of ACM CoNEXT*. ACM, 2018.

[30] Y. Qin, R. Jin, S. Hao, K. R. Pattipati, F. Qian, S. Sen, B. Wang, and C. Yue. A control theoretic approach to abr video streaming: A fresh look at PID-based rate adaptation. In *INFOCOM*, 2017.

[31] I. Sodagar. The MPEG-DASH standard for multimedia streaming over the internet. *IEEE Multimedia*, (4):62–67, 2011.

[32] K. Spiteri, R. Sitaraman, and D. Sparacio. From theory to practice: Improving bitrate adaptation in the dash reference player. In *Proc. of ACM MMSys*. ACM, 2018.

[33] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. BOLA: near-optimal bitrate adaptation for online videos. In *INFOCOM*. IEEE, 2016.

[34] R. T. Stefani, B. Shahian, and G. Hostetter. *Design of Feedback Control Systems*. Oxford University Press, 4th edition, 2002.

[35] 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 *SIGCOMM*. ACM, 2016.

[36] G. Tian and Y. Liu. Towards agile and smooth video adaptation in dynamic HTTP streaming. In *Proc. of ACM CoNEXT*, 2012.

[37] X. Xie, X. Zhang, S. Kumar, and L. E. Li. piStream: physical layer informed adaptive video streaming over LTE. In *Proc. of ACM MobiCom*, 2015.

[38] S. Xu, Z. M. Mao, S. Sen, and Y. Jia. Dissecting VOD services for cellular: Performance, root causes and best practices. In *Proc. of IMC*, 2017.

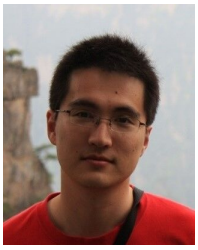
[39] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *Proc. of ACM SIGCOMM*, pages 325–338, 2015.

1536-1233 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2019.2929125, IEEE Transactions on Mobile Computing

[40] X. K. Zou, J. Erman, V. Gopalakrishnan, E. Halepovic, R. Jana, X. Jin, J. Rexford, and R. K. Sinha. Can accurate predictions improve video streaming in cellular networks? In *Proc. of HotMobile*, 2015.

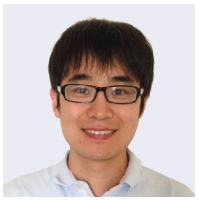
Yanyuan Qin received the B.S. degree (2011) in Automation from the Nanjing University of Aeronautics and Astronautics, in 2011, and the M.S. degree (2014) in Control Science and Engineering from Shanghai Jiao Tong University. He is currently pursuing the Ph.D. degree with the Computer Science and Engineering Department, University of Connecticut. His research interests are in video streaming and SDN.



Ruofan Jin received his B.S. and M.S. degree in Computer Science and Engineering from Beihang University in 2007 and 2010. He then received his Ph.D. degree at University of Connecticut in 2015. His research interests are in the areas of wireless networks and performance optimization.

Shuai Hao received the B.Comp. (2005) and M.S. (2006) degrees from the National University of Singapore, Singapore, and the Ph.D. (2014) degree from the University of Southern California, Los Angeles, CA, USA, all in Computer Science. He is currently a researcher at AT&T Labs, NJ. His research interests include networked systems, mobile computing, and video streaming.

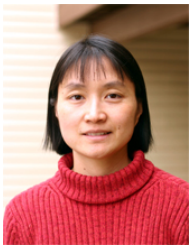
Krishna R. Pattipati (S'77-M'80-SM'91-F'95) is currently the Board of Trustees Distinguished Professor and the UTC Professor of Systems engineering in the Department of Electrical & Computer Engineering at UConn. He was elected a Fellow of the IEEE in 1995 for his contributions to discrete-optimization algorithms for large-scale systems and team decision-making. He served as Editor-in-Chief of the IEEE Transactions on SMC-Cybernetics during 1998-2001.



Feng Qian received the bachelor's degree from Shanghai Jiao Tong University, and the Ph.D. degree from the University of Michigan. He is currently an assistant professor in the Computer Science and Engineering Department at University of Minnesota. His research interests cover the broad areas of mobile systems, VR/AR, computer networking, and system security.

Subhabrata Sen (M'01-SM'14-F'16) received a Bachelor of Engineering (First Class with Honors) degree in Computer Science (1992) from Jadavpur University, India, and M.S. and Ph.D. degrees in Computer Science from the University of Massachusetts, Amherst, USA, in 1997 and 2001, respectively. Dr. Sen is currently a Lead Scientist at AT&T Labs–Research and an IEEE Fellow. He is a recipient of the AT&T Science and Technology Medal, the AT&T Labs President Excellence award, and the AT&T CTO Innovation Award. He is a co-inventor of the widely used open-source Application Resource Optimizer (ARO) tool for analyzing cellular friendliness of mobile app designs that earned top industry honors including the American Business Awards' Tech innovation of the Year Gold Stevie award, 2013. He is a past editor of the IEEE/ACM Transactions on Networking.

Chaoqun Yue is currently a Ph.D. student in the Computer Science & Engineering Department at the University of Connecticut. He received his M.S. degree in Computer Science from Shanghai Jiao Tong University, China in 2014. His research interests are in the areas of wireless networks and wireless sensing applications.



Bing Wang received a B.S. degree in Computer Science from the Nanjing University of Science & Technology, China, in 1994, and the MS degree in computer engineering from the Institute of Computing Technology, Chinese Academy of Sciences, in 1997. She then received M.S. degrees in computer science and applied mathematics, and a PhD degree in computer science from the University of Massachusetts, Amherst, in 2000, 2004, and 2005, respectively. She is currently a Professor in the Computer Science & Engineering Department at the University of Connecticut. She received an NSF CAREER award in February 2008.

1536-1233 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.