ABR Streaming of VBR-encoded Videos: Characterization, Challenges, and Solutions

Yanyuan Qin[1], Shuai Hao[2], K.R. Pattipati[1], Feng Qian[3],
Subhabrata Sen[2], Bing Wang[1] and Chaoqun Yue[1]
[1]University of Connecticut [2]AT&T [3]Indiana University Bloomington

ABSTRACT

Adaptive Bitrate (ABR) video streaming is widely used for over-the-top (OTT) video delivery. Recently, streaming providers have been moving towards using Variable Bitrate (VBR) encodings for the video content, spurred by the potential of improving user QoE (Quality of Experience) and reducing network bandwidth requirements compared to Constant Bitrate (CBR) encodings. However VBR introduces new challenges for ABR streaming, whose nature and implications are little understood. We explore these challenges across diverse video genres, encoding technologies, and platforms. We identify distinguishing characteristics of VBR encodings that impact user QoE and should be factored in any ABR adaptation decision. Traditional ABR adaptation strategies designed for the CBR case are not adequate for VBR. We develop novel best practice design principles to guide ABR rate adaptation for VBR encodings. As a proof of concept, we design a novel and practical control-theoretic rate adaptation scheme, CAVA (Control-theoretic Adaption for VBR-based ABR streaming), incorporating these concepts. Extensive evaluations show that CAVA substantially outperforms existing state-of-the-art adaptation techniques, validating the importance of these design principles.

CCS CONCEPTS

• **Information systems** → Multimedia streaming;

KEYWORDS

Adaptive Video Streaming; VBR videos; DASH

ACM Reference Format:
Yanyuan Qin, Shuai Hao, K.R. Pattipati, Feng Qian, Subhabrata Sen, Bing Wang, and Chaoqun Yue. 2018. ABR Streaming of VBR-encoded Videos: Characterization, Challenges, and Solutions. In *The 14th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '18), December 4–7, 2018, Heraklion, Greece.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3281411.3281439

1 INTRODUCTION

Adaptive Bitrate (ABR) streaming (specifically HLS [2] and DASH [14]) has emerged as the *de facto* video streaming technology in industry for dynamically adapting the video streaming quality based on varying network conditions. A video is compressed into multiple independent *streams* (or *tracks*), each specifying the same content but encoded with different bitrate/quality. A track is further divided into a series of *chunks*, each containing data for a few seconds' worth of playback. During streaming playback, to fetch the content for a particular play point in the video, the adaptation logic at the client player dynamically determines what bitrate/quality chunk to download.

The video compression for a track can be (1) Constant Bitrate (CBR) – encodes the entire video at a relatively fixed bitrate, allocating the same bit budget to both *simple scenes* (i.e., low-motion or low-complexity scenes) and *complex scenes* (i.e., high-motion or high-complexity scenes), resulting in variable quality across scenes, or (2) Variable Bitrate (VBR) – encodes *simple scenes* with fewer bits and *complex scenes* with more bits, while maintaining a consistent quality throughout the track. VBR presents some key advantages over CBR [22], such as the ability to realize better video quality for the same average bitrate, or the same quality with a lower bitrate encoding than CBR. Traditionally, streaming services mainly deployed only CBR, partly due to various practical difficulties including the complex encoding pipelines for VBR, as well as the demanding storage, retrieval, and transport challenges posed by the multi-timescale bitrate burstiness of VBR videos. Most recently content providers have begun adopting VBR encoding [16, 35, 46], spurred by the promise of substantial improvements in the quality-to-bits ratio compared to CBR, and by technological advancements in VBR encoding pipelines. Current commercial systems and ABR protocols do not fully make use of the characteristics of VBR videos; they typically assume a track's peak rate to be its bandwidth requirement when making rate adaptation decisions [46]. Existing ABR streaming research centered mostly on CBR encoding, and very little on the VBR case (see §7). Therefore there exists little prior understanding of the issues and challenges involved in ABR streaming of VBR videos.

In this paper, we explore a number of important open research questions around ABR streaming for VBR-encoded video. We make the following main contributions:

• We analyze the characteristics of VBR videos (§3) using a rich set of videos spanning diverse content genres, predominant encoding technologies (H264 [18] and HEVC (H.265) [17]) and platforms (§2). Jointly examining content complexity, encoding quality, and encoding bitrates, we identify distinguishing characteristics of VBR encodings that impact user QoE, and their implications for ABR rate adaptation and QoE metrics. We find that, for real-world VBR encoding settings, complex scenes, although encoded with more bits, have inferior encoding quality compared to other scenes in the same track, across a range of video quality metrics. We also find that different chunk sizes across the video can be used to identify relative scene complexity with high accuracy. These two key observations (i) suggest the need to provide complex scenes with *differential treatment* during streaming, as improving the quality of complex scenes brings more QoE enhancement compared to improving simple scenes [23], and (ii) provide us a practical pathway towards achieving this differential treatment in the context of today's *de facto* ABR protocols where the scene complexity or quality information is typically unavailable to the ABR logic.

• Based on our analysis, we enunciate three key design principles (*non-myopic*, *differential treatment*, and *proactive*) for ABR rate adaptation for VBR videos (§4). These principles explicitly account for the characteristics of VBR videos (including scene complexity, quality, and bitrate variability across the video). Specifically, the *non-myopic* principle dictates considering multiple future chunks when making rate adaptation decisions, leading to better usage of the available network bandwidth; the *differential treatment* principle strategically favors complex scenes over simple scenes for better user QoE; the last principle proactively reacts to chunks' variable sizes/bitrates by, for example, pre-adjusting the target buffer level, thus further facilitating the smoothness and adaptiveness of VBR streaming.

• We design CAVA (Control-theoretic Adaption for VBR-based ABR streaming), a novel, practical ABR rate adaptation algorithm based on concrete instantiations of these principles (§5). CAVA uses control theory that has shown promise in adapting to dynamic network bandwidth such as that in cellular networks [33, 47]. Its design involves two tightly coupled controller loops that work in synergy using easy-to-obtain information such as VBR chunk sizes and historically observed network bandwidth. Specifically, CAVA improves the QoE by (i) using Proportional-Integral-Derivative (PID) control concepts [4] to maintain a dynamic target buffer level to limit stalls, (ii) judiciously selecting track levels to maximize the quality while minimizing quality changes, and (iii) strategically favoring chunks for complex scenes by saving bandwidth for such chunks.

• We evaluate the performance of CAVA for a wide range of real-world network scenarios, covering both LTE and broadband (§6). Our results show that CAVA considerably outperforms existing state-of-the-art ABR schemes across several important dimensions. It provides significantly enhanced quality for complex scenes while not sacrificing the quality of other scenes. Depending on the video, the percentage of low-quality chunks under CAVA is up to 75% lower than that of other schemes. CAVA also leads to substantially lower rebuffering (up to 95%) and quality variation (up to 48%). The network data usage under CAVA is in the same ballpark or slightly lower than most of the other schemes. The above results demonstrate that CAVA achieves a much better balance in the multiple-dimension design space than other schemes. The performance improvements hold across a wide spectrum of settings, indicating that the three design principles hold broadly. Our implementation of CAVA in the open source `dash.js` [15] player further demonstrates that CAVA is lightweight and practical.

2 THE VBR VIDEO DATASET

Our video dataset includes 16 videos: 8 encoded by YouTube, and 8 encoded by us using `FFmpeg` [13], following specifications in [11]. Each video is around 10 minutes, and includes 6 tracks/levels (we use the terms *track* and *level* interchangeably) for ABR streaming, with resolutions of 144p, 240p, 360p, 480p, 720p, and 1080p, respectively. Our use of YouTube and `FFmpeg` encodings complement each other. YouTube represents the state-of-the-art practice of VBR encoding from a popular commercial streaming service. With `FFmpeg`, we explore state-of-the-art VBR encoding recommendations from another commercial streaming service, Netflix [11]. We also use `FFmpeg` to explore a variety of other settings (e.g., 4x-capped video (§3.3), H.265 encoding) beyond the above cases. We next explain in detail how we obtain our dataset.

Videos Encoded by `FFmpeg`. We select 4 publicly available *raw* videos from [45]. These videos include Elephant Dream (ED), Big Buck Bunny (BBB), Tears of Steel (ToS), and Sintel, in the categories of animation and science fiction. We encode these 4 raw videos into 4 H.264 and 4 H.265 videos using `FFmpeg` (8 encoded videos in total). For each video, we choose the aforementioned six tracks (144p to 1080p), consistent with [29]. To encode each track, we follow the per-title "three-pass" encoding procedure from Netflix [11]. In the first pass, a video clip is encoded using a fixed constant rate factor (CRF) value, which accounts for motion and has been shown to outperform other methods such as constant quantization parameter (QP) [36]. In the second and third passes, we use the output bitrates of the first step to drive a two-pass VBR encoder to avoid over-allocating bits for simple scenes and under-allocating bits for complex scenes. In practice, *capped VBR* (i.e., the chunk bitrate is capped at a certain high limit) is often used to limit the amount of bitrate variability and enable the player to estimate the incoming chunk sizes [46]. Following the latest recommendations from HLS [3], we encode the videos to be 2× capped (i.e., the peak bitrate is limited to 200% of the average) using `-maxrate` and `-bufsize`, two options in `FFmpeg`. After a track is encoded, we segment it into 2-sec chunks. Our encoding uses both the widely adopted H.264, and a more recent and efficient codec, H.265 [17]. We use CRF value of 25, which provides good viewing quality [11, 36]. For an encoded video, we assess its quality relative to a *reference video* (§3.1), which is its corresponding raw video footage.

Videos Encoded by YouTube. We upload the aforementioned 4 raw videos to YouTube and download the encoded videos using `youtube-dl` [48]. We also downloaded 4 other videos from YouTube, in the categories of sports, animal, nature and action movies. The highest quality track of these 4 videos has a resolution of 2160p, with average bitrate of 14 – 18 Mbps, significantly higher than lower tracks. To be consistent with the other videos which were 6-track with top track at 1080p, for these 4 videos, we use the bottom six tracks between 144p and 1080p as the set of tracks for ABR adaptation. We use the top track (2160p) as the reference video to measure the video quality of the lower tracks, as we do not have the raw source for these videos. All YouTube videos are encoded using H.264 codec [18], with chunk duration around 5 seconds, consistent with that in [25].

Chunk Duration and Bitrate Variability. Overall, our dataset includes two chunk durations: 2 (`FFmpeg` encodings) and 5 seconds (YouTube encodings). They are consistent with the range of chunk durations (2 to 10 seconds) that are used in commercial services [46], and allow us to investigate the impact of chunk duration on the performance of ABR streaming in §6. All encoded videos exhibit

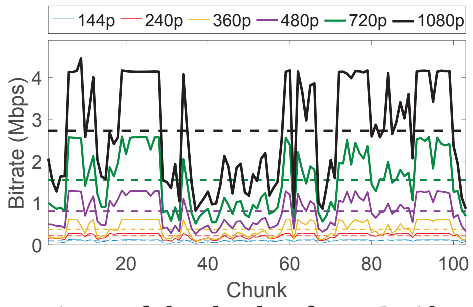

Figure 1: Bitrate of the chunks of a VBR video (Elephant Dream, H.264, YouTube encoded).

significant bitrate variability: the coefficient of variation of the bitrate in a track varies from 0.3 to 0.6. Fig. 1 shows an example YouTube video; the six horizontal dashed lines mark the average bitrate of the six tracks. All encodings are capped VBR: for YouTube videos, the maximum bitrate of a track ranges between 1.1× to 2.3× the average bitrate; for the FFmpeg videos, the corresponding range is 1.4× to 2.4×. Note that although we set the cap to 2× explicitly for FFmpeg, the resulting videos can exceed the cap slightly to achieve the specified quality. For all videos, the two lowest tracks have the lowest variability, since the low bitrate limits the amount of variability that can be introduced in VBR encoding; for all the other tracks, the ratio is 1.4 to 2.4.

3 VBR STREAMING: CHALLENGES

We now analyze our video dataset described in §2: we first present the characteristics of VBR encoding, and then show its implications for ABR streaming, highlighting challenges of streaming VBR videos.

3.1 VBR Video Characteristics

3.1.1 Scene Complexity and Chunk Size. Classifying chunks by scene complexity is useful for ABR streaming since the knowledge of scene complexity can be used for rate adaptation. For instance, since complex scenes in a video play a particularly important role in viewing quality [23], the player may choose higher tracks for complex scenes to bring more enhancement to viewing experience.

One way of determining scene complexity is through Spatial information (SI) and Temporal Information (TI) [19], two commonly used metrics for quantifying the spatial and temporal complexity of the scenes. This approach, however, requires computation-heavy content-level analysis. More importantly, such scene complexity information is not available in the complex ABR streaming pipeline in today's commercial services, and would involve non-trivial changes to realize, including resources to compute the complexity information and new features to the ABR streaming standard specifications. Below, we propose a simple and lightweight method that uses relative chunk size to approximate scene complexity, and can be readily used for ABR streaming (§3.2).

Our approach is motivated by the following two properties. (1) Since the VBR encoding principles dictate that simple scenes are encoded with fewer bits and complex scenes with more bits [12], we expect that the relative size of a chunk (in bytes) can be used as a proxy for the relative scene complexity. (2) Since the scene complexity is a function of the content itself (e.g., motion, details in the scene), we expect that the scene complexity at a given playback point in a video is consistent among different tracks. Combining with the property that the relative size of a chunk is correlated with the scene complexity, we expect that a chunk that is relatively large/small in one track (compared to other chunks in that track) is also relatively large/small in another track.

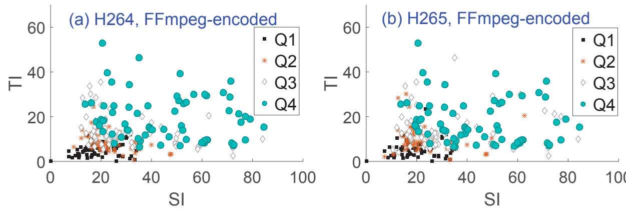

Figure 2: Chunk SI & TI (Elephant Dream, Track 3).

We verify the above two properties as follows. To verify Property (1), we calculate the SI and TI values (recall that SI and TI are two commonly used complexity metrics) for each chunk in a video. Specifically, we obtain SI and TI values from the raw video, which is not affected by encoding distortion, and hence the SI and TI values thus obtained reflect the inherent scene complexity more accurately. We then consider an encoded video, for each track, we classify the chunks into four categories, based on their size distribution. Specifically, a chunk with a size falling into the first quartile is called a Q1 chunk, and a chunk with a size falling into the second quartile is called a Q2 chunk, and so on. We observe that indeed chunks in lower quartiles have lower SI and TI values, while the Q4 chunks tend to have the largest SI and TI values. Fig. 2 shows an example. For the H.264 encoding, Fig. 2(a) shows that 78% of Q4 chunks have SI and TI larger than 25 and 7, respectively, while only 11% of Q1 chunks and 14% of Q2 chunks have SI and TI larger than these thresholds. For H.265 (Fig. 2(b)), 75% of Q4 chunks have SI and TI larger than 25 and 7, respectively, while only 5% of Q1 chunks and 14% of Q2 chunks have SI and TI larger than these thresholds. To verify Property (2), let $\{c_{\ell,1}, \ldots, c_{\ell,n}\}$ denote the sequence of the chunk categories for track/level ℓ, where $c_{\ell,i} \in \{1, 2, 3, 4\}$ represents the category of the ith chunk in track ℓ, and n is the number of chunks in a track. We then calculate the correlation between each pair of tracks. Our results confirm that all the correlation values are close to 1, indicating that chunks with the same index (i.e., at the same playback position) are indeed in consistent categories across tracks.

The above two properties together suggest that we can classify chunks into different categories of scene complexity simply based on chunk size. For consistent classification across tracks, we can leverage the consistent relative sizes across tracks, use one track as the reference track, and classify chunks at different playback positions based on their chunk sizes in that track (so that chunks with the same index will be in the same class). Specifically, we choose a reference track (e.g., a middle track), and classify the chunks at different playback positions into four categories, Q1, Q2, Q3, and Q4 chunks, based on which quartile that the size of a chunk falls into. After that, all chunks in the same playback positions are placed into the same category, regardless of the tracks.

The above classification method is based on quartiles. Other methods can also be used (e.g., using five classes instead of four

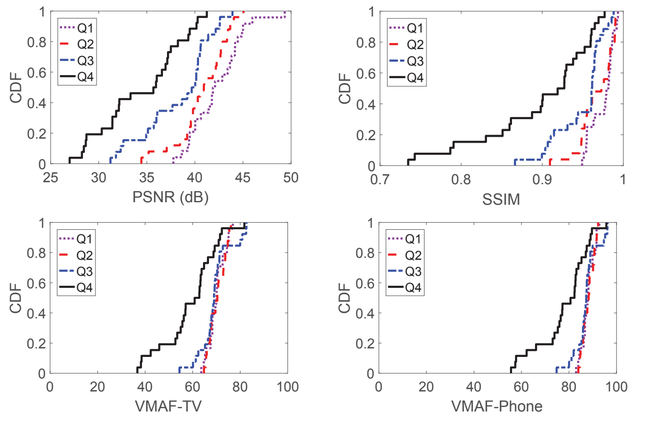

Figure 3: Quality of chunks in a VBR video (Elephant Dream, YouTube encoded, H.264).

classes); our design principles (§4) and rate adaptation scheme (§5) are independent of this specific classification method.

3.1.2 Scene Complexity and Encoding Quality. After classifying chunks into the four categories as above, we now analyze the encoding quality of the chunks in each category. VBR encoding allocates fewer bits to simple scenes and more bits to complex scenes so as to maintain a constant quality across the scenes [12]. We show below, however, the quality across the scenes is not constant in a specific track. Specifically, we use three quality metrics, Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index (SSIM) [43], and Video Multimethod Assessment Fusion (VMAF) [24, 27, 50]. PSNR is a traditional image quality metric. The PSNR of a chunk is represented as the median PSNR of all frames in that chunk. SSIM is a widely used perceptual quality metric; it utilizes local luminance, contrast and structure measures to predict quality. VMAF is a recently proposed perceptual quality metric that correlates quality strongly with subjective scores and has been validated independently in [34]. There are two VMAF models: *TV model* for larger screens (TV, laptop, tablet), and *phone model* for smaller screens (phones). The aggregate VMAF of a chunk can be calculated in multiple ways [30]. We use the median VMAF value of all frames in a chunk as the VMAF value for the chunk (using mean leads to similar values for the videos in our dataset).

Fig. 3 plots the CDF of the quality, measured in PSNR, SSIM, VMAF TV, and VMAF phone model, for a YouTube-encoded video. For each quality metric, the results of a middle track (480p) are shown in the figure. We observe that, while Q1 to Q4 chunks have increasing sizes (in bits), they have decreasing quality. In addition, the quality gap between Q4 and Q1–Q3 chunks is particularly large, despite that Q4 chunks have the most bits. This is because it is challenging to encode complex scenes to have the same quality of simple scenes. We make similar observations for videos encoded using `FFmpeg` and H.265 encoded videos (figures omitted).

3.2 Implications for Rate Adaptation

As described above, we have identified two important characteristics of VBR encoding: *(1) the relative size of a chunk (relative to the other chunks in the same track) is a good indication of the scene complexity, and (2) complex scenes, despite being encoded with more bits, have inferior qualities compared to other scenes in the same track, across a range of quality metrics.* They both have important implications on rate adaption during VBR streaming process.

In particular, the second observation is opposite to what one would like to achieve during playback, where having higher quality for complex scenes is more important [23]. It indicates that complex scenes need to receive differential treatment to achieve a higher quality, which requires (i) inferring the scene complexity, and (ii) carefully designing ABR streaming algorithms. The first requirement can be achieved based on our first observation, by using chunk sizes to infer scene complexity. It has two advantages compared to using those based on video content (e.g., SI and TI). First, it is simple to compute. Second, chunk sizes are already known at the client in current dominant ABR standards[1]. Achieving the second requirement is very challenging. Given that complex scenes are already encoded with more bits than other scenes in the same track, choosing higher tracks for complex scenes indicates that a significant amount of network bandwidth needs to be allocated to stream complex scenes, which may come at the cost of lower tracks for simple scenes. We therefore need to design ABR streaming schemes to judiciously use network bandwidth, favoring Q4 chunks while not degrading the quality of other chunks too drastically (§4, §5).

3.3 VBR with Larger Cap

In this paper, we mainly use 2× capped VBR videos. The bitrate cap, while limits the amount of bitrate variability and makes it easier to stream VBR videos over the Internet, also limits the quality of the videos, particularly for complex scenes. The recommendation on VBR encoding has been evolving. The original recommendation was to set the cap to 1.1× [2], and has now evolved to 2× [3]. It is conceivable that the cap will be even larger in the future. For this reason, we further encode one of the publicly available videos to be 4× capped. We observe that the characteristics described in §3 still hold for this video. Specifically, Q4 chunks, even under 4× cap, are still of significantly lower quality than other chunks. As an example, for the middle track (480p), the median VMAF value (phone model) for the Q4 chunks is 79, significantly lower than the values of 88, 88, and 85 for Q1 to Q3 chunks. This might be because it is inherently very difficult to encode complex scenes to reach the same quality as simple scenes [5]. Therefore, for VBR videos of larger caps, we still need to incorporate these characteristics when designing ABR schemes (§6.6).

4 DESIGN PRINCIPLES

We believe that a good rate adaptation scheme for VBR videos must (i) take the per-chunk bitrate information into consideration (also suggested by [46]), which is available to the clients in dominant ABR standards (see §3.2), and (ii) use per-chunk bitrate information properly, by incorporating the characteristics of VBR videos. We next propose three design principles for VBR video rate adaptation.

Be Non-myopic (P1). This requires considering not only the bitrate of the immediate next chunk, but also the bitrate of multiple future chunks. A myopic scheme only considers the bitrate of the immediate next chunk, leading to level selections that only match the current resource (in terms of network bandwidth or player

[1]Chunk size information is included in the manifest file sent from server to client in DASH, and more recently HLS has added this feature [46].

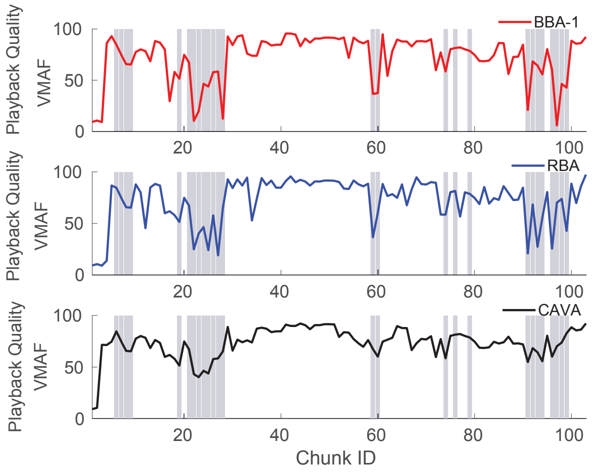

Figure 4: Two myopic schemes and CAVA.

buffer level). As a result, a myopic scheme oftentimes mechanically selects very high (low) levels for chunks with small (large) sizes – exactly the opposite to what is desirable for VBR videos (§3). We illustrate this using two myopic schemes: (i) BBA-1 [16], a buffer-based scheme, and (ii) a rate-based scheme [49], referred to as RBA henceforth. Both schemes are myopic: when selecting the track for the next chunk, BBA-1 selects the highest track based on a *chunk map*, which defines the allowed chunk sizes as a range from the average chunk size of the lowest track to that of the highest track; RBA selects the highest track so that after downloading the corresponding chunk, the player buffer will still contain at least four chunks, where the downloading time of a chunk is obtained as its size divided by the estimated network bandwidth. Fig. 4 shows an example that illustrates the performance of BBA-1 and RBA (the top two plots), where the shaded bars mark the playback positions of Q4 chunks. We observe that indeed BBA-1 and RBA tend to lead to low qualities for Q4 chunks (recall that these tend to correspond to complex scenes) and high qualities for Q1-Q3 chunks (with simpler scenes). For comparison, we also plot our scheme CAVA (to be described in §5) in Fig. 4. In this example, for BBA-1 and RBA, the average VMAF quality of Q4 chunks is 49 and 52, and the amount of rebuffering is 6s and 4s, respectively; both are significantly worse than our scheme CAVA, under which the average VMAF quality of Q4 chunks is 65 with no rebuffering.

Offer Differential Treatment (P2). This principle favors more complex scenes by explicitly accounting for the chunk size and quality characteristics of VBR videos (§3.2). The differential treatment can be achieved by using resources saved from simple scenes for complex scenes. While this suggests potentially lower quality for simple scenes, a good VBR rate adaptation scheme needs to achieve a balance: improve the quality of complex scenes while not degrading too much the quality of simple scenes. This principle is particularly important when complex scenes have lower quality than simple scenes in the same track, as we have observed in §3.1.2, since in this case, it is preferable to choose a higher track for complex scenes to achieve comparable quality as simple scenes. Even in an ideal VBR encoding, where complex scenes have similar quality as simple scenes in the same track, this principle is still important, since complex scenes, being of larger sizes, naturally need more

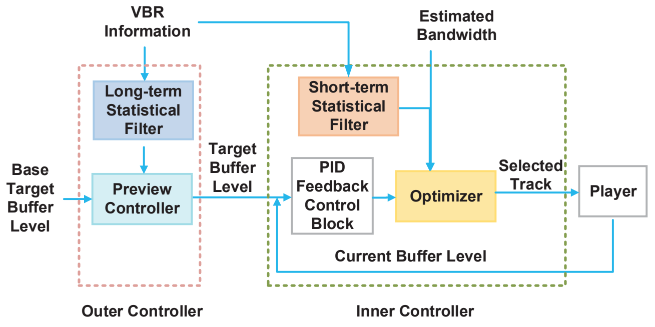

Figure 5: CAVA design diagram.

resources than simple scenes in the same track. Offering differential treatment thus helps the VBR algorithm make judicious rate adaptation decisions when the network bandwidth is limited.

Be Proactive (P3). Since VBR videos have high bitrate variability and the size/bitrate of future chunks is known beforehand, it is desirable to account for the variability proactively. This is especially true for complex scenes corresponding to large chunks. Suppose when making a decision at time t, the player notices that consecutive positions after t contain complex scenes. If the playback buffer is low and the network bandwidth is low, then rebuffering will inevitably occur. Fortunately, with the known chunk size information, the rate adaptation algorithm can, instead, react to the cluster of large chunks *early*—before time t instead of right at t—by, for example, pre-adjusting the target buffer level for buffer-related approaches.

Discussion. Several existing schemes [23, 33, 47] use a window-based approach that considers multiple future chunks in bitrate adaptation, which differs from our non-myopic principle that is motivated specifically by the dynamic chunk bitrates/sizes of VBR videos (§5.3). Note that **P1** and **P3** share in common that they both require considering future chunks. But they differ mainly in the timescale: the non-myopic principle considers a small number of future chunks to account for the variability of VBR videos in the short timescale, in order to facilitate the *current* decision; the proactive principle, on the other hand, considers chunks further away in future to benefit *future* decisions in a proactive manner.

5 CAVA DESIGN AND IMPLEMENTATION

We now instantiate the three principles (§4) and design a control-theoretic rate adaptation scheme, CAVA, for VBR streaming. The reason for adopting a control-theoretic approach is because it has shown promise in adapting to dynamic network bandwidth [33, 47]. The design of CAVA is compatible with the current dominant ABR standards (DASH and HLS), and can be readily used in practice.

5.1 Overview of CAVA

We design CAVA to address both chunk size variability, an inherent feature of VBR videos, and quality variability in VBR videos, a characteristic that we identified in today's VBR encodings (§3.1). Even under an ideal VBR encoding where complex scenes have similar quality as other scenes in the same track, CAVA still has its value in addressing the inherent chunk size variability of VBR videos.

Fig. 5 shows the design diagram for CAVA, which builds on the basic feedback control framework [33]. CAVA introduces significant innovations by (i) generalizing the control framework from plain CBR to VBR, (ii) designing an inner controller that reacts in short timescale, and incorporates the *non-myopic* and *differential treatment principles*, and (iii) designing an outer controller that reacts over a longer timescale, and incorporates the *proactive principle*.

The control framework uses PID control [4], a widely used feedback control technique. The reason for using PID control is its good performance and robustness [33]. As shown in Fig. 5, CAVA consists of two controllers that work in synergy to realize the aforementioned three principles.

• The *inner controller* is responsible for selecting the appropriate track level. It employs a PID feedback control block that maintains the target buffer level by properly adjusting the control signals (§5.2). Then based on the controller's output, together with the estimated bandwidth and a short-term average of future chunks' bitrate, CAVA invokes an optimization algorithm that incorporates non-myopic (**P1**) and differential treatment (**P2**) principles to perform VBR-aware track level selection (§5.3).

• The *outer controller*'s job is to determine the target buffer level that is used by the inner controller. When doing so, CAVA exercises proactive principle (**P3**) to adjust the target buffer level proactively based on dynamics of future chunks (§5.4).

5.2 PID Feedback Control Block

The PID feedback control continuously monitors an "error value", i.e., the difference between the target and current buffer levels of the player, and adjusts the control signal to maintain the target buffer level. To account for the characteristics of VBR videos, it considers per-chunk size information and uses a dynamic target buffer level. This differs from the PID controller design for CBR [33], which uses a fixed target buffer level and a fixed average bitrate for each track.

Specifically, let C_t denote the network bandwidth at time t. Let ℓ_t denote the track number selected at time t, with the corresponding bitrate of the chunk denoted as $R_t(\ell_t)$. We use $R_t(\ell_t)$ instead of $R(\ell_t)$ to reflect that it is a function of time t since for VBR videos, the bitrate varies significantly even inside a track. The controller output, u_t, is defined as

$$u_t = \frac{C_t}{R_t(\ell_t)}, \tag{1}$$

which is a unitless quantity representing the relative buffer filling rate. The control policy is defined as

$$u_t = K_p(x_r(t) - x_t) + K_i \int_0^t (x_r(t) - x_\tau)d\tau + \mathbf{1}(x_t - \Delta) \tag{2}$$

where K_p and K_i denote respectively the parameters for proportional and integral control (two key parameters in PID control), x_t and $x_r(t)$ denote respectively the current and target buffer level at time t (both in seconds), Δ denotes the playback duration of a chunk, and the last term, $\mathbf{1}(x_t - \Delta)$, is an indicator function (it is 1 when $x_t \geq \Delta$ and 0 otherwise), which makes the feedback control system linear, and hence easier to control and analyze. The target buffer level is dynamic, set by the outer controller (§5.4).

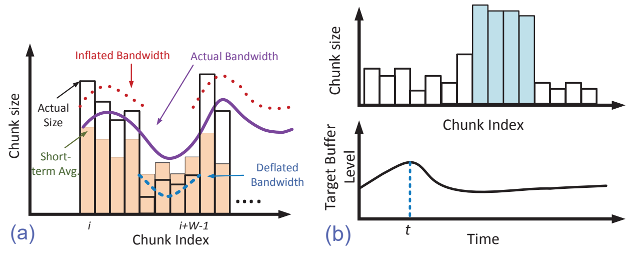

Figure 6: Illustration of (a) track selection, and (b) setting target buffer level.

5.3 The Inner Controller

The inner controller determines ℓ_t, the track at time t. Based on the definition in Eq. (1), once the controller output u_t is determined using Eq. (2), we can simply determine ℓ_t so that the corresponding bitrate $R_t(\ell_t)$ is the maximum bitrate that is below $\widehat{C}_t/u_t$, where $\widehat{C}_t$ is the estimated network bandwidth at time t. This approach is, however, myopic (only considering the next chunk), and is thus undesirable for VBR videos.

We next formulate an optimization problem that accounts for both the non-myopic and differential treatment principles (§4) to determine ℓ_t. Fig. 6(a) illustrates the approach. Suppose at time t, we need to select the track number for chunk i. Following the non-myopic principle (**P1**), we consider a window of W chunks, from i to $i + W - 1$, and use the average bitrate of these W chunks to represent the bandwidth requirement of chunk i. This is handled by the "Short-term Statistical Filter" in Fig. 5. Using the average bitrate of W chunks (instead of the bitrate of chunk i) leads to smoother bandwidth requirement from the chunks, avoiding mechanically choosing very high (low) levels for chunks with small (large) bitrates.

To provide differential treatment (**P2**) to chunks with different complexities, we assume either *inflated* or *deflated* network bandwidth, depending on whether chunk i contains complex or simple scenes. For complex scenes, we assume inflated network bandwidth, allowing a higher track to be chosen, while for simple scenes, we assume deflated bandwidth so as to save bandwidth for complex scenes. The track selection algorithm maximizes the quality level to match the assumed network bandwidth, while minimizing the number of quality level changes between two adjacent chunks. In the example in Fig. 6(a), chunk i is a Q4 chunk, and hence we assume inflated bandwidth, which allows a higher level track to be selected.

Optimization Formulation. We next describe our optimization formulation. Our current design uses track level to approximate the quality since this is currently the only available quality information in predominant ABR standards; using such information only allows CAVA to be easily deployable. In our performance evaluation (§6), we use a perceptual quality metric (VMAF) to evaluate CAVA, and show that it achieves good perceptual quality although the formulation does not explicitly use a perceptual quality metric.

The optimization aims to minimize a weighted sum of two penalty terms: the first term represents the deviation of the selected bitrate from the estimated network bandwidth, and the second term represents the track change relative to the previous chunk. Other formulations might also be possible; our empirical results show that this formulation works well in practice. Specifically, we aim to minimize

$$Q(\ell_t) = \sum_{k=t}^{t+N-1} \left(u_k \bar{R}_t(\ell_t) - \alpha_t \widehat{C}_k\right)^2 + \eta_t \left(r(\ell_t) - r(\ell_{t-1})\right)^2, \qquad (3)$$

where u_k and $\widehat{C}_k$ are respectively the controller output and the estimated link bandwidth for the k-th chunk, $\eta_t \geq 0$ is a parameter that represents the weight of the second term, $\bar{R}_t(\ell_t)$ is the average bitrate over a window of W chunks (according to **P1**), from t to $t+W$, at track ℓ_t (here we slightly abuse the notation to use t to represent the chunk index for time t), and $r(\ell)$ denotes the average bitrate of track ℓ.

Penalty on deviation from estimated bandwidth. The first term in Eq. (3) represents the penalty based on how much the required bandwidth deviates from the estimated network bandwidth. Specifically, minimizing this penalty term aims at selecting the largest ℓ_t (and hence the highest quality) so that the difference between the required bandwidth, $u_k \bar{R}_t(\ell_t)$, and the assumed bandwidth, $\alpha_t \widehat{C}_k$, is minimized (α_t is a factor that deflates or inflates the estimated bandwidth; see details later on). Following the non-myopic principle (**P1**), we use $\bar{R}_t(\ell_t)$ (instead of $R_t(\ell_t)$), which is the average bitrate of W future chunks, to represent the bandwidth requirement of chunk t. We refer to W as the *inner controller window size* and explore its configuration in §6.2.

When operating online, CAVA only has a limited view towards future network conditions. Therefore, the penalty term on deviation from estimated bandwidth considers the bandwidth needed to download a finite horizon of N future chunks. We assume that these N chunks are all from the same track, ℓ_t, to reduce the number of quality level changes. In the rest of the paper, we set N to 5 chunks, a relatively short horizon to accommodate network bandwidth dynamics.

Note that several existing schemes [23, 33, 47] also examine a finite horizon of chunks in their online algorithms (controlled by N in CAVA). But none of them utilizes concepts like our non-myopic principle in computing the bandwidth requirement from multiple future chunks (controlled by W in CAVA), whose sizes may vary significantly in VBR, for rate adaptation.

Penalty on track change. The second term in Eq. (3) represents the penalty on track change. Specifically, minimizing the second term aims at selecting ℓ_t so that its difference from ℓ_{t-1}, i.e., the track of the previous chunk, is minimized. We consider two categories of chunks, Q4 and non-Q4 chunks when setting η_t, a weight factor. If the current and previous chunks fall in different categories, we set η_t to 0; otherwise we set η_t to 1 (so as to give equal weight to the two terms in Eq. (3) since both are important for QoE). That is, we do not penalize quality change if two consecutive chunk positions are in different categories (and hence of different importance in viewing quality).

Recall that $r(\ell)$ denotes the average bitrate of track ℓ. The penalty term on track change uses $r(\ell_t) - r(\ell_{t-1})$, instead of $\ell_t - \ell_{t-1}$ or $R_t(\ell_t) - R_t(\ell_{t-1})$. This is because $r(\ell_t) - r(\ell_{t-1})$ reflects track change, and yet is in the unit of bitrate, which is the same as that of the first term in Eq. (3). The expression $\ell_t - \ell_{t-1}$ represents track change directly, whose unit is however different from that of the first term in Eq. (3); the expression $R_t(\ell_t) - R_t(\ell_{t-1})$ represents bitrate change on a per chunk level, which is not meaningful for VBR videos since even chunks in the same track can have highly dynamic bitrate.

Deflating/inflating network bandwidth estimate. In Eq. (3), $\alpha_t \widehat{C}_k$ represents the assumed network bandwidth for chunk k. As mentioned earlier, according to differential treatment principle (**P2**), we set $\alpha_t > 1$ for complex scenes to inflate the bandwidth, and set $\alpha_t < 1$ for simple scenes to deflate the bandwidth. Determining the value of α_t presents a tradeoff: a large value for complex scenes can lead to higher quality level, at the cost of potential stalls; similarly, a small α_t value for simple scenes, while can save bandwidth and eventually benefit complex scenes, may degrade the quality of simple scenes too much. We have varied α_t for complex scenes from 1.1 to 1.5, and varied α_t for simple scenes from 0.6 to 0.9. In the rest of the paper, we set $\alpha_t = 1.1$ for Q4 chunks and $\alpha_t = 0.8$ for Q1-Q3 chunks, which lead to a good tradeoff observed empirically.

In addition, we use the following heuristic for Q1-Q3 chunks: if the current chunk is a Q1-Q3 chunk and the selected level is very low (i.e., level 1 or 2) under the above strategy, while the buffer level is above a threshold (chosen as 10 seconds in §6) indicating low risk of stalls, we do not deflate network bandwidth, and instead use $\alpha_t = 1$. This heuristic avoids choosing unnecessarily low levels for Q1-Q3 chunks. Similarly, we can use a heuristic for Q4 chunks: if the current chunk is a Q4 chunk and the buffer level is below a threshold indicating high risk of stalls, we can also set α_t to 1 (i.e., do not inflate network bandwidth). This heuristic can lead to lower stalls at the cost of potentially reducing the quality of Q4 chunks. Our evaluations show that in the settings that we explore, the number of stalls in CAVA is already very low even without using this heuristic. Therefore, in §6, we only report the results when this heuristic is disabled.

Time complexity. Let $\mathcal{L}$ be the set of all track levels. The optimal track is

$$\ell_t^* = \arg\min_{\ell_t \in \mathcal{L}} Q(\ell_t). \qquad (4)$$

We can find ℓ_t^* easily by evaluating (3) using all possible values of $\ell_t \in \mathcal{L}$, and selecting the value that minimizes the objective function. For every $\ell_t \in \mathcal{L}$, obtaining $Q(\ell_t)$ requires computation of N steps. Therefore, the total computational overhead of solving (4) is $O(N|\mathcal{L}|)$.

5.4 The Outer Controller

When using the inner controller alone, we observe that rebuffering sometimes happens when the network bandwidth is very low and the buffer level is also low (due to relatively large chunks downloaded earlier). While the inner controller reacts by selecting the track with the lowest bitrate, the reaction appears to be too late. To deal with such cases, we incorporate the proactive principle (**P3**) by adopting the concept of *preview control* from control theory [40] to adjust the target buffer level proactively.

Fig. 6(b) illustrates our approach. The target buffer level is set to the base target buffer level, added by a term that is determined by future chunk size variability. Specifically, when there are large chunks in the future, and hence a higher chance of longer downloading time and slower buffer filling rate that can lead to buffer underrun, the target buffer level is adjusted to a higher level to react proactively (handled by the "Long-term Statistical Filter" in Fig. 5), as illustrated in the region around t in Fig. 6(b). This adjustment will reduce rebuffering because the controller aims to reach a higher target buffer level.

Specifically, let x_r denote a base target buffer level (we set it to 60 seconds in §6; setting it to 40 seconds leads to similar results). The target buffer level at time t, $x_r(t)$, is set to be x_r added by a term that accounts for the future chunk size variability. We use a track $\tilde{\ell}$ (e.g., a middle track) as the reference track. If the average size of the future chunks in that track is above the mean value of that track, we proactively increase the target buffer level as:

$$x_r(t) = x_r + \max\left(\frac{\Sigma_{k=t}^{t+W'} R_k(\tilde{\ell})\Delta - r(\tilde{\ell})W'\Delta}{r(\tilde{\ell})}, 0\right), \qquad (5)$$

where W' is referred to as *outer controller window size* that represents how much to look ahead, Δ is the chunk duration, $r(\tilde{\ell})$ is the average bitrate of the reference track $\tilde{\ell}$, and $R_t(\tilde{\ell})$ represents the bitrate of the chunk at time t (again we slightly abuse the notation to use t to represent chunk index). In the second term in Eq. (5), the numerator represents how much (in bits) the sum of the next W' chunks deviates from the average value; dividing the numerator by $r(\tilde{\ell})$ converts the unit of the deviation to seconds, compatible with how we represent buffer level. To avoid pathological scenarios where $x_r(t)$ becomes too large, we limit its value to $2x_r$, which is chosen empirically. The choice of W' is explored in §6.2.

5.5 Implementation

We have implemented CAVA in `dash.js`, a production quality open-source web-based DASH video player [15] (version v2.7.0). Under the `dash.js` framework, we implemented a new ABR streaming rule `CAVARule.js` (consisting of 520 LoC) to realize CAVA. In addition, the interface of `dash.js` only exposes limited information including the track format, buffer occupancy level, bandwidth estimation, and declared bitrate. The segment size information, however, is not accessible by default. We therefore add a new `LoadSegmentSize` class to expose such information to the ABR logic. We further develop a bandwidth estimation module that responds to playback progress events to estimate throughput using a harmonic mean of the past 5 chunks.

6 PERFORMANCE EVALUATION

We explore CAVA and its parameter settings across a broad spectrum of real-world scenarios (covering both LTE and broadband), using 16 video clips (§2). Each video is around 10 minutes long. The length is much longer than network RTT and the timescale of ABR rate adaptation. In addition, it allows the player to go beyond the start-up phase, allowing us to understand both start-up and longer-term behaviors. These videos cover different content genres (i.e., animation, science fiction, sports, animal, nature and action), track encodings (Youtube and `FFmpeg` [13]), codecs (H.264 [18] and H.265 [17]), and chunk durations (2 and 5 seconds).

6.1 Evaluation Setup

To ensure repeatable experimentations and evaluate different schemes under identical settings, we use real-world network trace-driven replay experiments. To make the large state space exploration scalable and tractable, we use both trace-driven simulations, and experiments with our prototype implementation in `dash.js`.

We use two sets of network traces. The first set, *LTE*, contains 200 cellular network traces we captured in commercial LTE networks with a collaborator driving coast-to-coast across the US. The LTE traces were represented as per-second network throughput, recorded when downloading a large file from a well-provisioned server to a mobile phone. The second set, *FCC*, contains 200 broadband traces, randomly chosen from the set of traces collected by the FCC [10], represented as per-5 second network throughput. Each trace contains at least 18 minutes of bandwidth measurements.

ABR logic operates at the application level, using application-level estimation of the network bandwidth (based on the throughput for recently downloaded chunks) as part of the decision process to determine the quality of the next chunk to download [16, 20, 46]. The impact of lower level network characteristics such as signal strength, loss, and RTT on ABR adaptation behavior is reflected through their impact on the network throughput. Therefore, our evaluation is through replaying the network traces that capture the network bandwidth variability over time.

ABR Schemes. We compare CAVA with the following state-of-the-art rate adaptation schemes:
• MPC and RobustMPC [47], which are based on model predictive control. RobustMPC is shown to outperform MPC under more dynamic network bandwidth settings [28].
• PANDA/CQ [23], which incorporates video quality information to optimize the performance of ABR streaming using dynamic programming. Specifically, it considers a window of N future chunks while making decisions. We investigate two variants, (i) *max-sum* that maximizes the sum of the quality of the next N chunks, and (ii) *max-min* that maximizes the minimum quality of the next N chunks.
• BOLA-E [37], which selects the bitrate to maximize a utility function considering both rebuffering and delivered bitrate. It is an improved version of BOLA [38].
• BBA-1 [16] and RBA [49]. Both are myopic schemes (§4). Through extensive evaluation, we find that, consistent with the example in §4, CAVA significantly outperforms these two schemes in all cases. In the interest of space, we omit the results for these two schemes.

Since our focus is VBR-encoded videos, following the recommendation of each scheme described above, we use the actual size of a video chunk in making rate adaptation decisions. Note that, of all the schemes, only PANDA/CQ relies on using video quality information. Such information, however, is not available in the dominant ABR protocols used today (i.e., HLS and DASH), limiting the practical applicability and deployability of such a scheme. Still, using video quality information as part of the adaptation decision process can be potentially valuable, and for that reason, we include PANDA/CQ in our evaluations. As we shall see later in §6.3, while CAVA does not use such video quality information and relies only on information available in ABR streaming protocols, it still outperforms PANDA/CQ. Learning based approaches such as [28] require extensive training on a large corpus of data, and are not included for comparison since our focus is on pure client-side based schemes that are more easily deployable.

ABR Configurations. An important parameter in all schemes is the playback startup latency, i.e., the minimum number of seconds worth of video data that needs to be downloaded before the client begins playback. We explore a range of values for this parameter, guided by commercial production players [46], and report results for one setting – 10 seconds (results for other practical settings were similar). A startup delay of 10 seconds corresponds to having two chunks in the buffer for chunk length of 5 seconds (as the case for YouTube videos), consistent with the recommendation of having at least two to three chunks before starting playback [46]. Another important factor is the network bandwidth estimation technique used by the ABR logic. For a fair comparison, for all the schemes that need bandwidth estimation, we use the harmonic mean of the past 5 chunks as the bandwidth prediction approach, as the technique has been shown to be robust to measurement outliers [20, 47]. We evaluate the impact of inaccurate bandwidth estimation on performance in §6.7. Unless otherwise stated, the maximum buffer size is set to 100 seconds for all the schemes for apple-to-apple comparison. The client does not download the next chunk when the maximum buffer size is reached.

For CAVA, the base target buffer level, x_r, is set to 60 seconds. The controller parameters, K_p and K_i, are selected adopting the methodology outlined in [33]. Specifically, we varied K_p and K_i, and confirmed that, similar to [33], a wide range of K_p and K_i values lead to good performance.

Performance Metrics. We use five metrics: four for measuring different aspects of user QoE and one for measuring resource efficiency, i.e., network data usage. All metrics are computed with respect to the delivered video, i.e., considering the chunks that have been downloaded and played back. We measure the video quality using VMAF (a recently proposed perceptual quality metric from Netflix, see §3.1). Specifically, we use the VMAF phone model (covers typical smartphone viewing settings) for the evaluations with the cellular traces, and the VMAF TV model (covers typical larger screen TV viewing settings) for the evaluations with the FCC traces, based on the typical device types and viewing conditions prevalent in cellular and home networks, respectively.

The performance metrics are (i) *quality of Q4 chunks*: captures the video quality for the most complex scenes. (ii) *low-quality chunk percentage*: among all the chunks played back in a given streaming session, this measures the proportion of chunks that were selected with low video quality. For the experiments, we identify VMAF values below 40 (representing poor or unacceptable quality [50]) as low quality. (iii) *rebuffering duration*: measures the total rebuffering/stall time in a video session. (iv) *average quality change per chunk*: is defined as the average quality difference of two consecutive chunks in playback order (since human eyes are more sensitive to level changes in adjacent chunks) for a session, i.e., $\sum_i |q_{i+1} - q_i|/n$, where q_i denotes the quality of i^{th} chunk (in terms of playback order), and n is the number of chunks. (v) *data usage*: captures the total amount of data downloaded (i.e., the total network resource usage) for a session. While comparing two schemes, for each of the last 4 metrics, a lower value is preferable; the only exception is the first metric (quality of Q4 chunks), where a higher value is preferable.

In the above performance metrics, we use perceptual quality metrics since they quantify the viewing quality more accurately compared to bitrate that is commonly used in the literature. As an example, the average bitrate of the delivered video, which is commonly used to quantify the quality of a streaming session, is an especially poor metric for VBR videos. To see this, consider two cases with the same average bitrate. In the first case, the lowest track is selected for Q4 chunks, while higher tracks are selected for Q1-Q3 chunks; in the second case, a medium track is selected for Q4 chunks, while low or medium tracks are selected for Q1-Q3 chunks. These two cases clearly lead to different viewing quality, while will be classified as being the same quality based on the average bitrate. In addition, we use metrics to capture both the average and tail behaviors. Specifically, the percentage of low quality chunks is an important metric on tail behavior since it has been reported that human eyes are particularly sensitive to bad quality chunks [30].

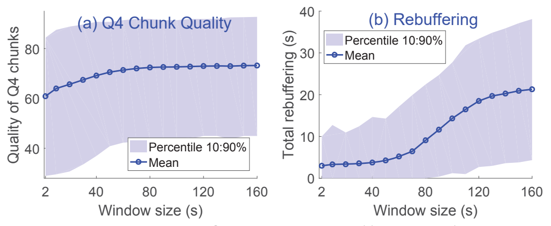

Figure 7: Impact of inner controller window size.

6.2 Choice of Parameters for CAVA

Inner controller window size. Recall that the inner controller in CAVA uses the average bitrate over a window of W future chunks to represent the bandwidth requirement of the current chunk. We investigate the impact of W on the performance of CAVA. Fig. 7 shows an example (Elephant Dream, `FFmpeg` encoded, H.264, LTE traces); the solid lines represent the average values, and the top and bottom of the shaded region represent the 90th and 10th percentile across the network traces, respectively. When increasing W, (i) the quality of the Q4 chunks first improves significantly and then flattens out, and (ii) the amount of rebuffering increases slightly and then sharply. Increasing W at the beginning is beneficial because even averaging over a few chunks can smooth out the bitrate of the chunks, which allows higher levels to be chosen for Q4 chunks (since the smoothed bitrate for a Q4 chunk tends to be lower than its actual bitrate). The impact on smoothing the bitrate diminishes after W becomes sufficiently large, leading to diminishing gains for Q4 chunks. The increase in rebuffering when W is large is because CAVA becomes less sensitive to the bitrate changes in that case. We set W to 40 seconds (i.e., 20 and 8 chunks for 2 and 5 seconds chunk durations, respectively) as this leads to a good tradeoff between quality and rebuffering.

Outer controller window size. The outer controller in CAVA uses a window of W' future chunks to adjust the target buffer level. We vary W' to explore its impact. In general, the amount of rebuffering decreases as W' increases since the controller reacts more proactively with larger W'. For some videos, however, the amount of rebuffering may start to increase as W' increases further. This is because using very large W' smoothes out the effect of the variability of the video (i.e., the average bitrate of the future W' chunks becomes closer to the average bitrate of the track, causing little increment in the target buffer level as shown in Eq. (5)). We set W' to 200 seconds (i.e., 100 and 40 chunks for 2 and 5 seconds chunk durations, respectively), which leads to good results in reducing rebuffering.

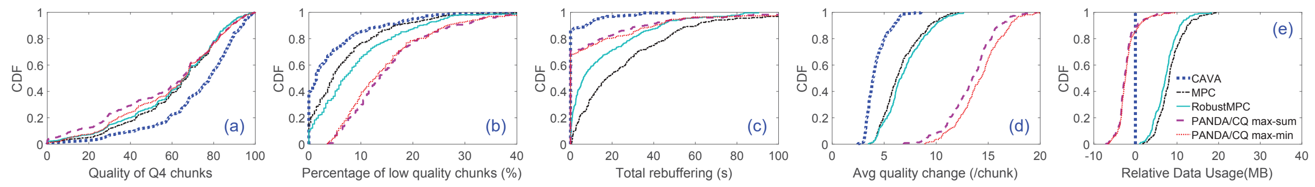

Figure 8: Performance comparison for one video (Elephant Dream, FFmpeg encoded, H.264) under LTE traces.

6.3 Comparison with Other Schemes

We compare CAVA with other ABR schemes using all the videos under both LTE and FCC traces (the comparison with BOLA-E is deferred to §6.8). Fig. 8(a)-(e) compare the performance of CAVA with other schemes for one `FFmpeg` encoded video under the LTE traces. Our results below focus on CAVA versus RobustMPC and PANDA/CQ max-min; MPC can have significantly more rebuffering than RobustMPC, and PANDA/CQ max-sum can have significantly lower quality for Q4 chunks than PANDA/CQ max-min. We observe that CAVA significantly outperforms RobustMPC and PANDA/CQ max-min. (i) With CAVA, a much larger proportion (79%) of the Q4 chunks are delivered at a quality higher than 60 (considered as good quality [50]), compared to only 59% and 57% for RobustMPC and PANDA/CQ max-min, respectively. The median VMAF value across Q4 chunks is 78 under CAVA, 11 and 12 larger than the corresponding values for RobustMPC and PANDA/CQ max-min, respectively, indicating noticeable perceptual improvements (It has been reported that a difference of 6 or more in VMAF would be noticeable to a viewer [31]). (ii) For the percentage of low-quality chunks, under CAVA, 40% of the traces have no low-quality chunks (VMAF score < 40); and 82% of the traces have less than 10% low-quality chunks, significantly larger than the corresponding values 65% and 40% for RobustMPC and PANDA/CQ max-min, respectively. (iii) CAVA has no rebuffering for 85% of the traces, compared to only 20% and 68% of traces under RobustMPC and PANDA/CQ max-min, respectively. (iv) For average quality change per chunk, CAVA realizes substantially lower changes than the other schemes (on average, 67% and 29% of the values of RobustMPC and PANDA/CQ max-min, respectively), resulting in less variability in playback quality, and therefore more consistent and smoother playback experience. (v) CAVA has 5% to 40% lower data usage than RobustMPC. PANDA/CQ max-min has slightly lower data usage than CAVA (10% at most), at the cost of lower Q4 quality and overall quality (see below). Figures 9(a) and (b) show the quality of Q1-Q3 chunks and all the chunks, respectively. We observe that while CAVA does not lead to very high quality for Q1-Q3 chunks, it does not choose low quality for these chunks either. Overall, CAVA leads to a desired tradeoff in choosing less low-quality chunks compared to other schemes.

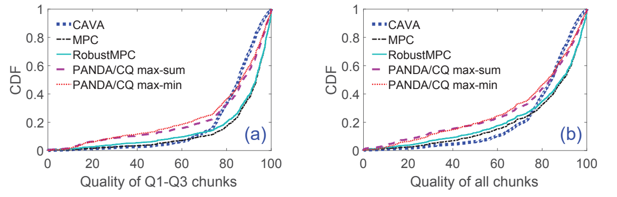

Figure 9: Quality of Q1-Q3 chunks and all the chunks.

The above results are for one `FFmpeg` video under LTE traces. Table 1 shows the results for several YouTube videos under both LTE and FCC traces. For Q4 chunk quality, we show two values: the metric value (averaged over all the runs) obtained by CAVA subtracted by that of RobustMPC and PANDA/CQ max-min, respectively. For each of the other four performance metrics, we show two percentage values, (i) the difference in the metric value (averaged over all runs) for CAVA subtracted by that for RobustMPC, as a percentage of the latter value, and (ii) the corresponding value for CAVA vs. PANDA/CQ max-min. In Table 1, ↑ indicates that CAVA has a higher value; while ↓ indicates that CAVA has a lower value. The first part of Table 1 shows the results under LTE traces. We observe that CAVA leads to average quality improvement for Q4 chunks in the range of 8-18 VMAF points compared to RobustMPC, and 3-9 VMAF points compared to PANDA/CQ max-min. Also CAVA substantially reduces the average stall/rebuffering duration by 62%-95%, reduces the quality change per chunk by 25%-48%, and reduces the percentage of low-quality chunks by 4%-61%. Last, the data usage of CAVA is 7%-11% lower than RobustMPC, and 2%-10% lower than PANDA/CQ max-min.

Table 1: Performance comparison (YouTube videos).

	Video	Q4 chunk quality	Low-qual. chunks (%)	Stall duration (%)	Quality changes (%)	Data usage (%)
LTE	BBB	↑13, ↑4	↓61, ↓31	↓62, ↓64	↓48, ↓37	↓11, ↓5
	ED	↑10, ↑4	↓54, ↓37	↓73, ↓77	↓38, ↓36	↓9, ↓4
	Sintel	↑8, ↑4	↓75, ↓42	↓83, ↓77	↓37, ↓32	↓8, ↓3
	ToS	↑9, ↑6	↓72, ↓55	↓84, ↓81	↓39, ↓40	↓9, ↓3
	Animal	↑15, ↑3	↓4, ↓7	↓83, ↓95	↓43, ↓38	↓11, ↓10
	Nature	↑10, ↑3	↓17, ↓11	↓80, ↓97	↓40, ↓41	↓7, ↓10
	Sports	↑18, ↑9	↓30, ↓2	↓83, ↓83	↓35, ↓31	↓8, ↓2
	Action	↑14, ↑6	↓49, ↓27	↓79, ↓74	↓38, ↓25	↓8, ↓3
FCC	BBB	↑11, ↑4	↓70, ↓28	↓26, ↓51	↓48, ↓32	↓8, ↓4
	ED	↑8, ↑3	↓33, ↓11	↓79, ↓34	↓36, ↓31	↓7, ↓2
	Sintel	↑7, ↑4	↓87, ↓70	↓94, ↓85	↓35, ↓26	↓6, ↓2
	ToS	↑6, ↑6	↓69, ↓62	↓90, ↓6	↓40, ↓38	↓7, ↓1
		↑ better	↓ better	↓ better	↓ better	↓ better

* The two numbers in each cell represents the changes by CAVA relative to RobustMPC and PANDA/CQ max-min, respectively.

The second part of Table 1 shows the results for FCC traces. Compared to the LTE trace results, the rebuffering for all the schemes becomes lower due to smoother network bandwidth profiles; CAVA still leads to lower rebuffering and better performance in all the performance metrics.

6.4 Impact of CAVA Design Principles

Recall CAVA incorporates three key design principles (§4). In the following, we investigate the contribution of each principle to the performance of CAVA. As detailed in §4, the first (i.e., non-myopic) principle is used in the inner controller of CAVA: it uses the average bitrate of multiple future chunks to represent the bandwidth requirement of the current chunk when deciding the track level for the current chunk. The second (i.e., differential treatment) principle favors Q4 chunks over Q1-Q3 chunks. The third (i.e., proactive) principle is used in the outer controller: it considers long-term future (within tens of chunks) and increases the target buffer level to be prepared for incoming large chunks in the future. Of the three principles, the first principle is the basic principle, while the other two principles are used on top of the basic principle. In the following, we consider three variants of CAVA, referred to as CAVA-p1, CAVA-p12 and CAVA-p123, which includes the first principle, the first two principles, and all three principles, respectively.

Our evaluation confirms that the three principles indeed successfully play their individual roles. Specifically, CAVA-p1 already leads to fast responses to dynamic network bandwidths, consistent playback quality and low rebuffering; CAVA-p12 leads to higher Q4 chunk quality, while CAVA-p123 leads to lower rebuffering. Fig. 10 shows an example for one video (ED, FFmpeg encoded, H.264) under LTE traces. Fig. 10(a) plots the Q4 chunk quality under the two variants with the differential treatment principle (i.e., CAVA-p12 and CAVA-p123) relative to CAVA-p1. Specifically, it shows the distribution of the metric value under CAVA-p12 and CAVA-p123 minus the value under CAVA-p1 across all runs. We observe that for around 40% of the Q4 chunks, CAVA-p12 and CAVA-p123 lead to higher Q4 chunk quality. For only around 5% of the Q4 chunks, the quality under CAVA-p12 and CAVA-p123 is lower than CAVA-p1. Fig. 10(b) shows the total rebuffering of CAVA-p123 relative to CAVA-p12 for traces that lead to rebuffering under either of these two variants (35 out of 200 traces fall into this category). Specifically, it shows the metric value under each variant minus the value under CAVA-p12. We see that CAVA-p123 leads to lower rebuffering than CAVA-p12 in 55% of these traces and the reduction is up to 20 seconds.

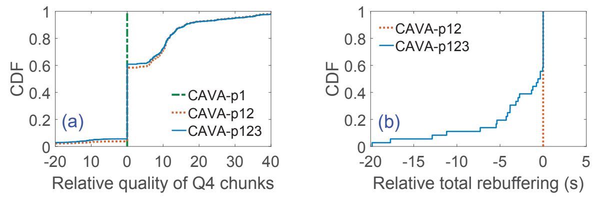

Figure 10: Impact of the design principles of CAVA.

6.5 Codec Impact

For each video, the performance of all the schemes under H.265 encoding is better than that under H.264 encoding (figures omitted). This is due to the significantly lower bitrate requirement of H.265 encoding. We observe that CAVA also significantly outperforms all the other schemes under H.265 encoding. Compared to RobustMPC and PANDA/CQ max-min, on average, Q4 chunk quality under CAVA is 7-12 higher, the percentage of low-quality chunks is 51%-82% lower, rebuffering is 52%-91% lower, and average quality change is 27%-72% lower. Also CAVA has similar data usage compared to other schemes.

6.6 Impact of Higher Bitrate Variability

We further use the 4× capped VBR-encoded video (Elephant Dream FFmpeg encoded, in H.264) to explore the impact of higher bitrate variability. In this case, we observe similar trends as the earlier results. For CAVA, the average Q4 chunk quality is 65 across the LTE traces, 8 and 7 larger than RobustMPC and PANDA/CQ max-min, respectively. The average quality change is 42% and 68% lower, the rebuffering is 90% and 89% lower, and the average percentage of low-quality chunks is 39% and 57% lower.

6.7 Impact of Bandwidth Prediction Error

So far, we have used a particular bandwidth estimation algorithm (harmonic mean of past 5 chunks) to predict available network bandwidth. We next investigate the impact of bandwidth prediction errors on different schemes in a controlled manner as follows. We assume the bandwidth prediction error is err. That is, if the actual network bandwidth at time t is C_t, we assume that the predicted network bandwidth is uniformly randomly distributed in $C_t(1 \pm err)$. We vary err from 0 to 50% with a step of 25%. We observe that CAVA is insensitive to bandwidth prediction errors: the quality of Q4 chunks, the amount of rebuffering and the percentage of low quality chunks when err=50% are similar to those when err=0 (i.e., perfect prediction), demonstrating that CAVA is adaptive and tolerant to relatively significant bandwidth prediction errors (due to the control-theoretical underpinning). In comparison, MPC leads to significantly more rebuffering and data usage when err=50%, compared to those when err=0. PANDA/CQ max-min leads to noticeably more rebuffering when err is increased from 0 to 50%. The reason why CAVA is resilent to bandwidth estimation errors is because it is a principled approach based on control theory – it continuously monitors the "errors" in the system, namely the difference between the target and current buffer levels of the player, and adjusts the control signal to correct "errors" caused by inaccurate bandwidth estimation.

6.8 DASH Implementation Results

We evaluate CAVA using the dash.js implementation (§5.5), and compare its performance with that of BOLA-E [37] implemented in dash.js version 2.7.0, which contains a stable release of BOLA-E. Our evaluations below use two computers (Ubuntu 12.04 LTS, CPU Intel Core 2 Duo, memory 4GB) with a 100 Mbps direct network connection to emulate the video server (Apache httpd) and client. At the client, we use Selenium [32] to programmatically run a Google Chrome web browser and use dash.js APIs to collect the streaming performance data. We use tc [26] to emulate real-world variable network conditions by programmatically "replaying" the network traces (§6.1).

We find that CAVA is very light-weight – even with the current prototype implementation, the runtime overhead (from profiling) is only around 56ms for a 10-minute video.

We compare the performance of CAVA with three versions of BOLA-E. Two versions are based on the original BOLA-E implementation, which takes a single bitrate value from each track (i.e., the declared bitrate from the manifest file). Specifically, we consider BOLA-E (peak), where the declared bitrate for a track is the peak bitrate of the track, and BOLA-E (avg), where the declared bitrate for a track is the average bitrate of the track. In the third version, referred to as BOLA-E (seg), we modified the original BOLA-E implementation to use the actual chunk size for each chunk, as suggested by the paper [38] for VBR encodings. All the three versions use the default parameters in BOLA-E. Fig. 11 shows the performance of CAVA and the three versions of BOLA-E for one Youtube video. We observe

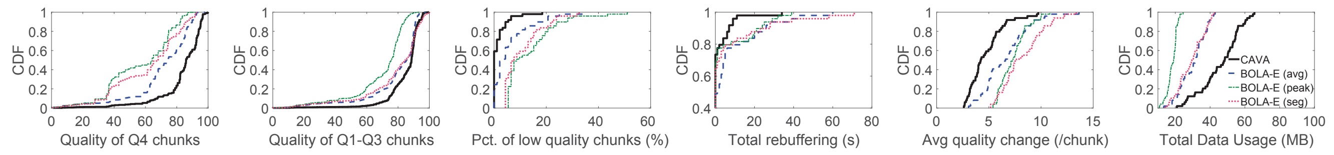

Figure 11: Performance comparison in `dash.js` (Big Buck Bunny, YouTube encoded, H.264, LTE traces).

Table 2: CAVA versus BOLA-E in `dash.js`.

Video	Q4 chunk quality	Low-qual. chunks (%)	Stall duration (%)	Quality changes (%)	Data usage (%)
BBB	↑21	↓87	↓65	↓45	↑56
ED	↑20	↓73	↓15	↓35	↑50
Sports	↑10	↓75	↓29	↓40	↑25
ToS	↑12	↓76	↓19	↓24	↑45

that CAVA outperforms the three versions of BOLA-E in having higher Q4 chunk quality, lower percentage of low-quality chunks, lower rebuffering, and lower quality changes. While the data usage of BOLA-E (all three variants) is lower than that of CAVA, it comes at the cost of worse performance in other performance metrics. The lower data usage of BOLA-E could be because it sometimes pauses before fetching the next chunk under various scenarios. In addition, the implementation uses bitrate capping when switching to a higher bitrate to avoid bitrate oscillations.

Of the three BOLA-E variants, not surprisingly, BOLA-E (peak) is the most conservative variant in choosing bitrate, since it uses the peak bitrate of a track as the bitrate of every chunk in the track, and hence overestimates the bandwidth requirements of the individual chunks. In comparison, BOLA-E (avg) is the most aggressive, and BOLA-E (seg) is between BOLA-E (peak) and BOLA-E (avg). We see that for Q1-Q3 chunks, BOLA-E (seg) can choose even higher tracks than BOLA-E (avg), since BOLA-E (seg) uses the actual chunk size and can choose higher quality for some small chunks (that fall into Q1-Q3). We further see that using the actual chunk size in BOLA-E (seg) leads to more significant quality changes compared to BOLA-E (peak) and BOLA-E (avg). The above behaviors demonstrate once again that while it is important to consider individual chunk sizes to handle ABR streaming of VBR videos, the actual decision logic needs to be developed with VBR characteristics in mind; simply plugging in the individual chunk sizes is insufficient.

Table 2 compares the performance of CAVA and BOLA-E (seg) for four YouTube videos, all under LTE traces. We observe that while the data usage of BOLA-E (seg) is lower, CAVA outperforms BOLA-E (seg) in all the other metrics: on average, Q4 chunk quality under CAVA is 10-21 higher, the percentage of low-quality chunks is 73%-87% lower, rebuffering is 15%-65% lower, and quality changes is 24%-45% lower.

7 RELATED WORK

There is a large body of existing work on ABR streaming [1, 6, 8, 16, 20, 23, 28, 33, 38, 39, 41, 44, 47, 49, 51]. Rate-based (e.g., [20, 21, 49]) ABR schemes select bitrate simply based on network bandwidth estimates. Buffer-based schemes, e.g., BBA [16], BOLA [38], and BOLA-E [37], select bitrate only based on buffer occupancy. Most recent schemes combine both rate-based and buffer-based techniques. For instance, MPC [47] and PIA [33] use control-theoretic approaches to design rate adaptation based on both network bandwidth estimates and buffer status. Oboe [1] dynamically adjusts the parameters of an ABR logic to adapt to various network conditions. Another group of related studies [7, 9, 28, 42] uses machine learning based approaches, which "learns" ABR schemes from data.

While content providers have recently begun adopting VBR encoding, their treatment to VBR videos is simplistic (e.g., by simply assuming that the bandwidth requirement of a chunk equals to the peak bitrate of the track) [46]. As we have seen in §6.8, such simplistic treatment can lead to poor performance. Of the large number of existing studies, only a few targeted VBR videos [16, 23, 49]; most others were designed for and evaluated in the context of CBR encodings. While several schemes in the latter category can be applied for VBR videos by explicitly incorporating the actual bitrate of each chunk, they were not designed ground up to account for VBR characteristics, nor had their performance been evaluated for that case. We evaluate the performance of schemes from both categories in this paper (see §4 and §6), and show that they can suffer from a large amount of rebuffering and/or significant viewing quality impairments when used for VBR videos.

Our study differs from all the above works in several important aspects, including developing insights based on analyzing content complexity, encoding quality, and encoding bitrates, identifying general design principles for VBR streaming, and developing a concrete, practical ABR scheme for VBR streaming that significantly outperforms the state-of-the-art schemes.

8 CONCLUSIONS AND FUTURE WORK

Through analyzing a number of VBR-encoded videos from a variety of genres, we identified a number of characteristics related to scene complexities, bandwidth variability, and quality levels, that are relevant to ABR streaming and QoE. Based on these insights, we proposed 3 key principles to guide rate adaptation for VBR streaming. We also designed CAVA, a practical control-theoretic rate adaptation scheme that explicitly leverages our proposed principles. Extensive evaluations demonstrate that CAVA substantially outperforms existing ABR schemes even in challenging network conditions common in cellular networks. The results demonstrate the importance of utilizing these design principles in developing ABR adaptation schemes for VBR encodings. In this paper, we primarily focused on the VoD setting. As future work, we will explore extending CAVA and its concepts to ABR streaming of live VBR encoded videos.

ACKNOWLEDGEMENTS

We express our sincerest gratitude towards the anonymous reviewers who gave valuable and insightful feedback to improve this work, and our shepherd, Behnaz Arzani, for guiding us through the revisions. The work of Feng Qian was partially supported by NSF under award CNS-1750890.

REFERENCES

[1] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. 2018. Oboe: auto-tuning video ABR algorithms to network conditions. In *Sigcomm*. ACM.

[2] Apple. 2017. Apple's HTTP Live Streaming. https://goo.gl/eyDmBc. (2017).

[3] Apple. 2017. HLS Authoring Specification for Apple Devices. (2017). https://goo.gl/kYrCW5

[4] Karl Johan Åström and Richard M. Murray. 2008. *Feedback Systems: An Introduction for Scientists and Engineers.* Princeton University Press.

[5] Chao Chen, Yao-Chung Lin, Anil Kokaram, and Steve Benting. 2017. Encoding Bitrate Optimization Using Playback Statistics for HTTP-based Adaptive Video Streaming. *arXiv preprint arXiv:1709.08763* (2017).

[6] J. Chen, R. Mahindra, M. A. Khojastepour, S. Rangarajan, and M. Chiang. 2013. A Scheduling Framework for Adaptive Video Delivery over Cellular Networks. In *Proc. of ACM MobiCom.*

[7] Federico Chiariotti, Stefano D'Aronco, Laura Toni, and Pascal Frossard. 2016. Online Learning Adaptation Strategy for DASH Clients. In *Proc. ACM Multimedia systems.* ACM.

[8] L. De Cicco, S. Mascolo, and V. Palmisano. 2011. Feedback Control for Adaptive Live Video Streaming. In *Proc. of ACM MMSys.*

[9] Maxim Claeys, Steven LatrÃĨ, Jeroen Famaey, Tingyao Wu, Werner Van Leekwijck, and Filip De Turck. [n. d.]. Design and optimisation of a (FA)Q-learning-based HTTP adaptive streaming client. *Connection Science* 26, 1 ([n. d.]).

[10] Federal Communications Commission. 2016. Measuring Fixed Broadband Report. https://goo.gl/STv4pg. (December 2016).

[11] Jan De Cock, Aditya Mavlankar, Anush Moorthy, and Anne Aaron. 2016. A large-scale video codec comparison of x264, x265 and libvpx for practical VOD applications. In *SPIE, Applications of Digital Image Processing.*

[12] Wolfgang Effelsberg, Otto Spaniol, André Danthine, and Domenico Ferrari. 1996. *High-speed networking for multimedia applications.* Springer.

[13] FFmpeg. 2017. FFmpeg Project. https://www.ffmpeg.org/. (2017).

[14] International Organization for Standardization. 2012. ISO/IEC DIS 23009-1.2 Dynamic adaptive streaming over HTTP (DASH). (2012).

[15] DASH Industry Forum. 2017. Reference Client 2.4.1. https://goo.gl/XJcciV. (2017).

[16] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proc. of ACM SIGCOMM.*

[17] MulticoreWare Inc. 2018. H.265 Video Codec. http://x265.org/hevc-h265/. (2018).

[18] ITU. 2017. H.264 : Advanced video coding for generic audiovisual services. https://www.itu.int/rec/T-REC-H.264. (2017).

[19] ITU-T P. 910. 2008. Subjective Video Quality Assessment Methods for Multimedia Applications. (2008).

[20] Junchen Jiang, Vyas Sekar, and Hui Zhang. 2012. Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with FESTIVE. In *Proc. of ACM CoNEXT.*

[21] Robert Kuschnig, Ingo Kofler, and Hermann Hellwagner. 2010. An evaluation of TCP-based rate-control algorithms for adaptive internet streaming of H.264/SVC. In *Proc. ACM Multimedia systems.*

[22] TV Lakshman, Antonio Ortega, and Amy R Reibman. 1998. VBR video: Tradeoffs and potentials. *Proc. IEEE* (1998).

[23] Zhi Li, Ali Begen, Joshua Gahm, Yufeng Shan, Bruce Osler, and David Oran. 2014. Streaming video over HTTP with consistent quality. In *ACM MMSys.*

[24] J. Y. Lin, T.-J. Liu, E. C.-H. Wu, and C.-C. J. Kuo. 2014. A Fusion-based Video Quality Assessment (FVQA) Index. *APSIPA Transactions on Signal and Information Processing* (2014).

[25] Yao-Chung Lin, Hugh Denman, and Anil Kokaram. 2015. Multipass encoding for reducing pulsing artifacts in cloud based video transcoding. In *Image Processing (ICIP), 2015 IEEE International Conference on.* IEEE, 907–911.

[26] Linux. 2014. tc. https://linux.die.net/man/8/tc. (2014).

[27] T.-J. Liu, J. Y. Lin, W. Lin, and C.-C. J. Kuo. 2013. Visual Quality Assessment: Recent Developments, Coding Applications and Future Trends. *APSIPA Transactions on Signal and Information Processing* (2013).

[28] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *Proc. of ACM SIGCOMM.*

[29] Netflix. 2015. Per-Title Encode Optimization. https://goo.gl/hahDG4. (2015).

[30] Netflix. 2016. VMAF score aggregation. https://github.com/Netflix/vmaf/issues/20. (2016).

[31] Jan Ozer. 2017. Finding the Just Noticeable Difference with Netflix VMAF. https://goo.gl/TGWCGV. (September 2017).

[32] Selenium Projects. 2017. Selenium Web Browser Automation. http://www.seleniumhq.org. (2017).

[33] Yanyuan Qin, Ruofan Jin, Shuai Hao, Krishna R Pattipati, Feng Qian, Subhabrata Sen, Bing Wang, and Chaoqun Yue. 2017. A Control Theoretic Approach to ABR Video Streaming: A Fresh Look at PID-based Rate Adaptation. In *INFOCOM.*

[34] Reza Rassool. 2017. VMAF reproducibility: Validating a perceptual practical video quality metric. In *IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB).*

[35] Andrew Reed and Benjamin Klimkowski. 2016. Leaky streams: Identifying variable bitrate DASH videos streamed over encrypted 802.11n connections. In *Consumer Communications & Networking Conference.* IEEE.

[36] Werner Robitza. 2017. CRF Guide (Constant Rate Factor in x264 and x265). http://slhck.info/video/2017/02/24/crf-guide.html. (February 2017).

[37] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. 2018. From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player *(MMSys '18).* ACM.

[38] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. 2016. BOLA: Near-Optimal Bitrate Adaptation for Online Videos. In *INFOCOM.* IEEE.

[39] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. 2016. CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *SIGCOMM.* ACM.

[40] Kiyotsugu Takaba. 2003. A tutorial on preview control systems. In *SICE.*

[41] Guibin Tian and Yong Liu. 2012. Towards agile and smooth video adaptation in dynamic HTTP streaming. In *Proc. of ACM CoNEXT.*

[42] Jeroen van der Hooft, Stefano Petrangeli, Maxim Claeys, Jeroen Famaey, and Filip De Turck. 2015. A learning-based algorithm for improved bandwidth-awareness of adaptive streaming clients. In *IFIP/IEEE International Symposium on Integrated Network Management.*

[43] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4 (2004).

[44] Xiufeng Xie, Xinyu Zhang, Swarun Kumar, and Li Erran Li. 2015. piStream: Physical Layer Informed Adaptive Video Streaming Over LTE. In *Proc. of ACM MobiCom.*

[45] Xiph.Org. 2016. Xiph.org Video Test Media. https://media.xiph.org/video/derf/. (2016).

[46] Shichang Xu, Z. Morley Mao, Subhabrata Sen, and Yunhan Jia. 2017. Dissecting VOD Services for Cellular: Performance, Root Causes and Best Practices. In *Proc. of IMC.*

[47] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proc. of ACM SIGCOMM.*

[48] youtube-dl developers. 2018. youtube-dl. https://goo.gl/mgghW8. (2018).

[49] Tong Zhang, Fengyuan Ren, Wenxue Cheng, Xiaohui Luo, Ran Shu, and Xiaolan Liu. 2017. Modeling and analyzing the influence of chunk size variation on bitrate adaptation in DASH. In *INFOCOM.* IEEE.

[50] Li Zhi, Aaron Anne, Katsavounidis Ioannis, Moorthy Anush, and Manohara Megha. 2016. Toward A Practical Perceptual Video Quality Metric. (2016). https://medium.com/netflix-techblog/toward-a-practical-perceptual-video-quality-metric-653f208b9652

[51] Xuan Kelvin Zou, Jeffrey Erman, Vijay Gopalakrishnan, Emir Halepovic, Rittwik Jana, Xin Jin, Jennifer Rexford, and Rakesh K. Sinha. 2015. Can Accurate Predictions Improve Video Streaming in Cellular Networks?. In *Proc. of HotMobile.*