C2: ABR Streaming in Cognizant of Consumption Context for Improved QoE and Resource Usage Tradeoffs

CHEONJIN PARK, School of Computing, University of Connecticut, Storrs, United States
CHINMAEY SHENDE, School of Computing, University of Connecticut, Storrs, United States
SUBHABRATA SEN, AT&T Labs–Research, Bedminster, United States
BING WANG, School of Computing, University of Connecticut, Storrs, United States

Smartphones have emerged as ubiquitous platforms for people to consume content in a wide range of *consumption contexts (C2)*, e.g., over cellular or WiFi, playing back audio and video directly on phone or through peripheral devices such as external screens or speakers. In this article, we argue that a user's specific C2 is an important factor to consider in Adaptive Bitrate (ABR) streaming. We examine the current practices of using C2 in five popular ABR players, and identify various limitations in existing treatments that have a detrimental impact on network resource usage and user experience. We then formulate C2-cognizant ABR streaming as an optimization problem and develop practical best-practice guidelines to realize it. Instantiating these guidelines, we develop a proof-of-concept implementation in the widely used state-of-the-art ExoPlayer platform and demonstrate that it leads to significantly better tradeoffs in terms of user experience and resource usage. Last, we show that the guidelines also benefit `dash.js` player that uses an ABR logic significantly different from that of ExoPlayer.

CCS Concepts: • **Information systems → Multimedia streaming;**

Additional Key Words and Phrases: Multimedia streaming, consumption context, quality of experience

ACM Reference Format:
Cheonjin Park, Chinmaey Shende, Subhabrata Sen, and Bing Wang. 2024. C2: ABR Streaming in Cognizant of Consumption Context for Improved QoE and Resource Usage Tradeoffs. *ACM Trans. Multimedia Comput. Commun. Appl.* 20, 9, Article 272 (August 2024), 27 pages. https://doi.org/10.1145/3652517

1 Introduction

We consume content on smartphones in a wide range of contexts: we listen to audio/music on our phones, using either the built-in speaker, or headphones that can even support spatial audio [69]; we watch video using the small built-in phone screen, a flip screen [49] that can be unfolded to a larger display, or a connected projector (either built-in [77] or external [38, 47]) that projects the

A preliminary version of this article [56] appeared in MMSys 2022.
Authors' Contact Information: Cheonjin Park, School of Computing, University of Connecticut, Storrs, United States; e-mail: cheon.park@uconn.edu; Chinmaey Shende, School of Computing, University of Connecticut, Storrs, United States; e-mail: chinmaey.shende@uconn.edu; Subhabrata Sen, AT&T Labs–Research, Bedminster, NJ, United States; e-mail: sen@research.att.com; Bing Wang, School of Computing, University of Connecticut, Storrs, Connecticut, United States; e-mail: bing@engr.uconn.edu.

ACM 1551-6865/2024/08-ART272
https://doi.org/10.1145/3652517

content to a larger space. In addition, we connect our phones to external peripherals such as a large-screen high-resolution TV/monitor and 5.1 or 7.2 channel surround sound system [4, 36, 55] when convenient (e.g., in Tailgate parties). The ubiquitous network connectivity to phones, through cellular or WiFi, further allows us to access the Internet anytime anywhere.

We broadly refer to the environment in which a user consumes content as the ***consumption context (C2)***. As examples, streaming a video over a cellular connection to the phone and viewing the content on the phone involves one type of C2, while viewing the video on an external TV would be another type. As we shall see, to deliver good **Quality of Experience** (**QoE**), different types of C2 can have very different resource requirements and user needs. Therefore, being C2-cognizant is important for many applications to appropriately optimize user experience and resource usage.

In this article, we explore using C2 in **Adaptive Bitrate** (**ABR**) streaming, the current de facto technology for video streaming. In ABR streaming, the origin server provides multiple *tracks* or *variants* that represent the same content, but are encoded at different bitrates and quality levels. Each track is divided into multiple *segments*, each containing a few seconds worth of content. During playback, for each segment position, the client uses an ABR rate adaptation logic to dynamically select a variant from the multiple available options to adapt to dynamic network conditions. We consider broadly **HTTP Adaptive Streaming** (**HAS**), including both DASH [42] and HLS [20] protocols, with or without CMAF packaging [11]. At the server, video and audio can be muxed together, or demuxed (i.e., stored and streamed as separate tracks) [60]. We focus on the demuxed paradigm because it has many advantages [60] and is widely adopted in popular streaming services. However, the C2-cognizant framework developed here also applies to the muxed case.

Existing literature on ABR streaming has focused primarily on a single aspect of C2, i.e., available network bandwidth, and produced a range of rate adaptation schemes (see Section 7). While this is certainly very important, many other important aspects of C2 have received little prior attention. We describe two such aspects next. First, existing works focused primarily on video; audio has received much less attention. Even in the well-studied area of rate adaptation, there is very little work on rate adaptation for the case of streaming demuxed video and audio tracks, which requires addressing subtle interactions between audio and video track selections [60]. Second, there has been very little attention on matching the ABR track selection to the contextual needs of the display/audio device where the content is consumed. Specifically, the video can be either displayed on a small phone screen or a large external display, while audio can be played either by the stereo speaker on the phone or an external surround sound system. Even in a single session, the display/audio device can change over time. As we shall see, the choice of the peripheral used for consuming the video or audio has important implications for both QoE and resource usage, and hence needs to be considered carefully.

We argue that using C2 holistically is important for the entire end-to-end path of ABR streaming that involves the server, CDN, client and the network, especially due to the large amount of resources and bandwidth consumed by ABR streaming. For last-mile networks, such considerations are clearly relevant for resource constrained cellular networks. Even for relatively resource-richer broadband settings, such considerations are relevant since typical uses often involve multiple devices running multiple applications concurrently and sharing network resources over the same broadband connection. It is important to ensure that the resource-intensive multimedia streaming applications do not use more bandwidth than necessary for the current C2.

In this article, we explore using C2 to guide existing ABR rate adaptation schemes toward better QoE and resource tradeoffs. In particular, our emphasis is on appropriately tailoring the video and audio track selections to better match the contextual needs of the specific audio and display devices used for the playback. This is important since not doing so can waste significant network bandwidth, without improving QoE (see examples in Section 2.1). In addition, user expectations may also be conditioned depending on the consumption device capabilities and context. For instance, when playing audio with a surround sound system, the user will clearly prefer the richer 5.1-channel experience over the 2-channel experience. However, when consuming the same content over a phone with a stereo speaker, the 2-channel experience may be perfectly acceptable.

To understand whether an audio/video track is suitable for particular C2, we need to be able to measure the corresponding delivered QoE. This is relatively straightforward to realize for video, given the availability of recent state-of-the-art metrics such as **Video Multimethod Assessment Fusion** (**VMAF**) [50], which allow for evaluating perceptual video quality under different screen sizes and viewing modes (e.g., phone, TV, and 4K screens). For audio, however, somewhat to our surprise, the same task is much more challenging. While various objective audio quality models exist (see Section 7), they suffer from key limitations (e.g., can only measure audio quality for mono and stereo cases). In addition, while there is some recent work [26, 58] that uses video quality metrics to guide video track selection, we are not aware of any work that uses perceptual audio quality metrics for driving audio track selection. In this article, we address the above limitations and make the following main contributions:

— We identify C2 as an important factor for achieving good tradeoffs between QoE and resource usage in ABR streaming. Using existing quality models for video and the methodology that we developed for audio quality evaluation, we quantify and highlight the benefits of being C2-cognizant in ABR streaming (Section 2).
— We examine the current practices of using C2 in five popular ABR players on a wide range of platforms (Android, iOS, and web browser) and identify various limitations (Section 3). Specifically, we consider ExoPlayer [33], `dash.js` [27], Shaka Player [34], AVPlayer [19], and the YouTube app. Our evaluation shows that, although the different players use elements of C2 to certain extent, they only provide limited treatment, lacking a holistic view of C2. We show that these limitations can lead to substantial resource usage and/or degradation in QoE.
— We formulate the problem of C2-cognizant ABR streaming as an optimization problem and develop practical best-practice guidelines to realize it (Section 4). These guidelines leverage information provided through standard APIs by the OS and the streaming server, and can be easily incorporated in ABR players. They enable appropriate tradeoffs between QoE and resources to be achieved automatically, without involving users in the complex decision process. We propose that these best-practice guidelines be used as first-class principles in ABR streaming pipeline, while allowing each player to tailor its own policies and instantiations to its specific use cases.
— To evaluate the design guidelines, we develop a proof-of-concept implementation in ExoPlayer and evaluate it in a wide range of realworld scenarios (Section 5). We show that it achieves significantly better tradeoffs between QoE and resource usage than the standard ExoPlayer. For example, under low network bandwidth settings, it improves video quality by reducing low-quality video segments (e.g., by 17%), while leading to similar audio quality and slightly lower data usage compared to the non-C2 case. When the available network bandwidth is high, it leads to significantly lower resource usage on the end-to-end path (e.g., using only 13% of the bandwidth used by the standard ExoPlayer), while still realizing good QoE for the specific C2. In addition, we demonstrate that our prototype can recognize and react to dynamic C2 changes, and adjust audio/video track selection accordingly.
— We further apply C2-based filtering to `dash.js` player (Section 6), whose rate adaptation strategy differs significantly from that in ExoPlayer. Our results show that C2-based filtering

Table 1. Video and Audio Tracks of Two Demuxed Media

Track	Attributes	Average, peak, and DASH declared bitrate (Kbps)	
		ED (Elephant Dream)	ToS (Tears of Steel)
A1	2, 48 kHz	66, 67, 65	66, 67, 67
A2	2, 48 kHz	131, 171, 128	130, 133, 133
A3	6, 48 kHz	197, 198, 198	197, 198, 198
A4	6, 48 kHz	392, 413, 383	392, 394, 388
V1	144p	102, 138, 117	89, 106, 91
V2	240p	225, 292, 259	198, 226, 202
V3	360p	390, 642, 560	342, 473, 429
V4	480p	844, 1365, 1186	763, 1003, 900
V5	720p	1622, 2716, 2325	1422, 2000, 1778
V6	1080p	2857, 4670, 3838	2314, 3249, 2931

can bring substantial benefit to `dash.js`, e.g., in reducing rebuffering when the network bandwidth is low.

— We highlight the ABR protocol as another important component of C2 that needs to be considered carefully (Section 2 and Section 3.2.4). Specifically, DASH [42] and HLS [20], the two predominant ABR protocols, have subtle differences in their specifications, which have significant implications that need to be explicitly accounted for by any streaming platform that serves both protocols.

The concept of C2 and C2-based filtering apply to both client- and server-based approaches. Our design and prototype (Sections 4 and 5) use a client-based approach, which can be easily adapted to a server-based setting. While our prototype implementation and evaluation focus on the use cases of displaying the video on the phone screen versus an external display, and playing the audio over the phone's built-in stereo speaker versus an external surround sound system, our C2 best practices can be applied directly to other C2 cases such as displaying video using a projector and playing the audio using headphones. In the rest of the article, we focus on the cases where the phone plays a central role in selecting the audio/video tracks from the server, whether in the standalone mode or connected to an external display or speaker (popularly referred to as *mirroring*). The cases where the external devices make their own decisions (e.g., when pressing the "casting" button on YouTube phone app, see Section 3.4) are not of interest to this study.

2 The Need for C2-cognizance

In this section, we highlight the importance of being C2-cognizant in ABR streaming, specifically, the importance of taking audio/video device capabilities and the ABR protocol into account during track selection.

2.1 Audio Device and Audio Quality

A common practice in ABR streaming is that the server provides multiple audio tracks with varying number of channels and encoding bitrate. Two examples are shown in the top half of Table 1 for two demuxed media, **ED** (**Elephant Dream**) and **ToS** (**Tears of Steel**) [73]. For each media, the server provides four audio tracks, A1–A4, all with sampling rate of 48 KHz, encoded using AAC-LC [1]. Among them, A1 and A2 have two audio channels; A3 and A4 have six channels, i.e., corresponding to 5.1 channel surround sound [7].

Do the two higher bitrate audio tracks, A3 and A4, lead to better quality than the two lower bitrate tracks, A1 and A2? As we show below, the answer is not straightforward—it depends on the number of channels that can be played out by the speaker. Specifically, we show two scenarios below: (i) playback over a stereo (i.e., 2-channel) speaker, and (ii) playback over a surround sound system.

Scenario 1 (playback over a stereo speaker). In this scenario, since the speaker only has two channels, while A3 and A4 have six channels, each has to be downmixed into two channels before it can be played back [21]. We follow the **ATSC** (**Advanced Television Systems Committee**) standard [21] for the downmixing, and refer to the downmixed tracks as A3' and A4'. Therefore, the four options of audio tracks that can actually be played back by the stereo speaker are A1, A2, A3', and A4'. To compare the quality of these four options, we explore three objective models for perceptual audio assessment: ITU-T Rec. P.1203.2 [46], ViSQOL [24, 37, 66], and **Perceptual Evaluation of Audio Quality** (**PEAQ**) [43]. While P.1203.2 is a recent standardized model (released in 2017, as part of ITU-T P.1203 series), its audio quality module algorithm is the same as an older standard, ITU-T P.1201.2 [44]. It adopts a no-reference model (also known as single-ended or non-intrusive), i.e., it does not use a reference audio stream when assessing the quality of a test stream. In contrast, both ViSQOL and PEAQ adopt full-reference models. ViSQOL v3 [24] was released in 2020 and improves upon its earlier versions [37, 66] in both design and usage. PEAQ is an ITU-T standard that predates P.1203.2 and ViSQOL. In the following, we only present the quality scores from ViSQOL and PEAQ since their results are sensitive to the audio content, while P.1203.2 does not consider content at all.

To use ViSQOL and PEAQ, we need a high-quality stereo audio track as the reference, referred to as A0. We obtain A0 from the content source website [73] as follows. The website provides six raw audio tracks, corresponding to front left and right, center, rear left and right, and low-frequency effect channels. To create A0, we first join the six raw audio tracks into a single file and then downmix it to A0 following the downmix standard [21]. We divide each such encoded audio track into multiple segments, each 5.33 seconds long to be compatible with video segmentation, and obtain a time series of scores for each tested audio using A0 as reference.

Figure 1(a) plots the ViSQOL-based **Mean Opinion Score** (**MOS**) for ED, which is on a 1 (bad) to 5 (excellent) scale. For the ViSQOL software we use, the maximum score observed is around 4.75 [12]. Figure 1(b) plots the PEAQ **Objective Difference Grade** (**ODG**) scores obtained using the PEAQ software from [10] for ED, which is on a scale from -4 (very annoying impairment) to 0 (imperceptible impairment). Figure 1(c)–(d) shows the corresponding results for ToS. We see that with ViSQOL, the scores of A4' and A2 are very close to each other, and the scores of A3' and A1 are very close to each other. Using PEAQ, A4' has the highest score, followed by A2, A3', and A1. For a given audio track, the PEAQ MOS mappings tend to be lower than the ViSQOL scores, consistent with the observations in [66]. For instance, for ED, ViSQOL rates A2 and A3' as good quality (i.e., around 4), while PEAQ rates A2 mostly as -1 (impairment not annoying), and rates most of the segments in A3' as -2 (impairment slightly annoying).

The above observations suggest that *choosing 6-channel audio tracks (A3 and A4) for a stereo speaker is problematic*: downloading A4 requires 3× the network bandwidth as A2, while the downmixed version A4' has comparable or only slightly better quality than A2; A3 requires 1.5× the bandwidth as A2, while A3' has lower quality than A2. The higher bandwidth requirement of A3 and A4 can significantly limit the bandwidth left over for streaming video. This can lead to lower video tracks being streamed and hence lower video QoE. Conversely, restricting the choice of audio tracks to those that match the speaker capability (i.e., A1-A2) can lead to better video quality, while not degrading audio quality.

We illustrate the above points using an example in Figure 2. It is obtained by running ExoPlayer for ED (video and audio tracks listed in Table 1) on a Pixel 4a phone using a cellular network bandwidth trace (average bandwidth as 1.5 Mbps). This network trace was collected by us; see

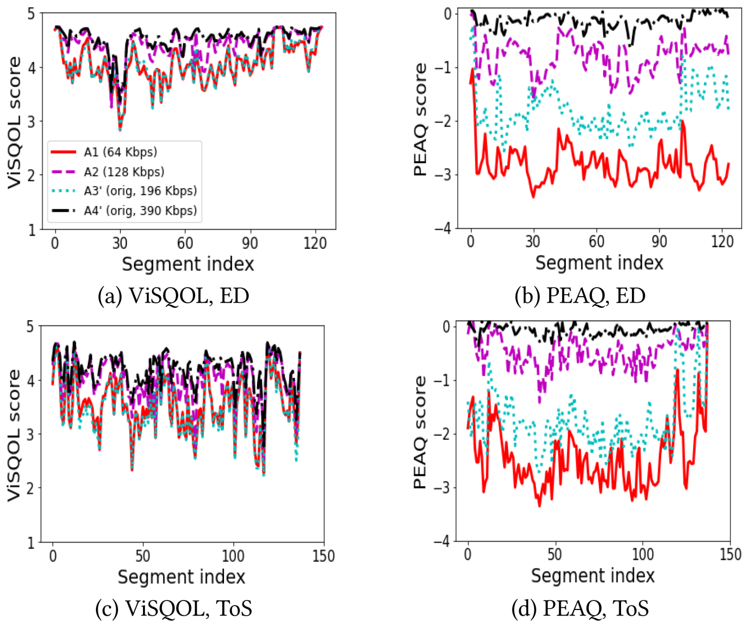


Fig. 1. Quality of the four audio tracks in ED and ToS assessed using ViSQOL and PEAQ.

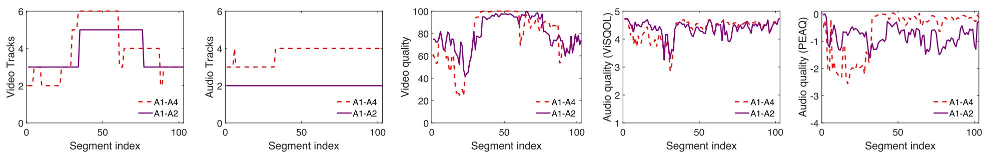


Fig. 2. An example illustrating the benefits of tailoring audio track selection to audio device capability (from ExoPlayer with the DASH protocol using a cellular network trace with average bandwidth of 1.5 Mbps). The y-axes in the first two plots are for video and audio tracks. The y-axes in the last three plots are for video and audio qualities.

more details in Section 5.2.1. Both the video and audio are directly played on the phone. Figure 2, from left to right, plots the video and audio track selection, video quality measured using VMAF, and audio quality measured using ViSQOL and PEAQ. The results are for two cases: one only allows 2-channel audio tracks (i.e., suitable for stereo speakers, marked as A1-A2) and the other has no restriction (i.e., all 4 audio tracks are allowed, marked as A1-A4). We see that the A1-A2 case leads to significantly better video quality than the A1-A4 case: 91% of the video segments have VMAF values above 60 and no segment has VMAF below 40 (VMAF value is in [0,100], the higher the better; below 40 is considered as poor quality, above 60 is considered as fair or better), while in the A1-A4 case, only 80% of the segments have VMAF above 60 and 7% of the segments have VMAF below 40. The improved video quality in the A1-A2 case does not come at the cost of lower audio quality; instead, it leads to similar or even better audio quality than the A1-A4 case. Specifically, Figure 2 shows that, for the earlier audio segments, the A1-A2 case have higher quality than the A1-A4 case. This is because the A1-A4 case selects A3, which has to be downmixed to the 2-channel A3' for playback on the stereo speaker, but A3' has lower scores than A2 that is selected by the A1-A2 case. For the later segments, Figure 2 shows that both cases have good quality (mostly above 4 for ViSQOL and above -1 for PEAQ).

Scenario 2 (playback over a surround sound system). Intuitively, since A3 and A4 can take advantage of surround sound, they can lead to better quality than the two stereo tracks, A1 and A2. However, we are not aware of any objective audio quality assessment tools that work for audio tracks with more than two channels. Hence we cannot provide any numeric comparisons here. Developing good quality evaluation tools for multi-channel such as surround sound is a promising research area.

2.2 Display Device and Video Quality

Matching the video track selection to the display context being used for consuming the video is important. However, there is a lot of focus on maximizing user experience, which in industry translates to delivering very high quality content in many services, even for small-screens such as phones. As an example, a new version of the YouTube player allows users to stream 4K videos on Android devices (even on small-screen phones) [14–16]. In a test, we were able to stream a 4K track over a LTE network to a Samsung phone whose screen resolution is only 1440p (significantly less than 4K). The data usage is very substantial (several tens of Mbps), while the associated benefit accruals in terms of better QoE is unclear for small screen context, due to human perception limitations. For example, studies have shown very little gain for delivering quality beyond 720p on small screens [58]. In addition, streaming and playing the high-bandwidth 4K resolution can lead to high phone energy consumption [76] and undesirable stalls. As we shall see in Section 3, ExoPlayer and Shaka player exhibit similar problematic behaviors as above.

Certain commercial streaming services provide options that account for partial C2 for video. As examples, the YouTube phone app provides an option called "Play HD on Wi-Fi only," which limits the video resolution when streaming over cellular networks; the Amazon Prime Video app provides multiple options ("Good", "Better", "Best") to tradeoff the video quality and data usage. In addition, some commercial services use device-specific ABR manifest files [20, 42], which can limit the highest resolution track allowed on small screens. These practices, while moving in the right direction, do not consider the entire C2. They are mainly motivated to conserve data, and do not consider other important C2 factors such as display and speaker capabilities. Our best-practice guidelines in Section 4 significantly extend the above practices in holistic ways.

2.3 ABR Protocol

DASH and HLS, the two predominant protocols, differ in important ways in terms of the actual information communicated between the server and client. As an example, for demuxed video and audio, DASH only specifies individual audio/video tracks and their bitrates, and does not specify desired audio and video track combinations. In contrast, with HLS, a top-level master playlist specifies both the allowed audio and video track combinations and the average and peak bitrate of each combination, but does *not* specify the bitrate of an individual audio or video track. The player logic needs to carefully account for these differences. As we shall see (Section 3.2.4), the current treatment of this aspect in a popular player is not adequate.

3 C2 Practices in Current Players

In this section, we investigate the current practices around C2 in five popular players and identify their limitations. We first describe our methodology, and then the behaviors of these players.

3.1 Players and Methodology

We study five players: ExoPlayer (v2.16.1) [33], `dash.js` (v3.1.1) [27], Shaka Player (v2.5.20) [34], AVPlayer (running on iOS 15.0.1) [19], and the YouTube app for both Android and iOS (versions 18.12.34 and 18.09.4, respectively). ExoPlayer is an application-level media player for the Android platform, and has been used by more than 140,000 apps in Google Play Store [32]. dash.js is a JavaScript based player and the reference player maintained by the DASH Industry Forum [2]. Shaka Player is a JavaScript library and has been used by more than 1,600 websites [65]. AVPlayer is based on AVFoundation [18], a full-feature framework currently recommended by Apple for audiovisual media on Apple devices. YouTube player is one of the most widely used players, accessed by millions of users every day [63].

Our main goal for studying these players is to understand (i) how and to what extent C2-related information (such as audio/video device capabilities and network type) is obtained or configured in each player, and (ii) whether/how such information is used to filter out unsuitable audio/video tracks. ExoPlayer, Shaka Player, and dash.js are open-source players. For them, we first use source code analysis to understand their behaviors and then use controlled lab experiments to verify our understanding. For AVPlayer and YouTube app, since no source code is available, we rely primarily on controlled blackbox experiments.

3.2 Open-Source Players

ExoPlayer and Shaka Player support both DASH and HLS, while dash.js only supports DASH. For clarity, we first describe the behavior of these three players with DASH (Sections 3.2.1 to 3.2.3), followed by ExoPlayer and Shaka Player with HLS (Sections 3.2.4 and 3.2.5). We use the built-in ABR logic of each player without any change. Briefly, ExoPlayer considers audio and video tracks together, and the rate adaptation is primarily rate based, with consideration of other factors (e.g., buffer level and past track selection). The default ABR logic in dash.js is DYNAMIC [67], and dash.js uses this ABR logic for audio and video rate adaptation separately. Shaka uses a simple rate-based adaptation scheme that selects the video and audio combination whose bandwidth requirement is closest to the current estimated network bandwidth. See more details on the ABR logic of each player in [56].

The description below focuses on each player's practice of audio and video track filtering; their behaviors of identifying and using network information also have limitations, which are omitted in the interest of space and can be found in [56].

3.2.1 ExoPlayer with DASH Protocol. For audio, ExoPlayer specifies that the maximum number of channels that can be played by an audio device as a fixed value 8, independent of the *actual* device capability. Due to the above over-specification, even for a stereo speaker, all audio tracks that have up to eight channels will be determined as playable. After that, ExoPlayer has a filtering mechanism based on the *primary audio track*. Specifically, ExoPlayer determines the primary audio track to be the one with the highest number of channels, the highest sampling rate, and the highest bitrate. After the primary audio track is determined, only the audio tracks that have the same number of channels and the same sampling rate as the primary audio track are retained for the subsequent ABR logic. As an example, for the media in Table 1, A4 is chosen as the primary audio track. Then only A3-A4 are retained; A1 and A2 are excluded since they both have only two channels. *The above practice does not consider the speaker capability at all*, which can have significant adverse impact on QoE (see later).

For video track filtering, ExoPlayer associates a variable, maxVideoSizeInViewPort, with each video track, whose value is a function of the resolution and width-to-height ratio of the display, as well as those of the video track. For example, consider a Pixel 4a phone with display resolution 2340×1080, and three video tracks of 360p (640×360), 1080p (1920×1080), and 2160p (3840×2160). The width-to-height ratio of the display is 29:6, while the width-to-height ratio of all the three tracks is 16:9, lower than that of the display. Then for each of three tracks, its maxVideoSizeInViewPort will be set to 1920×1080 for it to fit the display resolution (see details in ExoPlayer code [5]). After that, the video track filtering works as follows. The player checks whether both the height and width of a video track are larger than the corresponding values in `maxVideoSizeInViewPort` times a fraction (default as 0.98), that is, whether this video has to be downsampled [8] to fit the display resolution. If none of the video tracks satisfies the above condition, then no video track will be filtered. Otherwise, i.e., if one or more video tracks satisfy the above condition, their respective numbers of pixels are noted, and let n be the minimum of these values. After that, any video track with the number of pixels exceeding n will be filtered out. In the example above for the Pixel 4a phone and three video tracks, both the 1080p and 2160p tracks satisfy the condition, and hence their number of pixels will be noted, leading to $n = 1920 \times 1080$. Hence the 2160p (4K) track, which has more than n pixels, will be filtered out, and the 1080p track along with the 360p track will be retained. *While the above practice considers display capabilities to some extent, it is not sufficient* as we shall show later on.

As to display changes, ExoPlayer registers a listener to receive events regarding display changes. However, it only handles the case when the display is changed from an external display to the built-in display, and does not handle the other direction (i.e., changing the display from phone screen to an external display).

Summarizing the above, ExoPlayer's current treatment of C2 has various limitations in terms of optimizing QoE and resource usage, including:

— The audio track filtering mechanism does not consider the speaker capabilities at all and can exclude audio tracks with lower numbers of channels and lower bitrates. The example in Figure 2 shows that such a practice can lead to lower video quality, with no improvement in audio quality, compared to the practice of considering speaker capabilities.
— While the video track filtering mechanism considers display capabilities, it can still allow video tracks with overly high resolutions to be selected. One example is as follows. Consider a Samsung Galaxy S21 Ultra 5G with resolution 3200×1440, and seven video tracks with resolutions 144p, 240p, 360p, 480p, 720p, 1080p, and 2160p. We find that ExoPlayer sets `maxVideoSizeInViewPort` for each video track as 2560×1440. Consequently, only the number of pixels of the last track is noted and n is set to 3840×2160, and hence none of the video tracks will be filtered out, and the highest 2160p (4K) track may be selected under certain network conditions. However, streaming and playing 4K track on small-screen phones is problematic in both QoE and resource usage, as we have pointed out in Section 2.2.
— The video track filtering only considers the built-in screen's capabilities and does not handle the case where the screen resolution is increased to a larger value. For example, when connecting a Pixel 4a phone to an external 4K display, the 4K track should be allowed to be selected. However, as described earlier, it will not be selected since it has already been filtered out.
— The handling of dynamic display change is inadequate in that it cannot deal with the scenarios of changing the display during a streaming session, e.g., connecting a Pixel 4a phone to a 4K display in the middle of the playback.

3.2.2 `dash.js` *with DASH Protocol.* `dash.js` does not consider the audio device capabilities for filtering out audio tracks. It does not take account of the audio device capabilities in the ABR track selection either. For video, `dash.js` sets the display window using HTML, and provides a mechanism to limit the top video track based on the window size. This mechanism is, however, disabled by default. *As a result of the above behaviors, no audio or video track will be filtered.* The above simplifying treatment might be because `dash.js` is meant to be a prototype, instead of a full-fledged product. On the other hand, since `dash.js` is a reference player, identifying what can be improved is important for informing the practice.

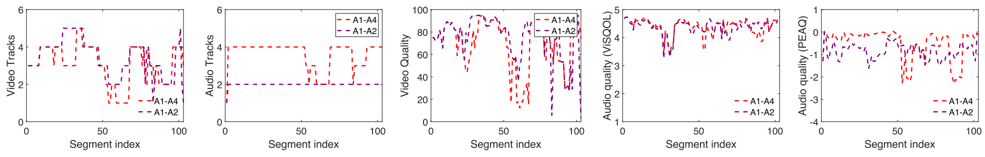


Fig. 3. An example showing the detrimental impact of not considering speaker capabilities in `dash.js` for ED using a cellular network trace. In addition to worse video quality, it also leads to significantly more rebuffering (5.5 seconds vs no rebuffering, not shown in the figure).

Since `dash.js` does not filter out any audio/video tracks based on C2, it has various drawbacks, including potentially using audio/video tracks that exceed the speaker/display capabilities, leading to low QoE and high data usage. We next show an example. It is obtained by running `dash.js` for ED under the same cellular network trace for Figure 2 (average bandwidth 1.5 Mbps) on a Pixel 4a phone. In the following, the A1-A4 case is for the current `dash.js` player, which allows all four audio tracks A1-A4 to be selected, while the A1-A2 case emulates the scenarios where only A1-A2 are allowed based on speaker capabilities. We observe that the A1-A4 case has 5.5 seconds of stalls, while the A1-A2 case has no stall at all. Figure 3 further shows the video and audio track selection and their respective qualities. We see that these two cases have similar audio quality, while the A1-A4 case has significantly lower video quality: 69% of the video segments have VMAF values above 60 and 14% of the segments have VMAF below 40, while the corresponding values for A1-A2 case are 79% and 8%, respectively.

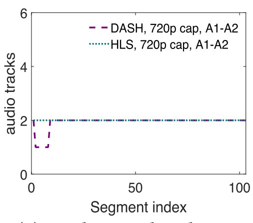
3.2.3 Shaka Player with DASH Protocol. Given multiple audio tracks, Shaka has an explicit logic to *prefer* those with two channels. Specifically, the audio tracks are placed into groups based on the number of channels. If there are tracks with two channels, these tracks are retained and the rest of the tracks are removed from subsequent ABR track selection. If none of the audio tracks has two channels, then the group of audio tracks with the lowest number of channels is retained and the rest of the audio tracks are removed. For the two media in Table 1, only A1 and A2 are retained, *independent of the audio device that is being used.*

For video, Shaka has a mechanism that filters video tracks based on the minimum/maximum allowed video width, height, number of pixels, frame rate, or bitrate. In the default setting, however, the minimum value is 0 and the maximum value is infinity for all these attributes, and hence *no video track will be filtered.*

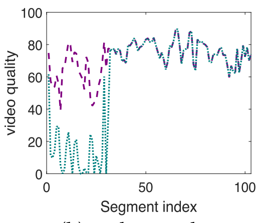
In summary, Shaka's conservative preference for low-channel audio tracks, independent of the audio device capabilities, is problematic: even if the audio is played over a surround sound system, multi-channel audio tracks with six or more channels will not be selected, which can lead to lower user experience than possible with such speaker systems. In contrast, for video, since no video tracks are filtered out, Shaka can lead to inappropriate video track selection. Specifically, we tested a setting where the playback uses the built-in screen of a Pixel 4a phone, the connection is over a cellular network, and the highest video track is 4K. We see that the 4K track can be selected, which, as we pointed out in Section 2.2, is problematic in both QoE and resource usage.

3.2.4 ExoPlayer with HLS Protocol. We highlight several differences between how HLS and DASH are handled by ExoPlayer.

— **Audio and video media attributes.** Since the bitrate of an individual audio/video track is not specified in the top-level HLS manifest file (see Section 2.3), ExoPlayer sets the bitrate of each audio track as -1 (i.e., undefined). It sets the bitrate of a video track to be the value specified in the first combination that contains this track.



(a) Audio track selection.



(b) Video quality.

Fig. 4. Results of ExoPlayer with HLS and DASH under one cellular network trace (with proper C2-based filtering).

- **Audio track filtering.** ExoPlayer uses the same logic to filter audio tracks for HLS as it uses for DASH (see Section 3.2.1). However, since all the audio tracks have the same bitrate for HLS (i.e., -1), only the first two rules (i.e., related to the number of channels and sampling rate) will be effective in determining the primary audio track. For example, for the two media in Table 1, when the HLS manifest file has four audio tracks, in the order of A1, A2, A3, and A4, ExoPlayer will determine A3 (instead of A4 in DASH) as the primary audio track. This is because A3 has the highest number of channels, sampling rate and bitrate, same as A4, and is listed earlier in the manifest file (in reality, A3 has a lower encoding bitrate than A4, but ExoPlayer regards the bitrate of both as the same -1).
- **ABR logic.** ExoPlayer uses the same logic for DASH and HLS. Specifically, ExoPlayer predetermines a set of audio and video combinations, and *ignores* the set of combinations that is *actually* specified in the HLS manifest file. In addition, since audio tracks have unspecified bitrate in HLS, ExoPlayer will retain a single audio track, i.e., the primary audio track, leading to a *fixed* single audio track selection. For ED (see Table 1), since A3 is determined to be the primary audio track and is the only audio track retained for ABR logic, the predetermined set of combinations for HLS only contains A3. See more details on the track combination chosen by DASH and HLS in ExoPlayer in [56].

Due to the above issues in ABR, even if there were proper C2-based filtering in place, the performance of ExoPlayer with HLS can still be undesirable. In fact, the QoE can be significantly lower than that with DASH, even under exactly the same network bandwidth profile and for the same set of audio/video tracks. Figure 4 shows an example obtained when the playback is on a Pixel 4a phone under a cellular network trace with the average bandwidth of 1.0 Mbps (see details on the network traces in Section 5.2.1). We assume proper C2-based filtering. Specifically, the video track is capped to 720p considering the small phone screen, and the audio tracks are A1-A2 considering the stereo speaker. The audio tracks are listed in the manifest in the order A2, A1, and hence A2 is determined to be the primary audio track. Figure 4(a) shows that a fixed audio track (i.e., A2) is selected for HLS, in contrast to the adaptive choice for DASH. As a result, the HLS case has significantly worse video quality than the DASH case as shown in Figure 4(b): it has 28% low-quality segments (VMAF below 40), versus 1% in the DASH case.

The above observations highlight the importance of understanding the subtle differences between DASH and HLS, and their implications for ABR streaming. A streaming service that supports both DASH and HLS protocols may need customized treatment for these two protocols. A service that builds on top of ExoPlayer may choose to support a single protocol for lower cost, and may very well choose to support HLS instead of DASH, since HLS is supported by both Android and Apple platforms, while DASH is only supported by Android. In such cases, it is even more important to address the limitations in how ExoPlayer handles HLS protocol.

3.2.5 Shaka with HLS Protocol. Unlike ExoPlayer with HLS, Shaka considers the set of audio and video track combinations specified in the HLS manifest file. As with DASH, it prefers audio tracks with two channels, independent of the audio device capabilities. Again, such choices can lead to audio quality that is below the current device capabilities, and undesirable user experience.

3.3 AVPlayer

AVPlayer only supports HLS. Since it is a proprietary player, we use controlled blackbox experiments to explore its behaviors. All the experiments below are carried out in a high-bandwidth (consistently above 300 Mbps) WiFi network, using the default setting in AVPlayer. We use an iPhone 11 pro (iOS 15.0.1) that has a built-in stereo speaker and resolution 2436×1125. The peripheral devices include two different smart TVs that each has two speakers and 4K display, and a 5.1 channel surround sound system.

Audio track filtering. We investigate which audio track AVPlayer prefers when given three audio tracks that have one, two, and six channels, respectively. All the three tracks are encoded in Dolby Digital Plus (E-AC-3) [3], recommended surround sound codec by Dolby [6]. In the standalone mode where the phone is not connected to any external device, AVPlayer selects the 6-channel audio track, which needs to be downmixed for the stereo speaker on the phone and is problematic (for reasons described in Section 2.1). When connecting the phone to a smart TV via HDMI, we experiment with two recent but different Samsung TVs. Both TVs have stereo speakers, and hence for each TV, we further explore two scenarios: connecting or not connecting a 5.1 channel surround sound system to the TV. We find that for one TV, the 6-channel audio track is selected in both modes, while for the other TV, the 2-channel audio track is selected in both modes. Neither behavior is undesirable. The preferred C2-cognizant behavior would be to select a 2-channel audio when the audio playback is over the built-in speakers, and select a 6-channel track when the playback is over an attached surround sound speaker system.

Video track filtering. We explore the track selection by AVPlayer using a manifest file with seven tracks, the two highest tracks being 1080p and 4K. In both the stand-alone and HDMI cases, the maximum selected track is 1080p, even though 4K track is a more appropriate choice for the latter based on C2. We find that AVPlayer has a configurable parameter `preferredMaxResolution` whose default setting appears to be limiting this highest track selection to 1080p, agnostic of the specific C2. If this parameter is configured appropriately, the player can indeed select a higher resolution (e.g., 4K) track. Therefore, C2-cognizant ABR streaming would need to determine the C2 over time and dynamically adapt this parameter setting appropriately.

3.4 YouTube Player

We explore the C2 behavior of the YouTube app on both the Android and iOS platforms using controlled blackbox experiments. As mentioned in Section 1, we focus on various scenarios where the decisions are made by the app on the phone.

Our experiments below use a Samsung Galaxy S21 Ultra 5G phone (running Android 12) and an iPhone 11 pro (iOS 15.0.1). The peripheral devices include (i) a smart TV that has a stereo speaker and 4K resolution, (ii) a 5.1 channel surround sound system, and (iii) an external monitor with resolution 1080p. All the experiments were conducted in the same WiFi network with high bandwidth as in Section 3.3. The audio and video selections by the YouTube app are obtained using YouTube's Stats for Nerds [9]. To verify the characteristics of the selected video/audio tracks (e.g., resolution of a video track and number of channels of an audio track), we use `youtube-dl` [75] to download the tracks and use FFmpeg [30] to analyze the encoding of the tracks.

YouTube Android Player. For audio filtering, we stream a movie that has both two and six channel audio tracks. The encoding is E-AC-3 [3]. We experiment in four scenarios: phone standalone mode, phone connected to the smart TV using HDMI, the smart TV is further connected to the 5.1 surround sound system using HDMI, and phone connected the 5.1 surround sound system using HDMI. In all these four cases, the app only selects the 2-channel audio track. While the selection is appropriate for the first two scenarios (the phone has a stereo speaker), it is not ideal for the last two scenarios, where the six channel audio track will better match the capability of the surround sound system and user's expectation.

For video filtering, we stream a video with seven tracks, the two highest tracks being 1440p and 4K. We configure the resolution of the Android phone display to three settings: 720p, 1080p, and 1440p (i.e., the maximum resolution that can be set since the phone display resolution is 3200×1440). For each setting, we experiment in three scenarios: phone standalone mode, phone connected to the smart TV using HDMI, and phone connected to the monitor using HDMI. In all the three scenarios, the video track selection is up to the resolution set on the phone, i.e., when the phone resolution is set to 720p, the maximum resolution of the video tracks that are selected is 720p, and so on. In other words, for the two cases where the phone is connected to external devices, the video track selection is agnostic to the external display resolution, while ideally the selection should consider the external device capabilities, e.g., allowing 4K track to be selected when playing the video on the smart TV.

YouTube iOS Player. For audio, we observe the same results on the iOS app as those on the Android app. Namely, even when a 6-channel audio track is available on the YouTube server and the network bandwidth is sufficiently high, the 6-channel audio track is never selected even when the phone is connected to a 5.1 channel surround sound system.

For video, the iOS phone does not allow changing the phone resolution to other values, and hence we only experiment with the actual phone resolution (2436×1125). We test four cases: three as those for the Android phone (see above), while in the 4th case we connect the phone to the smart TV using the screen-mirroring capability that is provided by iOS. In all the four cases, the app only chooses 1080p video tracks, although it is more desirable to allow 4K tracks when the phone is connected to the 4K smart TV.

3.5 Summary of Main Findings

We see all the five players have limitations in using C2. The limitations vary, including over-specification (e.g., ExoPlayer's setting on the maximum allowed number of audio channels), lack of restriction (e.g., `dash.js` allows all audio and video tracks to be played under all possible C2), conservative preference (e.g., Shaka prefers stereo audio tracks irrespective of the audio device capabilities), and not adapting to external peripheral capabilities (e.g., AVPlayer and YouTube player do not select 4K video when connected to a 4K display). As we have shown, such limitations can significantly affect their effectiveness, leading to performance and resource usage far from optimal.

4 Best-practice Guidelines

The limitations of the existing practice in using C2 cannot be simply resolved by changing some default parameters alone. Rather, it requires the right flow of C2 information, and C2-cognizant policies and actions based on that information. In this section, we first formulate the problem of C2-cognizant ABR streaming as an optimization problem, and then present our design and best-practice guidelines to realize it. While our findings on the limitations of the various players are specific to the players, our formulation, design and best-practice guidelines are general, applicable broadly to ABR streaming.

4.1 C2-Cognizant ABR Streaming

The goal of C2-cognizant ABR streaming is to use C2 to guide rate adaptation to optimize QoE and resource usage. Specifically, consider K video and audio segments. Let v_k and a_k represent the selected video and audio tracks for segment k, respectively. Let $C2_k$ represent the C2 at the decision time for segment k. Let QoE_1^K and R_1^K represent the QoE and resource usage for segment 1 to K. Then the optimization problem is

$$\begin{aligned} \text{maximize}_{v_k, a_k, k=1,\ldots,K}: \quad & QoE_1^K - \gamma R_1^K \\ \text{s.t.} \quad & QoE_1^K = \sum_{k=1}^{K} Q(v_k, a_k \mid C2_k) - \lambda \sum_{k=1}^{K} B(v_k, a_k \mid C2_k) - \mu \sum_{k=1}^{K-1} V_k \\ & R_1^K = \sum_{k=1}^{K} S(v_k) + S(a_k) \end{aligned}$$

where $Q(\cdot)$ represents the quality given the selected video and audio segments under the current C2, while $B(\cdot)$ represents the amount of rebuffering, $V_k = |Q(v_{k+1}, a_{k+1} \mid C2_{k+1}) - Q(v_k, a_k \mid C2_k)|$ represents the quality variation (in absolute value) between the $(k+1)$th and kth segment, and $S(\cdot)$ is the size of a video/audio segment in bits. Last, γ, λ, and μ are non-negative parameters that specify the weights for resource usage, quality variation, and rebuffering, respectively.

Given the above formulation, one way to optimize the QoE and resource usage tradeoffs cognizant of C2 is *C2-based filtering*, i.e., dynamically determining a subset of video and audio tracks that are appropriate for the current C2 and limiting track selection to this subset. Let $\mathcal{V}_k$ and $\mathcal{A}_k$ denote the full set of video and audio tracks for segment k, respectively. Given $C2_k$, i.e., the C2 when the player needs to select segments k, let $\mathcal{V}_k' \subseteq \mathcal{V}_k$ denote the appropriate subset of video tracks based on $C2_k$. Similarly, define $\mathcal{A}_k' \subseteq \mathcal{A}_k$ for audio. Specifically, $\mathcal{V}_k'$ and $\mathcal{A}_k'$ exclude some tracks with overly high bitrate relative to the current C2 (e.g., excluding 6-channel audio tracks, and 1080p and 4K video tracks when playing on a phone), while still keeping the lower bitrate tracks to accommodate bandwidth variations. As a result, choosing from $\mathcal{V}_k'$ and $\mathcal{A}_k'$ steers the ABR logic to C2 appropriate choices, which can lead to lower resource usage, potentially less rebuffering, while not degrading the quality of video and audio, and hence leading to improved QoE and resource usage tradeoffs.

4.2 High-Level Design Choices

C2-based filtering requires meshing various information from the server and client, including specifying properly the media attributes at the server, parsing the media attributes properly at the client, and obtaining the *current* C2 of the client. Such C2 information needs to be obtained during the *entire* streaming session since the display/speaker setting and the type of network connection may change over time. We next outline several design choices; our proposed design is deferred to Section 4.3.

Server-based or client-based. In a server-based design, the client sends its C2 information to the server, which, in turn, selects a subset of the audio/video tracks, creates a *C2-specific* manifest file with this subset of tracks, and sends it to the client so as to restrict the track selection to those suitable to the C2. This approach is a significant step forward over the use of device-specific manifest files in some services, which, although customize the manifest files separately for small-screen and large-screen devices (e.g., allowing high bitrate tracks only for large-screen devices), still does not address the scenario where a small-screen device might in reality be used with a different C2, e.g., when it displays the video on an external large-screen peripheral.

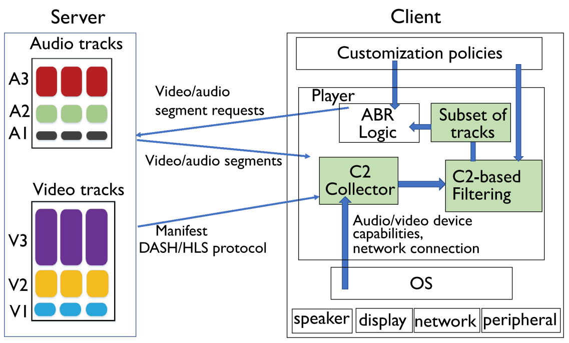


Fig. 5. High-level design of client-based C2-cognizant ABR streaming.

Importantly, the C2 is not static, it can change during the playback, and therefore needs to be tracked and accounted for automatically in a dynamic manner during playback. A challenge for the above server-based approach is that, if the C2 at the client changes in the middle of a streaming session, the changed information needs to be sent to the server so that the server can send an updated manifest file to the client. The above involves communication of device capabilities and C2 information to the server, and appropriate updates to the ABR manifest file and communication of the same to the client in the middle of the session. This communication can be realized in a variety of ways in the context of today's ABR workflows, e.g., using an HTTP-based query-response interface between the client and server. A longer term solution would involve building in such communication capability into the ABR protocol itself.

Compared to the server-based design, a client-based approach is easier to deploy: the client is cognizant of its current C2, and can filter out audio/video tracks accordingly. As a result, a subset of audio/video tracks is fed to the ABR logic, restricting it from selecting tracks that are not suitable for the current C2. On the other hand, as explained earlier, if desired, one can also adopt a server-based approach, at the cost of more communication between the server and client.

Inside or outside the player. Another design issue is whether C2-based filtering is best accounted for inside the player or outside the player (e.g., as additional customization policies determined by the users). We argue for the former design because: (i) making users account for C2 appropriately is not practical since it requires them to have detailed knowledge about the different dimensions of the C2 to be able to select the appropriate options in a meaningful way. When incorporating C2 inside the player, it becomes easier for a streaming service to create customized C2-cognizant experiences for end users based on its specific business needs. (ii) C2 needs to be accounted for *automatically* in a *dynamic* manner during the playback, and hence is ideally incorporated at the player level, based on dynamic C2 collected in real time by the player, requiring no manual input from the users.

4.3 Proposed Design

Summarizing the above, we develop a client-based design that resides inside the player software, as shown in Figure 5. Two new components are added in the player, highlighted in green: one is *C2 Collector*, which collects C2 information from the lower layers (the underlying OS and other sources) and the server, and the other is *C2-based Filtering*, which filters out audio/video tracks based on the current C2. The output of C2-based Filtering leads to a subset of audio and video tracks (also highlighted in green in Figure 5), which is fed to the ABR logic for track selection. Both components (C2 Collector and C2-based Filtering) run continuously during the playback, determining dynamically the subset of audio and video tracks for C2-cognization rate adaptation.

Existing customization polices such as “Play HD on WiFi only” can be provided as done today as UI-based user configurable options outside the core ABR player system. The C2-based Filtering module considers this policy information together with the C2 information received from the C2 Collector for making track filtering decisions.

Our design essentially extends the OS by adding a layer of automatic and realtime C2 information collection and filtering inside the player. This added layer is outside the ABR logic and can be easily incorporated in existing players, without any changes to the ABR logic. By filtering out audio/video tracks inappropriate for the current C2, and feeding only the C2-appropriate tracks to the player ABR logic, this added layer makes it easier for the ABR logic to then select the appropriate tracks to achieve a better tradeoff between QoE and resource usage. This approach also makes it easier and more practical to realize the benefits of C2-appropriate choices for ordinary users, who do not have deep technical understanding of the various resource and performance tradeoffs. In summary, using our approach, the player uses the choices appropriate for the current C2, not requiring users to know the exact C2 all the time, while still allowing users to provide their input to customize the C2-based filtering.

Under the above design, we provide the following best-practice suggestions for the design of C2 Collector and C2-based Filtering, and for the player to use C2 information.

4.3.1 C2 Collector. This module runs throughout the playback. When a player starts, it is desirable to obtain the C2 details such as (i) the specific setup of the audio and video devices, e.g., playing directly using the built-in speaker or native display of the phone or external peripherals, (ii) the capabilities of the audio/video device used for playback, e.g., stereo speaker, or 5.1 channel surround sound system, screen size and resolution of the display, and (iii) the network connection, i.e., cellular or WiFi. Such information can be obtained through the APIs provided by the underlying system and other auxiliary information if needed; see more discussion in Section 4.4.

During playback, it is necessary for C2 Collector to dynamically determine the current C2, and update the the C2-based Filtering module accordingly. This can be achieved by registering an intent to receive information about C2 changes from the underlying system as they occur.

4.3.2 C2-Based Filtering. This module also runs throughout the playback. At each point of time, it should take the C2 information from the C2 Collector, and use C2 holistically. The decision on what to be filtered should not be limited only by the available network bandwidth and what can be decoded and played, but also should take account of the capabilities of the output devices and the type of network connection. Both audio and video track selection needs to be moderated by taking C2 into account based on what really brings value to the users in the specific C2. For example, for a 1440p display, there is no need to bring in a 4K video track even if a phone can play it, since the player will need to downsample the resolution to the lower screen resolution (1440p) before displaying the video. In addition, it may not even need to bring in a 1440p track for a small phone screen (see Section 2.2). Similarly, given a 2-channel speaker, there is no utility in streaming in a high-bitrate 6-channel audio track, since it needs to be downmixed to a 2-channel version in any case (see Section 2.1). A non-holistic piecemeal treatment of C2 can lead to undesirable outcomes; see one example regarding ExoPlayer in [56].

4.3.3 C2-Cognizant Player. The client player should only consider the subset of audio and video tracks determined by the C2-based Filtering module in its rate adaptation. In addition, it should explicitly consider and tailor its C2-aware adaptation decisions based on the specific ABR protocol being used and the information it provides. Which ABR protocol is used between the server and

Table 2. APIs for Obtaining C2 Information on Android and Chrome Browser

	Android	Chrome
Audio device	AudioManager	navigator.mediaDevices
Display device	DisplayManager, WindowManager	devicePixelRatio, HTMLElement
Network type	ConnectivityManager	navigator.connection

client is an important component of C2, since it leads to different ways of representing information included in the manifest file (e.g., bitrate representation for audio and video tracks), and somewhat different types of information in the manifest file, which can have significant impact on the C2-aware adaptation and resultant QoE. In particular, as we showed in Section 3.2.4, the DASH and HLS specifications (the two predominant ABR protocols) have subtle but significant implications that need to be explicitly accounted for by any client player that serves both protocols. This is true even in the presence of CMAF packaging [11], which allows a single packaging to be used for DASH and HLS, obviating the need for multiple packaged copies of the same underlying content. In that case, the differences between DASH and HLS specifications still matter (e.g., different types of information exchanged in the manifest file) and require careful treatment.

In our approach, the client is cognizant of its current C2, and therefore can take all the required actions (e.g., C2-based filtering and track selection). Certainly, the server, which has more knowledge of the content, can assist the client in its C2 decisions through some additional hints. For example, for a high-motion sports content, it might indicate a subset of tracks suitable for a certain C2, e.g., a small screen versus a large TV. These hints could be included in the manifest file or shared out-of-band. But even with the information in today's manifest file, it is possible for the client side to be C2-cognizant, as demonstrated in our prototype implementation (Section 5).

4.4 Obtaining C2 from the Underlying Platform

Our C2-cognizant ABR streaming design requires knowing C2 information. We next investigate the availability of C2 information to the player through standard APIs on two state-of-the-art platforms: Android as an example popular mobile phone OS, and Chrome as an example popular browser-based platform. Table 2 lists the APIs for accessing C2 for audio/video devices and network type on these two platforms.

We test the Android APIs on two phones, Samsung Galaxy S21 Ultra 5G and Pixel 4a (both running Android 12), and test the Chrome APIs on a Windows laptop (Chrome 91.0.4472.124) and the Samsung phone (Chrome 91.0.4472.120). We find that the required C2 information is available for all the above cases, modulo some device-specific exceptions for the Pixel 4a phone (see Section 5). Since C2 information is readily available to the players, our best-practice guidelines can be easily incorporated into existing players.

5 Proof-of-Concept Implementation and Experimental Results

5.1 Proof-of-Concept Implementation for ExoPlayer

As a proof-of-concept, we modify ExoPlayer following our framework and best-practice guidelines in Section 4. The modifications include:

- **Audio.** For audio, we first remove the logic that filters audio tracks based on the primary audio track (Section 3.2.1). We then add a function in class `DefaultTrackSelector` in ExoPlayer to set the maximum number of channels that can be played as follows. When the player starts up, we use Android API `AudioManager` to get information about the capabilities of the audio device over which the audio will be played. If the built-in speakers are used, we use `AudioManager` to obtain the number of channels supported by the device (i.e., two channels) and then set the maximum number of channels to that value. After that, we use `setMaxAudioChannelCount` function in ExoPlayer to only retain the audio tracks whose number of channels do not exceed that maximum value. If an external audio device is attached to the phone (again detected using `AudioManager`), we similarly obtain the number of channels supported by the external device (e.g., 8eight channels for a surround sound system) and set the maximum number of channels accordingly.

— **Video.** For video, we use Android's `DisplayManager` API to detemrine the capabilities of the display device on which the video will be shown. (i) If the built-in phone screen is used for display, we restrict the video tracks to be no more than resolution k (e.g., 480p or 720p) by using `setMaxVideoSize` function in Exoplayer. This setting is based on the small phone screen and the diminishing gain of video perception quality [58]. Hence larger resolutions only bring marginal benefits to users, while, because of their significantly higher bitrates, they lead to substantially more energy and bandwidth consumption for the phone, the network, and the server [76]. (ii) If an external large display is connected to the phone, we determine the maximum resolution as the resolution of the external display (obtained from `DisplayManager`); the video tracks with resolution up to this maximum value are allowed to be selected. This design is made because when a user makes the effort of connecting the phone to an external display, the external display will most likely be the primary display device. Therefore, we use its resolution to dictate the selection of the maximum resolution.

— **Other choices.** The above choices are the defaults that we decide based on device capabilities. Other choices are possible and our implementation can be easily extended to other defaults. In addition, the default choices can be overwritten by users, e.g., through the customization policies in Figure 5. As an example, a user may specify that, if the connection is over a cellular network, then the phone should not choose 4K tracks even if it is connected to an external 4K large display. This can be easily translated into C2-based filtering, by detecting the network connection type using `ConnectivityManager` and limiting the maximum video track using `setMaxVideoSize`.

— **Dynamic settings.** We use Android's `BroadcastReceiver` to obtain notification on speaker/display device changes and network type changes. Specifically, we use `ACTION_AUDIO_BECOMING_NOISY` intent to catch changes in audio output. After that, we obtain the maximum number of audio channels supported by the current audio device and use it to filter audio tracks as described earlier. For display, we use the `ACTION_HDMI_AUDIO_PLUG` intent to catch when HDMI is plugged or unplugged, and then set the maximum allowed video resolution accordingly as described earlier. To identify changes in network type, we use `NetworkTypeObserver` class, which has a `onNetworkTypeChanged` listener that is called when network type is changed.

We tested the above modifications on two recent phones: Samsung Galaxy S21 Ultra 5G and Pixel 4a (both running Android 12). The testing includes various realworld scenarios, including standalone mode and connecting phones to peripheral devices (see details in Section 5.2). The Samsung phone was able to identify the current C2 correctly in all the settings and our modifications achieved the desired behaviors (see results in Section 5.2). Pixel 4a has several limitations in C2 information flow. First, `AudioManager` did not return the current number of channels for the built-in speaker. To address this issue, we added a mechanism in our implementation that involves using auxiliary device information: we use Android's `getDevices` to obtain the device model (e.g., "Pixel 4a"), and then use a lookup table to determine the number of audio channels for the built-in speaker for that device model (e.g., 2 for "Pixel 4a"). Such mappings can be created based on widely available device specifications. Second, unlike the Samsung phone, external audio/video devices that are connected to Pixel 4a cannot be directly detected through `AudioManager` and `DisplayManager`, which might be because Pixel phones do not support video/HDMI output [13, 28, 70].

5.2 Proof-of-Concept Results for ExoPlayer

We next use our proof-of-concept implementation for ExoPlayer in a wide range of realworld scenarios, including playback on the phone over cellular or home WiFi networks, playback when connected to peripheral devices over home WiFi networks. In addition, we test in dynamic settings, e.g., starting in a stand-alone mode and then connecting an external speaker or display to the phone, and vice versa. We next report the results in static settings; in dynamic settings, our prototype correctly identifies the changes and filters audio/video tracks accordingly. All the results are obtained using Exoplyer with DASH running on a Samsung Galaxy S21 phone. ExoPlayer with HLS needs further improvement in the ABR logic (see Section 3.2.4), which is beyond the scope of this article.

5.2.1 Experiment Setting. All the experiments are conducted between the Samsung phone and a server that we set up. The phone has a stereo speaker and a 6.7-inch built-in screen with the maximum resolution of 3200 × 1440. We consider three types of peripheral devices that can be attached to the phone: a 1080p large-screen (24-inch) display, a 4K large-screen (32-inch) display, and a surround sound system (VIZIO 5.1 channel).

Videos and audios. We consider the two 6-track videos, ED and ToS, both with four audio tracks, in Table 1. In the interest of space, we only report the results for ED; the results for ToS show similar trends. In addition, we consider another 7-track video, **BBB** (**Big Buck Bunny**), which has six video tracks of the same resolutions as those for ED, and an additional 2160p (4K) track.

Network settings. We use trace-driven experiments for apples-to-apples comparison of our prototype with the standard ExoPlayer. Specifically, the network bandwidth between the server and the client is controlled using `tc` [52] at the server to emulate realworld network scenarios. We consider both cellular and WiFi network settings. For cellular, we use 10 traces that we collected from two commercial LTE networks under various scenarios (stationary, walking, driving). The bandwidths of these traces are scaled to four settings, with the average bandwidth as 1, 1.5, 2.5, or 5 Mbps, respectively. For WiFi, we use four traces that we collected from a home network under loaded conditions, with the average bandwidth varying from 15.7 to 19.0 Mbps.

Performance metrics. We use (i) *quality of played back video segments:* measured using a state-of-the-art perceptual quality metric, VMAF [50]; we refer to the segments with VMAF below 40 as *low-quality* and those with VMAF above 80 as *good-quality*, (ii) *quality of playbed back audio segments:* measured using ViSQOL and PEAQ, (iii) *rebuffering duration*: the total duration of rebuffering in a streaming session, (iv) *quality changes*: including the magnitude of quality change and frequency of track level changes for video and audio, (v) *data usage*: the total amount of data downloaded, and (vi) *percentage of wasted data*: measured as the amount of wasted data due to chunk replacement (i.e., the sizes of the segments downloaded, but later replaced and not played back) divided by the total data usage.

5.2.2 Playback on Phone over Cellular Networks. We use ED in this setting. Since the phone has a stereo speaker, our prototype limits the audio tracks to the two 2-channel tracks, A1-A2, while the standard ExoPlayer allows only A3-A4. For video, our prototype caps the video tracks due to the small phone screen. Specifically, we show the results when capped to 480p or 720p; the

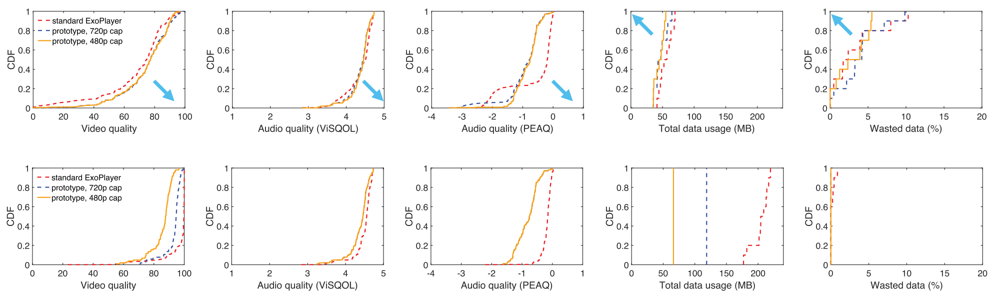


Fig. 6. Comparison of our prototype and the standard ExoPlayer for 6-track ED (playback on phone over cellular network settings). The two rows are the results when the average network bandwidth is 1.0 and 5.0 Mbps, respectively. The arrows point to the directions of better results (arrows in the bottom row omitted).

standard ExoPlayer allows all the six tracks (including 1080p) to be selected. We show the results of three scenarios: from the standard ExoPlayer, from our prototype when capping the video tracks to 480p or 720p. For all the scenarios, the percentage of chunk replacement tends to be lower when the network bandwidth is higher since higher video tracks were selected initially and hence less replacement is needed. For all the network settings, none of the runs has rebuffering.

The top row of Figure 6 plots the results when the average network bandwidth is 1.0 Mbps. **(i)** Our prototype leads to significantly better video quality than the standard ExoPlayer: across the 10 traces, our prototype with 480p and 720p caps both reduce the percentage of low-quality video segments by up to 17%, with the average reduction being 6%. This is because in our prototype, based on the audio device capabilities, only the two lower bitrate audio tracks, A1 and A2, are allowed to be selected, leaving more bandwidth for video selection, while in the standard ExoPlayer, the selected audio tracks are exclusively the higher bitrate A3 or A4 tracks. For our prototype, we see the video quality when capping to 480p and 720p is similar. This is because when the network bandwidth is low, the choice of video tracks is mainly limited by the network bandwidth, not the cap. **(ii)** When using ViSQOL, the audio quality of our prototype is slightly better than the standard ExoPlayer: for our prototype, 92% and 90% of the segments have ViSQOL quality above 4 with 480p and 720p caps, respectively, while the percentage for the standard ExoPlayer is 84%. This is because A2 has higher quality than A3', i.e., the downmixed 2-channel version A3 that is selected by the standard ExoPlayer. When using PEAQ, we see more lower quality audio segments from the standard ExoPlayer, again due to the selection of A3. Specifically, for our prototype with 480p and 720p caps, respectively 1% and 6% of the segments have PEAQ lower than -2, while the percentage for the standard ExoPlayer is 15%. **(iii)** The data usage of our prototype is slightly lower: 86% and 83% of what is used by the standard ExoPlayer for 720p and 480p caps, respectively, since the low network bandwidth is the main constraint.

The bottom row of Figure 6 shows the results when the average network bandwidth is 5 Mbps. We see all the three cases have good audio quality and close to zero low-quality segments. The data usages of our prototype with 720p and 480p caps are 58% and 33% of what is used by the standard ExoPlayer, respectively. For audio, our prototype chooses the C2 appropriate stereo A2 track, while the standard ExoPlayer chooses the 6-channel A4 track. Both tracks have good quality based on ViSQOL and PEAQ, with ViSQOL rating them as similar, while PEAQ rating A2 worse than A4. For video, our prototype with 720p cap has almost the same amount of good-quality segments as the standard ExoPlayer; capping to 480p leads to less good-quality segments, but more data savings. Considering both quality and data usage, capping to 720p appears to achieve the best tradeoffs.

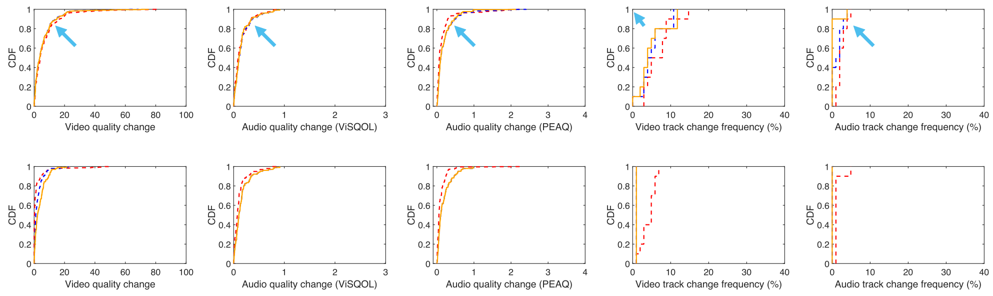


Fig. 7. Quality change results of our prototype (blue and orange curves for limit of 720p and 480p, respectively) and the standard ExoPlayer (red curve) for the two settings in Figure 6.

Quality change. Figure 7 compares the quality change of our prototype and standard ExoPlayer. In the top row (average network bandwidth of 1.0 Mbps), we see that our prototype has slightly lower quality change than the standard ExoPlayer for video. For audio, the quality change in ViSQOL is similar for the three cases, and the standard ExoPlayer has slightly lower quality change in PEAQ. The frequency of video track level changes for our prototype is comparable to that of the standard ExoPlayer, while for audio tracks, our prototype has slightly lower frequency of change. In the bottom row (average network bandwidth of 5.0 Mbps), the standard ExoPlayer has slightly lower quality change than our prototype, while our prototype has lower frequency of track changes for both video and audio.

Other bandwidth settings. Last, when the average network bandwidth is 1.5 or 2.5 Mbps, the performance shows similar trends as above: our prototype leads to less low-quality video segments and lower data usage than the standard ExoPlayer. The figures are omitted in the interest of space.

5.2.3 Playback on Phone over Home WiFi. For this setting, we first use ED and then BBB, both with the four WiFi network traces. For ED, since the network bandwidth is high, the standard ExoPlayer always selects A4 and the 1080p track, while our prototype always selects A2 and 480p or 720p (depending on the configured cap). The results are similar to the bottom row of Figure 6: on average, our prototype with 720p and 480p caps uses 54% and 30% of the data as that used by the standard ExoPlayer, the audio quality is similar, and the video quality under 720p cap is similar to that of the standard ExoPlayer, while the 480p cap leads to less number of good-quality segments.

For BBB, the average bitrates of the 480p, 720p, 1080p, and 4K are 0.4, 1.2, 2.1, and 10.8 Mbps, with the corresponding declared bitrate in the manifest file being 0.7, 2.0, 3.3, and 16.8 Mbps, respectively. We first test the case when only the first 6 tracks (up to 1080p) are included in the manifest file. Across the four WiFi traces, our prototype with 720p and 480p caps uses 1.2 and 0.4 Mbps bandwidth on average, 58% and 18% of what is used by the standard ExoPlayer, respectively; the video quality with the 720p cap is very close to that of the standard ExoPlayer, while the quality with 480p cap is lower (figures omitted). When including the 4K track in the manifest file, our prototype makes the same choices as before, while the standard ExoPlayer allows the 4K track to be selected. As a result, the bandwidth usage gap between our prototype and the standard ExoPlayer becomes even larger: our prototype only uses 13% and 4% of what is used by the standard ExoPlayer (9.1 Mbps on average). For the standard ExoPlayer, the downloaded 4K segments have to be downsampled to fit the phone screen resolution. Therefore, the resulting perceptual video quality of the displayed content is still comparable to what is achieved by our prototype with 720p cap, which already has very good quality.

5.2.4 Connected to Large Screen in Home WiFi. We again use the 7-track BBB video. The phone is attached to a 1080p or 4K external display in a home WiFi network setting. When connected to the 1080p display, our prototype only allows up to 1080p track, while the standard ExoPlayer allows up to the 4K track. The average bandwidth usage of our prototype is 2.1 Mbps, only 23% of what is used by the standard ExoPlayer (i.e., 9.1 Mbps). Again, the 4K segments downloaded by the standard ExoPlayer need to be downsampled to fit the 1080p display resolution. When connected to the 4K display, both our prototype and the standard ExoPlayer allow the 4K track to be selected (for different reasons: due to the resolution of the 4K display and the phone display resolution, respectively), and have identical results.

5.2.5 Connected to Surround Sound System in Home WiFi. We use ED with four audio tracks for this setting. Our prototype recognizes the surround sound system that supports up to eight channels, and hence allows 6-channel audio tracks to be selected, i.e., all the four audio tracks, A1-A4, can be selected. The standard ExoPlayer allows only A3-A4 to be selected due to over-specification, not based on C2 (see Section 3.2.1). Because of the high network bandwidth, both players choose the highest audio track (A4), and the respective highest resolution video track (480p or 720p for our prototype and 1080p for the standard ExoPlayer). As a result, they have similar audio quality; for video, the results are similar to those described in Section 5.2.3 for ED. Our prototype still uses less bandwidth: 62% and 38% of what is used by the standard ExoPlayer under the 720p and 480p caps, respectively.

6 Exploring C2-Cognizance for `dash.js`

We now present the results when C2 considerations are incorporated in `dash.js`. Since the original `dash.js` player has no limitation on video and audio track selection, we are particularly interested in scenarios where C2-based audio/video track selection is desirable. Specifically, we focus on ABR streaming on a phone over cellular networks (see Section 5.2.2), where it is desirable to filter out the 6-channel audio tracks to match the stereo speaker on the phone, and cap the video track selection to 480p or 720p to match the small phone display. This exploration on `dash.js` is complementary to that on ExoPlayer since these two players use very different ABR schemes that represent the ABR design space in existing players (see more details in [56]). In the following, instead of developing yet another full-fledged C2-cognizant prototype for `dash.js`, we emulate different types of C2-based filtering by passing different manifest files from the server to the player. Specifically, we use a manifest with only two audio tracks to emulate that only these two tracks are left after C2-based filtering. C2-based video filtering (up to 480p or 720p video track) is achieved in a similar manner.

The performance metrics are the same as those presented in Section 5.2.1 except for wasted amount of data due to chunk replacement (since it is not in the `dash.js` player we use). As in Section 5.2.1, we consider four settings of cellular network bandwidth (i.e., the average network bandwidth scaled to 1, 1.5, 2.5, or 5 Mbps), each with 10 cellular network traces.

Figure 8 (top row) shows the results for `dash.js` when the average network bandwidth is 1.0 Mbps. We see that the C2-based audio and video track filtering leads to significantly lower rebuffering than the original `dash.js` player: with the filtering, 70% and 80% of the traces have no rebuffering for 720p and 480p cap, respectively, while only 20% of the traces have no rebuffering for the original player. For the other performance metrics, we see similar results as those for ExoPlayer. **(i)** The C2-based filtering leads to significantly improved video quality compared to the original player: imposing 480p and 720p caps (and allowing up to stereo audio tracks) reduces the percentage of low-quality video segments by 8.6% and 7.5%, respectively. **(ii)** The C2-based filtering leads to slightly higher audio quality in ViSQOL (92% of the segments have ViSQOL quality

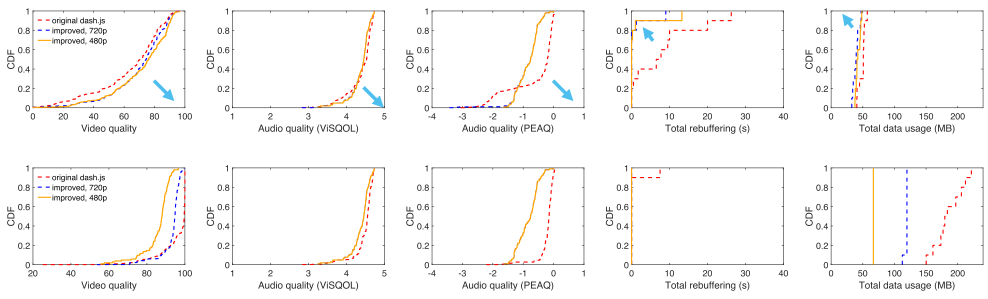


Fig. 8. Experimental results for `dash.js` over cellular networks; the two rows are the results when the average network bandwidth is 1.0 and 5.0 Mbps, respectively.

above 4 for our prototype, versus 85% for the original player), and less low-quality audio segments in PEAQ (0% and 1% of the segments have PEAQ quality below -2 for 480p and 720p caps, versus 12% for the original player). **(iii)** The data usage of the original `dash.js` player is slightly higher than the two cases with C2-based filtering since the low network bandwidth is the main constraint on data usage in this setting.

Figure 8 (bottom row) shows the results when the average network bandwidth is 5.0 Mbps. In this case, the original `dash.js` has rebuffering in one trace, while no rebuffering is present when using C2-based filtering. For the other performance metrics, the results are consistent with those for ExoPlayer: all three cases (with and without C2-based filtering) leads to good video and audio quality, while the original player uses significantly more data.

For the above two bandwidth settings, the quality change results are similar to those in Figure 7, and hence are omitted in the interest of space. Last, as in Section 5.2.2, the results for the two intermediate average network bandwidth (1.5 and 2.5 Mbps) show consistent trends as the above two bandwidth settings: C2-based filtering leads to lower rebuffering, less or similar low-quality video segments, and similar quality audio segments.

7 Related Work

This article substantially extended our preliminary version in [56]. First, we added the current C2 practices of YouTube player on both Andriod and iOS platforms (Section 3.4). Second, we formulated C2-cognizant ABR streaming as an optimization problem, and used it to illustrate the benefit of C2-cognizance in leading to better QoE and resource usage tradeoffs (Section 4.1). Last, we added experimental results that show the benefits of using C2-based filtering for `dash.js` player (Section 6). We next briefly review other related work.

Objective perceptual audio assessment. In Section 2.1, we consider three audio quality assessment models: ITU-T Rec. P.1203.2 [46], ViSQOL [24, 37, 66], and PEAQ [43]. In addition, there are other models, e.g., POLQA [45], PEMQ-Q [41], DNSMOS [62]. DNSMOS is recently proposed, but is used for content that has been noise suppressed, and does not use clean audio as reference. The above tools are only for assessing audio quality. We are not aware of any study that uses perceptual audio quality to drive audio track selection as in our study.

ABR rate adaptation. Many ABR schemes have been proposed in the literature (see surveys in [22, 48, 64]). Several schemes [40, 67, 68] are buffer-based. The schemes in [25, 29, 57, 59, 71, 74] are based on control theory. The schemes in Pensieve [53], D-DASH [31], Comyco [39], and GreenABR [72] leverage machine learning. QDASH [54] tries to reduce quality switches during adaptation. PANDA/CQ [51] jointly considers network bandwidth and video bitrate variability. Oboe [17] pre-computes the best possible ABR parameters for different network conditions and dynamically adapts the parameters at runtime. The main focus of the above ABR schemes is on addressing one aspect of C2, i.e., dynamic network bandwidth. Our study builds on existing ABR schemes, and focuses on incorporating other aspects of C2 (audio and video device capabilities, and ABR protocols) to supplement existing schemes. In addition, the above ABR schemes are not designed specifically for demuxed audio and video streaming. Our work expands the study in [60] to highlight the subtle interactions between audio and video rate adaptation in demuxed ABR streaming, and the benefits of incorporating C2 in such scenarios for the overall QoE.

Limiting track selection. Most ABR schemes aim to maximize quality; only several studies consider additional factors. The study in [58] assumes a user-specified target quality and aims to achieve the target quality to reduce data usage. The study in [61] proposes a framework and two strategies that use a per-session data budget to guide ABR rate adaptation to deliver good QoE while satisfying the data budget constraint. QBR [26] aims to improve existing ABR schemes by reducing the data usage while potentially increasing QoE. The work in [76] proposes two simple strategies to limit the track selection to achieve significant energy savings with little degradation in viewing quality. Some commercial streaming services provide options to cap the track selection to tradeoff the quality and data usage when streaming over cellular (see Section 2.2). Our work differs from them in that we identify C2 as an additional factor and demonstrate that C2-cognizance provides substantially improved tradeoffs in QoE and data usage.

8 Conclusions and Future Work

In this article, we identified C2 as an important factor to consider in ABR streaming, and explored its potential for realizing better tradeoffs between QoE and resource usage. We identified limitations in existing players in using C2 and developed practical best-practice guidelines that can be easily incorporated in existing player frameworks to realize the benefits of C2-cognizant ABR streaming. Furthermore, we developed a proof-of-concept prototype in the widely used ExoPlayer platform to instantiate and validate the guidelines. Our evaluations demonstrate that the approach has substantial benefits over the state-of-the-art.

As future work, we plan to extend C2-cognizant ABR streaming to a wider range of C2, e.g., streaming in moving vehicles versus stationary environment, in a quiet room versus a noisy environment, for far versus near user-screen distances, with and without other running applications on the device, and for different types of content being consumed. While some aspects have been explored in existing studies (e.g., [23, 35]), an interesting question is how to address the various aspects of C2 in a unifying framework to achieve C2-cognizance considering all these aspects. In addition, this study focuses on mobile devices. Another interesting direction is exploring C2 for other types of devices (e.g., laptops).

References

[1] Advanced Audio Coding. Retrieved December 10, 2022 from https://en.wikipedia.org/wiki/Advanced_Audio_Coding. (n.d.).
[2] DASH Industry Forum. Retrieved December 10, 2022 from https://dashif.org/. (n.d.).
[3] Dolby Digital Plus. Retrieved December 10, 2022 from https://en.wikipedia.org/wiki/Dolby_Digital_Plus. (n.d.).
[4] Everything You Need to Know About Screen Mirroring iPhone and iPad. Retrieved December 10, 2022 from https://tinyurl.com/57mw37ka. (n.d.).
[5] ExoPlayer. Retrieved December 10, 2022 from https://github.com/google/ExoPlayer/tree/r2.16.1. (n.d.).
[6] HTTP Live Streaming in Dolby. Retrieved December 10, 2022 from https://developer.dolby.com/platforms/apple/hls-in-dolby/. (n.d.).
[7] Surround sound. Retrieved December 10, 2022 from https://en.wikipedia.org/wiki/Surround_sound. (n.d.).

[8] Video scaler. Retrieved December 10, 2022 from https://en.wikipedia.org/wiki/Video_scaler. (n.d.).
[9] YouTube Stats for Nerds. Retrieved December 10, 2022 from https://tinyurl.com/37masafb. (n.d.).
[10] Martin Holters. 2015. GstPEAQ - A GStreamer plugin for Perceptual Evaluation of Audio Quality (PEAQ) v0.6.1. Retrieved November 2015 from https://github.com/HSU-ANT/gstpeaq
[11] International Organization for Standardization and International Electrotechnical Commission. 2018. Information technology–Multimedia application format (MPEG-A)–Part19: Common media application format (CMAF) for segmented media. Standard ISO/IEC 23000-19:2018, International Organization for Standardization and International Electrotechnical Commission. Retrieved December 10, 2022 from https://www.iso.org/standard/71975.html
[12] Stergios Terpinas and George Zachos. 2020. ViSQOL v3.1.0. Retrieved October 2020 from https://github.com/google/visqol
[13] 2021. Pixel 4a HDMI Output? Retrieved December 10, 2022 from https://www.reddit.com/r/GooglePixel/comments/i3n3t4/pixel_4a_hdmi_output/
[14] 2021. You can now play videos up to 4K60 on mobile regardless of screen resolution. Retrieved February 2021 from https://tinyurl.com/muuhan3x
[15] Michail. 2021. YouTube for Android now giving 4K playback option even if don't have a 4K screen. Retrieved February 2021 from https://tinyurl.com/23r7nfev
[16] Brittany Roston. 2021. YouTube on Android now lets you watch 4K videos on FHD screens. Retrieved February 2021 from https://tinyurl.com/ybyjxwb8
[17] 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*.
[18] Apple. AVFoundation. Retrieved December 10, 2022 from https://developer.apple.com/av-foundation/. (n.d.).
[19] Apple. AVPlayer. Retrieved December 10, 2022 from https://developer.apple.com/documentation/avfoundation/avplayer. (n.d.).
[20] Apple. 2017. Apple's HTTP Live Streaming. Retrieved December 10, 2022 from https://goo.gl/eyDmBc. (2017).
[21] ATSC (Advanced Television Systems Committee). 2012. ATSC Standard: Digital Audio Compression (AC-3, E-AC-3). (December 2012).
[22] Abdelhak Bentaleb, Bayan Taani, Ali C. Begen, Christian Timmerer, and Roger Zimmermann. 2019. A survey on bitrate adaptation schemes for streaming media over HTTP. *IEEE Communications Surveys & Tutorials* 21, 1 (2019), 562–585.
[23] Xianda Chen, Tianxiang Tan, Guohong Cao, and Thomas F. La Porta. 2022. Context-aware and energy-aware video streaming on smartphones. *IEEE Transactions on Mobile Computing* 21, 3 (March 2022), 862–877.
[24] Michael Chinen, Felicia S. C. Lim, Jan Skoglund, Nikita Gureev, Feargus O'Gorman, and Andrew Hines. 2020. ViSQOL v3: An open source production ready objective speech and audio metric. In *QoMEX*.
[25] L. De Cicco, S. Mascolo, and V. Palmisano. 2011. Feedback control for adaptive live video streaming. In *ACM MMSys*.
[26] William Cooper, Sue Farrell, and Kumar Subramanian. 2017. QBR Metadata to improve streaming efficiency and quality. In *SMPTE*.
[27] DASH Industry Forum. dash.js. Retrieved December 10, 2022 from https://goo.gl/XJcciV. (n.d.).
[28] Corbin Davenport. 2019. Pixel 4 has USB video output disabled in software. Retrieved November 2019 from https://tinyurl.com/3s6rwsbe.
[29] Luca De Cicco, Vito Caldaralo, Vittorio Palmisano, and Saverio Mascolo. 2013. ELASTIC: A client-side controller for dynamic adaptive streaming over HTTP (DASH). In *PV*. IEEE.
[30] FFmpeg. 2017. FFmpeg Project. Retrieved December 10, 2022 from https://www.ffmpeg.org/.
[31] M. Gadaleta, F. Chiariotti, M. Rossi, and A. Zanella. 2017. D-DASH: A deep Q-learning framework for DASH video streaming. *IEEE Transactions on Cognitive Communications and Networking* 3, 4 (December 2017), 703–718.
[32] Google. 2014. ExoPlayer: Adaptive video streaming on Android - YouTube. Retrieved December 10, 2022 from https://tinyurl.com/58j8n3yb. (2014).
[33] Google. 2016. ExoPlayer. Retrieved December 10, 2022 from https://github.com/google/ExoPlayer. (2016).
[34] Google. 2019. Shaka Player. Retrieved December 10, 2022 from https://github.com/google/shaka-player. (2019).
[35] S. He, Y. Liu, and H. Zhou. 2015. Optimizing smartphone power consumption through dynamic resolution scaling. In *ACM MobiCom* .
[36] Joe Hindy. 2021. 5 best screen mirroring apps for Android and other ways too. Retrieved December 10, 2022 from https://www.androidauthority.com/best-screen-mirroring-apps-android-ways-807191/. (2021).
[37] A. Hines, J. Skoglund, A. C. Kokaram, and N. Harte. 2015. ViSQOL: An objective speech quality model. *EURASIP Journal on Audio, Speech, and Music Processing* Article number 13, (2015).
[38] Tony Hoffman. 2021. The Best Portable Projectors for 2022. Retrieved November 2021 from https://tinyurl.com/mr3pfdue

[39] Tianchi Huang, Chao Zhou, Rui-Xiao Zhang, Chenglei Wu, Xin Yao, and Lifeng Sun. 2019. Comyco: Quality-aware adaptive video streaming via imitation learning. In *ACM MM*.

[40] 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 *ACM SIGCOMM*.

[41] R. Huber and B. Kollmeier. 2006. PEMO-Q: A new method for objective audio quality assessment using a model of auditory perception. *IEEE Audio, Speech, Language Process* 14, 6 (November 2006), 1902–1911.

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

[43] ITU. 2001. ITU-R Recommendation BS.1387-1: Method for objective measurements of perceived audio quality (PEAQ). (2001).

[44] ITU. 2012. Recommendation ITU-T P.1201.2: Parametric non-intrusive assessment of audiovisual media streaming quality – Higher resolution application area. (2012).

[45] ITU. 2014. ITU-T Rec. P.863: Perceptual objective listening quality assessment. (2014).

[46] ITU. 2017. Recommendation ITU-T P.1203.2: Parametric bitstream-based quality assessment of progressive download and adaptive audiovisual streaming services over reliable transport – Audio quality estimation module. (October 2017).

[47] K. Kellogg and T. Garcia. 2021. 13 Best Mini Projects for Indoor and Outdoor Use. Retrieved September 2021 from https://www.teenvogue.com/story/best-mini-projectors/

[48] Jonathan Kua, Grenville Armitage, and Philip Branch. 2017. A survey of rate adaptation techniques for dynamic adaptive streaming over HTTP. *IEEE Communications Surveys & Tutorials* 19, 3 (2017), 1842–1866.

[49] Lynn La. 2022. Top foldable phones for 2022: Motorola Razr 2020, Galaxy Flip, Galaxy Fold 2 and more. Retrieved January 2022 from https://www.cnet.com/tech/mobile/best-foldable-phones/. (2022).

[50] Zhi Li, Anne Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara. 2016. Toward A Practical Perceptual Video Quality Metric. Retrieved December 10, 2022 from https://goo.gl/ptjrWv.. (2016).

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

[52] Linux. 2014. tc. Retrieved December 10, 2022 from https://linux.die.net/man/8/tc. (2014).

[53] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *ACM SIGCOMM*.

[54] Ricky K. P. Mok, Xiapu Luo, Edmond W. W. Chan, and Rocky K. C. Chang. 2012. QDASH: A QoE-aware DASH system. In *ACM MMSys*.

[55] Mridula Nimawat. 2021. 15 Best Free Screen Mirroring Apps For Android & iPhone [2022]. Retrieved December 2021 from https://wethegeek.com/screen-mirroring-apps/

[56] Cheonjin Park, Chinmaey Shende, Subhabrata Sen, and Bing Wang. 2022. C2: Consumption context cognizant ABR streaming for improved QoE and resource usage tradeoffs. In *ACM MMSys*.

[57] Yanyuan Qin, Shuai Hao, Krishna R. Pattipati, Feng Qian, Subhabrata Sen, Bing Wang, and Chaoqun Yue. 2018. ABR streaming of VBR-encoded videos: Characterization, challenges, and solutions. In *CoNext*. ACM.

[58] 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 *MMSys*. ACM.

[59] Yanyuan Qin, Ruofan Jin, Shuai Hao, Krishna R. Pattipati, Feng Qian, Subhabrata Sen, Chaoqun Yue, and Bing Wang. 2020. A control theoretic approach to ABR video streaming: A fresh look at PID-based rate adaptation. *IEEE Transactions on Mobile Computing* 19, 11 (2020), 2505–2519.

[60] Yanyuan Qin, Subhabrata Sen, and Bing Wang. 2019. ABR streaming with separate audio and video tracks: Measurements and best practices. In *CoNext*. ACM.

[61] Yanyuan Qin, Chinmaey Shende, Cheonjin Park, Subhabrata Sen, and Bing Wang. 2021. DataPlanner: Data-budget driven approach to resource-efficient ABR streaming. In *ACM MMSys*.

[62] Chandan K. A. Reddy, Vishak Gopal, and Ross Cutler. 2022. DNSMOS: A non-intrusive perceptual objective speech quality metric to evaluate noise suppressors. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'22)*.

[63] Daniel Ruby. YouTube Statistics 2023: Data For Brands & Creators. Retrieved December 10, 2022 from https://tinyurl.com/bdec3hke. (n.d.).

[64] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia. 2015. A survey on quality of experience of HTTP adaptive streaming. *IEEE Communications Surveys & Tutorials* 17, 1 (2015), 469–492.

[65] SimilarTech Ltd. 2019. Facebook Video vs Shaka Player. Retrieved December 10, 2022 from https://www.similartech.com/compare/facebook-video-vs-shaka-player. (2019).

[66] Colm Sloan, Naomi Harte, Damien Kelly, Anil C. Kokaram, and Andrew Hines. 2017. Objective assessment of perceptual audio quality using ViSQOLAudio. *IEEE Transactions on Broadcasting* 63, 4 (2017), 693–705.

[67] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. 2018. From theory to practice: Improving bitrate adaptation in the DASH reference player. In *MMSys*.
[68] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. 2016. BOLA: Near-optimal bitrate adaptation for online videos. In *INFOCOM*. IEEE.
[69] Olivia Tambini. 2021. Spatial Audio: Our guide to immersive speakers, headphones, and streaming services. Retrieved December 10, 2022 from https://tinyurl.com/5n7bahwv. (2021).
[70] Ewdison Then. 2019. Pixel 4 USB-C video output exists but is disabled in source code. Retrieved November 2019 from https://tinyurl.com/2xbf8bxw
[71] Guibin Tian and Yong Liu. 2012. Towards agile and smooth video adaptation in dynamic HTTP streaming. In *ACM CoNEXT*.
[72] Bekir Oguzhan Turkkan, Ting Dai, Adithya Raman, Tevfik Kosar, Changyou Chen, Muhammed Fatih Bulut, Jaroslaw Zola, and Daby Sow. 2022. GreenABR: Energy-aware adaptive bitrate streaming with deep reinforcement learning. In *ACM MMSys*.
[73] Xiph. 2016. Xiph Video Test Media. Retrieved December 10, 2022 from https://media.xiph.org/video/derf/. (2016).
[74] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *SIGCOMM*. ACM.
[75] youtube-dl developers. 2018. youtube-dl. Retrieved December 10, 2022 from https://goo.gl/mgghW8. (2018).
[76] C. Yue, S. Sen, B. Wang, Y. Qin, and F. Qian. 2020. Energy considerations for ABR video streaming to smartphones: Measurements, models and insights. In *ACM MMSys*.
[77] Diekola Yusuf. 2019. 5 Best Projector Phones For Presenters. Retrieved September 2019 from https://tinyurl.com/mr2jh4dn

Received 21 April 2023; revised 5 December 2023; accepted 9 February 2024