Quality-aware Strategies for Optimizing ABR Video Streaming QoE and Reducing Data Usage

Yanyuan Qin[1], Shuai Hao[2], Krishna R. Pattipati[1], Feng Qian[3*],
Subhabrata Sen[2], Bing Wang[1], and Chaoqun Yue[1]
[1]University of Connecticut [2]AT&T Labs - Research [3]University of Minnesota

ABSTRACT

Streaming videos over cellular networks is highly challenging. Since cellular data is a relatively scarce resource, many video and network providers offer options for users to exercise control over the amount of data consumed by video streaming. Our study shows that existing data saving practices for Adaptive Bitrate (ABR) videos are suboptimal: they often lead to highly variable video quality and do not make the most effective use of the network bandwidth. We identify underlying causes for this and propose two novel approaches to achieve better tradeoffs between video quality and data usage. The first approach is Chunk-Based Filtering (CBF), which can be retrofitted to any existing ABR scheme. The second approach is QUality-Aware Data-efficient streaming (QUAD), a holistic rate adaptation algorithm that is designed ground up. We implement and integrate our solutions into two video player platforms (`dash.js` and ExoPlayer), and conduct thorough evaluations over emulated/commercial cellular networks using real videos. Our evaluations demonstrate that compared to the state of the art, the two proposed schemes achieve consistent video quality that is much closer to the user-specified target, lead to far more efficient data usage, and incur lower stalls.

CCS CONCEPTS

• **Information systems** → Multimedia streaming;

KEYWORDS

Adaptive Video Streaming; Quality-aware; QoE; Data Saving.

ACM Reference Format:
Yanyuan Qin, Shuai Hao, Krishna R. Pattipati, Feng Qian, Subhabrata Sen, Bing Wang, and Chaoqun Yue. 2019. Quality-aware Strategies for Optimizing ABR Video Streaming QoE and Reducing Data Usage. In *10th ACM Multimedia Systems Conference (MMSys '19), June 18–21, 2019, Amherst, MA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3304109.3306231

1 INTRODUCTION

Mobile video streaming is extremely popular, already accounting for the bulk of traffic on cellular networks [34]. Ensuring good viewing experience is important but challenging, especially when streaming over cellular networks which can exhibit high bandwidth variability, due to the conflicting goals of maximizing quality, minimizing rebuffering, and minimizing quality changes [12, 24, 29, 35]. Adaptive Bitrate (ABR) streaming (mainly using HLS [2] and DASH [15]) has emerged as the *de facto* over-the-top (OTT) video streaming technology in industry. On the server side, a video is compressed into a ladder of multiple independent *tracks*, each specifying the same content but with a different bitrate/quality. A track is further divided into a series of *chunks* (or *segments*), each containing data for a few seconds' worth of playback. For each chunk position in the video, the encoded bitrates and hence the picture quality generally increase from lower to higher tracks. During playback, the client downloads a *manifest file* containing the metadata about the different tracks and resource requirement (e.g., the peak rate). The ABR adaptation logic dynamically determines which quality (i.e., from which track) to fetch for each chunk position in the video, based on the available network bandwidth and possibly other factors.

This paper attempts to answer the following question: how can we effectively reduce the bandwidth consumption for mobile video streaming while minimizing the impact on users' quality of experience (QoE)? This is a highly important and practical research problem since cellular network bandwidth is a relatively scarce resource. The average data plan for a U.S. cellular customer is only 2.5 GB per month [13], while streaming just one-hour High Definition (HD) video on mobile Netflix can consume 3 GB data. Therefore, the capability to make more efficient use of data while still hitting quality targets is the key to enabling users to consume more content within their data budgets without adversely impacting QoE. In addition, downloading less data for a video session also translates to lower radio energy consumption, less thermal overhead on mobile devices, as well as potentially better QoE for other users sharing the same cellular RAN (radio access network) or base station.

Existing ABR adaptation schemes (§9) focus mainly on maximizing the video quality and user QoE. While some schemes do conservatively utilize the bandwidth, their decisions are primarily driven by QoE impairment concerns such as the possibility of stalls caused by the potentially inaccurate bandwidth estimation. For example, when selecting the next chunk, the default adaptation scheme in ExoPlayer [18] (a popular open-source player that is used in more than 10000 apps) only considers tracks whose declared bitrates are at least 25% lower than the estimated network bandwidth. Such a design saves data just by being conservative. As we show later, it can lead to significantly lower quality/QoE (§8.3). Existing ABR schemes do not explicitly consider bandwidth efficiency together with quality in making the track selection decisions.

Some mobile network operators and commercial video services provide users with certain "data saver" options. These options take either a service-based approach or a network-based approach. The

* The work of Feng Qian was done when he was at Indiana University.

former limits the highest level of quality/resolution/bitrate by, for example, streaming only standard definition (SD) [47] content over cellular networks. A network-based approach instead limits the network bandwidth. We survey the "quality throttling" mechanisms used by today's commercial video content providers for saving bandwidth for mobile devices, and pinpoint their inefficiencies (§2-§3). Despite their simplicity, we find such approaches often achieve tradeoffs between video quality and bandwidth usage that are far from what viewers desire. A key reason is that for state-of-the-art encoders (e.g., H.264 [21], H.265 [20] and VP9 [37]), the actual perceived picture quality exhibits significant variability across different chunks within the same track, for both Constant Bitrate (CBR) and Variable Bitrate (VBR) encodings. Therefore, a data-saving approach that simply removes high tracks leads to quality variations. Such quality variability impairs user QoE, and makes suboptimal use of the network bandwidth.

To address the drawbacks of the existing practices, we propose a *quality-aware data saving* strategy, which provides data savings while allowing users' direct control of the video quality. Specifically, a user selects from multiple quality options (e.g., good, better, best), and the player will map the selection to a *target quality* (§3). Within this strategy, we propose two new schemes that explicitly consider the bandwidth efficiency by matching the fetched content quality against a target quality. In this way, the player will avoid fetching chunks whose qualities are beyond the target quality, leading to bandwidth savings. Specifically, the two new schemes are:

• *Chunk-Based Filtering (CBF).* We propose a novel, quality-variability aware scheme called Chunk-Based Filtering (CBF) (§4) that can be retro-fitted into existing ABR adaptation schemes. Its high-level idea is as follows. For every chunk position in the video, CBF limits the choice of the highest quality chunk for ABR rate adaptation to the chunk whose quality is closest to the target quality. CBF thereby steers an ABR adaptation scheme to the set of more desirable choices (from the perspective of balancing the tradeoff between the video quality and bandwidth usage), and helps the ABR scheme achieve better streaming performance than it would by itself. Using CBF in conjunction with existing ABR adaptation schemes is also attractive from a practical incremental deployment perspective. CBF has no dependencies on and requires no changes to the complex ABR adaptation logic, and can be relatively easily inserted into the existing streaming workflow (§4.3).

• *Quality-aware ABR Adaptation.* From a performance and bandwidth-efficiency tradeoff perspective, it is possible to do even better, if the ABR scheme itself can explicitly integrate the goal of approaching the target quality into its rate adaptation logic. To this end, we develop *QUality-Aware Data-efficient streaming (QUAD),* a holistic rate adaptation algorithm that is designed ground up (§5). QUAD jointly considers three aspects when making rate adaptation decisions: pacing the selected chunks' bitrate to the estimated network bandwidth to prevent stalls, adapting to the target quality to reduce bandwidth consumption, and minimizing the inter-chunk quality change to enhance the playback smoothness. QUAD is robust and lightweight as it is developed upon solid control theoretic foundations.

We implemented CBF and QUAD (§6) in two popular state-of-the-art open source ABR video players: `dash.js` [17] and ExoPlayer [18]. Our evaluation of these two techniques uses a diverse set of real videos and different encoding schemes (VBR and CBR), under both emulated and real-world LTE networks (§6). The key evaluation results include the following.

• CBF significantly improves the performance of existing state-of-the-art ABR schemes (§7). Specifically, after employing CBF as a prefiltering step, the average deviation from the target quality is reduced by 37-67%; the average quality variation is reduced by 7-31%; and the data usage is reduced by 34-67% even in challenging network conditions (in easier conditions with ample bandwidth, the reduction is even more). We further experimentally demonstrate that CBF is significantly more effective in steering existing ABR schemes to more desirable rate adaptation decisions than traditional service-based or network-based data saving approaches.

• Compared to existing schemes enhanced with CBF, QUAD achieves even better performance in approaching the target quality with low quality variations while still achieving good QoE (§8). For instance, our evaluation on `dash.js` shows that, compared to a state-of-the-art ABR scheme, BOLA-E [43] enhanced with CBF, QUAD leads to 37% fewer low-quality chunks and 12% reduction in quality variation. Compared to the default rate adaptation of ExoPlayer, QUAD reduces the deviation from the target quality by 64%, reduces the number of low-quality chunks by 81%, and reduces the quality variation by 43%. Compared to an optimized version of ExoPlayer's algorithm enhanced with CBF, the corresponding reductions brought by QUAD are 40%, 46% and 22%, respectively.

2 QUALITY AND BITRATE TRADEOFFS

In this section, we describe the encoding characteristics of both VBR and CBR videos to motivate our proposed solutions.

2.1 Video Dataset

Our video dataset includes 18 VBR and 4 CBR videos. We consider both VBR and CBR videos because both are widely used in practice [49], with the trend of wider adoption of VBR videos thanks to their many advantages over CBR videos [25]. Each video is around 10 minutes long, and encoded at 6 tracks/levels[1] for ABR streaming, with resolutions of 144p, 240p, 360p, 480p, 720p, and 1080p.

VBR Videos. All the 18 VBR videos were encoded by YouTube (YouTube has adopted VBR encoding [28]). Four videos, Elephant Dream (ED), Big Buck Bunny (BBB), Sintel, and Tears of Steel (ToS), were encoded from publicly available *raw* videos [48]. Specifically, we uploaded the raw videos to YouTube and downloaded the encoded videos using `youtube-dl` [51]. These four videos are in the categories of animation and science fiction. We further downloaded 14 other videos, in a wide range of categories, including sports, animal, nature, action movies, family drama, comedy, and documentary, using `youtube-dl`. All the above videos are encoded using the H.264 codec [21], with chunk duration of around 5 seconds, consistent with [28] (the encoding is multi-pass; readers can find the detailed encoding settings in [28]). Ten out of the aforementioned 14 videos were also available in another codec, VP9 [37], encoded by YouTube. In addition, we further use `FFmpeg` to encode the four

[1]We use the terms *track* and *level* interchangeably in this paper. Other equivalent terms include representation and rendition, which are also widely used in the literature.

publicly available videos using a more recent and efficient codec, H.265 [20], following the “three-pass” encoding in [11].

CBR Videos. We also created CBR encodings for the four publicly available raw videos using FFmpeg [14], a popular open source encoder. Specifically, we used the one-pass CBR encoder, the default CBR encoder in FFmpeg, which is often used instead of multi-pass encoders due to latency considerations, particularly in live streaming. Each video is encoded using H.264 into six tracks (144p to 1080p), and each track is segmented into 5-sec chunks, consistent with the ABR track configurations of YouTube.[2]

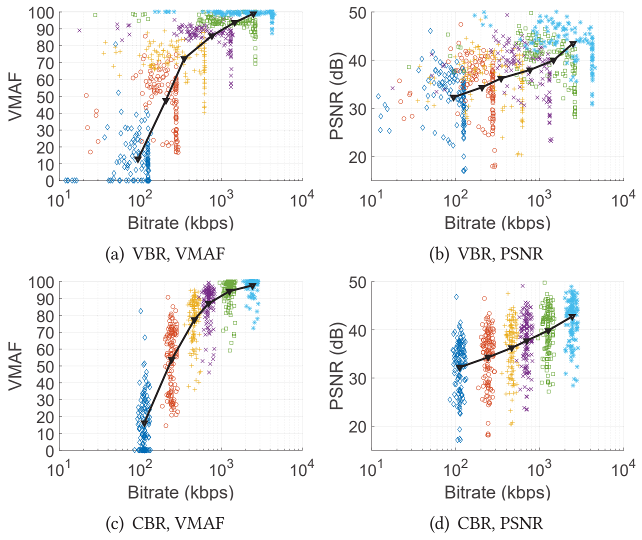
2.2 Video Quality Metrics

We use two quality metrics, Peak Signal-to-Noise Ratio (PSNR) and Video Multimethod Assessment Fusion (VMAF) [26]. PSNR is a traditional image quality metric. VMAF is a recently proposed perceptual quality metric that correlates quality strongly with subjective scores, and has been validated independently in [41]. VMAF provides different models tailored to various screen sizes, such as phone and TV. We focus on the VMAF phone model in this paper since phones are the dominant platform for viewing videos over cellular networks. A VMAF score is between 0 and 100: a score of 0-20 is considered as unacceptable, 20-40 as poor, 40-60 as fair, 60-80 as good, and 80-100 as excellent [26]. Similarly, different ranges of PSNR values are used to categorize picture qualities [11]. The aggregate VMAF of a chunk is set as the median VMAF of all the frames in a chunk (using mean leads to similar values for videos in our dataset) [33]. The same approach is used for PSNR.

To calculate PSNR and VMAF for a video, we need a *reference video*, i.e., a pristine high quality copy of the video against which to compute these metrics. For the four VBR and CBR videos that were encoded using the publicly available raw videos, the corresponding raw videos are used as the reference videos. For the other VBR videos downloaded from YouTube, we do not have the raw video footage, and use the top track (1080p) as the reference track to measure the video quality of the lower tracks. To understand the impact of this approximation, we calculate the quality values for the four videos that we have raw videos under two options, one with the raw video and the other with the 1080p track as the reference video. We find empirically that the latter leads to lower variability across the chunks in the same track for both PSNR and VMAF, a point that we will return to in §2.3.

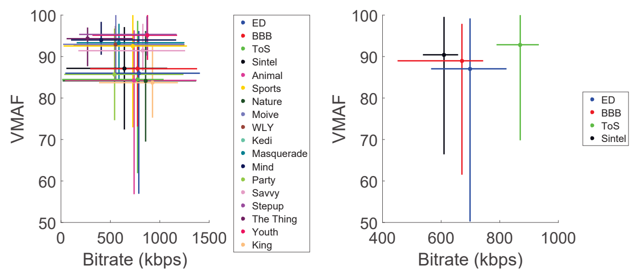
2.3 Perceptual Quality vs. Encoding Bitrate

We first use an example video to illustrate the tradeoffs between quality and bitrate, and then describe the observations across the video dataset. Figures 1(a) and (b) plot quality versus bitrate for a VBR video (with 123 chunks, each of 5-sec duration, and 6 tracks), using VMAF and PSNR as the quality metric, respectively. In these figures, different colors represent the chunks in the different tracks; the black curve shows the average quality versus average bitrate for each track. We see a diminishing gain in increasing bitrate on quality (note the log scale in x-axis), consistent with the observations



(a) VBR, VMAF (b) VBR, PSNR (c) CBR, VMAF (d) CBR, PSNR

Figure 1: Quality vs. bitrate for one video (ED).



(a) YouTube VBR (H264) (b) FFmpeg CBR (H264)

Figure 2: Quality vs. bitrate for track 4 (480p).

in [10]. Furthermore, even under VBR encoding, the chunks in the same track have significantly variable perceptual quality (a VMAF difference of 6 or more would be noticeable to a viewer [10, 36]). Specifically, for a track, the standard deviation of quality across the chunks varies from 2 to 14 for VMAF, 4.1 to 5.4 in PSNR (note PSNR values in log scale), with the middle tracks exhibiting more variability. We make similar observations for CBR video. Figures 1(c) and (d) plot VMAF quality versus bitrate for a CBR video, where all the chunks in same track have similar bitrates.

Fig. 2 plots the quality (in VMAF) and bitrate variations for track 4 for all the videos we consider (18 VBR videos in Fig. 2(a) and 4 CBR videos in Fig. 2(b)); the results under PSNR show a similar trend. Each video is represented by two error bars, representing the 1st and 99th percentile in quality and bitrate, respectively. We see the same two observations hold for all the videos we consider. Note that Fig. 2(a) includes the 14 VBR videos for which we use the 1080p tracks as the reference videos (due to unavailability of the raw videos), which tend to underestimate the quality variability across the chunks in the same track (see empirical observations in §2.2). We see significant variability even with the underestimation.

The above results are for H.264 [21] videos. To ensure that the above observations are general, we further investigate the quality and bitrate relationships for other encoders and content. Specifically, we examine (i) H.265 [20], a more recent and efficient codec than

[2]As an example, the command for encoding an input video into a 1080p track is `ffmpeg -i inputvideo -b:v 2500k -minrate 2500k -maxrate 2500k -bufsize 2500k -r 24 -profile:v high -x264-params nal-hrd=cbr:keyint=120 :min-keyint=120:scenecut=0 -vcodec libx264 -vf scale=1920:1080 -preset fast output.mp4`.

Option	Top Track	Declared Bitrate	Resolution
Data saver	3	120kbps	288p
Good	6	450kbps	360p
Better	7	650kbps	396p
Best	8	1000kbps	480p

Table 1: Data saving options in Amazon Prime video.

H.264, using the four publicly available videos, (ii) VP9 [37], another widely used codec in YouTube, using ten VP9 videos downloaded from YouTube, and (iii) Twitch, another popular video streaming service, by selecting five popular videos in different genres based on the number of viewers. For all the three cases, our results confirm the same two quality and bitrate relationships observed earlier: (1) increasing bitrate leads to diminishing gain in quality improvement, and (2) the chunks in the same track have significantly variable quality. These two observations are consistent with the results of Netflix encoded videos [9, 10], indicating that they hold widely across encoding platforms and videos.

The property that chunks within the same track have highly variable quality holds obviously for CBR encodings, which encode 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). The fact that it also holds for VBR is somewhat counterintuitive since VBR allocates bits according to scene complexity to achieve a more consistent quality throughout a track. Part of the reason is the inherent complexity of encoding and the difficulty of handling scenes of diverse complexities [10, 28].

3 REDUCING DATA USAGE

The diminishing gains in increasing bitrate on quality improvement demonstrated in the previous section indicate that an ABR logic that simply aims to maximize quality is not bandwidth efficient. In the following, we first describe the current data saving practices and show that they are inefficient. We then propose a quality-aware strategy for reducing data usage.

3.1 Current Data Saving Practices

Certain cellular network operators provide users with options to limit the network bandwidth for a streaming session (e.g., [4, 45]). The rationale is that the bandwidth cap may lead an ABR player to avoid bandwidth-consuming High-Definition (HD) tracks so as to save data.

Various commercial video streaming services have also provided users with options to save data. For instance, the YouTube phone app provides an option called "Play HD on Wi-Fi only", i.e., only Standard Definition (SD) videos will be streamed over cellular networks. To understand the behavior of this option, we stream eight videos in different categories using the YouTube app over a commercial LTE network. We observe that, when the option is on, even if the network bandwidth is very high (over tens of Mbps), the 480p track is selected throughout the video. The fact that the selected tracks never exceed 480p despite significantly higher bandwidth indicates that the data saving is achieved by capping the top track to the 480p track. Henceforth, we refer to this practice as *Track-based Filtering (TBF)*. We find that data saving options in the Amazon Prime Video app are also achieved by capping the top track. Table 1 summarizes the measurement results, showing the top track for the four options varies from track 3 to 8.

The above two current practices both have drawbacks. The network-based approach forces an ABR scheme to choose lower tracks due to the network bandwidth limit. It provides no explicit control on what quality will be chosen for a particular chunk position, thus leading to highly variable quality across the rendered chunks (§7.2). The practice of TBF does not account for the high quality variability across chunks within the same track. For example, the purple points in Fig.1(b) represent chunks encoded at track 4 (or 480p). When using TBF with 480p as top track, the quality for some chunk positions is lower than 60 (i.e., the threshold for good quality in VMAF [26]), *no matter what ABR scheme is being used and how much network bandwidth is available.*

3.2 Quality-aware Data Saving

To address the drawbacks of the current data saving practices, we propose a quality-aware strategy for reducing data usage. We assume that a user is provided with multiple viewing quality options (e.g., good, better, best), with the understanding that the saving is higher under a lower viewing quality option and vice versa. For an option chosen by a user, the player will map it to a particular quality value, referred to as *target quality*, and the goal of the ABR logic is to maintain the quality to be close to the target quality, subject to the network bandwidth constraints. In contrast to the current data saving practices, the above target quality based strategy provides data savings while directly controlling the quality level.

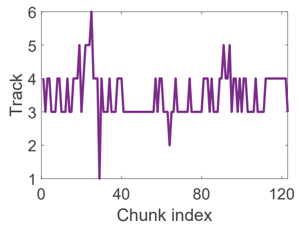
The target quality can be specified in terms of a wide range of perceptual quality metrics. While there is no single agreed-upon way of defining a good perceptual quality, existing literature has established certain metrics, e.g., through threshold values in VMAF and PSNR (see §2.2). As an example, VMAF values of 60 and 80 are the thresholds for "good" and "very good" quality, respectively [26]. The player can set the target quality in VMAF values based on the viewing quality option that a user chooses: when the "good", "better", "best" option is chosen, the target quality is set to VMAF 60, 70, 80, respectively. Users do not need to numerically specify the target quality. Instead, they only need to select a desired viewing quality option such as good/better/best—similar to the current practice in commercial streaming systems. For a given video, the quality metrics such as PSNR and VMAF can be calculated by the server after the video is encoded, and then shared with the client. In addition, a video player can also automatically decide the target quality based on the user's cellular data plan, the cellular data budget, the video content type, and the user's historical preferences. Furthermore, the target quality can be changed over time during the playback; our schemes in §4 and §5 can be applied to dynamic target qualities.

4 CHUNK-BASED FILTERING (CBF)

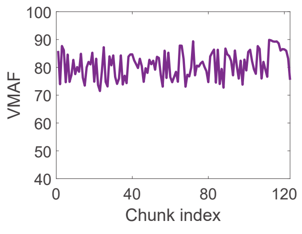
We first describe the CBF approach, and then detail its deployment scenarios and how to leverage it in ABR streaming.

4.1 CBF Approach

CBF is motivated from the two video quality and bitrate tradeoffs in §2, i.e., (i) increasing bitrate leads to diminishing gain in improving quality, and (ii) the chunks in the same track exhibit significantly

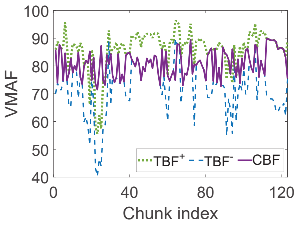


(a) Top track when applying CBF.

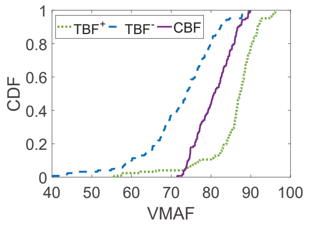


(b) Quality for top track with CBF.

Figure 3: Illustration of CBF (ED, $Q_r = 80$).



(a) Quality for top track



(b) CDF for quality of top track

Figure 4: CBF, TBF$^-$, and TBF$^+$ (ED, $Q_r = 80$).

variable quality. Specifically, for a given target quality, Q_r, CBF filters the tracks that are undesirable on *a per-chunk basis* as follows. For the i-th chunk position, let $q_{i,\ell}$ denote the quality for track ℓ, which can be obtained right after the encoding process at the server. Then, for a given Q_r, CBF sets the top level for chunk position i to $\bar{\ell}_i$ so that the corresponding quality, $q_{i,\bar{\ell}_i}$, is closest to the target quality, Q_r, among all the tracks for chunk position i (i.e., $|q_{i,\bar{\ell}_i} - Q_r|$ is the smallest). In other words, for chunk position i, all the encodings (i.e., tracks) that are above $\bar{\ell}_i$ will not be considered in ABR streaming.

Fig. 3(a) illustrates CBF using an example. It plots the top track after filtering by CBF for each chunk position when the target quality is 80 (VMAF). The video has 6 tracks originally. We see that the top track after CBF varies from 1 to 6 for the different chunk positions (with 53% and 39% as track 3 and 4, respectively, and 6% above 4, and 2% below 3). As an example, for the 29th chunk, the lowest track (i.e., track 1) is sufficient to achieve the target quality 80. A manual inspection reveals that this chunk contains very simple scenes that require less bits to encode. Fig. 3(b) plots the highest quality variant for each chunk position that CBF retains. We see that 95% of the chunk positions have top quality within 10% of the target quality (i.e., between 72 and 88).

4.2 CBF vs. TBF

We next compare CBF and TBF. Specifically, we consider two variants of TBF as follows. For a video, let $\widetilde{Q}(\ell)$ represent the average quality of all the chunks in track ℓ. Let ℓ^- and ℓ^+ denote two adjacent tracks, $\ell^- = \ell^+ - 1$, satisfying that $\widetilde{Q}(\ell^-) \leq Q_r$ and $\widetilde{Q}(\ell^+) > Q_r$. The first variant of TBF, denoted as TBF$^-$, caps the top track to ℓ^-, i.e., it removes all the tracks that are higher than ℓ^-. The second variant of TBF, denoted as TBF$^+$, caps the top track to ℓ^+. Clearly, TBF$^-$ is more aggressive in filtering out tracks than TBF$^+$.

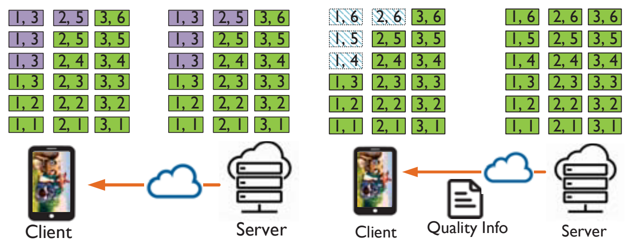
Fig. 4(a) plots the quality of the top track for each chunk position under CBF, TBF$^-$, and TBF$^+$ for one video when the target quality is 80. For this setting, $\ell^- = 3$ and $\ell^+ = 4$. In TBF$^+$, all the tracks above track 4 (with resolution 480p) are removed, which coincides with YouTube's data saving option (§3.1). In Fig. 4(a), the quality for a given chunk position represents the maximum achievable quality when the network bandwidth is sufficiently large. We see that the quality under CBF is overall much closer to the target quality than that under the two TBF variants. Fig. 4(b) shows the cumulative distribution function (CDF) corresponding to the quality values in Fig. 4(a). For TBF$^-$ and TBF$^+$, only 56% and 53% of the chunk positions have quality within 10% of the target quality, compared to 95% under CBF. We observe similar results as above for other videos and target quality investigated. It is easy to prove that, for any chunk position, the top quality under CBF is no farther away from the target quality than that under the two TBF variants.

4.3 Deployment Scenarios

From a practical perspective, a key advantage of CBF is that it can be *incrementally* deployed in the existing DASH and HLS streaming pipelines at either the server or client side. In both deployment scenarios, the server does not remove any chunk from its storage. Rather, it modifies or extends the manifest file that it transmits to the client. We next describe the two deployment scenarios, and end the section with a brief description of using CBF in ABR streaming.

Server Side Deployment. Two approaches, called *chunk variant trimming* and *chunk variant substitution*, can be used for server side deployment. In chunk variant trimming, the server simply modifies the manifest file so that, for each chunk position, it only lists the chunk variants that remain after CBF filtering. As an example, in Fig. 5(a), for chunk 1, only the three lowest track variants will be listed in the manifest file. In chunk variant substitution, the server makes the filtering completely transparent to the client operation by substituting the information for certain chunk variants as follows. Consider the chunks at position i. Let $i.\ell$ represent the chunk at level ℓ. Let $\bar{l}$ denote the top track after the filtering by CBF. Then the server modifies the manifest file so that the information for chunk $i.\ell$, $\ell > \bar{\ell}$, is replaced with the information of chunk $i.\bar{\ell}$. In this way, each playback position still has the same number of levels, and the changes through CBF is transparent to the client. For example, in Fig. 5(a), for chunk 1, levels 4, 5 and 6 are filtered according to CBF; the server modifies the manifest file to replace the information for chunks 1.4, 1.5 and 1.6 with that of 1.3.

We have verified that the above two approaches work in the context of both DASH and HLS protocols and common packaging formats such as Fragmented MP4 and MPEG-2 TS [2, 15]. The chunk variant substitution approach clearly works when the media format does not include separate initialization segments (i.e., each chunk is self-initializing). It can also be realized for media formats where separate initialization segments (each containing information required to initialize the video decoder to decode a particular chunk) are included, as long as a proper initialization segment is specified for each chunk in the manifest file. We have confirmed through experiments that both DASH and HLS have ways to specify such associations, and that, when presented with the appropriately modified manifest file, the player was able to correctly decode and play the associated video. In DASH, this can be achieved using



(a) Server side deployment (b) Client side deployment

Figure 5: Illustration of CBF deployment.

"Period" construct [15]. In HLS, this can be achieved through extra "EXT-X-MAP" tags [2].

Client Side Deployment. When CBF is deployed at the client, the server further needs to transmit the quality information for each chunk to the client, e.g., by including the information in the manifest file, as our prototype implementation of CBF in two ABR streaming platforms (§6). The client, when making rate adaptation decision for a chunk position, will exclude the levels that are above the top track for that chunk position (illustrate as shaded chunks in Fig. 5(b)). Note that while quality metrics can be carried in the media file, e.g., following the ISO standard [16], for rate adaptation purposes, the quality information needs to be known to the client beforehand to assist the decision making, instead of extracting after a chunk is downloaded. Therefore, our implementation embeds the quality information in the manifest file instead of the media file.

Leveraging CBF in ABR Streaming. CBF can be retrofitted to, and improve the performance and data efficiency of existing ABR schemes that may not be quality-aware themselves. This can be achieved either through server-side or client-side deployment of CBF. Specifically, under server-side deployment of CBF, an existing ABR scheme simply selects from the remaining levels for each chunk position. The scheme does not need to leverage or even be aware of any quality information. It can simply aim at maximizing the bitrate (as an indirect way of maximizing the quality) with other QoE considerations, as in most existing schemes. Under client-side deployment of CBF, the client can add a function that applies CBF, and then pass the information of the remaining tracks for a chunk position to an existing ABR algorithm.

5 GROUNDS-UP DESIGN: QUAD

Besides integrating CBF into existing ABR schemes, another approach to design target quality aware ABR adaptation schemes is to develop them from the ground up. Such schemes, since explicitly designed with the target quality in mind, have the potential to outperform existing schemes enhanced with CBF. To demonstrate this approach, we propose one design, called QUAD (QUality Aware Data-efficient streaming), based on control theory. QUAD explicitly integrates the goal of approaching the target quality into its online optimization framework. As a result, it is more capable of maintaining the target quality, and more adaptive to the fluctuating network conditions compared to using CBF with existing schemes.

As shown in Fig. 6, QUAD takes a target quality as input, and leverages an optimization formulation and feedback control to optimize the QoE metrics while approaching the target quality.

Target Quality based Optimization. The optimization formulation below aims to make the chosen chunks' quality approach the target quality while minimizing rebuffering and quality changes. Let ℓ_t denote the track number selected at time t. Let $R_t(\ell_t)$ denote the corresponding bitrate of the selected chunk. We use $R_t(\ell_t)$ instead of $R(\ell_t)$ to accommodate VBR encoding whose bitrate is both a function of ℓ_t and time t since the bitrate can vary significantly even within a track. The client selects ℓ_t from the set of levels that remains after CBF. The optimization problem is to determine the track, ℓ_t, so that the following objective function is minimized:

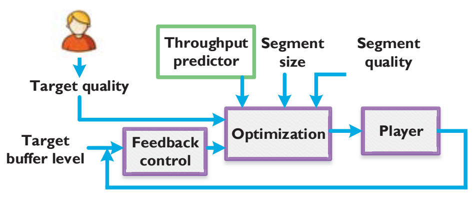


Figure 6: Design diagram of QUAD.

$$\begin{aligned} J(\ell_t) &= \left\| \max\left(0, u_t R_t(\ell_t) - \widehat{C}_t\right) \right\|^2 + \alpha \left\| Q_r - Q_t(\ell_t) \right\|^2 \\ &+ \eta \left\| Q_t(\ell_t) - Q_{t-1}(\ell_{t-1}) \right\|^2, \end{aligned} \tag{1}$$

where u_t is the controller output and $\widehat{C}_t$ is the estimated link bandwidth at time t, $\alpha > 0$ and $\eta > 0$ are parameters for the second and third terms, respectively, Q_r is the target quality specified by the user, $Q_t(\ell_t)$ denotes the quality of the chunk at track ℓ_t for time t, and $Q_{t-1}(\ell_{t-1})$ represents the quality of the previous chunk (here we slightly abuse the notation by using t to represent the index of the chunk for time t and use $t-1$ to represent the index of the previous chunk).

The formulation in Eq. (1) is a least-square optimization problem. In the first term, $u_t R_t(\ell_t)$ represents the bandwidth requirement of the selected track, derived from the feedback control that we will explain shortly. For now readers can regard it as a black box. The first term is zero if the bandwidth requirement of the selected track is no more than the estimated network bandwidth; otherwise, a stall may potentially occur, so it incurs a penalty that equals to the amount of bandwidth that is exceeded by the bandwidth requirement. The second term depends on how much the chosen quality for the chunk deviates from the target quality. The sum of the first and second terms allows the chosen track to be as close to the target quality as possible, while does not overly exceed the network bandwidth (to avoid stalls). The last term penalizes quality changes between two adjacent chunks, in order to maintain a more consistent quality and smooth playback.

We apply normalization in Eq. (1) since the first term is of a different unit from the second and third terms. Specifically, we normalize all the three terms to be unitless as follows. The first term is normalized by $\widehat{C}_t$, the estimated bandwidth, and the other two terms are normalized by Q_r (since QUAD selects from the tracks that remain after CBF, the maximum quality is approximately Q_r). In Eq. (1), α and η represent the weights for the second and third terms, respectively. We set both of them to 1 since all the three terms in Eq. (1) are important QoE metrics.

We see from Eq. (1) that choosing ℓ_t above the target quality Q_r is not beneficial in minimizing the objective function. Therefore, we may apply CBF before solving the optimization problem to reduce the problem space and improve the runtime efficiency. Let $\mathcal{L}_t$ be the set of track levels for chunk t (retained after CBF). We can find the optimal solution to (1) by evaluating Eq. (1) using all possible values of $\mathcal{L}_t$, leading to computational overhead $O(|\mathcal{L}_t|)$. In §8, we show that QUAD is very lightweight using our implementation.

Feedback Control Block. In the first term in (1), $u_t R_t(\ell_t)$, is derived from the feedback control block shown in Fig. 6. We use PID control [3] as the underlying control framework since it is simple and robust for ABR streaming [39]. Specifically, the PID control block works by continuously monitoring the difference between the target and current buffer levels of the video player, and adjusting the control signal to maintain the target buffer level, which helps to avoid stalls. We define the controller output, u_t, as:

$$u_t = \frac{C_t}{R_t(\ell_t)}, \tag{2}$$

where C_t denotes the network bandwidth at time t, and $R_t(\ell_t)$ denotes the bitrate of the chunk selected for time t. The control policy is defined as:

$$u_t = K_p(x_r - x_t) + K_i \int_0^t (x_r - x_\tau)d\tau + \mathbf{1}(x_t - \Delta) \tag{3}$$

where K_p and K_i denote respectively the parameters for proportional and integral control (two key parameters in PID control), x_r is the target buffer level, x_t is the current buffer level (in seconds) at time t, Δ denotes the playback duration of a chunk, and the last term, $\mathbf{1}(x_t - \Delta)$, is an indicator function (1 when $x_t \geq \Delta$ and 0 otherwise), which makes the feedback control system linear, and hence easier to control and analyze. From (2), we derive $C_t = u_t R_t(\ell_t)$ and plug it into (1).

Further Reducing Rebuffering. To avoid rebuffering when the current buffer level is low, we further use a heuristic. Specifically, if $x_t < 4\Delta$, i.e., there are less than four chunks in the buffer, then the track is selected as $\min(\ell_f, \widehat{C}_t/u_t)$, where ℓ_f is the lowest track with fair quality for that chunk position. In other words, when the current buffer level is low, we ignore the goals of achieving the target quality and reducing quality changes (i.e., the last two terms in (1)), and only consider the first term (to reduce the risk of stalls). In that case, we first select the track based on $\widehat{C}_t/u_t$. Since $\widehat{C}_t$ can be an overestimate of the actual network bandwidth, we further bound the selected track to be no more than level ℓ_f. For the videos that we use in our evaluation (see §6), we set ℓ_f to 2, which has significantly higher quality than track 1.

6 IMPLEMENTATION & EVALUATION SETUP

Implementation. We implemented client-based CBF and QUAD in two popular players, `dash.js` [17] and ExoPlayer [18]. We made a set of non-trivial changes. First, we included the quality information of each chunk in the manifest file. We then modified `dash.js` and ExoPlayer so that the chunk quality information can be passed to the ABR logic. We also implemented a new module `cbf.js` to realize CBF, and a new rate adaptation module `quad.js` to realize QUAD in `dash.js`. In ExoPlayer, we created two new classes, `cbf.java` and `quad.java`, for CBF and QUAD. The total number of LoC that is involved in the above changes is 336 in `dash.js` and 396 in ExoPlayer. In `dash.js`, we developed a bandwidth estimation module that responds to playback progress events and estimates network throughput using the harmonic mean of the last 5 chunks.

Evaluation Setup. Our evaluations use a combination of controlled lab experiments with our implementations in `dash.js` and ExoPlayer, as well as simulations, all driven by real-world network bandwidth traces collected from commercial cellular networks. This methodology allows repeatable experiments and to evaluate different schemes under identical settings. In §8.4, we also run in-the-wild tests using ExoPlayer on a phone over an LTE network.

Network Traces and Videos. We collected a total of 42 hours of network traces over two large commercial LTE networks in the U.S. Our traces consist of per-second network bandwidth measurements, which are collected on a phone, by recording the throughput of a large file downloading from a well-provisioned server. They cover a diverse set of scenarios, including different time of day, different locations, and different movement speeds (stationary, walking, local driving, and highway driving). We selected 50 *challenging* traces, each of 700 seconds, with the average network bandwidth below 1 Mbps. The reason for choosing such traces is because even with target quality of 80 (in VMAF, regarded as very good quality [26], the highest quality we evaluate), the average bandwidth requirement is below 1 Mbps (varies from 500 to 800 kbps) for the videos we use. Using higher bandwidth traces will diminish the differences among different schemes; these low-bandwidth traces, measured from commercial cellular networks, represent challenging conditions that do occur in real networks, e.g., when the network is congested or the signal is poor. We use 4 VBR and 4 CBR videos in our evaluation. They are encoded using the four raw videos (ED, BBB, Sintel, ToS) that we have access to (see §2.1), and hence we can calculate the perceptual quality using raw videos as the reference. The results in §7 and §8 focus on VBR videos; the results for CBR videos are consistent and omitted due to space.

CBF with Existing ABR Schemes. We use CBF as a prefilter for the following state-of-the-art rate adaptation schemes: (i) **RobustMPC** [50], which is a well-known scheme based on model predictive control. (ii) **PANDA/CQ** [27], which directly incorporates video quality information in ABR streaming. It maximizes the minimum quality for the next N chunks, which achieves better fairness (in terms of quality) among multiple chunks. (iii) **BOLA-E** [43, 44], which selects the bitrate to maximize a utility function considering both rebuffering and delivered bitrate. (iv) **ExoPlayer**'s ABR adaptation [18], referred to as *Exo* henceforth, which is essentially a rate based logic, i.e., selecting the track based on bandwidth estimation. The results for RobustMPC and PANDA/CQ are obtained through trace-driven simulation; the evaluation involving BOLA-E and ExoPlayer ABR logic is done using open source implementation in `dash.js` and ExoPlayer, respectively.

Offline Optimal Scheme. We further use an offline optimal solution as a baseline to evaluate the performance of various schemes. This offline scheme assumes that the entire network bandwidth is known beforehand. It considers three QoE metrics related to target quality, quality changes, and stalls. Specifically, for a video with n chunks, it selects tracks $\ell_1, \ldots, \ell_n$ to minimize

$$\begin{aligned} J(\ell_1, \ldots, \ell_n) \quad &= \quad \sum_{t=1}^{n} (Q_r - Q_t(\ell_t))^2 + \sum_{t=1}^{n-1} (Q_{t+1}(\ell_{t+1}) - Q_t(\ell_t))^2 \\ &+ \quad \gamma T_r(\ell_1, \ldots, \ell_n) \end{aligned}$$

where Q_r is the target quality, $Q_t(\ell_t)$ is the quality for ℓ_t, and $T_r(\ell_1, \ldots, \ell_n)$ is the rebuffering duration, and γ is the weight for rebuffering. The results below use $\gamma = 100^2$, i.e., we penalize each second of rebuffering with square of the maximum VMAF quality.

ABR Configurations. Following practices in commercial production players [49], we set the player to start the playback when two chunks are downloaded into the buffer. The track for the first chunk is selected to be the middle track (i.e., track 3). Unless otherwise stated, we use the harmonic mean of the average download throughout for the past 5 chunks as the bandwidth prediction, as it has been shown to be robust to measurement outliers [22, 50]. For the results using ExoPlayer, the bandwidth prediction uses the built-in sliding percentile technique [18]. Unless otherwise stated, for all schemes, we set the maximum client-side buffer size to 120 seconds. This is reasonable for Video on Demand (VOD) streaming, and is consistent with existing practices that set the maximum buffer limit to hundreds of seconds [17, 19, 49]. An ABR scheme may use thresholds to control when to stop and resume downloading based on the buffer level. When comparing with another scheme, we ensure that our schemes use the same thresholds as used in that scheme. For QUAD, the controller parameters, K_p and K_i, are selected by adopting the methodology outlined in [39]. Specifically, we varied K_p and K_i, and confirmed that a wide range of K_p and K_i values lead to good performance.

Perceptual Quality Metric. We measure video quality using VMAF (§2). Specifically, we use VMAF phone model (instead of TV model) given our focus on cellular networks. In the testing, we assume that a user selects from three quality options, good/better/best. Correspondingly, the player maps these quality options to VMAF values of 60 (for "good"), 70 (for "better"), and 80 (for "best").

Performance Metrics. We use five metrics: four for measuring different aspects of user QoE and one for measuring data usage. All metrics are computed with respect to the delivered video, i.e., considering the chunks that have been downloaded and played back. The metrics are listed as follows. (i) *Quality of all the chunks*: measures how the quality of each chunk differs from the target quality. (ii) *Low-quality chunk percentage*: measures the proportion of the chunks that were selected with low quality during a streaming session. The reason for using this metric is because human eyes are sensitive to bad quality chunks [33]. We identify VMAF values below 40 as low-quality based on [26]. (iii) *Rebuffering duration*: measures the total rebuffering/stall time in a streaming session. (iv) *Average quality change per chunk*: defined as the average quality difference of two consecutive chunks in playback order for a streaming session (since human eyes are more sensitive to level changes in adjacent chunks). (v) *Data usage*: measures the total amount of data downloaded for a streaming session. For metrics (ii)-(v), a lower value is preferable; for (i), we measure how close it is to the target quality. In addition to the above metrics, we further explored the number of stalls during a session in various cases. Our results show that using CBF in existing schemes reduces the number of stalls, and QUAD leads to fewer stalls compared to existing scheme enhanced with CBF in almost all the cases.

7 EVALUATION EXISTING ABR SCHEMES ENHANCED WITH CBF

We evaluate the performance of adding CBF to existing ABR schemes. We focus on two existing schemes, RobustMPC and PANDA/CQ. The performance of other schemes with CBF is deferred to §8.

Table 2: RobustMPC vs. RobustMPC+CBF.

	Video	Avg. dev. from target quality	% of traces w/ > 20% low-qual. chunks	Avg. stall dura. (s)	Avg. quality change	Data usage (MB)
VBR, 60	ED	25, 9	48%, 10%	8, 0	13, 9	39, 19
	BBB	27, 9	37%, 8%	6, 0	14, 11	36, 14
	Sintel	28, 11	6%, 0%	8, 0	10, 12	55, 18
	ToS	26, 14	56%, 40%	5, 0	13, 12	46, 22
VBR, 80	ED	23, 15	48%, 12%	8, 2	13, 10	39, 28
	BBB	22, 14	37%, 8%	6, 1	14, 13	36, 21
	Sintel	19, 9	6%, 0%	8, 0	10, 9	55, 29
	ToS	24, 16	56%, 21%	5, 2	13, 10	46, 35

* The two numbers in each cell are the results for RobustMPC and RobustMPC+CBF, respectively.

7.1 Benefits of CBF

Fig. 7 shows the performance of RobustMPC and PANDA/CQ with and without CBF for one video across the network traces when the target quality is 80. The results for the five performance metrics are shown in the five subplots in the figure: the first subplot is the CDF across individual chunks across all runs; the rest four are CDFs across runs, with each run corresponding to a different network trace. We observe that CBF improves performance for both ABR schemes, across all metrics. Specifically, compared to the original schemes, adding CBF leads the quality to be closer to the target quality, and reduces the percentage of low-quality chunks, the quality changes, rebuffering, and data usage.

Table 2 summarizes the results for four VBR videos for RobustMPC with and without CBF for target quality of 60 and 80, respectively. The first column is the average deviation from the target quality across all the network traces; the value for one trace is $(\sum |q_i - Q_r|)/n$, where q_i is the quality of chunk i in the rendered video, Q_r is the target quality, and n is the number of chunks in the video. The second column shows the percentage of the traces that have more than 20% of low-quality chunks. The third column shows the average rebuffering duration for the traces where either case (i.e., with or without CBF) has rebuffering. The last two columns show the average quality changes and the data usage, averaged across the network traces. We see that CBF reduces the deviation from the target quality by 37-67%, reduces the number of traces with frequent low-quality chunks by 6-42%, and reduces the average quality change by 7-31%. For many videos, across all network traces, using CBF leads to no stalls compared to substantial amount of stalls without CBF. The data usage with CBF is 34-67% lower than that without CBF. In the extreme case when the bandwidth is sufficiently high, RobustMPC without CBF will choose the highest track (i.e., track 6). The data usage for the four videos will be 203, 179, 240, and 209 MB respectively, 5.6-7.5 times higher than the corresponding values with CBF under target quality 80 (the ratios are even higher for target quality 60).

We see similar results for PANDA/CQ with and without CBF. The above results demonstrate that CBF can significantly improve the performance of existing ABR schemes by prefiltering the chunks whose qualities are higher than the target quality, and hence steering the schemes to the set of more desirable choices.

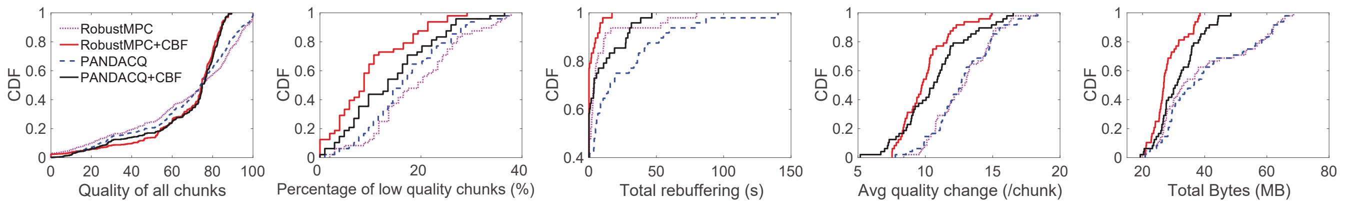


Figure 7: Two existing ABR schemes with and without CBF (ED, YouTube encoded, target quality 80).

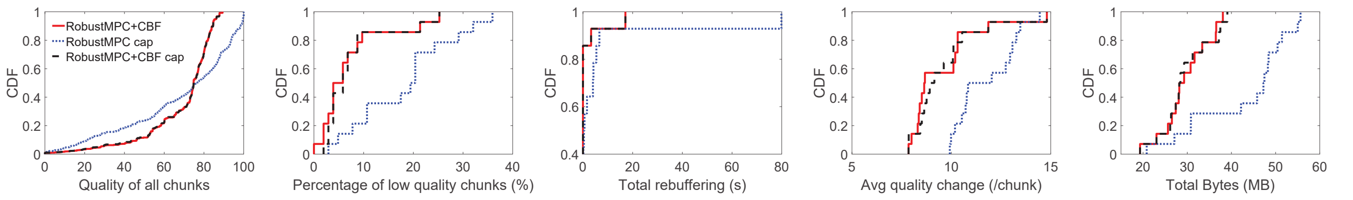


Figure 8: CBF versus capping network bandwidth (ED, YouTube encoded, target quality 80).

7.2 CBF vs. Network Bandwidth Cap

We now compare CBF with an existing practice for saving data by capping network bandwidth (§3.1). Specifically, for an existing ABR scheme, we consider three cases: (i) we assume that the cellular network provider caps the network bandwidth to 1.5 Mbps; (ii) there is no cap on the network bandwidth, while the scheme is used together with CBF; and (iii) there is a bandwidth cap and the scheme is used with CBF. Since the network bandwidth of some traces is low and imposing the constraint of 1.5 Mbps leads to little impact, we choose a subset of network traces where the cap meaningfully changes the available bandwidth. Specifically, a network trace is chosen if the bandwidth estimated using a window of 1 second is larger than 1.5 Mbps for at least 10% of the time. The results below are obtained from 20 traces selected as above.

Fig. 8 shows the results for RobustMPC with target quality 80; the results for PANDA/CQ show a similar trend. We see that the two variants with CBF achieve similar performance. Both of them significantly outperform the other variant (i.e., with bandwidth cap but no CBF) in all performance metrics. The results demonstrate the effectiveness of CBF compared to the network bandwidth capping approach. They also indicate that CBF can co-exist with the network bandwidth capping approach. The above results are for one video with target quality 80. We observe similar results for other videos and target qualities. For lower target qualities (i.e., 60 and 70), we observe that CBF achieves even better performance.

7.3 CBF vs. TBF

We now compare CBF with TBF, which is used by commercial streaming services such as YouTube and Amazon for reducing data usage (see §3.1). Specifically, we consider two variants of TBF, i.e., TBF$^-$ and TBF$^+$ (see §4.2).

Fig. 9 plots the performance of RobustMPC with CBF, and with the two TBF variants, referred to as RobustMPC$^-$ and RobustMPC$^+$. The results are for one VBR video when the target quality is 80. For this setting, the top track in RobustMPC$^-$ and RobustMPC$^+$ is track 3 and 4, respectively. We make the following observations. (i) RobustMPC$^-$ is quite conservative. It provides low rebuffering time. However, the negative side is that it undershoots the target quality (its average deviation from the target quality is 20 VMAF points across the chunks, compared to 15 under RobustMPC+CBF), and leads to a higher percentage of low-quality chunks (it has at least 10% low-quality chunks for 100% of the traces, compared to 58% and 68% of the traces under RobustMPC+CBF and RobustMPC$^+$, respectively). (ii) RobustMPC$^+$, on the other hand, is too aggressive. It overshoots the target quality (it exceeds the target quality in 50% of the chunks, compared to 30% under RobustMPC+CBF), incurs the highest rebuffering time, and consumes the highest network bandwidth among the three schemes. Also, it results in highly variable quality by choosing high quality for some chunks while leaving a higher percentage of chunks with low quality. (iii) RobustMPC+CBF strikes a better balance among the aforementioned factors. It also achieves a quality that is closest to the target quality. We observe similar results for other videos and target qualities. We further compare the three variants of PANDA/CQ and observe that PANDA/CQ+CBF achieves better performance than the others. The reason, as explained in §4.2, is that compared to TBF, CBF filters out chunks at a finer granularity (on the basis of chunks instead of tracks), thus allowing it to make better choices.

8 EVALUATION OF QUAD

In this section, we compare QUAD and existing schemes enhanced with CBF. We also evaluate them in `dash.js` and ExoPlayer.

8.1 QUAD vs. CBF

Fig. 10 plots the performance of QUAD and two existing schemes (RobustMPC and PANDA/CQ) with CBF for two videos, based on trace-driven simulations. It also plots the results of the offline optimal scheme (see §6). We observe that for all five metrics except the data usage, QUAD achieves performance closest to that of the offline optimal; for data usage, the offline optimal uses more data than other schemes, consistent with the best quality that it achieves. QUAD outperforms RobustMPC+CBF and PANDA/CQ+CBF for both videos in Fig. 10. For video BBB (Fig. 10 bottom row), while RobustMPC+CBF has similar rebuffering as QUAD, it leads to a noticeably worse quality (the average deviation from the target quality is 14 VMAF points across the runs under RobustMPC+CBF, and 10 under QUAD). For video ED (Fig. 10 top row), the overall quality of the two schemes is similar, while RobustMPC+CBF has

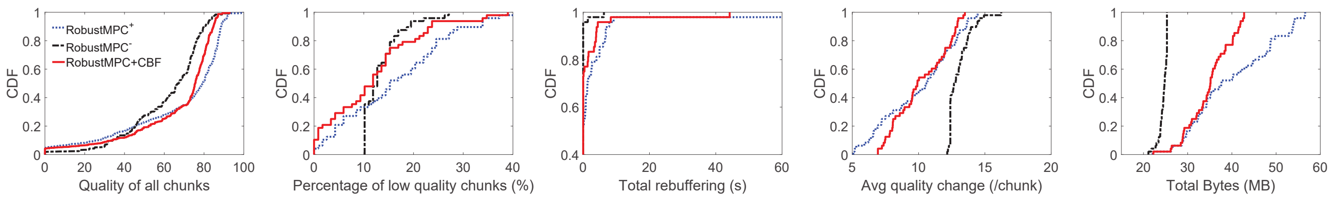


Figure 9: RobustMPC with CBF vs TBF (ToS, YouTube encoded, target quality 80).

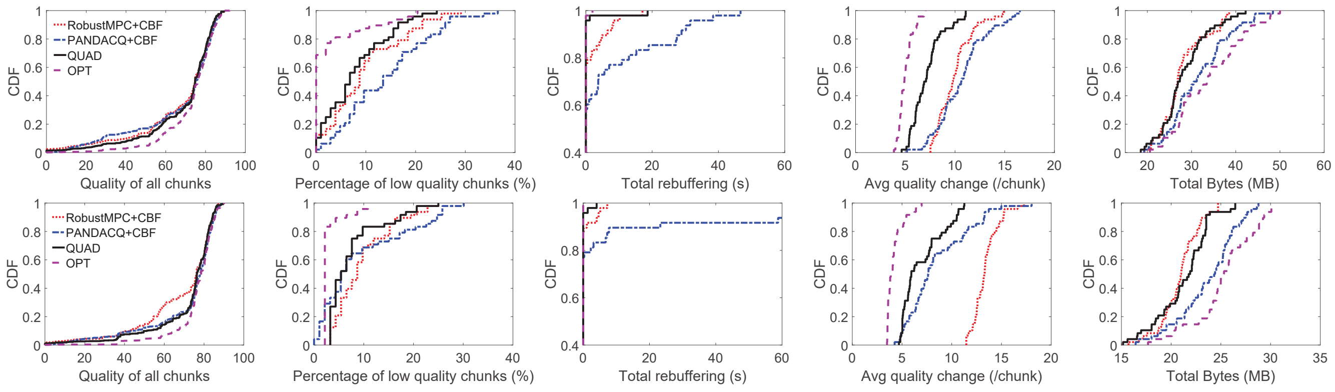


Figure 10: QUAD vs. two existing schemes with CBF (ED, BBB, YouTube encoded, target quality 80).

more rebuffering than QUAD (it has rebuffering in 23% of the runs, compared to only 4% in QUAD). For both videos, the average quality change per chunk under QUAD is much lower compared to RobustMPC+CBF: 7 under QUAD for both videos, compared to 10 and 13 under RobustMPC+CBF. The overall quality of PANDA/CQ+CBF is comparable to that of QUAD for both videos, but has significantly more rebuffering, higher quality changes, and uses more data.

8.2 `dash.js` based Evaluation

Using our developed QUAD in `dash.js` (version 2.4.1), we compare its performance with BOLA-E [44] and BOLA-E+CBF. The original BOLA-E design takes a single bitrate rate for each track, which is suitable for CBR videos; for VBR videos, we improved the design by using the actual chunk sizes in the ABR logic based on [44]. Our evaluation below uses two computers (Ubuntu 12.04 LTS, CPU Intel Core 2 Duo, 4GB memory) with a 100 Mbps direct network connection to emulate the video server (Apache `httpd`) and client. At the client, we use `selenium` [42] to run a Google Chrome web browser and use `dash.js` APIs to collect the streaming performance data. We use `tc` to emulate real-world variable mobile network conditions by "replaying" the network traces (§6). We find QUAD is very light-weight. With the current prototype, the total execution time of QUAD is only about 10ms for a 10mins video.

Fig. 11 shows the performance of QUAD, BOLA-E, and BOLA-E+CBF for one video with target quality 80. We observe that BOLA-E+CBF significantly outperforms BOLA-E in all performance metrics. QUAD further outperforms BOLA-E+CBF in achieving 37% reduction in average percentage of low-quality chunks and 12% reduction in average quality changes across the runs. QUAD has rebuffering in 8% of the runs, compared to 15% under BOLA-E+CBF. The chunk quality of BOLA-E+CBF is close to that of QUAD, with data usage close to than that of QUAD. The results for other videos and target qualities show a similar trend.

8.3 ExoPlayer based Evaluation

We compare the performance of QUAD (using our implementation) with Exo (the default ABR algorithm, see §6, with and without CBF) in ExoPlayer (version 2.4.4). The client is an LG V20 phone (Qualcomm Snapdragon 820, 64GB storage and 4 GB RAM) with Android 7.0. The experiments are conducted over a WiFi network (with consistent tens of Mbps bandwidth). We again emulate the 50 challenging cellular network traces (§6) by "replaying" the traces.

Fig. 12 shows the results for two variants of Exo: the first using the default parameters, and the second (referred to as *ExoTuned*) using tuned parameters to improve its quality. In the first variant (i.e., the default ABR logic), the downloading stops when the buffer level reaches 30 seconds and resumes when the buffer is less than 15 seconds. In ExoTuned, the parameters are set so that the player stops downloading when the buffer reaches 120 seconds (the value we have used for other schemes, see §6), and resumes downloading when the buffer level drops to 105 seconds. Correspondingly, the two parameters for QUAD are set to 120 and 105 seconds as well. We see that ExoTuned indeed achieves better performance compared to Exo since the larger buffer allows the player to download and store more content in the buffer. For both Exo and ExoTuned, adding CBF avoids downloading chunks with excessively high quality, which reduces rebuffering and the data usage (for these two schemes, adding CBF does not lead to much improvement in quality since their quality selections are already quite conservative). We further see that QUAD significantly outperforms all Exo variants. Compared to Exo, QUAD reduces the average deviation from the target quality by 64%, reduces the average percentage of low-quality chunks across all runs by 81%, reduces the average rebuffering (over

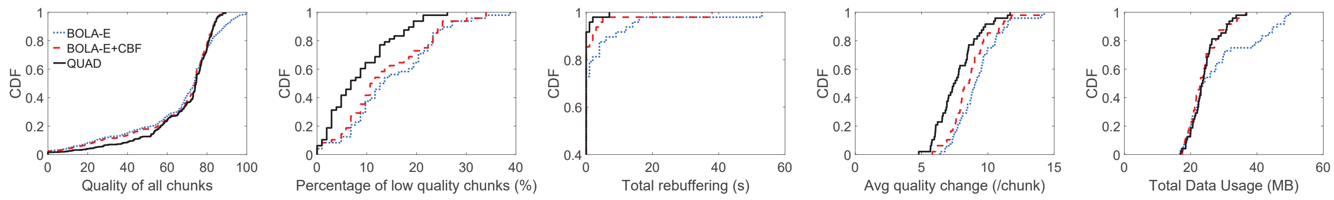


Figure 11: QUAD vs. BOLA-E with and without CBF in dash.js (ED, YouTube encoded, target quality 80).

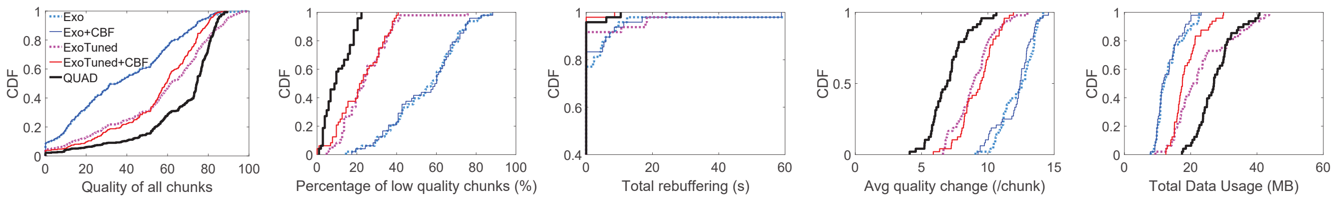


Figure 12: QUAD vs. Exo with and without CBF in ExoPlayer (ED, YouTube encoded, target quality 80).

Table 3: In-the-wild tests using ExoPlayer.

	Avg. dev. from target quality	Avg. % of low quality chunks	Avg. stall dura. (s)	Avg. quality change	Data usage (MB)
Low bw.	19.5±6.1	17.3±10.2	2.4±4.6	7.8±0.9	26.1±6.5
	20.0±6.2	14.2±8.5	0	8.5±1.1	21.0±4.3
	11.9±2.9	6.5±3.7	0	5.6±1.2	28.8± 3.7
High bw.	17.9±0.6	0	0	1.5±0.3	181.0±9.9
	3.9±0.1	0	0	5.0±0.1	47.7±0.6
	4.2±0.1	0	0	4.6±0.1	45.2±0.1

* The three rows in each cell are the results for ExoTuned, ExoTuned+CBF, and QUAD, respectively.

of the subset of runs where either algorithm has rebuffering) by 86%, and reduces the average quality change across the runs by 43%. Compared to ExoTuned+CBF, the corresponding reductions in the deviation from the target quality, the average percentage of low-quality chunks, and the quality changes are 40%, 46%, and 22%, respectively, albeit QUAD uses more data.

8.4 In-the-wild Tests

So far, our evaluation has been through large-scale trace-driven simulation and experimentation using real systems. We next present in-the-wild test results, by running QUAD and ExoTuned (which outperforms Exo) over a commercial LTE network. The video we use is ED (YouTube encoded). We consider two settings, one with poor signal conditions and hence low bandwidth (consistently less than 1Mbps and unstable), and the other with good signal conditions and hence high bandwidth. The first setting is in a residential home, and the second one is in an office building. For each setting, we make 10 runs, each consisting of three schemes (QUAD, ExoTuned and ExoTuned+CBF, in a random order). The results are shown in Table 3, which lists the mean and standard deviation across the runs for each case. Under high bandwidth conditions, compared to ExoTuned alone, we see significant benefits of CBF and QUAD in reducing data usage and achieving quality close to the target quality. Under low bandwidth conditions, the results exhibit more variations due to the fluctuating network bandwidth caused by the poor signal strength. Despite that, compared to ExoTuned, we still observe that CBF significantly reduces the percentage of low-quality chunks, and QUAD achieves the best QoE overall. ExoTuned+CBF and QUAD have no rebuffering, while ExoTuned shows non-negligible rebuffering duration for the 10-minute video.

9 RELATED WORK

Improving Video QoE. In addition to the schemes already described in §6, QDASH [31] tries to reduce quality switches during adaptation. BBA [19] proposes adaptation schemes based on client-side buffer information. PIA [39] designs a PID-based framework to account for various requirements of ABR streaming. Pensieve [30] proposes a system that generates ABR algorithms using reinforcement learning. Oboe [1] pre-computes the best possible ABR parameters for different network conditions and dynamically adapts the parameters at run-time. CAVA [38] proposes design principles for VBR-based ABR streaming and a concrete scheme that instantiates these design principles. PANDA/CQ [27] directly incorporates video quality information in ABR streaming and maximizing QoE by dynamic programming. The study in [5] outlines the challenges and open issues in consistent-quality streaming such as the scheme in [27]. None of the above studies explicitly considers data efficiency.

Reducing Data Usage. A number of studies have proposed techniques for reducing data in the content encoding process. Chen et al. [6] propose an optimization framework to identify the optimal encoding bitrates that minimize the average streaming bitrate, subject to a given lower bound on delivered quality. De Cock et al. [9] present a constant-slope rate allocation approach to improve the Bitrate-Distortion rate. Aaron et al. [32] propose per-title encoding, i.e., each title should receive a bitrate ladder, tailored to its complexity characteristics. Katsavounidis et al. [23] develop a dynamic optimizer framework that searches for optimized encoding parameters. Toni et al. [46] determine the optimal selection of tracks for encoding. Our work differs from the above by jointly improving video QoE and reducing data usage in the streaming process. Therefore, our work is orthogonal to those on reducing data in the encoding process, and can complement those efforts.

The study in [7] manages the tradeoff between monthly data usage and video quality by leveraging the compressibility of videos and predicting consumer usage behavior throughout a billing cycle. Our study differs from it in that we consider the data usage and quality tradeoffs when streaming a video. The study in [40] observes that, for some chunks, lower bitrate tracks may be of similar perceptual quality as higher bitrate tracks. Given a set of encoded ABR tracks, it proposes to perform a server-side chunk replacement (within the same resolution) so that a higher bitrate chunk can be replaced by a lower bitrate chunk with a perceptually similar quality. Unlike our work, this study does not perform an in-depth exploration of how their approach interacts with existing ABR rate adaptation algorithms. QBR [8] aims to improve the efficiency of existing ABR schemes by reducing the data usage while potentially increasing QoE. Specifically, a QBR server provides additional metadata hints to a client, allowing the client to request a reduced bitrate for chunks of low complexity. QBR is not designed to achieve a target quality based on a user-specified option. In addition, it does not provide a grounds-up design as QUAD.

10 CONCLUDING REMARKS

Existing data saving practices for ABR videos often incur undesired and highly variable video quality, without making the most effective use of the available network bandwidth. We identify underlying causes for this behavior and design two novel approaches, CBF and QUAD, to achieve better tradeoffs among video quality, rebuffering, quality variations, and cellular data usage. Evaluations demonstrate that compared to the state of the art, these two schemes achieve quality closer to desired levels, lower stalls, and more efficient data usage. Specifically, using CBF with existing schemes leads to significant benefits in all performance metrics, and QUAD achieves even better QoE compared to existing schemes enhanced with CBF.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers who gave valuable feedback to improve this work, and our shepherd, Roger Zimmermann, 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*.
[2] Apple. 2017. Apple's HTTP Live Streaming. https://goo.gl/eyDmBc. (2017).
[3] Karl Johan Åström and Richard M. Murray. 2008. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press.
[4] AT&T. 2018. Stream Saver. (2018). https://www.att.com/offers/streamsaver.html
[5] Ali Begen. 2016. Spending "Quality"' Time with the Web Video. *IEEE Internet Computing* (2016).
[6] 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).
[7] Jiasi Chen, Amitabha Ghosh, Josphat Magutt, and Mung Chiang. 2012. QAVA: Quota Aware Video Adaptation. In *Proc. of ACM CoNEXT*. 121–132.
[8] William Cooper, Sue Farrell, and Kumar Subramanian. 2017. QBR Metadata to Improve Streaming Efficiency and Quality. In *SMPTE*.
[9] Jan De Cock and Anne Aaron. 2016. Constant-slope rate allocation for distributed real-world encoding. In *Picture Coding Symposium (PCS), 2016*. IEEE, 1–5.
[10] Jan De Cock, Zhi Li, Megha Manohara, and Anne Aaron. 2016. Complexity-based consistent-quality encoding in the cloud. In *ICIP*. IEEE.
[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] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. 2011. Understanding the impact of video quality on user engagement. In *ACM SIGCOMM*.
[13] Ericsson. 2017. Ericsson Mobility Report. https://goo.gl/mjkwSH. (2017).
[14] FFmpeg. 2017. FFmpeg Project. https://www.ffmpeg.org/. (2017).
[15] International Organization for Standardization. 2012. ISO/IEC DIS 23009-1.2 Dynamic adaptive streaming over HTTP (DASH). (2012).
[16] International Organization for Standardization. 2015. ISO/IEC 23001-10:2015 Carriage of timed metadata metrics of media in ISO base media file format. (2015).
[17] DASH Industry Forum. 2017. Reference Client 2.4.1. https://goo.gl/XJcciV. (2017).
[18] Google. 2016. ExoPlayer. https://github.com/google/ExoPlayer. (2016).
[19] 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*.
[20] MulticoreWare Inc. 2018. H.265 Video Codec. http://x265.org/hevc-h265/. (2018).
[21] ITU. 2017. H.264 codec. https://goo.gl/AjvnTs. (2017).
[22] Junchen Jiang, Vyas Sekar, and Hui Zhang. 2012. Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with FESTIVE. In *CoNEXT*.
[23] Ioannis Katsavounidis. 2018. Dynamic optimizer a perceptual video encoding optimization framework. https://goo.gl/zHdium. (2018).
[24] S Shunmuga Krishnan and Ramesh K Sitaraman. 2013. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. *IEEE/ACM Transactions on Networking* 21, 6 (2013), 2001–2014.
[25] TV Lakshman, Antonio Ortega, and Amy R Reibman. 1998. VBR video: Tradeoffs and potentials. *Proc. IEEE* (1998).
[26] Zhi Li, Anne Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara. 2016. Toward A Practical Perceptual Video Quality Metric. (2016). https://goo.gl/ptjrWv.
[27] Zhi Li, Ali Begen, Joshua Gahm, Yufeng Shan, Bruce Osler, and David Oran. 2014. Streaming video over HTTP with consistent quality. In *ACM MMSys*.
[28] Yao-Chung Lin, Hugh Denman, and Anil Kokaram. 2015. Multipass encoding for reducing pulsing artifacts in cloud based video transcoding. In *ICIP*. IEEE.
[29] Yao Liu, Sujit Dey, Fatih Ulupinar, Michael Luby, and Yinan Mao. 2015. Deriving and Validating User Experience Model for DASH Video Streaming. *IEEE Transactions on Broadcasting* 61, 4 (December 2015).
[30] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *Proc. of ACM SIGCOMM*.
[31] Ricky KP Mok, Xiapu Luo, Edmond WW Chan, and Rocky KC Chang. 2012. QDASH: a QoE-aware DASH system. In *ACM MMSys*.
[32] Netflix. 2015. Per-Title Encode Optimization. https://goo.gl/1J5vBv. (2015).
[33] Netflix. 2016. VMAF score aggregation. https://goo.gl/v38JMB. (2016).
[34] Cisco Networks. 2016. Cisco VNI: Global Mobile Data Traffic Forecast Update, 2016-2021. https://goo.gl/64zqTT. (2016).
[35] Pengpeng Ni, Ragnhild Eg, Alexander Eichhorn, Carsten Griwodz, and Pål Halvorsen. 2011. Flicker effects in adaptive video streaming to handheld devices. In *Proc. of ACM Multimedia*.
[36] Jan Ozer. 2017. Finding the Just Noticeable Difference with Netflix VMAF. https://goo.gl/TGWCGV. (September 2017).
[37] The WebM Project. 2017. VP9 Video Codec. https://goo.gl/Xep8rr. (2017).
[38] 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 *ACM CoNEXT*.
[39] 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*.
[40] Benjamin Rainer, Stefan Petscharnig, Christian Timmerer, and Hermann Hellwagner. 2017. Statistically indifferent quality variation: An approach for reducing multimedia distribution cost for adaptive video streaming services. *IEEE Transactions on Multimedia* 19, 4 (2017), 849–860.
[41] Reza Rassool. 2017. VMAF reproducibility: Validating a perceptual practical video quality metric. In *IEEE BMSB*.
[42] Selenium. 2017. Selenium Browser Automation. https://goo.gl/2RaANN. (2017).
[43] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. 2018. From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player. In *MMSys*.
[44] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. 2016. BOLA: Near-Optimal Bitrate Adaptation for Online Videos. In *INFOCOM*. IEEE.
[45] T-Mobile. 2018. T-Mobile Binge On. (2018). https://goo.gl/Q9fbw6
[46] Laura Toni, Ramon Aparicio, Telecom Bretagne, Karine Pires, Gwendal Simon, Alberto Blanc, and Pascal Frossard. 2015. Optimal selection of adaptive streaming representations. *ACM Trans. Multimedia Comput. Commun. Appl.* (2015).
[47] Wikipedia. 2018. Standard-definition television. https://goo.gl/Y5uULb. (2018).
[48] Xiph. 2016. Xiph Video Test Media. https://media.xiph.org/video/derf/. (2016).
[49] Shichang Xu, Z. Morley Mao, Subhabrata Sen, and Yunhan Jia. 2017. Dissecting VOD Services for Cellular: Performance, Root Causes and Best Practices. In *IMC*.
[50] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *SIGCOMM*. ACM.
[51] youtube-dl developers. 2018. youtube-dl. https://goo.gl/mgghW8. (2018).