ABR Streaming with Separate Audio and Video Tracks: Measurements and Best Practices

Yanyuan Qin[1], Subhabrata Sen[2], and Bing Wang[1]
[1]University of Connecticut [2]AT&T Labs - Research

ABSTRACT

Adaptive bitrate (ABR) streaming is the predominant approach for video streaming over the Internet. When the audio and video tracks are stored separately (i.e., in *demuxed* mode), the client needs to dynamically determine which audio and which video track to select for each chunk/playback position. Somewhat surprisingly, there is very little literature on how to best mesh together audio and video adaptation in ABR streaming. In this paper, we first examine the state of the art in the handling of demuxed audio and video tracks in predominant ABR protocols (DASH and HLS), as well as in real ABR client implementations in three popular players covering both browsers and mobile platforms. Combining experimental insights with code analysis, we shed light on a number of limitations in existing practices both in the protocols and the player implementations, which can cause undesirable behaviors such as stalls, selection of potentially undesirable combinations such as very low quality video with very high quality audio, etc. Based on our gained insights, we identify the underlying root causes of these issues, and propose a number of practical design best practices and principles whose collective adoption will help avoid these issues and lead to better QoE.

CCS CONCEPTS

• **Information systems** → Multimedia streaming;

KEYWORDS

Adaptive Video Streaming; Audio and Video; DASH; HLS

ACM Reference Format:
Yanyuan Qin, Subhabrata Sen, and Bing Wang. 2019. ABR Streaming with Separate Audio and Video Tracks: Measurements and Best Practices. In *The 15th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '19), December 9–12, 2019, Orlando, FL, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3359989.3365419

1 INTRODUCTION

Adaptive bitrate (ABR) streaming allows adaptation to dynamic network conditions by providing multiple *tracks/variants* that all represent the same content but are encoded at different bitrates and quality levels. Each track is divided into multiple *chunks*, each containing a few seconds worth of content. During playback, the client dynamically selects a chunk from the multiple available variants based on existing network conditions. On the server, a video track and its corresponding audio can be combined together as a single multiplexed track (in *muxed* mode), where each chunk in the track contains the associated video and audio content. Alternatively, the video and audio content can be maintained separately as demultiplexed tracks (in *demuxed* mode) on the server, as shown in Fig. 1. The demuxed approach offers a number of advantages for services that need to have more than one audio variant – e.g., to support multiple languages, or multiple audio quality levels or both. First, it requires significantly less storage at the origin server. To see this, consider a hypothetical service that uses M video and N audio tracks. The server only needs to store M video and N audio tracks for the demuxed mode, while it has to store a much larger set of $M \times N$ muxed tracks for the muxed mode. Second, *a subtle effect is that* the demuxed mode increases CDN cache hits. As an example, for a given chunk/playback position, suppose user A requests video track variant $V1$ and audio track variant $A2$, and at a later time, user B requests the same video track variant $V1$ and a different audio track variant $A1$. In muxed mode, B needs to download both V1 and A1 from the original server, while in demuxed mode, B can get the video chunk for V1 from the CDN cache (which cached it due to A's earlier request), and only needs to get the audio chunk for A1 from the original server. Due to the above advantages, services are increasingly moving towards using demuxed audio and video tracks [24]. We focus on ABR streaming for demuxed video and audio tracks in this paper.

The literature on video rate adaptation is extensive [2, 12, 13, 18, 23, 25]. A common assumption regarding audio is that its bitrate is significantly lower than that of the video, and hence the decision on audio track selection has little impact on the video track selection [22]. The reality, however, is that increasingly an audio track can be of similar or even much higher bitrate than some of the video tracks. Based on the HLS authoring specification for Apple devices [4], the audio bitrate can be as high 384 Kbps for mobile devices, which can be much higher than the typical encoding bitrates of lower-rung video tracks in real services (e.g., the two lowest video tracks have peak rates around 100 and 250 Kbps respectively for one popular service). In addition, more and

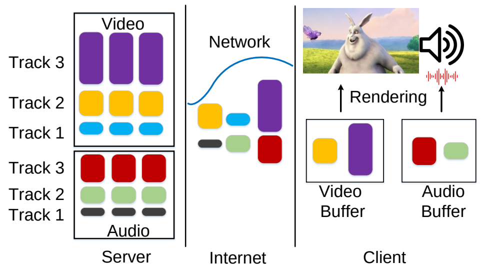


Figure 1: ABR streaming of demuxed audio and video (the different colors represent the chunks from different tracks).

more devices support Dolby Atmos audio [5], for which the audio bitrate can be up to 768 Kbps [19]. As a result, audio can in reality consume a significant fraction of the available network bandwidth, and audio track selection can therefore significantly affect video selection and the overall viewing experience (see §3). Concurrently with the availability of higher bitrate audio tracks, the community has also realized the importance of selecting higher audio quality tracks when possible and appropriate. For example, Netflix very recently adopted audio adaptation at the player side to boost the overall video streaming experience [19, 20].

Little is known on how best to mesh together audio and video rate adaptation in ABR streaming. In this paper, we first examine the current practice in the predominant DASH [6] and HLS [3] protocols and three popular players, ExoPlayer [8], Shaka Player [10], and `dash.js` player [7] (see §2). Using a combination of code inspection and experiments, we identify a number of limitations in existing practices both in the protocols and the player implementations (see §3). We identify the root causes of these limitations, and present a set of best practices and guidelines (see §4). Our work makes the following contributions:

• We find that all three players have various issues in handling demuxed audio and video tracks (§3). These issues are not simple implementation bugs, but rather arise due to a confluence of sometimes subtle interacting factors, ranging from a lack of sufficient insights, design/engineering inadequacies to architectural deficiencies. Specifically, these issues include (i) limitations in how DASH/HLS handles demuxed content, (ii) player designs to circumvent the limitations in DASH/HLS, (iii) little understood differences between HLS and DASH with important implications, (iv) attempts in applying the same player logic designed for one protocol (DASH/HLS) to the other, and (v) inadequate attention to synchronizing a player's video and audio downloading.

• Our findings demonstrate that meshing together audio and video rate adaptation is a very challenging problem – it involves many pieces (in ABR protocols, server and player designs) and needs a holistic solution. As a starting point in addressing the challenges, we provide a set of best practices and guidelines (§4) for the ABR protocol level (manifest file specification enhancements), the server side (appropriately using manifest to convey needed information to the client), and the player side (audio/video prefetching, and joint adaptation for audio and video).

Note that in this paper we consider coordinated ABR rate adaptation for streaming demuxed video and audio to the *same user* to maximize QoE. The demuxed video and audio tracks may be located at different servers and hence may not necessarily share the same bottleneck link. This problem is very different from the multi-user rate adaptation problem where videos are streamed to *multiple users* while sharing the same bottleneck link [1, 11, 13, 14].

We hope that our study brings the problem of ABR streaming with demuxed audio and video tracks to the community's attention With coordinated efforts, we hope that the community can come up with effective solutions soon.

2 MOTIVATION AND METHODOLOGY

2.1 Motivation

One way of treating demuxed audio and video tracks is determining their rate adaptation independently, and combining the audio and video chunks at playback time. While this approach is easy to implement, it has two major drawbacks. First, it can lead to some audio and video combinations that are clearly undesirable (e.g., lowest quality audio with highest quality video, or vice versa). Second, it can cause the prefetched audio and video streams to be severely unbalanced, e.g., audio (resp. video) buffer underruns and hence leads to stalls, while there is a lot of prefetched data for the corresponding video content (resp. audio).

Although the audio/video rate adaptation is decided by the player at the client, we argue that the server needs to provide sufficient information to facilitate the decision. Determining a good set of audio and video combinations is not a trivial task. It depends on the nature of the specific content, device characteristics (e.g., screen size, sound system), and the business rules encoding the content provider's preferences for that content. For instance, for music shows, the sound quality may be relatively more important than video quality, and hence it might be more desirable to combine high audio tracks with low/medium video tracks; while for an action movie, the desirable combinations may be the opposite. The origin server knows the content information, client device types, and the business rules, and hence is at a better position for deciding the combinations. Ideally, the server can determine a good set of audio and video combinations beforehand, and pass the information to the player through the manifest file or other out-of-the-band mechanisms.

2.2 High-level Methodology

To understand the current practice for handling demuxed audio and video tracks, we examine both predominant ABR streaming protocols (DASH and HLS) and the treatment in three open-source popular players, specifically, ExoPlayer [8] and Shaka [10] that are widely used in industry, and `dash.js` [7] that showcases the DASH standard. For each player, our focus is on understanding the interaction between audio and video rate adaptation and their impact on each other. Specifically, we use a combination of source code analysis (all the three players are open-source players) and controlled experiments (with additional instrumentation in the source code when necessary). The combined approach is necessary because as a large-scale often multithreaded software system, the player source code contains a large number of interlocking components with complex interactions, and just code analysis by itself may not be sufficient to understand the behavior of the system. The controlled experiments therefore play a crucial role in profiling the overall behavior and also help us to confirm our understanding. While our study focuses on these three players, our findings of specific issues (§3) in these players have wide applicability to the services that use these players. Our best practice recommendations (§4) are not player specific; we expect them to be widely applicable to any ABR streaming service.

2.3 ABR Streaming Protocols

DASH [6] and HLS [3] are the two predominant ABR streaming protocols. Both of them now support demuxed audio and video tracks. The DASH protocol defines an Adaptation Set as a set of interchangeable encoded versions of one or several media content components. For demuxed audio and video tracks, it defines one Adaptation Set for video tracks and another set for audio tracks.

A `bandwidth` attribute is associated with each track (termed *Representation* in DASH) to declare the required bandwidth (which is close to the peak bitrate; see one example in Table 1). In HLS, a top-level master playlist uses the EXT-X-STREAM-INF tag to specify an audio and video track combination (termed *Variant* in HLS). With HLS, one can specify all possible combinations or a subset thereof. A `BANDWIDTH` attribute is associated with a video and audio track combination to declare the aggregate bandwidth requirement of the combination (which is the sum of the peak bitrates of the audio and video tracks in the combination). In the rest of the paper, depending on the context, the phrase *bandwidth requirement* refers to the `bandwidth` attribute in DASH or `BANDWIDTH` attribute in HLS.

There are some key differences between DASH and HLS specifications in terms of how they treat demuxed audio and video tracks. These differences have important implications for both player ABR logic design and the resulting QoE. One key difference is that DASH does not provide a mechanism for a content provider to specify only the desired subset of audio and video track combinations, while HLS provides an explicit mechanism to do so. *Therefore, DASH provides a player more freedom to select combinations from the available audio and video tracks, potentially making it more vulnerable to choosing undesirable combinations (see §3).*

Another key difference between DASH and HLS is that DASH specifies the bandwidth requirements of individual audio and video tracks, while HLS top-level manifest only supports specifying the aggregate bandwidth requirement of each audio and video track combination. This difference should be recognized by the player to avoid undesirable behaviors, as we shall see later.

2.4 Players

We focus on three widely used and open-source players, ExoPlayer (v2.10.2) [8], Shaka Player (v2.5.1) [10], and `dash.js` player (v2.9.3) [7], and try to understand their behaviors towards audio and video rate adaptation using their latest versions. ExoPlayer is a widely used application level media player for Android platforms. More than 140,000 applications in Google Play Store use ExoPlayer to play media [9]. Shaka Player is an open-source JavaScript library for adaptive media, and has been used by more than 1,600 websites [17]. `dash.js` is an open-source JavaScript based player and is the reference player maintained by the DASH Industry Forum. ExoPlayer and Shaka Player support both DASH and HLS streaming; `dash.js` only supports DASH.

3 PRACTICE OF POPULAR PLAYERS

3.1 Experimental Setup

In the experiments below, we use a drama show downloaded from YouTube using `youtube-dl` [27]. It is around 5 minutes long, containing 6 video tracks and 3 audio tracks. Table 1 lists the average and peak bitrates of the video and audio tracks, as well as other key characteristics. The bitrate ladder in Table 1 is commonly used by YouTube [26]; the issues we point out below are related to the bitrate ladder (not particular to the specific content), and hence are broadly applicable.

We use the Bento4 toolkit [16] to create two sets of manifest files, complying respectively with DASH and HLS standards. For DASH, we create one manifest file, with the six video tracks and three audio tracks specified in two Adaptation Sets. The declared bitrate for each audio/video track is shown in Table 1. For HLS, we create two manifest files. The first, H_{all}, specifies all 18 combinations of video and audio tracks[1]; the second, H_{sub}, specifies a subset of 6 combinations, i.e., V1+A1, V2+A1, V3+A2, V4+A2, V5+A3, V6+A3, where high quality video tracks are associated with high audio quality tracks, and vice versa. The bitrates (both peak and average bitrates) for the combinations contained in these two manifest files are listed in Tables 2 and 3 in the Appendix.

Table 1: Video and audio of a YouTube drama show.

Audio/ Video Track	Average Bitrate (Kbps)	Peak Bitrate (Kbps)	Declared Bitrate for DASH (Kbps)	Audio channels, sampling rate Video resolution
A1	128	134	128	2 channels, 44 kHz
A2	196	199	196	6 channels, 48 kHz
A3	384	391	384	6 channels, 48 kHz
V1	111	119	111	144p
V2	246	261	246	240p
V3	362	641	473	360p
V4	734	1190	914	480p
V5	1421	2382	1852	720p
V6	2728	4447	3746	1080p

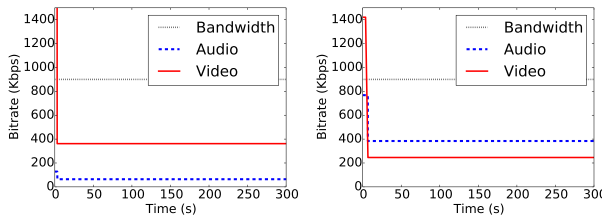
For controlled experiments, we set up a HTTP server as the origin server. The network bandwidths from the server to client are controlled by using `tc` [15] at the server. Our goal here is to identify demuxed audio-video related performance problems for each player, not to compare the performance across the multiple players. Therefore, we choose the experimental settings targeting the different issues of the different players. Whenever appropriate, we choose fixed network bandwidths, since the behavior of a player is more easily understood under such bandwidth profiles.

3.2 ExoPlayer

ExoPlayer adopted joint audio and video rate adaptation only very recently (starting from version v2.10.0, released in May 2019). Prior to that, it did not support simultaneous audio and video rate adaptation. Specifically, for multiple demuxed video and audio tracks, it selected a fixed audio track and used it throughout the session without any audio rate adaptation. In the following, we use ExoPlayer version v2.10.2, the latest version that is publicly available.

DASH. Since a DASH manifest file does not restrict the combinations of audio and video tracks that can be selected, ExoPlayer designs a specific logic that uses the per-track declared bitrate in the manifest file to determine a subset of combinations that combines higher bitrate/quality audio tracks with higher bitrate/quality video tracks. Specifically, for the audio and video tracks listed in Table 1, the resultant combinations, in increasing order of bandwidth requirement, are V1+A1, V2+A1, V2+A2, V3+A2, V4+A2, V4+A3, V5+A3, and V6+A3, where two adjacent combinations have either the same video or audio track. We refer to these combinations as the *predetermined* combinations by ExoPlayer. The subsequent rate adaptation process *only* considers these predetermined combinations. Specifically, during rate adaptation, the player estimates the

[1]While HLS provides a mechanism to specify a subset of video and audio combinations, it does not have an explicit recommendation that only carefully curated combinations should be specified. A content provider may choose to list all the combinations; we use H_{all} to illustrate the issues that may arise from such practices.



(a) Selected bitrates (experiment 1). (b) Selected bitrates (experiment 2).

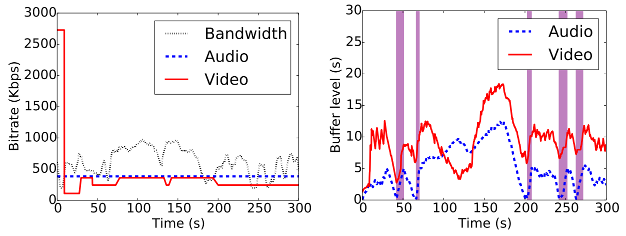
Figure 2: ExoPlayer results (DASH).

available network bandwidth by considering both video and audio downloading, and conservatively assumes that the actual network bandwidth is 75% of the estimated bandwidth. It then selects an audio and video combination out of the predetermined combinations (based on the bitrate of each of the combinations, network bandwidth estimate, and the buffer status).

The above treatment has the following limitation. The predetermined combinations by ExoPlayer, while may be reasonable for some bitrate ladder or content types, may not be suitable for others. As an example, for the above drama show, it may be desirable to allow V5+A2 (i.e., high quality video with medium quality audio), which is, however, not in the predetermined combinations, and hence will not be selected by ExoPlayer.

We next further illustrate the above point using two experiments. In the first experiment, we create an audio adaptation set with three audio tracks B1, B2 and B3 with the declared bitrate as 32, 64 and 128 Kbps, respectively. The video tracks are the same as listed in Table 1. In this case, the predetermined combinations are V1+B1, V2+B1, V2+B2, V3+B2, V4+B2, V5+B2, V5+B3, and V6+B3. The network bandwidth is fixed at 900 Kbps. Fig. 2(a) plots the average bitrate of the selected tracks. We see that V3+B2 is selected, while V3+B3 would be a better choice, since it has higher audio quality, and its bandwidth requirement (601 Kbps) is below the available network bandwidth. This more desirable combination (V3+B3) is, however, not in the predetermined combinations, and hence will not be selected. In the second experiment, we switch the audio adaptation set to three audio tracks with relatively higher bitrate, referred to as C1, C2 and C2 with the declared bitrate as 196, 384 and 768 Kbps, respectively. In this case, the predetermined combinations are V1+C1, V2+C1, V2+C2, V3+C2, V4+C2, V5+C2, V5+C3, and V6+C3. Fig. 2(b) shows that ExoPlayer selects V2+C2, resulting in very low video quality and high audio quality. The combination V3+C1 would be a better choice (V3+C1 has declared video and audio bitrates as 473 and 196 Kbps, respectively, versus 246 and 384 Kbps in V2+A2), which is however not in the predetermined combinations.

While clearly undesirable, the root cause of the above problem flows from a limitation in the DASH standard. As we mentioned in §2, the server is better placed to specify the desirable set of video and audio combinations. However, the DASH standard lacks such specifications. The DASH client, typically lacks detailed domain knowledge of the content being streamed, and therefore is forced to either (i) consider all possible combinations of video and audio track variants, or (ii) apply its own policy to determine a subset of combinations (as in ExoPlayer). Both approaches have drawbacks. The former can lead to a player selecting an unsuitable combination for a certain type of content, while the latter can potentially exclude desirable combinations of audio and video tracks for a content.



(a) Selected bitrates. (b) Buffer levels.

Figure 3: ExoPlayer results (HLS).

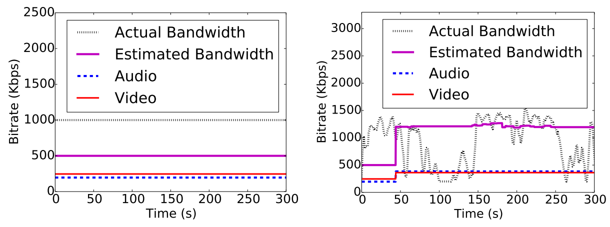
HLS. ExoPlayer uses the same rate adaptation code for both DASH and HLS. In contrast to the behavior with DASH, for HLS, this unfortunately leads to selecting a fixed audio track. In the following, we first illustrate this behavior using experiments and then explain the root cause.

In the first experiment, we specify a subset of audio and video combinations, H_{sub}, in the top-level master playlist; the first audio track in the manifest file is A3 (i.e., the highest bitrate/quality audio track). The network bandwidth is set to be time-varying, with the average as 600 Kbps. The evolution of the selected audio and video tracks is shown in Fig. 3(a); the audio and video buffer levels over time are shown in Fig. 3(b). We note that ExoPlayer selects A3 throughout the playback, resulting in *5 stall events and 36.9 seconds of rebuffering (marked as the shaded regions)*. In addition, the ABR selection has the effect of *disobeying the subset of audio and video combinations specified in the HLS manifest* as it selects some combinations (e.g., V1+A3) that are not in the specified subset. In the second experiment, we modify the manifest file so that A1 (the lowest quality audio track) is listed as the first audio track and fix the network bandwidth to 5 Mbps. ExoPlayer selects A1 throughout the playback despite plenty of available network bandwidth (figures omitted), leading to unnecessarily poor audio QoE.

What causes such behavior? Inspecting the ExoPlayer source code, we find that it is due to the lack of specification of the individual audio and video track's bitrate information in the top-level manifest file, which is needed by ExoPlayer to predetermine a subset of audio and video combinations. To accommodate the lack of this information, ExoPlayer simply assumes that all the audio tracks have the same quality, thereby leading to a fixed audio track selection. For a video track, it uses the aggregate bitrate of the first variant in the top-level manifest file that contains this video track as its bitrate, which is clearly an overestimation. The overestimation will be even more severe when the order of the variants is not specified properly (i.e., the first variant for a video track contains the video track and the highest bitrate audio track). In §4.1, we outline practical solutions to this problem.

3.3 Shaka Player

HLS. We use two experiments with the HLS manifest file H_{all} to understand the behavior of Shaka Player. In the first experiment, the network bandwidth is set to a constant 1 Mbps. Fig. 4(a) shows that the bandwidth estimated by Shaka is a constant 500 Kbps, only half of the actual specified network bandwidth. As a result, V2+A2 (with aggregate peak bitrate of only 460 Kbps) is selected. In the second experiment, the specified network bandwidth is dynamic (with the average as 600 Kbps, see Fig. 4(b)). In this case, Shaka first underestimates the network bandwidth, and then overestimates



(a) Selected bitrates (experiment 1). (b) Selected bitrates (experiment 2).

Figure 4: Shaka Player results (HLS).

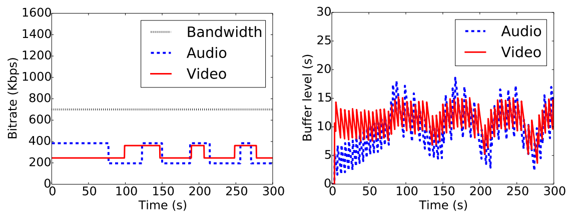
the bandwidth starting around 50 s. Correspondingly, the selected video and audio tracks are initially low (V2+A2), and then overly high (V3+A3), leading to a total rebuffering of 39 s. Inspecting the player code, we find that the above undesirable behaviors are due to Shaka's network bandwidth estimation algorithm. Specifically, Shaka uses past video and audio downloading throughputs as samples to estimate the bandwidth. It treats video and audio downloading separately. While downloading a video track (the same applies to audio downloading), Shaka considers each interval ($\delta = 0.125$s), calculates the amount of data d downloaded in that interval, and only counts the resultant throughput as a valid sample if $d \geq 16$ KB. To estimate the overall available network bandwidth, it considers a set of samples $\{c_1, \ldots, c_n\}$, where c_i is the throughput from either video or audio downloading, and sets the estimated bandwidth as an exponential weighted average of these samples. For time intervals when the audio and video streams are being concurrently downloaded over a shared network bottleneck link, the above strategy will severely underestimate the available network bandwidth. In addition, the filtering rule (i.e., only consider samples with $d \geq 16$ KB) itself can lead to bandwidth underestimation or overestimation, depending on the network conditions. In the first experiment above, none of the throughput samples satisfies the filtering rule, and hence the default bandwidth estimation of 500 Kbps is used throughout the streaming session. In the second experiment, only the throughput samples under high network bandwidth satisfy the rule, while those under low network bandwidth are discarded, leading to significant bandwidth overestimation.

Another issue is that Shaka uses a simple rate based adaptation scheme (i.e., selects the combination with the bandwidth requirement closest to the estimated bandwidth), which can cause the selected audio and video tracks to fluctuate frequently even if the bandwidth estimation is accurate. The above fluctuation problem is more severe for the case of demuxed audio and video since a large number of audio and video combinations may have close bandwidth requirements. For example, suppose manifest file H_{all} is used and the estimated network bandwidth varies between 300 to 700 Kbps. Then the selected combinations can fluctuate among V1+A2, V2+A1, V2+A2, V1+A3 and V2+A3, with bandwidth requirements as 318, 395, 460, 510 and 652 Kbps, respectively.

DASH. Under DASH, since no combinations of audio and video tracks are specified in the manifest file, the player creates all the combinations of video and audio tracks when parsing the DASH manifest file. Therefore, the result is the same as that for HLS when using manifest file H_{all}.

3.4 `dash.js` Player

The default ABR algorithm used in `dash.js` is `DYNAMIC` [22], which switches between two schemes, `THROUGHPUT` and `BOLA`. `THROUGHPUT` is a rate based approach that chooses tracks whose declared peak bitrates are close to the estimated bandwidth. `BOLA` is a buffer based approach that optimizes a utility function [23]. `DYNAMIC` starts by using `THROUGHPUT` rule. It switches to `BOLA` when the buffer level is above 12 s and `BOLA` selects a bitrate at least as high as that selected by `THROUGHPUT`; it switches back to `THROUGHPUT` if the buffer is less than 6 s and `BOLA` selects a bitrate lower than that selected by `THROUGHPUT`.



(a) Bitrate selections. (b) Buffer levels.

Figure 5: dash.js results (DASH).

Examination of the source code of `dash.js` shows that it utilizes `DYNAMIC` strategy for both audio and video, and performs rate adaptation for audio and video separately. In addition, the bandwidth estimation for audio (video) is based on past audio (video) downloading only. Fig. 5 plots the results when using a fixed network bandwidth of 700 Kbps. We see that the selected video and audio combinations includes V2+A3, V2+A2, V2+A3 and V3+A3. Some of these combinations are clearly undesirable, e.g., V2+A3. The combination, V3+A2, fits the network bandwidth profile (it actually has lower bandwidth requirement than V2+A3), while it has higher video quality and slightly lower audio quality than V2+A3, which might be more preferable from the user perspective for the drama show. We further see from Fig. 5(b) that the buffer levels for audio and video can be unbalanced, which is undesirable since when one of them underruns, stalls will happen, even if there is a lot of content in the other buffer.

3.5 Summary of Findings

In summary, we observe the following problems with current ABR streaming protocols and player implementations.

• When a player does not perform rate adaptation for audio (e.g., ExoPlayer for HLS), it can lead to poor performance (e.g., a large amount of rebuffering). Therefore, it is desirable to perform rate adaptation for both audio and video.

• Some players select undesirable audio and video combinations, which can lead to poor viewing quality (e.g., the lowest video and highest audio tracks). The problem is more severe for DASH, which does not provide a mechanism to specify a subset of desirable audio and video combinations, making the player more vulnerable to selecting bad combinations.

• Some players make the rate adaptation decisions for audio and video completely independently (e.g., `dash.js`). Considering audio and video jointly is better than considering them independently. But considering them jointly in an overly simplistic way is also not desirable (e.g., the practice in Shaka player can lead to frequent audio/video track changes).

• Some players do not synchronize audio and video downloading explicitly (e.g., `dash.js`). This can lead to unbalanced content in video and audio buffers. Since we need both video and audio in the playback, having a lot more content in video/audio is not helpful. Instead, it is more desirable to synchronize them on a finer granularity (e.g., on per chunk level as in ExoPlayer).

• An HLS manifest can specify a subset of audio and video combinations, but some players do not conform to the manifest file. For example, ExoPlayer may select some bad combinations that are not in the manifest file.

4 SUGGESTED BEST PRACTICES

Based on the analysis in §3, we suggest the following best practices for demuxed audio and video tracks for the server side as well as the player side.

4.1 Server Side Manifest Specifications

Audio and video combinations. *The content provider should identify desirable combinations of audio and video tracks based on content type and domain expertise, and specify these combinations in the manifest file. A player should only select chunks from the allowed combinations.* This practice provides the following advantages: (i) allows the content owner to specify combinations that are suitable for the specific content (e.g., music video v.s. action movie) and devices (considering screen size, connected sound system, e.g., speaker or headphone), and (ii) simplifies the rate adaptation task for the player. HLS already supports this capability to include specific combinations in the manifest file. However it is important for the content provider to utilize and leverage this capability appropriately, and not specify all possible combinations unless they are all desirable. Unlike HLS, DASH at present does not offer a way to list only specific allowed combinations in the manifest. A practical short term workaround would be for the client to get this additional information from the server using HTTP. In the longer term, the DASH specification can be expanded to support this feature.

Audio and video bandwidth declaration. *In the manifest file, it would be good to specify sufficient information about the aggregate bandwidth requirements of audio and video combinations, as well as the bandwidth requirements of individual audio/video tracks.* This is particularly important when audio and video are fetched over different network paths that have different network characteristics (e.g., when they are stored at different servers). Currently, DASH specifies the bandwidth for each audio and video track, and the aggregate bandwidth requirement for audio and video combinations (if provided following the earlier suggestion) can be calculated from the individual tracks. In HLS, the top-level master playlist *only* specifies the aggregate bitrate for each specified audio and video combination. The bitrate of each individual audio/video track is not in the top-level master playlist, but can be obtained from the second-level media playlists as follows (commercial players only use the information to identify the content address for an already selected chunk): (i) When all the video/audio chunks are packaged into a single file, the media playlists specify the EXT-X-BYTERANGE information (i.e., the start and end byte positions), which can be used to obtain the audio/video bitrate, as pointed out in [21]. (ii) When each video/audio chunk is packaged into an individual file, no byte range information is provided in the media playlists. However, HLS recently added an optional EXT-X-BITRATE tag that can be used to obtain per-chunk bitrate. We recommend that this option should be made mandatory. In addition, for both of the above two cases in HLS, since the information required for computing the per-track bitrates is in the second-level manifest files, we suggest that the player should download these files and read the information before making rate adaptation decisions[2]. A more robust longer term solution is to enhance the HLS specification so that the top-level master playlist directly provides per-track audio and per-track video bitrate information.

4.2 Player Side ABR Logic

Adopt audio rate adaptation. *For dynamic network bandwidth scenarios, it is important to perform audio rate adaptation.* High quality audio tracks can have similar or even higher bitrates than lower-rung video tracks. Audio rate adaptation is just as important as video rate adaptation to avoid adverse impact on QoE (e.g., rebuffering as observed in §3.2 or low audio quality).

Select only from allowed audio and video combinations. *The ABR logic should only select from the set of allowed audio and video combinations if provided by the server (e.g., via manifest file or certain out-of-band mechanism).* As mentioned in §4.1, the allowed combinations reflect the desirable combinations between audio and video based on the content. Therefore, it is important for the client to select only from those combinations.

Joint adaptation of audio and video. *We suggest that the selection of the audio and video tracks for each chunk position (in playback order) be considered jointly.* For both video and audio rate adaptation, it is desirable to satisfy conflicting goals in maximizing quality, minimizing stalls and minimizing quality variation. Since the selection of audio and video is inherently coupled (so that good combinations of audio and video tracks are chosen), we recommend considering the combinations of audio and video (i.e., those specified in the manifest file if provided) while making rate adaptation decisions, instead of considering audio and video individually. The rate adaptation should be done carefully to avoid frequent changes in either audio or video tracks.

Maintain balance between audio and video prefetching. *It is desirable to keep the audio and video buffer levels (in seconds) balanced.* This is because either empty audio or video buffer leads to stalls. The balance can be achieved by synchronizing the duration of prefetched audio and video content at a fine granularity, e.g., at the chunk level or in terms of a small number of chunks.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we examined the handling of ABR adaptation for demuxed video and audio tracks in predominant ABR streaming protocols (DASH and HLS), and in three popular ABR players. We identified a number of limitations in existing practices that can adversely impact user QoE, and traced their root causes. We then proposed a number of best practices and principles for the server, ABR streaming protocols and client. We hope these findings will encourage further studies on this important topic. As future work, we are working to further refine the suggested practices, and plan to design and implement rate adaptation schemes following the suggested practices.

[2]We suggest avoiding the practice of "lazy" fetching, which only fetches the playlist manifest file for a track when a chunk from that track has been selected, at which point the player needs to know the address of that chunk.

REFERENCES

[1] Saamer Akhshabi, Lakshmi Anantakrishnan, Ali C Begen, and Constantine Dovrolis. 2012. What happens when HTTP adaptive streaming players compete for bandwidth?. In *Proceedings of International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*. ACM, 9–14.

[2] 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 *Proc. of ACM SIGCOMM*.

[3] Apple. 2017. Apple's HTTP Live Streaming. https://developer.apple.com/streaming/. (2017).

[4] Apple. 2017. HLS Authoring Specification for Apple Devices. (2017). https://developer.apple.com/documentation/http_live_streaming/hls_authoring_specification_for_apple_devices

[5] Dolby. 2018. The Dolby Atmos Difference. https://www.dolby.com/us/en/brands/dolby-atmos.html. (2018).

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

[7] DASH Industry Forum. 2019. DASH IF Reference Client 2.9.3. https://github.com/Dash-Industry-Forum/dash.js. (2019).

[8] Google. 2016. ExoPlayer. https://github.com/google/ExoPlayer. (2016).

[9] Google. 2017. ExoPlayer: Flexible Media Playback for Android (Google I/O '17). https://youtu.be/jAZn-J1I8Eg?t=552. (2017).

[10] Google. 2019. Shaka Player. https://github.com/google/shaka-player. (2019).

[11] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. 2012. Confused, timid, and unstable: picking a video streaming rate is hard. In *Proc. of IMC*. ACM.

[12] 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*.

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

[14] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. Begen, and D. Oran. 2014. Probe and adapt: Rate adaptation for HTTP video streaming at scale. *IEEE JSAC* 32, 4 (2014), 719–733.

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

[16] Axiomatic Systems LLC. 2016. Bento4 MP4 and DASH Class Library, SDK and Tools. (2016). https://www.bento4.com

[17] SimilarTech Ltd. 2019. Facebook Video vs Shaka Player. (2019). https://www.similartech.com/compare/facebook-video-vs-shaka-player

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

[19] Netflix. 2019. Bringing Studio Quality Sound to Netflix. (2019). https://media.netflix.com/en/company-blog/bringing-studio-quality-sound-to-netflix

[20] Netflix. 2019. Engineering a Studio Quality Experience With High-Quality Audio at Netflix. shorturl.at/cfhx5. (2019).

[21] Yanyuan Qin, Shuai Hao, K. R. Pattipati, Feng Qian, Subhabrata Sen, Bing Wang, and Chaoqun Yue. 2018. ABR Streaming of VBR-encoded Videos: Characterization, Challenges, and Solutions. In *Proc. of ACM CoNEXT*.

[22] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. 2018. From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player. In *Proc. of ACM MMSys*.

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

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

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

[26] YouTube. 2019. Recommended upload encoding settings. https://support.google.com/youtube/answer/1722171?hl=en. (2019).

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

A BITRATE OF AUDIO AND VIDEO COMBINATIONS

Table 2 lists the full set of the 18 audio and video combinations for the drama show in Table 1. They are the combinations listed in the HLS manifest file H_{all}. For each combination, the peak bitrate is the sum of the peak bitrates of the audio and video tracks; the average bitrate is sum of their average bitrates. The combinations are placed in increasing order of the peak bitrate.

Table 3 lists a subset of 6 audio and video combinations for the drama show in Table 1. They are the combinations listed in the HLS manifest file H_{sub}. For each combination, both the peak and average bitrates are listed in the table.

Table 2: Bitrates of the full set of audio and video combinations (used in HLS manifest file H_{all}).

Video/Audio Combination	Average Bitrate (Kbps)	Peak Bitrate (Kbps)
V1 + A1	239	253
V1 + A2	307	318
V2 + A1	374	395
V2 + A2	442	460
V1 + A3	495	510
V2 + A3	630	652
V3 + A1	490	775
V3 + A2	558	840
V3 + A3	746	1032
V4 + A1	862	1324
V4 + A2	930	1389
V4 + A3	1118	1581
V5 + A1	1549	2516
V5 + A2	1617	2581
V5 + A3	1805	2773
V6 + A1	2856	4581
V6 + A2	2924	4646
V6 + A3	3112	4838

Table 3: Bitrates of a subset of audio and video combinations (used in HLS manifest file H_{sub}).

Video/Audio Combination	Average Bitrate (Kbps)	Peak Bitrate (Kbps)
V1+A1	239	253
V2+A1	374	395
V3+A2	558	840
V4+A2	930	1389
V5+A3	1805	2773
V6+A3	3112	4838