BOLA: Near-Optimal Bitrate Adaptation for Online Videos

Kevin Spiteri[1]*, Rahul Urgaonkar[2]†, Ramesh K. Sitaraman[1,3]
[1]University of Massachusetts Amherst, [2]Amazon Inc, [3]Akamai Technologies Inc
{kspiteri,ramesh}@cs.umass.edu, urgaonka@amazon.com

***Abstract*—Modern video players employ complex algorithms to adapt the bitrate of the video that is shown to the user. Bitrate adaptation requires a tradeoff between reducing the probability that the video freezes and enhancing the quality of the video shown to the user. A bitrate that is too high leads to frequent video freezes (i.e., rebuffering), while a bitrate that is too low leads to poor video quality. Video providers segment the video into short chunks and encode each chunk at multiple bitrates. The video player adaptively chooses the bitrate of each chunk that is downloaded, possibly choosing different bitrates for successive chunks. While bitrate adaptation holds the key to a good quality of experience for the user, current video players use ad-hoc algorithms that are poorly understood. We formulate bitrate adaptation as a utility maximization problem and devise an online control algorithm called BOLA that uses Lyapunov optimization techniques to minimize rebuffering and maximize video quality. We prove that BOLA achieves a time-average utility that is within an additive term O(1/V) of the optimal value, for a control parameter V related to the video buffer size. Further, unlike prior work, our algorithm does not require any prediction of available network bandwidth. We empirically validate our algorithm in a simulated network environment using an extensive collection of network traces. We show that our algorithm achieves near-optimal utility and in many cases significantly higher utility than current state-of-the-art algorithms. Our work has immediate impact on real-world video players and for the evolving DASH standard for video transmission.**

***Index Terms*—Internet Video, Video Quality, Adaptive Bitrate Streaming, Lyapunov Optimization, Optimal Control**

I. Introduction

Online videos are the "killer" application of the Internet with videos currently accounting for more than half of the Internet traffic. Video viewership is growing at a torrid pace and videos are expected to account for more than 85% of all Internet traffic within a few years [1]. As all forms of traditional media migrate to the Internet, video providers face the daunting challenge of providing a good quality of experience (QoE) for users watching their videos. Video providers are diverse and include major media companies (e.g., NBC, CBS), news outlets (e.g., CNN), sports organizations (e.g., NFL, MLB), and video subscription services (e.g., Netflix, Hulu). Recent research has shown that low-performing videos that start slowly, play at lower bitrates, and freeze frequently can cause viewers to abandon the videos or watch fewer minutes of the videos, significantly decreasing the opportunity for generating revenue for the video providers [2]–[4], underscoring the need for a high-quality user experience.

A preliminary version of this paper appeared at INFOCOM 2016.
*Supported in part by NSF grant CNS-1413998.
†This work was performed when Rahul Urgaonkar was at IBM Research.

Providing a high-quality experience for video users requires balancing two contrasting requirements. The user would like to watch the highest-quality version of the video possible, where video quality can be quantified by the bitrate at which the video is encoded. For instance, watching a movie in high definition (HD) encoded at 2 Mbps arguably provides a better user experience than watching the same movie in standard definition (SD) encoded at a bitrate of 800 kbps. In fact, there is empirical evidence that the user is more engaged and watches longer when the video is presented at a higher bitrate. However, it is not always possible for users to watch videos at the highest encoded bitrate, since the bandwidth available on the network connection between the video player on the user's device and the video server constrains what bitrates can be watched. In fact, choosing a bitrate that is higher than the available network bandwidth[1] will lead to video freezes in the middle of the playback, since the rate at which the video is being played exceeds the rate at which the video can be downloaded. Such video freezes are called *rebuffers* and playing the video continuously *without rebuffers* is a key factor in the QoE perceived by the user [3]. Thus, balancing the contrasting requirements of playing videos at a high bitrate while at the same time avoiding rebuffers is central to providing a high-quality video watching experience.

A. Adaptive Bitrate (ABR) Streaming

Achieving a high QoE for video streaming is a major challenge due to the sheer diversity of video-capable devices that include smartphones, tablets, desktops, and televisions. Further, the devices themselves can be connected to the Internet in a multitude of ways, including cable, fiber, DSL, WiFi and mobile wireless, each providing different bandwidth characteristics. The need to adjust the video playback to the characteristics of the device and the network has led to the evolution of adaptive bitrate (ABR) streaming that is now the de facto standard for delivering videos on the Internet.

ABR streaming requires that each video is partitioned into *chunks,* where each chunk corresponds to a few seconds of play. Each chunk is then encoded in a number of different bitrates to accommodate a range of device types and network connectivities. When the user plays a video, the video player can download each chunk at a bitrate that is appropriate for the available bandwidth of the network connection. Thus, the player can switch to a chunk with a lower bitrate when

[1] Throughout this paper, we say **bandwidth** when talking about network throughput and **bitrate** when talking about encoding quality.

the available bandwidth is low to avoid rebuffering. If more bandwidth becomes available at a future time, the player can switch back to a higher bitrate to provide a richer experience. The video player has a buffer that allows it to fetch and store chunks *before* they need to be rendered on the screen. Thus, the video player can tolerate brief network disruptions without interrupting the playback of the user by using the buffered chunks. A large disruption, however, will empty the buffer, resulting in rebuffering. The decision of which chunks to download at what bitrates is made by a *bitrate adaptation algorithm* within the video player, the design of such algorithms being the primary focus of our work.

Several popular implementations of ABR streaming exist, including Apple's HTTP Live Streaming (HLS) [5], Microsoft's Live Smooth Streaming (Smooth) [6] and Adobe's Adaptive Streaming (HDS) [7]. Each has its own proprietary implementation and slight modifications to the basic ABR technique described above. A key recent development is a unifying open-source standard for ABR streaming called MPEG-DASH [8]. DASH is broadly similar to the other ABR protocols and is a particular focus in our empirical evaluation.

B. Our Contributions

Our primary contribution is a principled approach to the design of bitrate adaptation algorithms for ABR streaming. In particular, we formulate bitrate adaptation as a utility maximization problem that incorporates both key components of QoE: the average bitrate of the video experienced by the user and the duration of the rebuffer events. An increase in the average bitrate increases utility, whereas rebuffering decreases it. A strength of our framework is that utility can be defined in arbitrary ways, say, depending on the content, video provider, or user device. This contrasts with bitrate adaptation algorithms currently in use that provide no such flexibility.

Using Lyapunov optimization, we derive an online bitrate adaptation algorithm called BOLA (Buffer Occupancy based Lyapunov Algorithm) that provably achieves utility that is within an additive factor of the maximum possible utility. While numerous bitrate adaptation algorithms have been proposed [9]–[12] and implemented within video players, our algorithm is the first to provide a *theoretical guarantee* on the achieved utility. Further, BOLA provides an explicit knob for video providers to set the relative importance of a high video quality in relation to the probability of rebuffering.

While not an explicit part of the Lyapunov optimization framework, we also show how BOLA can be adapted to avoid frequent bitrate switches during video playback. Bitrate switches are arguably less annoying than rebuffering, but it is still of some concern to video providers and users alike if such switches occur too frequently.

Most algorithms implemented in practice use a *bandwidth-based* approach where the available bandwidth between the server and the video player is predicted and the predicted value is used to determine the bitrate of the next chunk that is to be downloaded. A complementary approach is a *buffer-based* approach that does not predict the bandwidth, but only uses the amount of data that is currently stored in the buffer of the video player. Recently, there has been empirical evidence that a buffer-based approach has desirable properties that bandwidth-based approaches lack and has been adopted by Netflix [11]. An intriguing outcome of our work is that the optimal algorithm within our utility maximization framework requires only knowledge of the amount of data in the buffer and no estimate of the available bandwidth. Thus, our work provides the first theoretical justification for why buffer-based algorithms perform well in practice and adds new insights to the ongoing debate [12] within the video streaming and DASH standards communities of relative efficacy of the two approaches. Further, since our algorithm BOLA is buffer-based, it avoids the overheads of more complex bandwidth prediction present in current video player implementations and is more stable under bandwidth fluctuations. Note that our results imply that the buffer level is a *sufficient statistic* that indirectly provides all information about past bandwidth variations required for choosing the next bitrate.

We also *empirically* evaluate BOLA on a wide set of network traces that include 12 test cases provided by the DASH industry forum [13] and 85 publicly-available 3G mobile bandwidth traces [14]. As a benchmark for comparison, we develop an optimal *offline* algorithm that uses dynamic programming and is guaranteed to produce the maximum achievable time-average utility for any given set of network traces. Unlike BOLA that works in an online fashion, the offline optimal algorithm makes decision based on perfect knowledge of future bandwidth variations. Remarkably, the utility achieved by BOLA is within 84–95% of offline optimal utility for all the tested traces.

Besides comparing BOLA with the offline optimal, we also empirically compared our algorithm with two state-of-the-art algorithms proposed in the literature. In *all* test cases, BOLA achieved a utility that is as good as or better than the best state-of-the-art algorithm. In half of the tested scenarios, BOLA did even better by achieving a utility that is nearly 1.75 times the utility of the best state-of-the-art algorithm.

We are also implementing BOLA as the default ABR algorithm in dash.js, the open-source DASH reference player [15].

II. System Model

Our system model closely captures how ABR streaming works on the Internet today. We consider a video player that downloads a video file from a server over the Internet and plays it back to the user. The video file is segmented into chunks that are downloaded in succession. The available bandwidth between the server and the player varies over time. This can be due to reasons such as network congestion and wireless fading among others. The viewing experience of the user is determined by both the video quality as quantified by the bitrates of the chunks that are played back and the playback characteristics such as rebuffering. The objective of the player is to maximize a utility associated with the user's viewing experience while adapting to time-varying (and possibly unpredictable) changes in the available bandwidth.

Video Model: The video file is segmented into N chunks indexed as $\{1, 2, \ldots, N\}$ where each chunk represents p seconds of the video. On the server, each chunk is available in M different bitrates where a chunk encoded at a higher bitrate has a larger size in bits and its playback provides a better user experience and higher utility. Suppose the size (in bits) of a chunk encoded at bitrate index m is S_m bits[2] and suppose the utility derived by the user from viewing it is given by v_m where $m \in \{1, 2, \ldots, M\}$. WLOG, let the chunk bitrates be non-increasing in index m. Then, the following holds.

$$v_1 \geq v_2 \geq \ldots \geq v_M \iff S_1 \geq S_2 \geq \ldots \geq S_M. \quad (1)$$

Note that the actual encoding bitrate for bitrate index m is given by S_m/p bits/second.

Video Player: The video player downloads successive chunks of the video file from the server and plays back the downloaded chunks to the user. Each chunk must be downloaded in its entirety before it can be played back. We assume that the player sends requests to the server to download one chunk at a time. Also, the chunks are downloaded in the same order as they are played back. The video player has a finite buffer of size $Q_{\max}$ chunks[3] to store the downloaded but yet-to-be-played-back chunks. Measuring the buffer in chunks is equivalent to measuring it in seconds since the chunk duration p is fixed. If the buffer is full the player cannot download any new chunks and waits for a fixed period of time given by Δ seconds before attempting to download a new chunk. The chunks that are fully downloaded are played back at a fixed rate of $1/p$ chunks/second without any idling.

When sending a download request for a new chunk, the player also specifies the desired bitrate for that chunk. This enables the player to tradeoff the overall video quality with the likelihood of rebuffering that occurs when there are no chunks in the buffer for playback. Note that while each chunk has a fixed playback time of p seconds, the size of the chunk (in bits) can be different depending on its bitrate. Thus, the choice of bitrate for a chunk impacts its download time.

Network Model: The available bandwidth (in bits/second) between the server and player is assumed to vary continuously in time according to a stationary random process $\omega(t)$. We do not make any assumptions about knowing the statistical properties or probability distribution of $\omega(t)$ except that it has finite first and second moments as well as a finite inverse second moment. Suppose the player starts to download a chunk of bitrate index m at time t. Then the time t' when the download finishes satisfies the following:

$$S_m = \int_t^{t'} \omega(\tau) d\tau \quad (2)$$

Let $\mathbb{E}\{\omega(t)\} = \omega_{\text{avg}}$. Then, $\mathbb{E}\{t' - t\} = S_m/\omega_{\text{avg}}$.

[2]For simplicity, we assume that the chunk size (in bits) is S_m for all chunks of a given bitrate index m. However, our framework can be easily extended to the case where the chunk size for the same bitrate can vary across chunks.

[3]It is common practice for video players to measure the buffer in seconds of playback time rather than in bits.

III. Problem Formulation

We consider two primary performance metrics[4] that affect the overall QoE of the user: (1) time-average playback quality which is a function of the bitrates of the chunks viewed by the user and (2) fraction of time spent not rebuffering. To formalize these metrics, we consider a time-slotted representation of our system model. The timeline is divided into non-overlapping consecutive slots of variable length and indexed by $k \in \{1, 2, \ldots\}$. Slot k starts at time t_k and is $T_k = t_{k+1} - t_k$ seconds long. We assume that $t_1 = 0$. At the beginning of each slot, the video player makes a control decision on whether it should start downloading a new chunk, and if yes, its bitrate. If a download decision is made, then a request is sent to the server and the download starts immediately[5]. This download takes T_k seconds and is completed at the end of slot k. Note that T_k is a random variable whose actual value depends on the realization of the $\omega(t)$ process as well as the choice of chunk bitrate. If the player decides not to download a new chunk in slot k (for example, when the buffer is full), then this slot lasts for a fixed duration of Δ seconds.

We define the following indicator variable for each slot k:

$$a_m(t_k) = \begin{cases} 1 & \text{if the player downloads a chunk} \\ & \text{of bitrate index } m \text{ in slot } k, \text{ and} \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

Then, for all k, we must have $\sum_{m=1}^{M} a_m(t_k) \leq 1$. Moreover, when $\sum_{m=1}^{M} a_m(t_k) = 0$, then no chunks are downloaded. Let K_N denote the index of the slot in which the N^{th} (i.e., last) chunk is downloaded. Also, denote the time at which the player finishes playing back the last chunk by T_{end}. Then the first performance metric of interest is the time-average expected *playback utility* $\overline{v}_N$ which is defined as

$$\overline{v}_N \triangleq \frac{\mathbb{E}\left\{\sum_{k=1}^{K_N} \sum_{m=1}^{M} a_m(t_k) v_m\right\}}{\mathbb{E}\{T_{\text{end}}\}} \quad (4)$$

where the numerator denotes the expected total utility across all N chunks. Note that a chunk can only be played back after it has been downloaded entirely. Thus, T_{end} is greater than the last chunk's download finish time, i.e., $T_{\text{end}} > t_{K_N} + T_{K_N}$.

The second performance metric of interest is the expected fraction of time $\overline{s}_N$ that is spent not rebuffering and can be interpreted as a measure of the average playback "smoothness". This can be calculated by observing that the actual playback time for all N chunks is Np seconds. Thus, the expected *playback smoothness* $\overline{s}_N$ is given by

$$\overline{s}_N \triangleq \frac{Np}{\mathbb{E}\{T_{\text{end}}\}} = \frac{\mathbb{E}\left\{\sum_{k=1}^{K_N} \sum_{m=1}^{M} a_m(t_k) p\right\}}{\mathbb{E}\{T_{\text{end}}\}} \quad (5)$$

where in the last step we use the relation that $Np = \sum_{k=1}^{K_N} \sum_{m=1}^{M} a_m(t_k) p$. Note that $T_{\text{end}} \geq Np$ (since at most

[4]We do not include the secondary objective of avoiding frequent bitrate switches in our formulation, but we deal with it empirically in Section V-E.

[5]Any delays associated with sending the request can be added to the overall download time.

one chunk can be played back at any time), so that $\overline{s}_N \leq 1$.

Performance Objective: We want to design a control algorithm that maximizes the joint utility $\overline{v}_N + \gamma \overline{s}_N$ subject to the constraints of the model. $\gamma > 0$ is an input weight parameter for prioritizing playback utility with the playback smoothness.

This problem can be formulated as a stochastic optimization problem with a time-average objective over a finite horizon and dynamic programming (DP) based approaches can be used to solve it [16]. However, traditional DP-based methods have two major disadvantages. First, they require knowledge of the distribution of the $\omega(t)$ process which may be hard to obtain. Second, even when such knowledge is available, the resulting DP can have a very large state space. This is because the state space for this problem under a DP formulation would consist of not only the timeslot index k and value t_k, but also the buffer occupancy and the quality types of the chunks in the buffer. Further, an appropriate discretization of the $\omega(t)$ process would be required to obtain a tractable solution.

A. Problem Relaxation

In order to overcome the above mentioned challenges associated with traditional DP based methods, we take the following approach. We consider this problem in the limiting regime when the video size becomes large, i.e., $N \to \infty$. In this regime, we can get the following two simplifications. First, the optimal control policy becomes independent of the slot index k. That is, it is sufficient to consider the class of stationary (and potentially randomized) algorithms that make control decisions only as a function of the buffer occupancy. Second, instead of considering the total playback finish time T_{end}, we can consider total download finish time in the objective. Specifically, in the limit $N \to \infty$, the metrics $\overline{v}_N$ and $\overline{s}_N$ can be expressed as

$$\overline{v} \triangleq \lim_{N\to\infty} \overline{v}_N = \frac{\lim_{N\to\infty} \mathbb{E}\left\{\frac{1}{K_N}\sum_{k=1}^{K_N}\sum_{m=1}^{M} a_m(t_k) v_m\right\}}{\lim_{N\to\infty} \mathbb{E}\left\{\frac{1}{K_N}\sum_{k=1}^{K_N} T_k\right\}} \quad (6)$$

$$\overline{s} \triangleq \lim_{N\to\infty} \overline{s}_N = \frac{\lim_{N\to\infty} \mathbb{E}\left\{\frac{1}{K_N}\sum_{k=1}^{K_N}\sum_{m=1}^{M} a_m(t_k) p\right\}}{\lim_{N\to\infty} \mathbb{E}\left\{\frac{1}{K_N}\sum_{k=1}^{K_N} T_k\right\}} \quad (7)$$

This follows by noting that the difference between the expected total playback finish time $\mathbb{E}\{T_{\text{end}}\}$ and the expected total download finish time $\mathbb{E}\left\{\sum_{k=1}^{K_N} T_k\right\}$ is upper bounded by a finite value due to the finite buffer size $Q_{\max}$. Specifically, this upper bound is given by $Q_{\max} p$.

Let us denote the optimal time-average values of these metrics in the large N regime under an optimal policy by v^* and s^* respectively. Note that while the optimal policy in the large N regime does not depend on the slot index, it can still depend on the buffer occupancy state. To address this, we temporarily replace the finite buffer constraint of our model with a *rate stability* constraint [17]. This constraint only requires that the time-average arrival rate into the buffer is equal to the time-average playback rate. It is clear that optimal time-average values of the metrics under this relaxation cannot be smaller than v^* and s^* respectively since the optimal policy for the finite buffer constrained model is rate stable. Moreover, the following can be shown under this relaxation.

Lemma 1: In the large N regime, there exists a *buffer-state-independent* stationary policy that makes i.i.d. control decisions in every slot and satisfies the rate stability constraint while achieving time-average utility no smaller than $v^* + \gamma s^*$.

Proof: This follows from Theorem 4.5 in [17] and is omitted for brevity. ■

Note that such a buffer-state-independent stationary policy is not necessarily feasible for our finite buffer system. Further, calculating it explicitly would require knowledge of the distribution of $\omega(t)$. However, instead of calculating this policy explicitly, we will use its existence and characterization per Lemma 1 to design an *online* control algorithm using Lyapunov optimization [17]. We will show that this online algorithm is feasible for our finite buffer system and achieves a time-average utility that is within $O(1/Q_{\max})$ of $v^* + \gamma s^*$ without requiring any knowledge of the distribution of $\omega(t)$.

IV. BOLA: An Online Control Algorithm

Our online control algorithm for bitrate adaptation makes use of the current buffer level (measured in number of chunks) that we denote by $Q(t_k)$. This is updated at the start of each slot using the following equation:

$$Q(t_{k+1}) = \max[Q(t_k) - \frac{T_k}{p}, 0] + \sum_{m=1}^{M} a_m(t_k) \quad (8)$$

Here, the arrival value into this queue in slot k is given by $\sum_{m=1}^{M} a_m(t_k)$ which is 1 if a download decision is made in slot k and 0 otherwise. The departure value is T_k/p which represents the total number of chunks (including fractional chunks) that could have departed the buffer in slot k. Note that the actual value of T_k is revealed at the end of slot k. We assume that the buffer level is initialized to 0, i.e., $Q(t_1) = 0$.

The *Lyapunov optimization-over-renewal-frames* method [17] can be used to derive an algorithm that optimizes the metrics in (6)–(8). The method greedily minimizes the ratio of *drift plus penalty* to *frame length* over each slot. We now give a high-level intuition of how to derive the algorithm. In slot k, the buffer is kept stable by minimizing the drift defined as $\mathbb{E}\left\{(Q(t_{k+1})^2 - Q(t_k)^2)/2 \mid Q(t_k)\right\}$. Using (8), we achieve buffer stability by minimizing $Q(t_k)(\sum_{m=1}^{M} a_m - T_k/p)$. Using (6)–(7), the performance objective to maximize $\overline{v} + \gamma \overline{s}$ is achieved by maximizing $(\sum_{m=1}^{M} a_m(t_k)(v_m + \gamma p))$. The expected frame (slot) length has a linear relation to $\sum_{m=1}^{M} a_m(t_k) S_m$. We use a control parameter $V > 0$ related to the maximum buffer size to allow a tradeoff between the buffer size and the distance from the optimal utility.

In every slot k, given the buffer level $Q(t_k)$ at the start of the slot, our algorithm makes a control decision by solving

the following deterministic optimization problem:

$$\text{Maximize} \quad \frac{\sum_{m=1}^{M} a_m(t_k)\big(V\upsilon_m + V\gamma p - Q(t_k)\big)}{\sum_{m=1}^{M} a_m(t_k) S_m}$$

$$\text{subject to} \quad \sum_{m=1}^{M} a_m(t_k) \leq 1, a_m(t_k) \in \{0,1\} \tag{9}$$

The constraints of this problem result in a very simple solution structure. Specifically, the optimal solution is given by:

1) If $Q(t_k) > V(\upsilon_m + \gamma p)$ for all $m \in \{1, 2, \ldots, M\}$, then the no-download option is chosen, i.e., $a_m(t_k) = 0$ for all m. Note that in this case $T_k = \Delta$.
2) Else, the optimal solution is to download the next chunk at bitrate index m^* where m^* is the index that maximizes the ratio $\big(V\upsilon_m + V\gamma p - Q(t_k)\big)/S_m$ among all m for which this ratio is positive.

Notice that solving this problem does not require any knowledge of the $\omega(t)$ process. Further, the optimal solution depends only on the buffer level $Q(t_k)$. That's why we call our algorithm *BOLA: Buffer Occupancy based Lyapunov Algorithm*. These properties of BOLA should be contrasted with the bandwidth prediction based strategies that have been recently proposed for this problem that require explicit prediction of the available bandwidth for control decisions.

The following theorem characterizes the theoretical performance guarantees provided by BOLA.

Theorem 2: Suppose BOLA as defined by (9) is implemented in every slot using a control parameter $0 < V \leq \frac{Q_{\max}-1}{\upsilon_1+\gamma p}$. Assume $Q(0) = 0$. Then, the following hold.

1) The queue backlog satisfies $Q(t_k) \leq V(\upsilon_1 + \gamma p) + 1$ for all slots k. Further, the buffer occupancy in chunks never exceeds $Q_{\max}$.
2) The time-average utility achieved by BOLA satisfies

$$\overline{\upsilon}^{\text{BOLA}} + \gamma \overline{s}^{\text{BOLA}} \geq \upsilon^* + \gamma s^* - \frac{p^2 + \Psi}{2p^2 V} \tag{10}$$

where Ψ is an upper bound on $\mathbb{E}\{T_k^2\}$ under any control algorithm and is assumed to be finite.

Proof: See the Appendix. ∎

Remarks: The performance bounds in Theorem 2 show a $O(1/V, V)$ utility and backlog tradeoff that is typical of Lyapunov based control algorithms for similar utility maximization problems. Specifically, the time-average utility of BOLA is within an $O(1/V)$ additive term of the optimal utility and this gap may be made smaller by choosing a larger value of V. However, the largest feasible value of V is constrained by the buffer size and there is a linear relation between them.

A. Understanding BOLA With an Example

We now present a sample run to illustrate how BOLA works. We slice a 99-second video using 3-second chunks and encode it at five different bitrates. While BOLA only requires the utilities to be a non-decreasing function of the chunk bitrate, it is natural to consider concave utility functions with diminishing returns, e.g., a 1 Mbps increase in chunk bitrate likely provides a larger utility gain for the user when that increase is from 0.5 Mbps to 1.5 Mbps than when it is from 5 Mbps to 6 Mbps. A natural choice for our example is the logarithmic utility function: let $\upsilon_m = \ln(S_m/S_M)$. Pick $\gamma = 5.0/p$ and $V = 0.93$. The bitrates and utilities are below.

bitrate (Mbps)	6.000	2.962	1.427	0.688	0.331
S (Mb)	18.00	8.886	4.281	2.064	0.993
υ	2.897	2.192	1.461	0.732	0.000

For any slot we choose the chunk bitrate to maximize $(V\upsilon_m + V\gamma p - Q)/S_m$ for $1 \leq m \leq M$. Fig. 1 shows the relationship between the expression and the buffer level Q for different m. The line intersections mark the buffer levels that correspond to decision thresholds. Fig. 2 summarizes BOLA's bitrate choices as a function of the buffer level.

Fig. 3 shows how BOLA works. We use a synthetic network bandwidth profile as shown in Fig. 3(a). We can see the feedback loop involving the bitrate in (a) and the buffer level in (b). BOLA chooses the bitrate based directly on the buffer level using Fig. 2. The bitrate affects the download time, thus it indirectly affects the buffer level at the beginning of the following slot. Finally, when all the chunks are downloaded, the video player plays out the chunks remaining in the buffer.

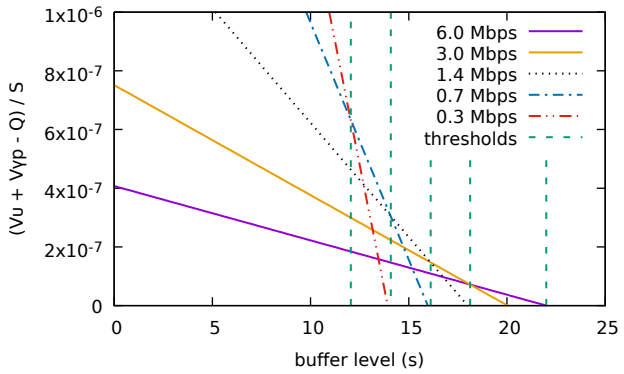

Fig. 1. The value of $(V\upsilon_m + V\gamma p - Q)/S_m$ for different bitrates depends on buffer level. ($\gamma p = 5$ and $V = 0.93$.) Note that the buffer level is Qp seconds.

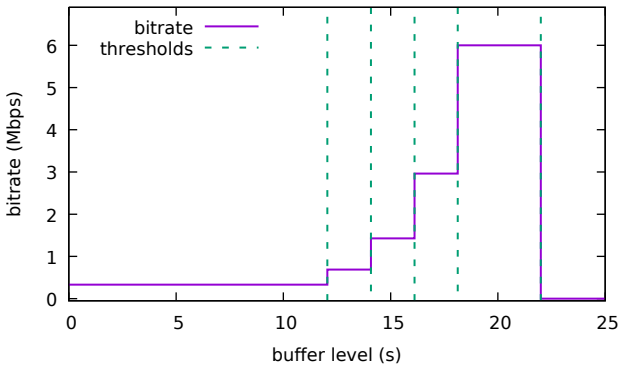

Fig. 2. BOLA's bitrate choice as function of buffer level. ($\gamma p = 5$,$V = 0.93$.) Note that the buffer level is Qp seconds.

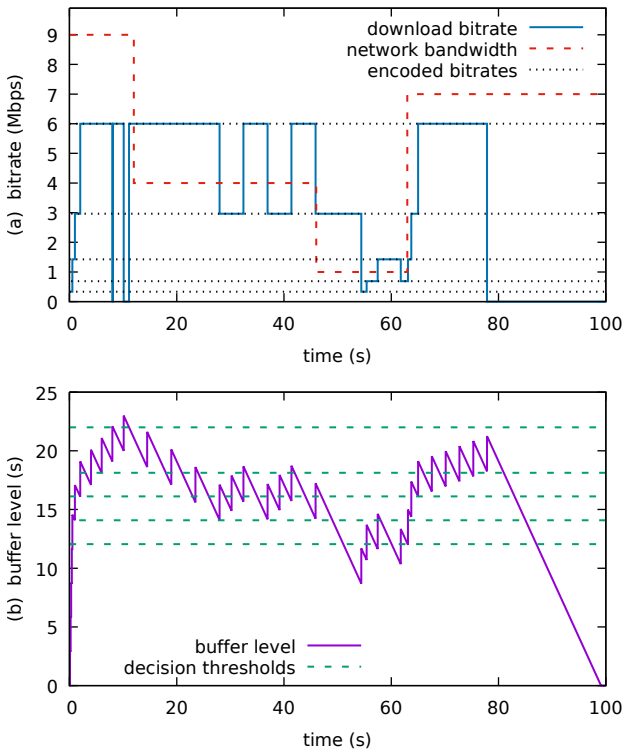

Fig. 3. Sample video download and playback using BOLA. (a) The video is encoded at 5 different bitrates. The network bandwidth varies from high to low and back to high. The downloaded chunk bitrate adapts to the network bandwidth. (b) The buffer level variation triggers bitrate changes when it crosses the thresholds.

B. Choosing Utility and Parameters γ and V

While we chose a logarithmic utility function for the example, a video provider can use any utility function satisfying (1). The utility function might also take into account system characteristics such as the type of device a viewer is using.

γ corresponds to how strongly we want to avoid rebuffering. Increasing γ translates the graphs in Figs. 1 and 2 to the right, effectively shifting the thresholds higher without changing their relative distance. BOLA will thus download more low-bitrate chunks to maintain a larger (and safer) buffer level.

Increasing V expands the graphs in Figs. 1 and 2 horizontally about the origin. If we have a maximum buffer level $Q_{\max}$ we want to avoid downloading unless there is enough space for one full chunk on the buffer, that is unless $Q \leq Q_{\max} - 1$. For a given $Q_{\max}$ we can set $V = (Q_{\max} - 1)/(\upsilon_1 + \gamma p)$.

After choosing a utility function, a video provider might want to specify a safe buffer level such that BOLA will always choose the lowest bitrate when the buffer falls below the level. γ and V can be calculated to satisfy the safe buffer level constraint and a maximum buffer level constraint.

V. Implementation and Empirical Evaluation

We first implemented a basic version of BOLA, named BOLA-BASIC, directly from (9). Recall that when the buffer level is full BOLA does not download a chunk but waits for Δ seconds. Rather than picking an arbitrary value for Δ, we use a dynamic wait until $Q(t_k) \leq V(\upsilon_1 + \gamma p)$. This has the same effect as picking a fixed but very small Δ, so the theoretical analysis still holds. We also implemented other versions of BOLA, namely BOLA-FINITE, BOLA-O, and BOLA-U, that we describe later in this section.

TABLE I
Bitrates used for Big Buck Bunny Test Video

Bitrate Index m	Bitrate (Mbps) Mean	Bitrate (Mbps) Standard Deviation	Chunk Size S (Mb) Mean	Chunk Size S (Mb) Standard Deviation	Utility $\upsilon = \ln(S/S_M)$
1	6.000	1.078	18.00	3.232	3.261
2	5.027	0.891	15.08	2.673	3.084
3	2.962	0.564	8.886	1.691	2.556
4	2.056	0.394	6.168	1.182	2.190
5	1.427	0.275	4.281	0.825	1.825
6	0.991	0.182	2.973	0.545	1.461
7	0.688	0.120	2.064	0.360	1.096
8	0.477	0.096	1.431	0.287	0.729
9	0.331	0.054	0.993	0.162	0.364
$M = 10$	0.230	0.038	0.690	0.113	0.000

TABLE II
Network Profiles for the DASH Benchmarks

1 Mbps (ms)	3 Mbps (ms)	5 Mbps (ms)	7 Mbps (ms)	9 Mbps (ms)	11 Mbps (ms)
5.0 (38)	5.0 (13)	5.0 (11)			
4.0 (50)	4.0 (18)	4.0 (13)	9.0 (25)	9.0 (10)	9.0 (6)
3.0 (75)	3.0 (28)	3.0 (15)	4.0 (50)	4.0 (50)	4.0 (13)
2.0 (88)	2.0 (58)	2.0 (20)	2.0 (75)	2.0 (150)	2.0 (20)
1.5 (100)	1.5 (200)	1.5 (25)	1.0 (100)	1.0 (200)	1.0 (25)
2.0 (88)	2.0 (58)	2.0 (20)	2.0 (75)	2.0 (150)	2.0 (20)
3.0 (75)	3.0 (28)	3.0 (15)	4.0 (50)	4.0 (50)	4.0 (13)
4.0 (50)	4.0 (18)	4.0 (13)			

A. Test Methodology

We simulated all versions of BOLA using the Big Buck Bunny movie [18]. The 10-minute movie was encoded at 10 different bitrates and sliced in 3-second chunks. Although each quality index has a specified average bitrate, chunks may have variable bitrate (VBR) because of the varying nature of the movie. We simulate playback times longer than 10 minutes by repeating the movie. Again we choose a logarithmic utility function: $\upsilon_m = \ln(S_m/S_M)$. Table I shows the mean and standard deviation of the bitrate and chunk size for each quality index and the respective utility values.

The DASH Industry Forum provides benchmarks for various aspects of the DASH standard [13]. The benchmarks include twelve different network profiles. Profiles 1–6 have network bandwidths ranging from 1.5 to 5 Mbps while profiles 7–12 have bandwidths ranging from 1 to 9 Mbps. Different latencies are provided for each bandwidth, where the latency is half the round-trip time (RTT). Table II shows the odd-numbered bandwidth characteristics. Profile 1 spends 30s at each of 5, 4, 3, 2, 1.5, 2, 3 and 4 Mbps respectively, then starts back at the top. Even-numbered profiles are similar to the preceding odd-numbered profiles but start at the low bandwidth stage. For example, profile 2 starts at 1.5 Mbps.

In addition, we also tested our algorithms using a set of 86 3G mobile bandwidth traces that are publicly available [14]. One trace was excluded because it had an average bandwidth

$r(0,t,b) \leftarrow \{0 \text{ for } t = b = 0,\ -\infty \text{ otherwise}\}$
for n in $[1, N]$ **do**
  initialize $r(n,t,b) \leftarrow -\infty$ for all t, b
  for all (t', b') such that $r(n-1, t', b') > -\infty$ **do**
    for m in $[1, M]$ **do**
      $x \leftarrow$ download time(n, t', m)
      $x_\delta \leftarrow \lfloor x/\delta \rfloor \cdot \delta$
      $x'_\delta \leftarrow \max[x_\delta, b' + p - b_{\max}]$
      $y \leftarrow \max[x'_\delta - b', 0]$
      $t \leftarrow t' + x'_\delta$
      $b \leftarrow b' - x'_\delta + y$
      $r' \leftarrow r(n-1, t', b') + v_m - \gamma y$
      $r(n,t,b) \leftarrow \max[r(n,t,b), r']$
    end for
  end for
end for
$r^* \leftarrow \max_{(t,b)} \dfrac{r(N,t,b)}{(t+b)}$

Fig. 4. Calculating the Offline Optimal Utility Upper Bound

of 80 kbps; our lowest video bitrate is 230 kbps. Since the traces do not include latency measurements, we used 50 ms latency giving a RTT of 100 ms throughout. This is the median RTT measured empirically in [19].

B. Computing an Upper Bound on the Maximum Utility

In order to evaluate how well BOLA performs on the traces, it is important to derive an upper bound on the maximum utility that is obtainable by *any* algorithm on a given trace. We derive an *offline* optimal algorithm that provides the maximum achievable utility using dynamic programming. We define a table $r(n,t,b)$ that contains the maximum utility possible when we download the n^{th} chunk and finish at time t with buffer level b. We initialize the table with $r(0,0,0) = 0$. Let $x(n,t,m)$ be the time to download the n^{th} chunk at bitrate index m starting at time t. Note that the dependency of x on n is due to VBR. We quantize the time with granularity δ. While some accuracy is lost, we ensure the final result will still be an upper bound by rounding the download time down.

$$x_\delta(n,t,m) = \lfloor x(n,t,m)/\delta \rfloor \cdot \delta$$

We cap the buffer level at $b_{\max}$.

$$x'_\delta(n,t,b,m) = \max[x_\delta(n,t,m), b + p - b_{\max}]$$

Let $y(n,t,b,m)$ be the rebuffering time.

$$y(n,t,b,m) = \max[x'_\delta(n,t,b,m) - b, 0]$$

We generate entries for $r(n,\cdot,\cdot)$ from $r(n-1,\cdot,\cdot)$ using

$$r(n,t,b) = \max_{m,t',b'} \Big(r(n-1,t',b') + v_m - \gamma y(n,t',b',m) \Big)$$

such that $t = t' + x'_\delta(n,t',b',m)$ and
$b = b' - x'_\delta(n,t',b',m) + y(n,t',b',m)$.

The dynamic programming algorithm is shown in Fig. 4.

C. Evaluating BOLA-BASIC

Fig. 5 shows the time-average utility of BOLA-BASIC when the video length is 10, 30 and 120 minutes. We set $\gamma p = 5$ and varied V for different buffer sizes. We compared the utility of BOLA-BASIC with the offline optimal bound described in

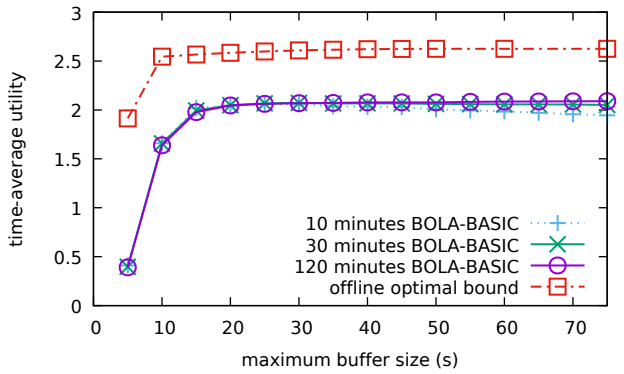

Fig. 5. Time-average utility for $\gamma p = 5$ using profile 1 for BOLA-BASIC.

for n in $[1, N]$ **do**
  $t \leftarrow \min[\text{playtime from begin, playtime to end}]$
  $t' \leftarrow \max[t/2, 3p]$
  $Q^{\text{D}}_{\max} \leftarrow \min[Q_{\max}, t'/p]$
  $V^{\text{D}} \leftarrow (Q^{\text{D}}_{\max} - 1)/(v_1 + \gamma p)$
  $m^*[n] \leftarrow \arg\max_m (V^{\text{D}} v_m + V^{\text{D}} \gamma p - Q)/S_m$
  if $m^*[n] < m^*[n-1]$ **then**
    $r \leftarrow$ bandwidth measured when downloading chunk $(n-1)$
    $m' \leftarrow \min m$ such that $S_m/p \le \max[r, S_M/p]$
    if $m' \le m^*[n]$ **then**
      $m' \leftarrow m^*[n]$
    else if $m' > m^*[n-1]$ **then**
      $m' \leftarrow m^*[n-1]$
    else if some utility sacrificed for fewer oscillations **then**
      pause until $(V^{\text{D}} v_{m'} + V^{\text{D}} \gamma p - Q)/S_{m'} \ge (V^{\text{D}} v_{m'-1} + V^{\text{D}} \gamma p - Q)/S_{m'-1}$ ▷ BOLA-O
    else
      $m' \leftarrow m' - 1$ ▷ BOLA-U
    end if
    $m^*[n] \leftarrow m'$
  end if
  pause for $\max[p \cdot (Q - Q^{\text{D}}_{\max} + 1), 0]$
  download chunk n at bitrate index $m^*[n]$, possibly abandoning
end for

Fig. 6. The BOLA Algorithm.

Section V-B. The offline optimal gave nearly the same utility for the different video lengths. BOLA-BASIC only obtains about 80% of the offline optimal bound. Also, the utility of BOLA-BASIC decreases slightly when the buffer size is increased because it must download more lower-bitrate chunks during startup before it can reach the buffer levels required to switch to higher-bitrate chunks. Our results suggests that there is room to improve BOLA-BASIC that motivates our next version.

D. Adapting BOLA to Finite-Sized Videos

BOLA-BASIC was derived under the assumption that the videos are infinite. Thus, some adaptations are needed for BOLA to work effectively with smaller videos. Motivated by our initial experiments, we implemeted two adaptations to BOLA-BASIC to derive a version we call BOLA-FINITE.

1) Dynamic V value for startup and wind down: A large buffer allows BOLA-BASIC to perform better but it has two drawbacks. First, it takes longer to prime a large buffer during startup. Lower bitrate chunks are preferred until the buffer

1: **function** SHALLABANDON(m, S_m^{R})
2: $r_m \leftarrow (V^{\mathrm{D}}\upsilon_m + V^{\mathrm{D}}\gamma p - Q)/S_m^{\mathrm{R}}$
3: $r_{m'} \leftarrow (V^{\mathrm{D}}\upsilon_{m'} + V^{\mathrm{D}}\gamma p - Q)/S_{m'}$
4: **return** *true* if $r_{m'} > r_m$ for some m' subject to $m < m' \le M$
5: **end function**

Fig. 7. BOLA-FINITE's Download Abandonment Heuristic: m is the current chunk bitrate and S_m^{R} is the number of bits remaining to download in the current chunk.

level reaches steady state. Second, at some late stage all downloads are complete and any remaining buffered video is played out. Any available bandwidth during this period is not utilized. Shortening this period would result in less unutilized available bandwidth. We mitigate these effects by introducing a dynamic V^{D} which corresponds to a dynamic buffer size $Q_{\max}^{\mathrm{D}}$, shown in lines 2–5 in Fig. 6. BOLA-FINITE does not try to fill the whole buffer too soon and does not try to maintain a full buffer too long. We still need a minimum buffer size $3p$ for the algorithm to work effectively.

2) Download abandonment: BOLA-BASIC takes control decisions just before the download of each chunk. Consider a scenario where the player is downloading high-bitrate 6 Mbps chunks in good network conditions. The network bandwidth suddenly drops to 1 Mbps as the player has just started a new chunk download. The chunk will take $6p$ seconds to download, depleting the buffer and possibly causing rebuffering. BOLA-FINITE mitigates this problem by monitoring download progress and possibly abandoning a download. Fig. 7 shows how BOLA-FINITE decides whether or not to abandon the download. If a chunk at bitrate index m is being downloaded, the remaining size S_m^{R} is less than S_m. The chunk can be abandoned and downloaded at some bitrate index m' subject to $m < m' \le M$ when $(V^{\mathrm{D}}\upsilon_m + V^{\mathrm{D}}\gamma p - Q)/S_m^{\mathrm{R}} < (V^{\mathrm{D}}\upsilon_{m'} + V^{\mathrm{D}}\gamma p - Q)/S_{m'}$. The control idea remains the same, but the current bitrate m has a smaller corresponding size S_m^{R} because part of the chunk has already been downloaded. Fig. 3 illustrates a scenario where abandonment might help. At 46s a 3 Mbps chunk download starts. Since there is a bandwidth drop at the time, the chunk takes almost 9s to download. The buffer is depleted and BOLA-BASIC switches to downloading at a bitrate of 0.3 Mbps. BOLA-FINITE with abandonment logic would have detected the rapidly depleting buffer and stopped the long download, with the system only dropping to the 1.4 and 0.7 Mbps download bitrates in the low-bandwidth period.

Fig. 8 shows the time-average utility of BOLA-FINITE for 10, 30 and 120 minutes of playback time with $\gamma p = 5$. Comparing with BOLA-BASIC in Fig. 5, we see that the time-average utility is much closer to the offline optimal bound. The benefit of the adjustments is also evident as the buffer grows larger, as there is no significant decrease in utility caused by filling the buffer with low-bitrate chunks in the earlier stages of the video.

E. Avoiding Bitrate Oscillations

While our performance objective optimizes playback utility and playback smoothness, users are also sensitive to excessive

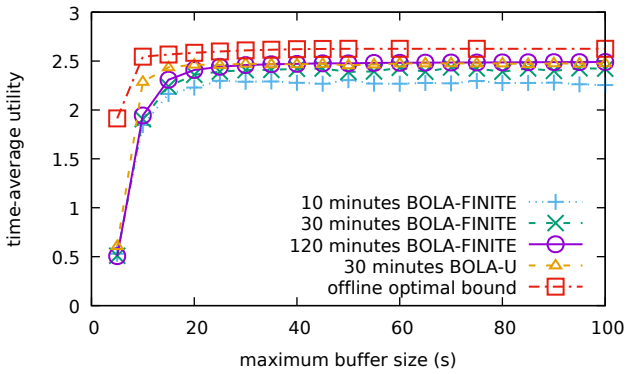

Fig. 8. Time-average utility for $\gamma p = 5$ using profile 1 for BOLA-FINITE and BOLA-U.

bitrate switching. We discuss three causes of bitrate switches.

1) Bandwidth variation: As the network conditions change, the player varies the bitrate, tracking the network bandwidth. Such switches are acceptable; the player has no control on the bandwidth and should adapt to different network conditions.

2) Dense buffer thresholds: Either a larger number of bitrate levels and/or a smaller buffer size may push the threshold levels closer. If the differences between threshold levels are less than the chunk duration p, adding one downloaded chunk to the buffer may push the buffer level over several threshold levels at once. This might cause BOLA-FINITE to overshoot and choose a bitrate that is too high for the available bandwidth. Consequently, the chunk download would take much more than p seconds, leading to excessive buffer depletion, causing BOLA-FINITE to switch down its bitrate by more than one level. In such a scenario BOLA-FINITE can oscillate between bitrates, even when the available bandwidth is stable.

3) Bitrate quantization: Having a stable network bandwidth and widely-spaced thresholds still does not avoid all bitrate switching. Suppose the bandwidth is 2.0 Mbps and it lies between two encoded bitrates of 1.5 and 3.0 Mbps. While the player downloads 1.5 Mbps chunks, the buffer keeps growing. When the buffer crosses the threshold the player switches to 3.0 Mbps, depleting the buffer. After the buffer gets sufficiently depleted, the player switches back to 1.5 Mbps, and the cycle repeats. In this example, a viewer might prefer the video player to stick to the 1.5 Mbps bitrate, sacrificing some utility in order to have fewer oscillations. Or, a viewer might want to maximize utility and play a part of the video in the higher bitrate of 3.0 Mbps at the cost of more oscillations. We describe two variants of BOLA below to suit either viewer.

The first variant that we call BOLA-O mitigates **o**scillations by introducing bitrate capping (lines 7–20 in Fig. 6) when switching to a higher bitrate. BOLA-O verifies that the higher bitrate is sustainable by comparing it to the bandwidth as measured when downloading the previous chunk (lines 8–11). Since the motive is to limit oscillations rather than to predict future bandwidth, this adaptation does not drop the bitrate to a lower level than in the previous download (lines 12–13). Continuous downloading at a bitrate lower than the

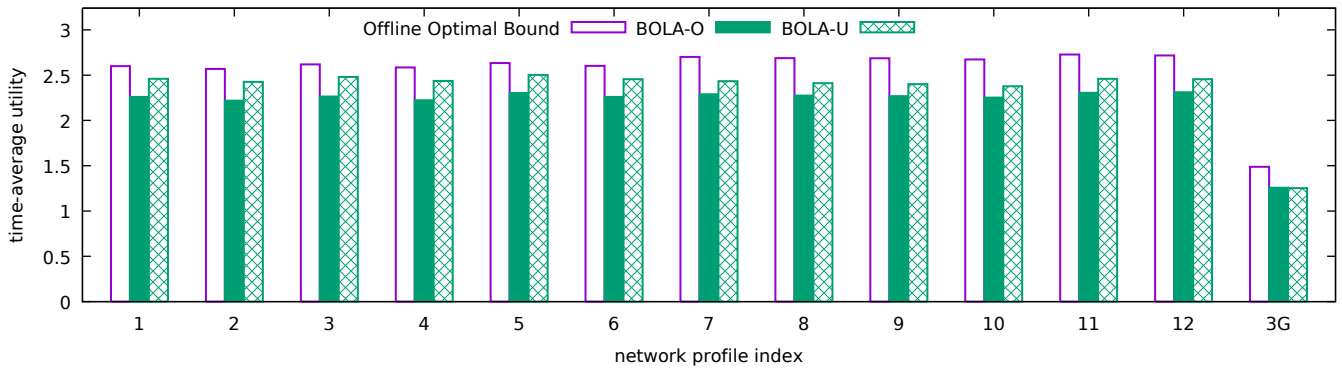

Fig. 9. The time-average utility of BOLA-O and BOLA-U with $\gamma p = 5$ and a 25-second buffer playing a 30-minute video for the DASH test network profiles 1–12 and mobile traces (3G). BOLA utility is within 84–95% of offline optimal utility.

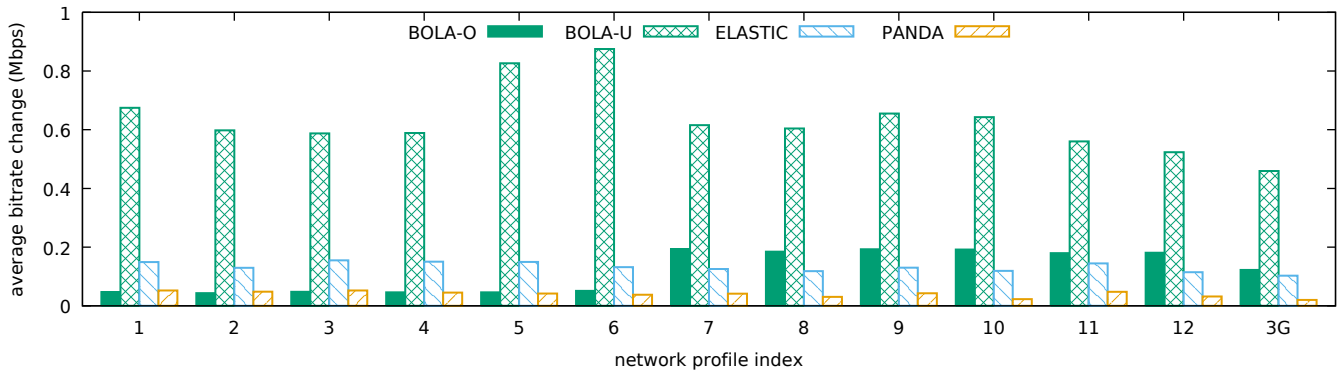

Fig. 10. The average bitrate change between adjacent chunks was smaller for BOLA-O than for BOLA-U, but some bitrate change is needed to accurately track the network bandwidth. In our experiments, ELASTIC and PANDA tracked the bandwidth less accurately than BOLA-O.

bandwidth would cause the buffer to keep growing. BOLA-O avoids this by allowing the buffer to slip to the appropriate threshold before starting the download (line 15).

The second variant that we call BOLA-U does not sacrifice **u**tility. Excessive buffer growth is avoided by allowing the bitrate to be one level higher than the sustainable bandwidth (line 17). This allows the player to choose 3 Mbps in the example. While BOLA-U does not handle the third type of oscillations, it handles the more severe second type.

Looking back at Fig. 8, we see that the added stability of BOLA-U pays off when using a small buffer size and BOLA-U achieves a larger utility than BOLA-FINITE. Fig. 9 shows the time-average utility of BOLA-O and BOLA-U with $\gamma p = 5$ and $Q_{\max} p = 25$s playing a 30-minute video. The utility lost by BOLA-O to avoid oscillations is clearly evident. In practice the lost utility is limited by the distance between encoded bitrates; if the next lower bitrate level is not far from the network bandwidth, then little utility will be lost.

We measure oscillations by comparing consecutive chunks. The change in bitrate between a chunk and the next is the absolute difference between bitrates (in Mbps) of the two chunks. Fig. 10 shows the bitrate change averaged across all the chunks. While BOLA-U has a high average bitrate change because of the quantization, BOLA-O only switches bitrate because of network bandwidth variations.

F. Comparison With State-of-the-Art Algorithms

We now compare BOLA with two state-of-the art algorithms, ELASTIC [9] and PANDA [10]. We use the default design parameters in [9] and [10]. We test both BOLA-O and BOLA-U. Although BOLA performs better with larger buffers, we limited the buffer size to 25s for the tests to ensure fairness. ELASTIC targets a buffer level of 15s but the buffer level varies higher. PANDA targets a minimum buffer level of 26s.

Fig. 11 compares the algorithms using each of the 12 network profiles and the mobile traces. BOLA-U consistently performs significantly better than PANDA. While BOLA-U and ELASTIC perform similarly for profiles 1–6, BOLA-U performs significantly better for the other profiles that have larger bandwidth variations. Recall that BOLA-O always performs within a small margin of BOLA-U in Fig. 9.

Since ELASTIC and PANDA were not designed for the utility score we repeat the comparison using the average bitrate and rebuffering metrics in Fig. 12. For profiles 1–6, BOLA-U has approximately the same bitrate as ELASTIC. ELASTIC has a higher bitrate for profiles 7–12, but that comes at a significant cost in terms of rebuffering. For these profiles, the ratio of the rebuffering time to the play time is more than

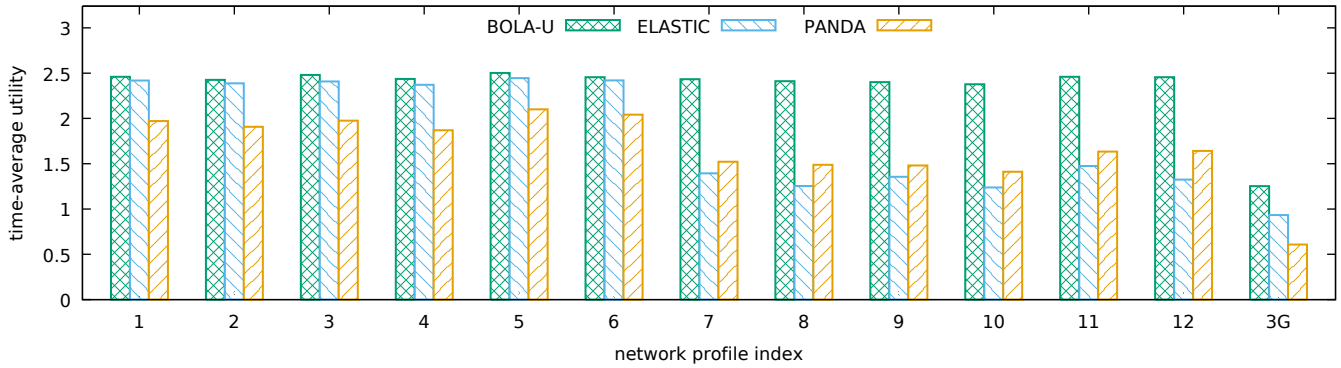

Fig. 11. The time-average utility of BOLA-U, ELASTIC and PANDA with $\gamma p = 5$ playing a 30-minute video for the DASH test network profiles 1–12 and mobile traces (3G). Compared with ELASTIC and PANDA, BOLA-U has about 1.75 times the utility of the other algorithms in roughly half the cases.

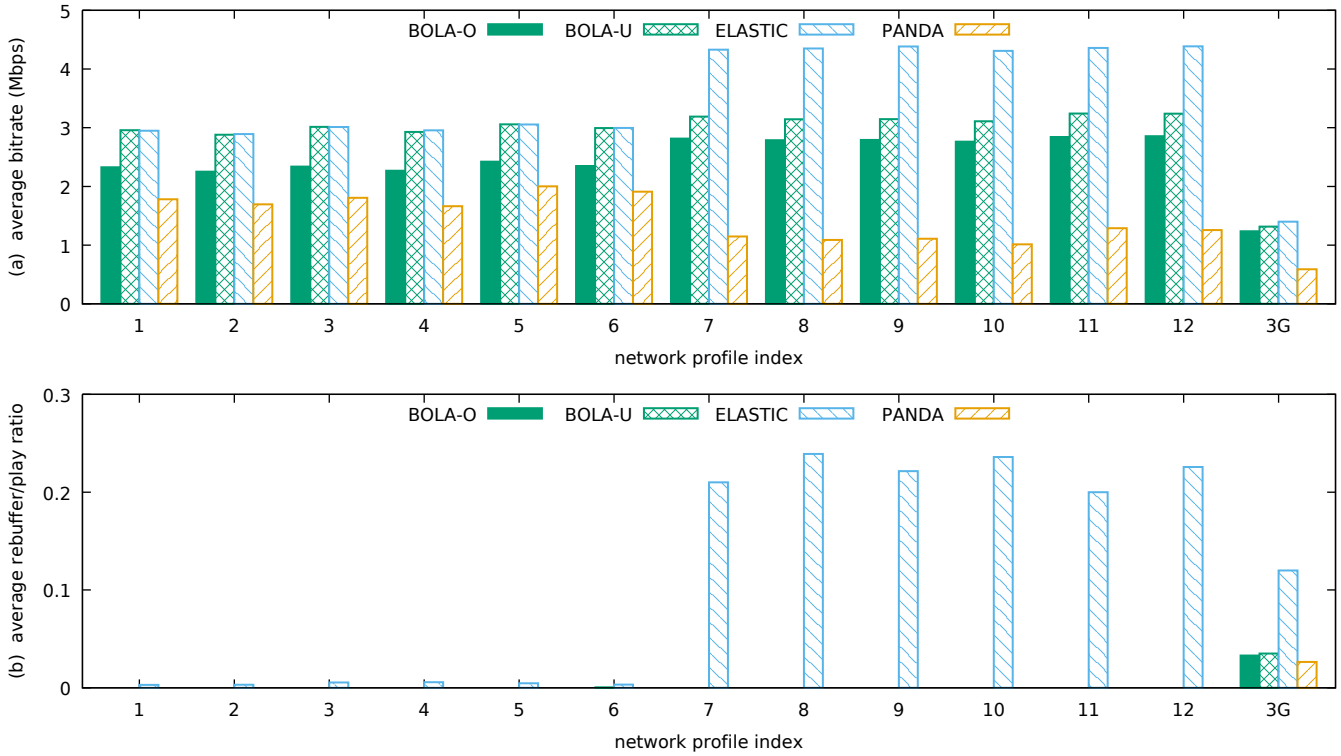

Fig. 12. Comparing BOLA with ELASTIC and PANDA using raw metrics: average bitrate and rebuffer-to-play ratio. BOLA and PANDA do not rebuffer for profiles 1–12. ELASTIC has almost no rebuffering for profiles 1–6, but it has a rebuffer-to-play ratio greater than 20% for profiles 7–12.

20% for ELASTIC, while BOLA-U has no rebuffering. For the 3G traces, ELASTIC has marginally higher bitrate than BOLA-U but has a 12.0% rebuffer-to-play ratio compared with BOLA-U's 3.5%. ELASTIC rebuffers significantly more because it does not react in time when the bandwidth drops.

Comparing BOLA-U with PANDA, both do not rebuffer for profiles 1–12. For the 3G traces, BOLA-U and PANDA have a rebuffer-to-play ratio of 3.5% and 2.6% respectively. However, PANDA has significantly lower bitrate than BOLA-U. The reason is that PANDA is more conservative and in some cases does not change to a higher bitrate even if it is sustainable.

In Fig. 10 we show our results for our secondary metric of bitrate oscillations. BOLA-U does not perform well in this metric, since it attempts to maximize utility at the cost of increased oscillations. Comparing BOLA-O with ELASTIC and PANDA, ELASTIC has a lower average change than BOLA-O only in the cases where it has a slow reaction and excessive rebuffering. PANDA has a lower average change because it is more conservative and in some cases does not change to a higher bitrate even if that bitrate is sustainable.

Thus, from our empirical analysis, we can conclude that BOLA achieves higher utility, and performs more consistently across different scenarios in comparison with ELASTIC and PANDA. One reason for the consistency of BOLA is that

it does not have a large number of parameters. BOLA has two design parameters γ and V, which have an intuitive significance as discussed in Section IV-B, and an option of whether or not to trade off some utility to reduce oscillations. Other algorithms have a number of different parameters and tuning the parameters for a particular scenario might make the system less suited for other scenarios.

VI. Related Work

There has been a lot of recent work on bitrate adaptation algorithms, much of which is based on estimating the bandwidth of the network connection. Notable among this is ELASTIC [9] that uses control theory to adjust the bitrate so as to keep the buffer occupancy at a constant level. Another notable algorithm is PANDA [10] which also estimates the network bandwidth. PANDA drops the download bitrate as soon as low bandwidth is detected but only increases the bitrate slowly to probe the real capacity when a higher bandwidth is detected. In [12], an algorithm using model predictive control (MPC) is proposed to optimize a comprehensive set of metrics. In this approach, the bitrate for the current chunk is chosen based on a network bandwidth prediction for the next few chunks. But, its performance depends on the accuracy of such a prediction. The approach also requires significant offline optimization to be performed outside of the client for an exhaustive set of scenarios. In [11], a buffer-based algorithm is proposed, but assumes that the buffer size is large (in the order of minutes), thereby making it not suitable for short videos. Further, it does not provide any theoretical guarantees for its buffer-based approach. Unlike prior work, we derive a buffer-based algorithm with theoretical guarantees that is simple to implement within the client and we empirically show its efficacy on extensive network traces.

VII. Conclusion

We formulated video bitrate adaptation for ABR streaming as a utility maximization problem and derived BOLA, an online control algorithm that is provably near-optimal. Further, we empirically demonstrated the efficacy of BOLA using extensive traces. In particular, we showed that our online algorithm achieves utility close to the optimal offline algorithm. We also showed that our algorithm significantly outperformed two well-known algorithms in nearly half the test scenarios. We are also implementing BOLA as the default ABR algorithm in dash.js, the open-source DASH reference player [15].

VIII. Acknowledgements

We would like to thank Daniel Sparacio and Will Law of Akamai for their key insights on real-world player implementations. Further, Daniel was instrumental in helping us implement BOLA in the DASH reference player.

Appendix
Proof of Theorem 2

We first show part 1 using induction. Note that the bound $Q(t_k) \leq V(\upsilon_1 + \gamma p) + 1$ holds for $k = 1$ since $Q(t_1) = Q(0) = 0$. Now suppose it holds for some k. We will show that it will also hold for $k+1$. We have two cases.

Case 1: $Q(t_k) \leq V(\upsilon_1 + \gamma p)$
From the queueing equation (8), it follows that the maximum that $Q(t_k)$ can increase in slot k is by 1. This implies that $Q(t_{k+1}) \leq V(\upsilon_1 + \gamma p) + 1$.

Case 2: $V(\upsilon_1 + \gamma p) < Q(t_k) \leq V(\upsilon_1 + \gamma p) + 1$
We have $Q(t_k) > V(\upsilon_m + \gamma p)$ for all $m \in \{1, 2, \ldots, M\}$ (using (1)). It follows from the structure of optimal solution to (9) that BOLA will choose the no-download option in this case. As a result, $Q(t_k)$ cannot increase and we have that $Q(t_{k+1}) \leq V(\upsilon_1 + \gamma p) + 1$.

$Q(t_k)$ denotes the total number of chunks in the buffer. This can be at most $Q_{\max}$ using the relation

$$V \leq \frac{Q_{\max} - 1}{\upsilon_1 + \gamma p}.$$

In part 2, we show the bound in (10) using the technique of Lyapounov optimization over variable size frames [17]. We first define a Lyapunov function $L(Q(t_k))$ as

$$L(Q(t_k)) = \frac{1}{2} Q^2(t_k)$$

and define the per-slot conditional Lyapunov drift $D(t_k)$ as

$$D(t_k) \triangleq \mathbb{E}\left\{ L(Q(t_{k+1})) - L(Q(t_k)) | Q(t_k) \right\}.$$

We use the queueing equation (8), to bound $D(t_k)$. We consider two cases for (8): $Q(t_k) \leq T_k/p$ and $Q(t_k) > T_k/p$. In the first case we have

$$D(t_k) = \mathbb{E}\left\{ \frac{1}{2} \left(\sum_{m=1}^{M} a_m(t_k) \right)^2 - \frac{1}{2} Q^2(t_k) | Q(t_k) \right\}.$$

In the second case we have

$$D(t_k) = \mathbb{E}\left\{ \frac{1}{2} \left(\frac{T_k}{p} - \sum_{m=1}^{M} a_m(t_k) \right)^2 - Q(t_k) \left(\frac{T_k}{p} - \sum_{m=1}^{M} a_m(t_k) \right) | Q(t_k) \right\}$$

In both cases, $D(t_k)$ is bounded by

$$D(t_k) \leq \frac{p^2 + \Psi}{2p^2} - Q(t_k) \mathbb{E}\left\{ \frac{T_k}{p} - \sum_{m=1}^{M} a_m(t_k) | Q(t_k) \right\}$$

where Ψ is an upper bound on $\mathbb{E}\left\{T_k^2\right\}$ under any control algorithm and is assumed to be finite.

Following the methodology of the Lyapunov optimization technique, we subtract $V \times$ reward term from both sides of the

above to get

$$D(t_k) - V\mathbb{E}\left\{\sum_{m=1}^{M} a_m(t_k)(\upsilon_m + \gamma p)|Q(t_k)\right\} \leq \frac{p^2+\Psi}{2p^2} - Q(t_k)\mathbb{E}\left\{\frac{T_k}{p} - \sum_{m=1}^{M} a_m(t_k)|Q(t_k)\right\} - V\mathbb{E}\left\{\sum_{m=1}^{M} a_m(t_k)(\upsilon_m + \gamma p)|Q(t_k)\right\} \quad (11)$$

Let us denote the control decisions (and resulting slot lengths) under our control algorithm by the superscript BOLA while those under the stationary policy of Lemma 1 by STAT. Since BOLA greedily maximizes over a frame, it ensures that

$$\mathbb{E}\left\{\sum_{m=1}^{M} a_m^{\mathrm{BOLA}}(t_k)(Q(t_k) - V(\upsilon_m + \gamma p))|Q(t_k)\right\} \leq \eta \times \mathbb{E}\left\{\sum_{m=1}^{M} a_m^{\mathrm{STAT}}(t_k)(Q(t_k) - V(\upsilon_m + \gamma p))|Q(t_k)\right\} \quad (12)$$

where $\eta = \frac{\mathbb{E}\left\{T_k^{\mathrm{BOLA}}|Q(t_k)\right\}}{\mathbb{E}\left\{T_k^{\mathrm{STAT}}|Q(t_k)\right\}}$. To see this, compare the ratio on the left hand side above with the objective in (9) while noting that we can express the denominator as $\mathbb{E}\left\{T_k^{\mathrm{BOLA}}|Q(t_k)\right\} = (\sum_{m=1}^{M} a_m^{\mathrm{BOLA}}(t_k)S_m)/\omega_{\mathrm{avg}}$. It should be noted that this ratio can be minimized without requiring knowledge of ω_{avg}. Then we use (12) to express (11) as

$$D^{\mathrm{BOLA}}(t_k) - V\mathbb{E}\left\{\sum_{m=1}^{M} a_m^{\mathrm{BOLA}}(t_k)(\upsilon_m + \gamma p)|Q(t_k)\right\} \leq \frac{p^2+\Psi}{2p^2} - Q(t_k)\mathbb{E}\left\{\frac{T_k^{\mathrm{BOLA}}}{p} - \eta\sum_{m=1}^{M} a_m^{\mathrm{STAT}}(t_k)|Q(t_k)\right\} - V\eta\mathbb{E}\left\{\sum_{m=1}^{M} a_m^{\mathrm{STAT}}(t_k)(\upsilon_m + \gamma p)|Q(t_k)\right\}$$

Substituting the time-average values for the stationary policy we get

$$D^{\mathrm{BOLA}}(t_k) - V\mathbb{E}\left\{\sum_{m=1}^{M} a_m^{\mathrm{BOLA}}(t_k)(\upsilon_m + \gamma p)|Q(t_k)\right\} \leq \frac{p^2+\Psi}{2p^2} - Q(t_k)\Big(\frac{1}{p} - r^{\mathrm{STAT}}\Big)\mathbb{E}\left\{T_k^{\mathrm{BOLA}}|Q(t_k)\right\} - V(\upsilon^* + \gamma s^*)\mathbb{E}\left\{T_k^{\mathrm{BOLA}}|Q(t_k)\right\} \quad (13)$$

where r^{STAT} denotes the expected arrival rate under the stationary policy and cannot exceed $1/p$ since it is rate stable. Thus we have

$$D^{\mathrm{BOLA}}(t_k) - V\mathbb{E}\left\{\sum_{m=1}^{M} a_m^{\mathrm{BOLA}}(t_k)(\upsilon_m + \gamma p)|Q(t_k)\right\} \leq \frac{p^2+\Psi}{2p^2} - V(\upsilon^* + \gamma s^*)\mathbb{E}\left\{T_k^{\mathrm{BOLA}}|Q(t_k)\right\} \quad (14)$$

Taking conditional expectation of both sides and summing over $k \in \{1, 2, \ldots, K_N\}$, we get

$$\mathbb{E}\left\{L(Q(t_{K_{N+1}}))\right\} - V\mathbb{E}\left\{\sum_{k=1}^{K_N}\sum_{m=1}^{M} a_m^{\mathrm{BOLA}}(t_k)(\upsilon_m + \gamma p)\right\} \leq \frac{(p^2+\Psi)K_N}{2p^2} - V(\upsilon^* + \gamma s^*)\mathbb{E}\left\{\sum_{k=1}^{K_N} T_k^{\mathrm{BOLA}}\right\} \quad (15)$$

Dividing both sides by $V\mathbb{E}\left\{\sum_{k=1}^{K_N} T_k^{\mathrm{BOLA}}\right\}$ and taking the limit as $N \to \infty$ yields the bound in (10).

REFERENCES

[1] Cisco, "Visual Networking Index," http://bit.ly/KXDUaX.

[2] 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. ACM SIGCOMM*, 2011.

[3] S. S. Krishnan and R. K. Sitaraman, "Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs," in *Proc. ACM IMC*, 2012.

[4] R. K. Sitaraman, "Network performance: Does it really matter to users and by how much?" in *Proc. COMSNETS*, 2013.

[5] "Apple HTTP Live Streaming," https://developer.apple.com/resources/http-streaming/, accessed: September, 25, 2014.

[6] "Microsoft Smooth Streaming," http://www.iis.net/downloads/microsoft/smooth-streaming, accessed: September, 25, 2014.

[7] "Adobe HTTP Dynamic Streaming," http://www.adobe.com/products/hds-dynamic-streaming.html, accessed: September, 25, 2014.

[8] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP – Standards and Design Principles," in *Proc. ACM MMSys*, 2011.

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

[10] 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*, vol. 32, no. 4, pp. 719–733, 2014.

[11] 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. ACM SIGCOMM*, 2014.

[12] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," in *Proc. ACM SIGCOMM*, 2015.

[13] "Guidelines for Implementation: DASH-AVC/264 Test cases and Vectors," http://dashif.org/guidelines/, DASH Industry Forum, January 2014.

[14] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3G networks: analysis and applications," in *Proc. ACM MMsys*, 2013.

[15] http://dashif.org/reference/players/javascript/index.html.

[16] D. Bertsekas, *Dynamic programming and optimal control*. Athena Scientific Belmont, MA, 1995, vol. 1.

[17] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[18] "Big Buck Bunny Movie," https://peach.blender.org/, accessed: July 31, 2015.

[19] P. Romirer-Maierhofer, F. Ricciato, A. D'Alconzo, R. Franzan, and W. Karner, "Network-wide measurements of TCP RTT in 3G," in *Traffic Monitoring and Analysis*. Springer Berlin Heidelberg, 2009, pp. 17–25.