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

Cheonjin Park[1], Chinmaey Shende[1], Subhabrata Sen[2], Bing Wang[1]
[1]University of Connecticut [2]AT&T Labs - Research

## ABSTRACT

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, etc. In this paper, we argue that a user's specific C2 is an important factor to consider in Adaptive Bitrate (ABR) streaming. We examine the current practice of using C2 in four popular ABR players, and identify various limitations in existing treatments that have a detrimental impact on network resource usage and user experience. We then develop practical best-practice guidelines for C2-cognizant ABR streaming. 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.

## CCS CONCEPTS

• **Information systems** → Multimedia streaming.

## KEYWORDS

ABR streaming; Consumption context; Resource efficiency.

**ACM Reference Format:**
Cheonjin Park, Chinmaey Shende, Subhabrata Sen, and Bing Wang. 2022. C2: Consumption Context Cognizant ABR Streaming for Improved QoE and Resource Usage Tradeoffs. In *13th ACM Multimedia Systems Conference (MMSys '22), June 14–17, 2022, Athlone, Ireland.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3524273.3528185

## 1 INTRODUCTION

Smartphones have emerged as ubiquitous platforms for us to consume content in a wide range of contexts: we listen to audio/music on our phones, using either the built-in speaker, or headphones that even support spatial audio [62]; we watch video using either the small built-in phone screen, a flip screen [44], or a mini projector [34, 42, 68] that projects the content to a larger space. In addition, our phones can be conveniently connected to play content on external peripherals, e.g., a large-screen and high resolution TV/monitor and 5.1 or 7.2 channel surround sound system [4, 32, 50]. One example of this latter use case is "Tailgate parties" for games or concerts [8]. 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 we shall see, to deliver good QoE, different types of C2 can have very different resource requirements and user needs. Therefore, C2-cognizant behaviors are important for various applications to appropriately optimize user experience and resource usage.

In this paper, we explore using C2 in ABR streaming, the current de facto technology for video streaming. In ABR streaming, the origin server provides multiple *tracks* or *variants* that all 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 variants to adapt to dynamic network conditions. At the server, video and audio can be muxed together, or demuxed (i.e., stored and streamed as separate tracks) [54]. In this paper, we focus on the demuxed paradigm because it has many advantages [54] and is widely adopted in popular streaming services. However, the C2-cognizant framework developed here also applies to the muxed case.

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 video 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 video streaming applications do not use more bandwidth than necessary for the current C2.

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 to accommodate dynamic bandwidth conditions (see §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 [54]. 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 displayed on a small phone built-in screen or a large external display, while audio can be played by the built-in stereo speaker on the phone or an external immersive multi-channel sound system (e.g., surround sound). 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 in the end-to-end streaming decisions.

In this paper, we explore using C2 to guide existing ABR rate adaptation schemes towards 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 for the following two main reasons. (i) Not tailoring the track selection to the current C2 can waste significant network bandwidth, without improving QoE. For instance, 5.1-channel surround sound and 2-channel stereo audio tracks deliver very different experiences when being played over a 5.1-channel capable sound system. However, with a built-in stereo speaker, while a player can still stream in and decode a 5.1 channel track, it will need to downmix [21] the 5.1 channel track to a 2-channel track to match the capability of the stereo speaker. The resulting perceptual audio quality would be comparable to or even worse than that of a 2-channel track, while the higher bandwidth usage of the 5.1 channel track can leave less bandwidth available for streaming the video part of the content, leading to lower video quality (see §2.1). (ii) 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. Summarizing the above, different resources may be needed to deliver good-quality experience in different C2.

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) [45], 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 §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 [25, 52] 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 paper, 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 (§2).
- We examine the current practice of using C2 in four popular ABR players spanning multiple platforms: ExoPlayer [30], `dash.js` [26], Shaka Player [31], and AVPlayer [19], and identify various limitations (§3). Our evaluation shows that, although the different players use elements of C2 to certain extents, 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 develop practical best-practice guidelines for achieving C2-cognizant ABR streaming (§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 instantiations to its specific use cases.
- To evaluate the design guidelines, we develop a proof-of-concept implementation in ExoPlayer (§5). Through a wide range of experiments under realworld scenarios (§6), 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. We further demonstrate that our prototype can recognize and react to dynamic C2 changes, and adjust audio/video track selection accordingly.
- We highlight the ABR protocol as another important component of C2 that needs to be considered carefully (§2 and §3.5). Specifically, DASH [37] 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.

Note that while our prototype implementation and evaluations focus on the use cases of displaying the video on the phone screen vs an external display, and playing the audio over the phone's built-in stereo speaker vs an external surround sound system[1], 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.

## 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 used between the server and client into account during track selection.

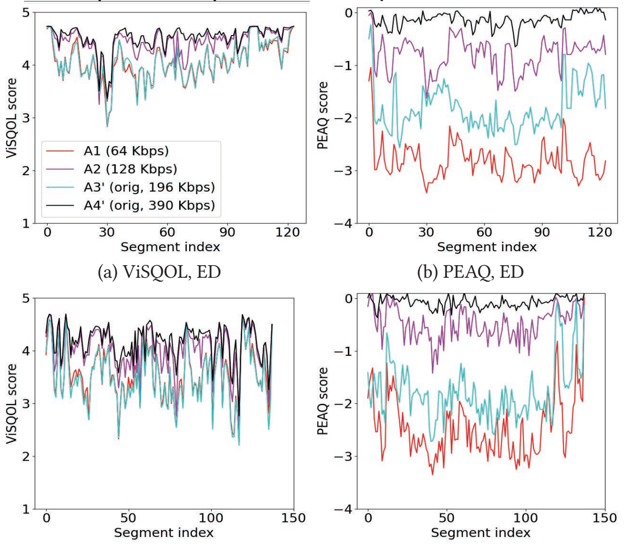
### 2.1 Audio Device and Audio Quality

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

[1]When a phone is attached to a peripheral device (display/speaker), we focus on the case where the phone streams the content from the server to the peripheral device, popularly referred to as *mirroring. In this case, the phone plays a central role in selecting the audio/video tracks from the server.*

**Table 1: Video and audio tracks of ED and ToS.**

| Track | Attributes | Avg., peak, DASH declared bitrate (Kbps) | |
|---|---|---|---|
| | | ED | ToS |
| 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 |



(a) ViSQOL, ED (b) PEAQ, ED

(c) ViSQOL, ToS (d) PEAQ, ToS

**Figure 1: Quality of four audio tracks of ED and ToS.**

for two demuxed media, ED (Elephant Dream) and ToS (Tears of Steel) [65]. 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 6 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 6 channels, either track has to be downmixed into 2 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 [41], ViSQOL [23, 33, 59], and Perceptual Evaluation of Audio Quality (PEAQ) [38]. 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 [39]. It adopts a no-reference model, 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 [23] was released in 2020 and improves upon its earlier versions [33, 59] 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 [65] 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.

Fig. 1a 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]. Fig. 1b plots the PEAQ Objective Difference Grade (ODG) scores for ED obtained using the software from [10], which is on a scale from -4 (very annoying impairment) to 0 (imperceptible impairment). Fig. 1c-d show the corresponding results for ToS. We see that with ViSQOL, A4' and A2 have similar scores, both higher than those of A3' and A1. 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 [59]. For instance, for ED, ViSQOL rates A2 and A3' in the range of good quality (i.e., around 4), while PEAQ rates A2 mostly as -1 (i.e., "impairment not annoying"), and rates most of the segments in A3' as -2 (i.e., "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, especially when the overall available network bandwidth is low. 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 Fig. 2. It is obtained by running ExoPlayer with DASH for ED on a Pixel 4a phone using a cellular network bandwidth trace (average bandwidth 1.5 Mbps). Both the video and audio are directly played on the phone. Fig. 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

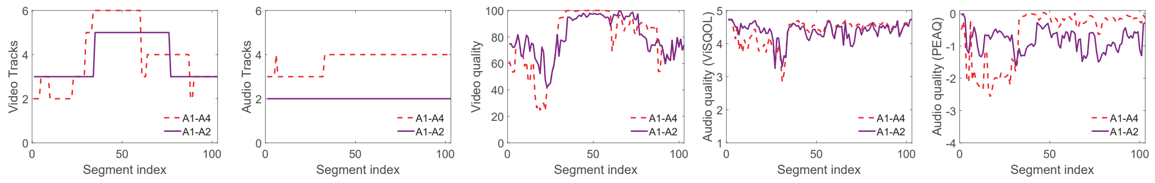


**Figure 2: An example illustrating the benefits of tailoring audio track selection to audio device capability (from ExoPlayer).**

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[2], 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, Fig. 2 shows that the A1-A4 case selects A3 for most of the early segments, 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.

**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 2 channels. Hence we cannot provide any numeric comparisons here. Developing good quality evaluation tools for multi-channel audio such as surround sound is a promising research area.

## 2.2 Display Device and Video Quality

It is important to match the video track selection to the requirement of the display context being used for consuming the video. However, there is a lot of focus currently 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. The resulting 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 [52]. In addition, streaming and playing the high-bandwidth 4K resolution can lead to high phone energy consumption [67] and undesirable stalls. As we shall see in §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, 37], 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 §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 as well as 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 (§3.5), the current treatment of this aspect in a popular player is not adequate.

## 3 CURRENT PLAYER PRACTICE

In this section, we investigate the current practice around C2 in four popular players and identify their limitations. Our main goal is to understand (i) how and to what extent C2-related information such as audio/video device capabilities and network type are obtained or configured in each player, and (ii) whether/how such information is used to filter out unsuitable audio/video tracks.

## 3.1 Players and Methodology

We study the latest version of the four players: ExoPlayer (v2.16.1), Shaka Player (v2.5.20), `dash.js` player (v3.1.1), and AVPlayer (running on iOS 15.0.1). 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 [29]. Shaka Player is a JavaScript library and has been used by more than 1,600 websites [58]. `dash.js` is a JavaScript based player and the reference player maintained by the DASH Industry Forum [2]. AVPlayer is based on AVFoundation [18], a full-feature framework currently recommended by Apple for audiovisual media on Apple platforms. 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, since it is proprietary, we rely primarily on controlled blackbox experiments.

While our study focuses on these four players, our findings have wide applicability to the services that use these players. In addition, the best-practice guidelines we present in §4 are applicable to any ABR streaming service, and are not limited to these players.

[2]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.

ExoPlayer and Shaka Player support both DASH and HLS, while `dash.js` only supports DASH, and AVPlayer only supports HLS. For clarity, we first describe the behavior of ExoPlayer, `dash.js`, and Shaka Player with DASH (§3.2 to §3.4), followed by ExoPlayer, Shaka Player and AVPlayer with HLS (§3.5 to §3.6). We use the built-in ABR logic of each player without any change. For the open-source players, we further describe the ABR logic briefly to provide insights on the observed results.

## 3.2 ExoPlayer with DASH Protocol

**Audio device and audio track filtering.** 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 device-capability agnostic behavior, even for a stereo speaker, all audio tracks that have up to 8 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 2 channels. *The above practice does not consider the speaker capability at all*, which can have significant adverse impact on QoE (see later).

**Display device and 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 [9] to fit the display resolution. If none of the video tracks satisfies the above condition, then no video track will be filtered. Otherwise, for the video tracks that 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 3 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. *While the above practice considers display capabilities to some extents, it is not sufficient* as we shall show later on.

ExoPlayer registers a listener to receive events regarding display addition, removal, and changes. However, currently, while it is aware of any change in the display being used, it updates some display related parameters (which is desirable) only when the change is from an external display to the built-in phone screen. It does not take any such action in other display change situations (e.g., display changes from the built-in phone screen to an external display). *The latter behavior is undesirable from a C2-cognizance perspective, as discussed later.*

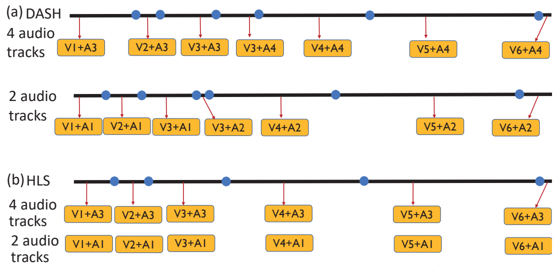


**Figure 3: Check point algorithm in ExoPlayer: (a) is for DASH and shows two cases, with 4 and 2 audio tracks in the manifest file, respectively; (b) shows the two cases for HLS.**

**Network type.** At the start of playback, ExoPlayer obtains the current network type and uses it to set the initial available network bandwidth estimate. When the network type is changed later, ExoPlayer detects the change (by registering a listener), updates the network type and resets the bandwidth information. Depending on the network type (e.g., WiFi, LTE), ExoPlayer sets the initial bandwidth estimate based on the country using a fixed lookup table [5], and *not based on the actual measurement of the network conditions.*

**ABR logic.** ExoPlayer has a specific logic for selecting the combinations of audio and video tracks. This logic calculates an approximate quality for each of audio/video track, based on the per-track declared bitrate in the manifest file, and then determines a subset of combinations, called *check points*, that combines higher bitrate/quality audio tracks with higher bitrate/quality video tracks. During rate adaptation, the player estimates the available network bandwidth, conservatively assumes that the available network bandwidth is 70% of the measured value . Let $b_i$ be the bitrate corresponding to the $i$-th checkpoint (i.e., the sum of the bitrates of the video and audio tracks). If the estimated network bandwidth is between $b_{i-1}$ and $b_i$, then the combination to the left of the $i$-th checkpoint can be selected (ExoPlayer also considers other factors, e.g., buffer level, in track selection).

Fig. 3a shows two examples for ED in Table 1: (i) when the manifest file has 4 audio tracks, A1-A4, and (ii) when it has 2 audio tracks, A1-A2. In (i), each check point, marked as circle on the line, includes one audio track out of A3-A4 (recall that only A3-A4 are retained after the audio filtering mechanism) and one video track out of V1-V6. In (ii), each check point includes one audio track out of A1-A2 and one video track out of V1-V6. These two cases have significantly different check points. As a result, their performance can differ significantly even under the same network conditions and the same ABR logic; see one example in Fig. 2.

**Impact of not using C2 adequately.** 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/bitrates. The example in Fig. 2 shows that such a

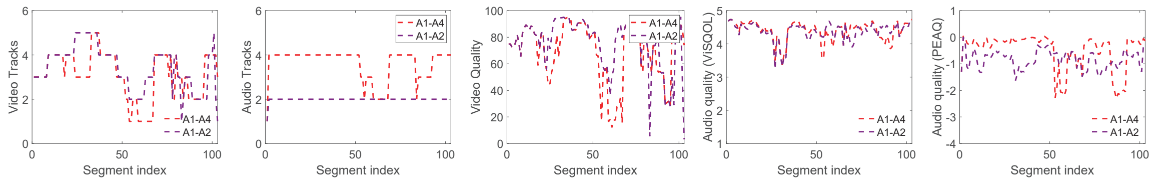


**Figure 4: An example showing the detrimental impact of not considering speaker capabilities in `dash.js`. In addition to worse video quality, it also leads to significantly more rebuffering (5.5 seconds vs no rebuffering, not shown in the figure).**

practice is problematic: it can lead to lower video quality, with no improvement in audio quality.

• 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 7 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 §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.

• The network type is only used to estimate the network condition, and not considered in audio/video track filtering. In cellular networks, it might be reasonable to further limit the video/audio track selection to save data and energy consumption [67].

## 3.3 `dash.js` Player with DASH Protocol

**Audio and video track filtering.** `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 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 both research and practice.

**Network type.** We observe that `dash.js` neither identifies nor uses network type information. It provides four modes for the initial track selection (e.g., select the highest bitrate track, or the first track); but none are based on the network connection type.

**ABR logic.** The default ABR logic in `dash.js` is `DYNAMIC` [60]. `dash.js` uses the above ABR logic for audio and video separately, using the bandwidth estimation for audio (video) that is based on past audio (video) downloading only. This practice is very different from that in ExoPlayer, which considers audio and video together.

**Impact of not using C2 adequately.** 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 by running `dash.js` for ED under a cellular network trace (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. Fig. 4 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 above 60 and 14% of the segments have VMAF below 40, while the corresponding values for A1-A2 case are 79% and 8%, respectively.

## 3.4 Shaka Player with DASH Protocol

**Audio and video track filtering.** Given multiple audio tracks, Shaka has an explicit logic to *prefer* those with 2 channels. Specifically, the audio tracks are placed into groups based on the number of channels. If there are tracks with 2 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 2 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.*

**Network type.** Shaka provides an option to determine whether network information is used to get initial network bandwidth estimate. This option is set to true by default, and Shaka uses the *downlink* attribute returned by Chrome API `navigator.connection` to set the initial network bandwidth estimate. Shaka also uses Chrome API to receive changes in network types in a streaming session.

**ABR logic.** 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. For DASH, since no combinations of audio and video tracks are specified in the manifest file, Shaka creates all possible combinations of the video and audio tracks listed in the manifest file, and considers all these combinations in the ABR track selection.

**Impact of not using C2 adequately.** 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 6 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 manifest file contains a high-resolution high birate 4K track. We see that the 4K track can be selected, which, as we pointed out in §2.2, is problematic in terms of both QoE and resource usage.

## 3.5 ExoPlayer and Shaka with HLS Protocol

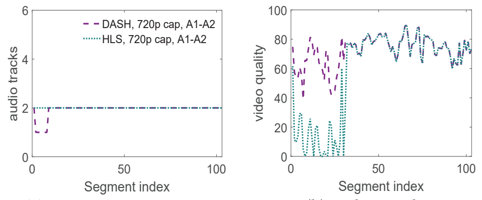
**ExoPlayer with HLS.** We highlight several differences between how HLS and DASH are handled by ExoPlayer.

• **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 §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.

• **Audio track filtering.** ExoPlayer uses the same logic to filter audio tracks for HLS as it uses for DASH (see §3.2). 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 4 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.** Again, ExoPlayer uses the same logic for DASH and HLS. This means that ExoPlayer still predetermines a set of audio and video combinations (see §3.2), 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 alone. This is shown in the top part of Fig. 3b, which differs significantly from that for DASH (the top part of Fig. 3a), although they are for *exactly the same* set of video and audio tracks, and only differ in the ABR protocol that is being used.

For comparison, the lower part of Fig. 3b shows the predetermined set of combinations when only A1-A2 are listed in the manifest file, in the order of A1, A2. In this case, ExoPlayer determines A1 to be the primary audio track and only uses A1 in the predetermined set of combinations. Again, the set of combinations differs significantly from that for DASH (the lower part of Fig. 3a), despite being for the same set of video and audio tracks. It also differs from the A1-A4 HLS case described earlier (the top part of Fig. 3b): these two cases have exactly the same set of check points, however, for a given check point, A1 versus A3 is combined with a video track.



(a) Audio track selection. (b) Video quality.

**Figure 5: Results of ExoPlayer with HLS and DASH under one cellular network trace (with proper C2-based filtering).**

• **Impact on QoE.** As described above, in addition to not handling C2 adequately, ExoPlayer with HLS also has the issue that it will choose a fixed audio track. Therefore, 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. Fig. 5 shows an example obtained when the playback is on a Pixel 4a phone under a cellular network trace (average bandwidth 1.0 Mbps). 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 built-in 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. Fig. 5a 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 Fig. 5b: 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 iOS 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.

**Shaka Player with HLS.** Unlike ExoPlayer, Shaka considers the set of audio and video track combinations specified in the HLS manifest file. As with DASH, it prefers audio tracks with 2 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.

## 3.6 AVPlayer on iPhone with HLS Protocol

Since AVPlayer is a proprietary player, we use controlled blackbox experiments to explore its behaviors. All the experiments below are carried out in a high-bandwidth 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, 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 1, 2 and 6 channels, respectively. All the three tracks are encoded in Dolby Digital Plus [3], recommend surround sound codec by Dolby [6]. In the *standalone* mode, i.e., 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 §2.1). When connecting the phone to a smart TV via HDMI, we experiment with two recent but different Samsung TVs. Both TVs have 2 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. Both these behaviors are undesirable. The preferred C2-cognizant behavior would be to select a 2-channel audio when the audio playback is over the TV's built-in speakers, and select a 6-channel track when the playback is over the attached surround sound system.

**Video track filtering.** We explore AVPlayer's video track selection using a manifest file with 7 tracks, the two highest tracks being 1080p and 4K. In both the standalone 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. Our testing shows that if this parameter is configured appropriately, the player can indeed select a higher resolution (e.g., 4K) track. Therefore, a C2 cognizant ABR streaming of video involving AVplayer would need to determine the C2 over time and dynamically adapt this parameter setting appropriately.

### 3.7 Summary of Main Findings

We see that all the four players have limitations in using C2, variously caused by behaviors such as having device-capability agnostic rules, not making appropriate device-capability cognizant selections, handling ABR protocol differences inadequately, etc. For example, ExoPlayer always specifies that a maximum number of 8 channels can be played; Shaka always prefers stereo audio tracks; AVPlayer in some cases prefers 2-channel content even when the peripheral is able to play multi-channel surround sound content; `dash.js` allows all audio and video tracks to be played irrespective of C2. As we showed earlier, such limitations can significantly impact the resulting QoE and resource usage.

## 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 present our best-practice guidelines in incorporating C2 in ABR streaming. We start with a high-level design, and then describe the details.

### 4.1 High-level Design

Tailoring the audio/video track selection to C2 requires meshing various information from the server and client, including specifying (at the server) properly the media attributes, 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 and then present our design.

**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 does 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.

Importantly, C2 can change during the playback session, 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 C2 information to the server, 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 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.

**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 and/or settings that are then appropriately selected 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.

**Proposed design.** Summarizing the above, we develop a client-based design that resides inside the player software, as shown in Fig. 6. 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,

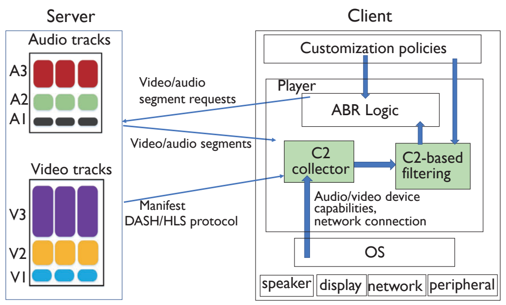


**Figure 6: High-level design of client-based C2-cognizant ABR streaming.**

and the other is *C2-based filtering*, which filters out audio/video tracks based on the current C2. Both components run continuously during the playback. The C2 collector feeds the C2 information to the C2-based filtering module, which interacts with the ABR logic. Existing customization polices such as "Play HD on Wi-Fi 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.

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 can 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 to be C2-cognizant, as we shall show in §5.

## 4.2 Leveraging C2 Information

Under the above design, we provide the following best-practice suggestions for the player to obtain and use C2 information:

• When a player starts, it is desirable to obtain the C2 details such as (i) the specific setup of the audio/video device, 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 later.

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

• The C2-based filtering module should 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 1440p screen resolution before displaying the video. In addition, due to the nature and limitations of human visual perception, it may not even need to bring in a 1440p track for a small phone screen (see §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 §2.1). A non-holistic piecemeal approach that considers only a single C2 dimension, or considers them separately can lead to undesirable behaviors such as high resource usage, QoE impairments, etc.

• The client player should explicitly consider and tailor its context-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 client is an important component of C2, since it leads to different ways of representing information and somewhat different types of information in the manifest file, which can have significant impact on the context-aware adaptation and resultant QoE. In particular, the DASH and HLS specifications 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 and requires careful treatment.

**Table 2: APIs for obtaining C2 information.**

| | Android | Chrome |
|---|---|---|
| Audio device | AudioManager | navigator.mediaDevices |
| Display device | DisplayManager<br>WindowManager | devicePixelRatio<br>HTMLElement |
| Network type | ConnectivityManager | navigator.connection |

**C2 from the underlying platform.** We 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 §5). Since C2 information is readily available to the players, our best-practice guidelines can be easily incorporated into existing players.

**Using C2 holistically.** Our best-practice guidelines above stress the importance of using C2 holistically, taking account of the various relevant factors and their relationships. A non-holistic piecemeal treatment of C2 can in fact lead to undesirable outcomes. As an illustration, an earlier version of ExoPlayer (v2.11.7) has a different design from the current version (v2.16.1). Specifically, the earlier version disables audio rate adaptation by default; the latest version takes a step in the right direction by enabling audio rate adaptation by default. However, the exact audio adaption scheme suffers from some design issues with respect to C2. The earlier version allows all audio tracks to be considered in ABR track selection (when audio rate adaptation is enabled), while the current version only allows the tracks that have properties matching those of the primary audio track, which is problematic (see §3.2). Using our best-practice guidelines, the above problems in both versions can be avoided, as we show next.

## 5 PROOF-OF-CONCEPT IMPLEMENTATION

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

• **Audio.** For audio, we first remove the logic that filters audio tracks based on the primary audio track (§3.2). 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., 2 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., 8 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 [52]. 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 [67]. (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 Fig. 6. 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 Android's `ConnectivityManager` API and limiting the maximum video track using `setMaxVideoSize`.

• **Dynamic settings.** We use Android's `BroadcastReceiver` API to obtain notification on speaker/display 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 ExoPlayer's `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 standalone mode and connecting phones to peripheral devices (see §6). The Samsung phone was able to identify the current C2 correctly in all the settings and our modifications achieved the desired behaviors. 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` function 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, 27, 63].

## 6 PROOF-OF-CONCEPT RESULTS

In this section, we use our proof-of-concept implementation 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., when starting in a standalone mode and then connecting an external speaker/display to the phone, and vice versa. In the following, we report the quantitative 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 §3.5), which is beyond the scope of this paper.

## 6.1 Experiment Setting

**Devices.** 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 4 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 6 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` [47] at the server to emulate realworld network scenarios. We consider both cellular and WiFi network settings. For cellular, we use 10 traces collected from two commercial LTE networks. 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 4 traces 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 [45]; 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) *data usage*: the total amount of data downloaded, and (v) *percentage of wasted data*: the amount of wasted data due to chunk replacement divided by the total data usage.

## 6.2 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 (see §3.2). 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 standard ExoPlayer allows all the 6 tracks (including 1080p) to be selected. We show the results of 3 scenarios: from the standard ExoPlayer, from our prototype with 480p or 720p cap. For all the scenarios, the percentage of chunk replacement tends to be lower when the network bandwidth is higher, since higher quality video tracks were already selected initially and hence less replacement is needed. For all the network settings, none of the runs has rebuffering.

The top row of Fig. 7 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, since A2 has higher quality than A3', 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. **(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 Fig. 7 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. 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. The performance when the average network bandwidth is 1.5 or 2.5 Mbps shows similar trends as above (figures omitted): our prototype leads to less low-quality video segments and lower data usage than the standard ExoPlayer.

## 6.3 Playback on Phone over Home WiFi

For this setting, we first use ED and then BBB. With ED, our prototype does the same C2-based filtering as that over cellular networks (see §6.2). For the four WiFi network traces, 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 Fig. 7: 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. Fig. 8 shows the bandwidth usage and quality of the played back video segments across the 4 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. When we include 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. In this case, the bandwidth usage gap between our prototype and the standard ExoPlayer becomes even larger: our prototype with 720p and 480p

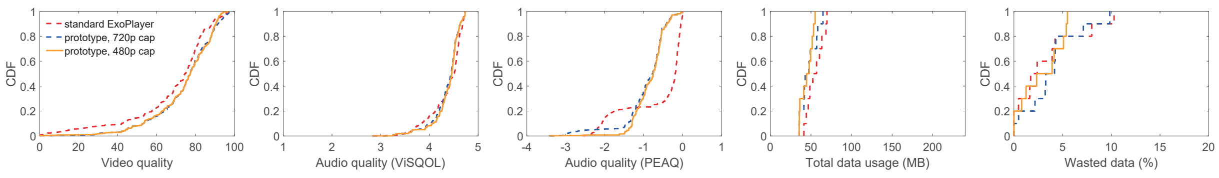
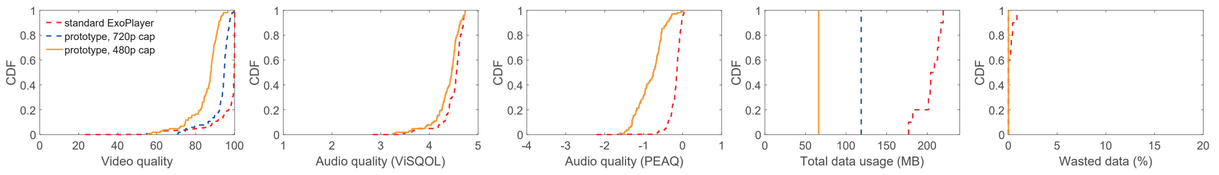



**Figure 7: Comparison of our prototype and the standard ExoPlayer for 6-track ED (playback on phone over cellular network settings). The two rows are the results when the average network bandwidth is 1.0 and 5.0 Mbps, respectively.**

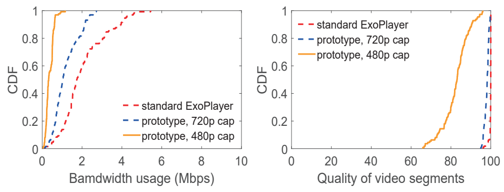


**Figure 8: Bandwidth usage (left) and video quality (right) for 6-track BBB (playback on phone over home WiFi network).**

caps only uses 13% and 4% of what is used by the standard ExoPlayer (9.1 Mbps on average), respectively. 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 a 720p cap, which already has very good quality.

## 6.4 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). The 4K segments downloaded by the standard ExoPlayer need to be downsampled to fit the 1080p display resolution, and hence the perceptual video quality will be similar to what is delivered by our prototype that uses much lower bandwidth. 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.

## 6.5 Connected to Surround Sound in Home WiFi

We use ED with 4 audio tracks for this setting. Our prototype recognizes the surround sound system that supports up to 8 channels, and hence allows 6-channel audio tracks to be selected. Hence, 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 §3.2). 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 §6.3 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.

## 7 RELATED WORK

**Objective perceptual audio assessment.** We are not aware of any study that uses perceptual audio quality to drive audio track selection as in our study. In addition to the three models in §2.1, there are other models, e.g., POLQA [40], PEMQ-Q [36], DNSMOS [56]. DNSMOS is recently proposed, but is used for content that has been noise suppressed, and does not use clean audio as reference.

**ABR rate adaptation.** The schemes are surveyed in [22, 43, 57]. The state-of-the-art schemes are buffer-based [35, 60, 61], control-theory based [24, 28, 51, 53, 64, 66], or using other techniques [17, 46, 48, 49]. We use the current ABR schemes in existing players, and focus on incorporating C2 to supplement existing ABR schemes.

**Limiting track selection.** Most ABR schemes aim to maximize quality; only several studies [25, 52, 55] consider additional factors to limit track selection. Our work differs from them in that we consider C2 as an additional factor and demonstrate that C2-cognizance provides substantially improved tradeoffs in QoE and data usage.

## 8 CONCLUSIONS

In this paper, 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 various limitations in how existing ABR players leverage C2 and developed practical best-practice guidelines. that can be easily incorporated in existing player frameworks. 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.

## ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their valuable feedback. We also thank Manuel Briand and Yanyuan Qin for their helpful discussions and feedback, and Ginger Chien for her insightful comments on an earlier version of the paper.

## REFERENCES

[1] [n.d.]. Advanced Audio Coding. https://en.wikipedia.org/wiki/Advanced_Audio_Coding.
[2] [n.d.]. DASH Industry Forum. https://dashif.org/.
[3] [n.d.]. Dolby Digital Plus. https://en.wikipedia.org/wiki/Dolby_Digital_Plus.
[4] [n.d.]. Everything You Need to Know About Screen Mirroring iPhone and iPad. https://www.airsquirrels.com/reflector/resources/everything-you-need-to-know-about-screen-mirroring-iphone-ios-ipad-ipados.
[5] [n.d.]. ExoPlayer. https://github.com/google/ExoPlayer/tree/r2.16.1.
[6] [n.d.]. HTTP Live Streaming in Dolby. https://developer.dolby.com/platforms/apple/hls-in-dolby/.
[7] [n.d.]. Surround sound. https://en.wikipedia.org/wiki/Surround_sound.
[8] [n.d.]. Tailgate party. https://en.wikipedia.org/wiki/Tailgate_party.
[9] [n.d.]. Video scaler. https://en.wikipedia.org/wiki/Video_scaler.
[10] 2015. GstPEAQ - A GStreamer plugin for Perceptual Evaluation of Audio Quality (PEAQ) v0.6.1. https://github.com/HSU-ANT/gstpeaq.
[11] 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. https://www.iso.org/standard/71975.html.
[12] 2020. ViSQOL v3.1.0. https://github.com/google/visqol.
[13] 2021. Pixel 4a HDMI Output? 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. https://www.reddit.com/r/youtube/comments/ln3dy8/you_can_now_play_videos_up_to_4k60_on_mobile/.
[15] 2021. YouTube for Android now giving 4K playback option even if don't have a 4K screen. https://www.gsmarena.com/youtube_for_android_showing_4k_playback_option_for_some_users-news-47842.php.
[16] 2021. YouTube on Android now lets you watch 4K videos on FHD screens. https://www.slashgear.com/youtube-on-android-now-lets-you-watch-4k-videos-on-fhd-screens-21660339.
[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. [n.d.]. AVFoundation. https://developer.apple.com/av-foundation/.
[19] Apple. [n.d.]. AVPlayer. https://developer.apple.com/documentation/avfoundation/avplayer.
[20] Apple. 2017. Apple's HTTP Live Streaming. https://goo.gl/eyDmBc.
[21] ATSC (Advanced Television Systems Committee). 2012. ATSC Standard: Digital Audio Compression (AC-3, E-AC-3).
[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).
[23] 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 *Proc. of International Workshop on Quality of Multimedia Experience (QoMEX)*.
[24] L. De Cicco, S. Mascolo, and V. Palmisano. 2011. Feedback Control for Adaptive Live Video Streaming. In *ACM MMSys*.
[25] William Cooper, Sue Farrell, and Kumar Subramanian. 2017. QBR Metadata to Improve Streaming Efficiency and Quality. In *SMPTE*.
[26] DASH Industry Forum. [n.d.]. dash.js. https://goo.gl/XJcciV.
[27] Corbin Davenport. 2019. Pixel 4 has USB video output disabled in software. https://www.androidpolice.com/2019/11/03/pixel-4-has-usb-video-output-disabled-in-software/.
[28] Luca De Cicco, Vito Caldaralo, Vittorio Palmisano, and Saverio Mascolo. 2013. ELASTIC: a client-side controller for dynamic adaptive streaming over HTTP (DASH). In *Proc. of Packet Video Workshop (PV)*. IEEE.
[29] Google. 2014. ExoPlayer: Adaptive video streaming on Android - YouTube. https://www.youtube.com/watch?v=6VjF638VObA.
[30] Google. 2016. ExoPlayer. https://github.com/google/ExoPlayer.
[31] Google. 2019. Shaka Player. https://github.com/google/shaka-player.
[32] Joe Hindy. 2021. 5 best screen mirroring apps for Android and other ways too. https://www.androidauthority.com/best-screen-mirroring-apps-android-ways-807191/.
[33] 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* (2015).
[34] Tony Hoffman. 2021. The Best Portable Projectors for 2022. https://www.pcmag.com/picks/the-best-portable-projectors/.
[35] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proc. of ACM SIGCOMM*.
[36] 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.
[37] International Organization for Standardization. 2012. ISO/IEC DIS 23009-1.2 Dynamic adaptive streaming over HTTP (DASH).
[38] ITU. 2001. ITU-R Recommendation BS.1387-1: Method for objective measurements of perceived audio quality (PEAQ).
[39] ITU. 2012. Recommendation ITU-T P.1201.2: Parametric non-intrusive assessment of audiovisual media streaming quality – Higher resolution application area.
[40] ITU. 2014. ITU-T Rec. P.863: Perceptual objective listening quality assessment.
[41] 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.
[42] K. Kellogg and T. Garcia. 2021. 13 Best Mini Projects for Indoor and Outdoor Use. https://www.teenvogue.com/story/best-mini-projectors/.
[43] 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).
[44] Lynn La. 2022. Top foldable phones for 2022: Motorola Razr 2020, Galaxy Flip, Galaxy Fold 2 and more. https://www.cnet.com/tech/mobile/best-foldable-phones/. access in January 2022.
[45] Zhi Li, Anne Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara. 2016. Toward A Practical Perceptual Video Quality Metric. https://goo.gl/ptjrWv..
[46] Zhi Li, Ali Begen, Joshua Gahm, Yufeng Shan, Bruce Osler, and David Oran. 2014. Streaming video over HTTP with consistent quality. In *ACM MMSys*.
[47] Linux. 2014. tc. https://linux.die.net/man/8/tc.
[48] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *Proc. of ACM SIGCOMM*.
[49] Ricky KP Mok, Xiapu Luo, Edmond WW Chan, and Rocky KC Chang. 2012. QDASH: a QoE-aware DASH system. In *ACM MMSys*.
[50] Mridula Nimawat. 2021. 15 Best Free Screen Mirroring Apps For Android & iPhone [2022]. https://wethegeek.com/screen-mirroring-apps/.
[51] 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.
[52] 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.
[53] 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).
[54] Yanyuan Qin, Subhabrata Sen, and Bing Wang. 2019. ABR Streaming with Separate Audio and Video Tracks: Measurements and Best Practices. In *CoNext*. ACM.
[55] 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*.
[56] Chandan KA Reddy, Vishak Gopal, and Ross Cutler. 2020. DNSMOS: A Non-Intrusive Perceptual Objective Speech Quality metric to evaluate Noise Suppressors. *arXiv e-prints* (2020), arXiv–2010.
[57] 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).
[58] SimilarTech Ltd. 2019. Facebook Video vs Shaka Player. https://www.similartech.com/compare/facebook-video-vs-shaka-player
[59] 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).
[60] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. 2018. From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player. In *MMSys*.
[61] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. 2016. BOLA: Near-Optimal Bitrate Adaptation for Online Videos. In *INFOCOM*. IEEE.
[62] Olivia Tambini. 2021. Spatial Audio: our guide to immersive speakers, headphones, and streaming services. https://www.techradar.com/news/spatial-audio-your-complete-guide-to-immersive-speakers-headphones-and-streaming-services.
[63] Ewdison Then. 2019. Pixel 4 USB-C video output exists but is disabled in source code. https://www.slashgear.com/pixel-4-usb-c-video-output-exists-but-is-disabled-in-source-code-03598259/.
[64] Guibin Tian and Yong Liu. 2012. Towards agile and smooth video adaptation in dynamic HTTP streaming. In *ACM CoNEXT*.
[65] Xiph. 2016. Xiph Video Test Media. https://media.xiph.org/video/derf/.
[66] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *SIGCOMM*. ACM.
[67] 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*.

[68] Diekola Yusuf. 2019. 5 Best Projector Phones For Presenters. https://www.novabach.com/5-best-projector-phones-for-presenters/.