BETA: Bandwidth-Efficient Temporal Adaptation for Video Streaming over Reliable Transports

Cyriac James, Mea Wang
University of Calgary
{cyriac.james,mea}@ucalgary.ca

Emir Halepovic
AT&T Labs – Research
emir@research.att.com

ABSTRACT

To cope with diverse network conditions, HTTP Adaptive Streaming (HAS) enables video players to dynamically change the video quality throughout the video stream. However, effective adaptation that minimizes stalls and start-up time while maximizing quality and stability remains elusive, especially when available bandwidth is variable or multiple players compete for the bottleneck capacity. Conventional approach to adaptation is to make a decision on the next video segment quality based on hysteresis of prior throughput measurements. This approach is not robust to bandwidth fluctuation at small time scales, which can consequently lead to stalls, bandwidth waste, and unstable quality, mainly due to the inability to mitigate significant bandwidth reduction during the segment download. We propose BETA– Bandwidth-Efficient Temporal Adaptation, an agile approach that allows HAS players to refine the quality level within video segments on the fly, according to the actual bandwidth conditions experienced *while* downloading each segment. We define a new HAS-oriented transmission order of video frames within segments that facilitates decodability of partial frames and paves the way for changing the paradigm from discrete to continuous bitrate ladders for HAS. BETA can work with any adaptation algorithm or HAS player to significantly improve robustness and efficiency in dynamic network environments and for low-latency streams, as well as to dramatically reduce content storage and encoding infrastructure requirements. Our evaluation using the real player implementation shows that BETA improves video quality by up to 20%, reduces number of stalls by 20%-100% in nearly 80% of cases, and cuts down wasted bandwidth by 22%-100%.

CCS CONCEPTS

• **Information systems → Multimedia streaming**;

KEYWORDS

DASH, Adaptive Streaming, Codec, Loss tolerance, Efficiency

ACM Reference Format:
Cyriac James, Mea Wang and Emir Halepovic. 2019. BETA: Bandwidth-Efficient Temporal Adaptation for Video Streaming over Reliable Transports. In *Proceedings of 10th ACM Multimedia Systems Conference (MMSys '19).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3304109.3306235

MMSys '19, June 18–21, 2019, Amherst, MA, USA

ACM ISBN 978-1-4503-6297-9/19/06…$15.00
https://doi.org/10.1145/3304109.3306235

1 INTRODUCTION

HTTP Adaptive Streaming has flourished in recent years and now accounts for the vast majority of Internet video delivered to user devices. Its success is mainly based on its ability to be deployed via existing Content Delivery Network (CDN) infrastructure, and broadly accepted standards for reliable delivery and video coding, such as HTTP(S) over TCP, and H.264 [42]. However, even with the ability to switch between multiple video quality levels (i.e., bitrates) by splitting the video into independent segments, HAS clients are still not robust to sudden reductions in available bandwidth, as caused by congestion from cross-traffic or competition among HAS players, normal variation in wireless link quality, or other reasons.

Quality adaptation in state-of-the-art HAS players is client-driven and typically based on averaging past observations, i.e., prior video segment throughput, combined with the playback buffer state. This creates an inherently delayed response to the current throughput observed at the application layer. If the HAS client makes an inappropriate quality adjustment (e.g., requests a high-bitrate segment when bandwidth is about to go down), it could eventually stall due to depleted buffer. The buffer depletes due to one fundamental reason – the client commits to downloading the segment that cannot complete before it is due to play. Industry reports indicate that 5.1 billion viewing hours of streaming TV were lost in the 2nd quarter of 2018 due to stalls and re-buffering [2].

Playback stalls caused by re-buffering events have the largest impact on end-user Quality of Experience (*QoE*), and content providers have the highest incentive to eliminate them [20]. To mitigate stalls, various client-based solutions exist. Most resort to conservative quality selection in practice, which leads to lower Visual Quality (VQ) and bandwidth under-utilization [15]. For Video on Demand (VoD) content, some players maintain a very large buffer to "ride out" disruptions – a very inefficient approach that wastes bandwidth and energy when users abandon the video before it ends. A few clients also use a *drop-and-replace* approach, where the transmission of a video segment times out, i.e., takes too long to complete, so the partially received segment is dropped and replaced by the same segment at lower (often the lowest) quality [4, 36, 38]. This may prevent some stalls, but does not guarantee the stall-free experience [33], since the lower quality segment may not arrive in time either. Moreover, bandwidth and time have already been wasted on the abandoned segment.

With emerging HAS applications, such as live TV, live events (e.g., auctions, gambling), 360 video, etc., the requirements for low latency to live broadcast are very high [9], calling for shorter, even sub-second segments. This means that the luxury of large buffers no longer exists, and that the time window to drop and replace the segment is diminishing. Hence, an ideal solution should be agile and robust to bandwidth variations in real time, and mitigate

the consequences of sudden bandwidth reductions in a graceful manner.

In this paper, our objective is to challenge the conventional design of HAS systems for improved *robustness* and *efficiency* in coping with bandwidth fluctuation, while also guarding VQ and visual stability. To this end, we propose BETA– **B**andwidth-**E**fficient **T**emporal **A**daptation, which enables HAS systems to adjust quality at sub-segment granularity and to play partially received segments by exploiting temporal property. BETA is inspired by the key insight in how stalls relate to the nature of HAS delivery over reliable transport protocols like TCP and QUIC [29]. With in-order delivery of packets of each video segment, a stall happens when the tail of a segment is still being transmitted at its playback time. Hence, a stall can be prevented if the partial segment is decodable with acceptable VQ.

BETA achieves this objective by exploiting the temporal property from the encoding process and prioritizing frames so that reference frames (essential for successfully decoding temporal information) are delivered first, followed by non-reference frames (optional frames), within each segment. This new transmission order allows the client to drop the tail of the segment when needed, and simply move on to requesting the next segment. The partial segment received is still decodable and can be played, with BETA client algorithm controlling the VQ using the model based on motion activity metric [24]. BETA loss tolerance combined with the VQ control model can not only reduce or eliminate stalls, but also allow more efficient VoD streaming, by allowing players to maintain smaller buffers for improved efficiency.

Although BETA consists of both the server (encoder and packager) and client (player and decoder) components, it has a very low adoption barrier. We demonstrate that today's open-source encoders can be adapted to use BETA transmission order, while the standard Android video player can already play BETA-encoded videos in several configurations we tested, including both H.264 [42] and H.265 [40], and both with and without client-side loss-tolerance feature. Therefore, a gradual transition to BETA-enabled encoders can proceed independently from client-side player evolution to support BETA features.

Our main contributions can be summarized as follows:

- We propose a concept of sub-segment temporal adaptation, which facilitates loss-tolerant HAS based on a new importance-based transmission order of video frames that allows partial segment decoding.
- We introduce the *VQ threshold* metric that allows the video players to judiciously manage segment downloads under dynamic network conditions.
- We design, implement and evaluate a BETA prototype using the standard encoder and Android player to demonstrate the practicality and benefits to QoE and efficiency – improved video quality by up to 20%, reduced number of stalls by 20%-100% in nearly 80% of cases, and lowered wasted bandwidth by 22%-100%.
- We show that BETA can be used to reduce the set of encoding bitrates (quality levels) for a HAS stream to save on storage and encoding requirements (up to 37% for a sample bitrate set), as well as to achieve nearly continuous video quality ladder.

The rest of the paper is organized as follows. Section 2 presents related work. Section 3 reviews video coding basics and overall BETA design. Section 4 and Section 5 present the detailed designs of the server and client sides of BETA, respectively. Section 6 discusses implementation, and Section 7 performance evaluation. Finally, Section 8 concludes the paper and outlines future directions.

2 RELATED WORK

In HAS, the source video content is transcoded into different quality levels, also referred to as bitrates, variants or tracks. The quality levels are typically pre-determined according to the discrete ladder or depending on the content. Each variant is divided into small segments of typically constant duration, from 1 to 10 seconds of play time. The Adaptive BitRate (ABR) nature of HAS is based on the ability of clients to switch between variants at segment boundaries. The variant information and its segment URLs are compiled into the manifest file(s) that are shared with the client when initiating the streaming session or provided periodically for live streams. These manifests are known as Media Presentation Description (MPD) file in DASH [39] and playlists in HLS [8], the two dominant HAS protocols today. Segments are either stored in individual files or at offsets in the respective variant video files. The core of any existing HAS system is the client-controlled adaptation logic, hereinafter also referred to as *ABR logic*. ABR logic includes the quality adaptation algorithm based on network conditions as the main switching control mechanism, but may also factor in other rules and constraints, such as user settings or device capabilities.

Despite different approaches to quality adaptation [17, 25, 27, 30, 31], all algorithms strive to improve visual stability (i.e., reduce quality switches and avoid playback stalls), and provide higher average bitrate (i.e., better VQ). To cope with variable bandwidth conditions and to prevent playback stalls, a client maintains a buffer for video segments until their playback time. The buffer size varies from a few seconds to a few minutes. The client requests segments in their playback order at the quality level appropriate to the estimated bandwidth.The bandwidth is typically estimated based on the throughput that previous segments achieved. Several recently proposed algorithms employ a harmonic mean rate estimator to stabilize quality [17, 27, 30]. The buffer level may also be used to influence or control adaptation [38, 43, 45]. For example, when the buffer level is low, the client may request the lower quality of the next segment in order to fill up the buffer faster.

ABR logic can also be based on codec features (particularly, the layering features), but often implemented on the server side. The layered coding is supported in High Efficiency Video Coding (HEVC) [40], as well as in Scalable Video Coding (SVC) [37], with the latter one being more or less obsolete nowadays. The basic idea here is to encode video into multiple layers, starting from a base layer on which additional enhancement layers can be added to improve viewing experience. The layering can be defined in one of three dimensions: temporal, spatial and quality. The base layer has the minimum quality that the video can be decoded at, which must be received by the client. The enhancement layers may be transmitted for better visual quality according to bandwidth availability. SVC

supports all three dimensions of layering, and the basic HEVC only supports temporal layers. Muller et al. demonstrated that exploiting spatial layering property of SVC leads to higher average video quality [34] compared to conventional Advanced Video Coding (AVC) [42]. However, due to its computational complexity, SVC is not used in practice.

Similar to BETA, Deshpande exploited temporal layer property of HEVC to vary the frame rate subject to bandwidth availability [19]. However, the study was done on low resolution and frame rate videos, i.e., Standard Definition (SD) videos of resolution in the range of 176×144 to 480×360 with frame rates between 15 and 30 frames per second (fps). Moreover, being a server-centric approach, this cannot be easily integrated into HAS.

Recent work proposed to help adaptation by controlling playback speed and also by dropping frames, with the assumption that missing frames can be handled irrespective of their importance [23]. For example, it is assumed that the decoder can regenerate missing reference frames used to decode other frames. This work is not making use of the dependency between frames. Another proposal exploits multiple TCP paths between the server and the client to re-order video packets (according to priority) before transmission [18]. However, how exactly would the video packets be prioritized was not discussed in the work. Moreover, the proposed approach requires kernel level changes at the server and client ends, and hence has deployability and backward-compatibility challenges. Yet another work addresses the traditional TCP's inability to deliver out-of-order video packets (*i.e.,* reliability is prioritized over timeliness) in HAS by using a variant of TCP [13]. However, this requires major changes to the existing HAS in terms of HTTP request/response semantics and default ABR logic on the client, as well as kernel level modifications at server and client ends. Our work aims to fill this gap by proposing a system that exploits temporal coding property with only small and backward-compatible modifications to the server (encoder) and client side, and works along with the default ABR logic on the client.

3 BACKGROUND AND BETA DESIGN OVERVIEW

In this section, we review the video coding basics (Section 3.1), especially the importance of different frames in decoding segments, which inspired the design of BETA. The high-level design of BETA is presented in Section 3.2.

3.1 Video coding basics

In recent coding standards, such as H.264/AVC and H.265/HEVC, the sequence of source video frames (also called pictures) is converted into a coded *bitstream* and then transmitted over the network. A bitstream is structured into Network Abstraction Layer (NAL) units, which are essentially video packets that can be transmitted separately over the network. An encoded frame is placed into a single or multiple NAL units. The bitstream is converted back to pictures by the decoder before playback.

There are two types of redundancies that an encoder takes advantage of to perform video compression: spatial and temporal. Each frame in the source video is divided into blocks of pixels. The spatial redundancy refers to the intra-dependency among blocks within a frame, while the temporal redundancy refers to the inter-dependency among blocks across frames. Thus, spatial redundancies lead to intra-frame dependency encoding, while temporal redundancies are exploited to form inter-frame dependency encoding of a video. The coded bitstream (terminology used in HEVC standard) is a sequence of GOPs (Group Of Pictures), each consisting of a number of coded frames. These frames could be of type I, P or B, with the position of I-frame indicating the beginning of a GOP. One or more GOPs form a video segment in the context of HAS. Since each segment used in HAS is independently transmitted at potentially different quality levels, there must not be any coding dependency between segments, hence closed GOPs are used (with no temporal dependency across GOPs). The GOP length is fixed.[1] The GOP structure is also fixed (i.e., the relative positions of I, P and B frames does not change from one segment to another).

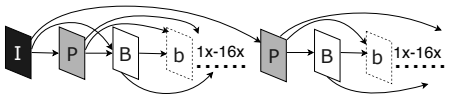

Figure 1: An example of GOP structure and frame encoding/decoding dependency

The GOP structure follows the frame decoding order. As illustrated in Figure 1, each GOP begins with an I-frame that is used to decode the P-frames and some B-frames within the GOP. Following the I-frame, there is a series of PBb* sequences. This sequences begins with a P-frame, and then a reference B-frame, followed by a sequence of non-reference B-frames (denoted by 'b'). B-frames are used for decoding other B- or b-frame(s), while b-frames are not used to decode any other frames. The number of consecutive non-reference b-frames could be anywhere between 1 and 16. In terms of temporal dependency in a closed GOP, an I-frame is independent of all other frames, while a P-frame can depend on at most one other frame, which could be either an I-frame or a P-frame. A B-frame can depend on at most two other frames, which could be either an I-frame, P-frame or B-frame. The relationship between the frames is defined by the temporal coding property.

3.2 BETA design overview

The importance of frames in encoding and decoding process inspired us to design the BETA streaming system, a new HAS system that exploits temporal adaptation of video content to achieve better robustness and efficiency in coping with bandwidth fluctuation. The specific motivating question we ask is *how can we help the client decode and play a partial segment, and discard some part of it when needed*, to avoid a stall in the case of a sudden bandwidth reduction? If that were possible, then a more general question arising is *can we dynamically adjust the quality of the segment being downloaded by manipulating the temporal property*?

To reiterate the root cause of stalls in HAS, it is necessary to download the entire segment for it to be decoded, even when some tail part of the segment must be delivered to the application after it is due to play. Only then can the client move on to request the next segment. The classic transmission (decoding) order results in critical frames possibly occupying the tail of the segment, whose delay

[1] The length of a GOP is the number of encoded frames from one I-frame to the next.

in delivery results in a stall. We note that even if multiple closed GOPs were delivered as a single segment (e.g., within fragments of a CMAF segment [1]), and if the client is able to decode the leading GOPs before the full segment is delivered, stalls may still happen, since a complete segment must still be downloaded. In addition, the buffer will be nearly or completely depleted, leading to quality down-switch for the next segment. Similar issue exists when a HAS player uses QUIC [29]. Although transport layer Head-of-Line Blocking (HoLB) is mitigated by QUIC and total delay in segment delivery can be reduced, HoLB will still exist at the segment level at the application layer, as next chunk can not be played, even if delivered, until previous one has played fully. Therefore, the ability to play the partial segment is still needed.

BETA system introduces a new approach to dynamic quality adaptation, by allowing reliable in-order delivery of a segment to terminate if needed, once the critical number of frames for decoding has been delivered. This way, the partial segment is accepted and decodable, while the tail is dropped. This approach therefore enables quality adjustment at sub-segment level. To make this possible, instead of transmitting frames of a segment in the decoding order as in Figure 1, BETA defines a new transmission order reflecting the importance of frames in the decoding process on the client side of HAS, henceforth referred to as BETA-order. In other words, frames in each segment are ordered according to their importance in the decoding process and are transmitted in this particular order to the client over HTTP.

In current H.264/H.265 coding standards, frames can be re-ordered differently from their playback order for coding efficiency purposes [22, 40, 42]. Transmission in playback order was considered using a theoretical analysis [41] for legacy codecs, but it was not investigated in the context of HAS and reliable transports. In this paper, we propose an end-to-end solution with a working prototype and analysis carried out on real mobile device and using popular open source HAS player. We use the term *frame re-ordering* to refer the proposed transmission order that enables successful download and playback of partial segments.

While the BETA order enforces the importance of frames during transmission, quality distortion at the client will become noticeable as more tail-end frames of a segment are dropped due to delay, even as the partial segment is reconstructed to its original duration. This can be perceived as either (or both) quality distortion and quality instability (appearance of a *pause or mini-stall*) at the client. To maintain quality and stability, we introduce a *VQ threshold* that defines the number of frames in each segment (in BETA-order) that must be received, so that the impact on VQ is minimized. The *VQ threshold* is determined on the server side while encoding the video, and is used by the BETA client-side algorithm to terminate the segment download. The detailed designs of the server and client sides of BETA are presented in Section 4 and Section 5, respectively.

Since the design of BETA is based on the tolerance of missing b-frames during the decoding process, it is important to verify the feasibility of this idea and the impact on VQ with real video content. More specifically, we need to answer the questions: (i) *how is VQ impacted as more b-frames are dropped*, and (ii) *how to derive the appropriate VQ threshold value?* For this reason, we conduct visual quality analysis of real videos. The results from the analysis guide several important design decisions, including the derivation of the *VQ threshold* value (Section 4.3) and per-segment timeout (Section 5.1) as well as the challenge of dropping consecutive frames (Section 4.4) and a potential opportunity to save on storage (Section 4.2).

Table 1: Characteristics of test videos

Video	Genre	Duration	MA	CSD	HTD	EHD
Aspen	Scene	19.08 s	0.68	44.98	140.71	3.46
Burn	Scene	19.08 s	0.25	41.22	142.25	3.74
Dinner	Animation	31.75 s	1.09	23.03	63.66	2.90
Life	Animation	27.58 s	0.91	32.28	95.66	3.60

Table 2: Target and actual bitrates (*Mbps*) of test videos

Quality	A	A'	B	B'	C
Target	*5.3*	*3.5*	*2.3*	*1.5*	*1.0*
Aspen	6.8	4.5	3.0	1.9	1.3
Burn	5.4	3.5	2.3	1.5	1.0
Dinner	4.6	2.9	2.0	1.2	0.9
Life	5.0	3.3	2.2	1.5	1.0

We select `ExoPlayer` as the test client, the most popular open-source media player for Android OS [5], running on a Samsung J3 device with standard codecs. We analyze four videos from the publicly available Xiph dataset [12]. The test videos are encoded in 1920×1080 spatial resolution with frame rate of 24 fps, to satisfy the device constraints (see Section 6 for more details). We also select the segment size of 1 second, each containing 1 GOP. We choose videos exhibiting both background and foreground changing over time, as well as diverse measures of basic video features, including Motion Activity (MA) [24], Color Structure Descriptor (CSD), Homogeneous Texture Descriptor (HTD), and Edge Histogram Descriptor (EHD) [32]. Table 1 shows the diverse features of test videos in their respective units, which ensure that our workload covers a wide range of video content types. All videos are coded using x265 HEVC encoder [11] in Variable BitRate (VBR) with the default rate factor value (QP) of 28. This means that the encoder would do its best to achieve the target bitrate, but the actual bitrate may sometimes exceed it. Such a situation is more relevant for live video streaming (and therefore, real-time encoding) where the encoder cannot leverage on various optimization techniques due to the limitation of few seconds of look-ahead buffer and can result in low compression efficiency.

All test videos are encoded in five quality levels at different target bitrates, as listed in Table 2. The target bitrates and the gap between adjacent ones (factor of 1.5 to 2) are chosen to cover a range suggested by popular content providers [7] and commonly used in practice [8], as well as to span the often found bandwidth fluctuations in real networks, especially cellular.

4 BETA SERVER SIDE

In this section, we present the detailed design of the server side of BETA. Although we conceptually refer to it as a *server* side for brevity, BETA actually affects the encoder, and the packager (segmenter) to generate additional meta-data for the MPD file. The server side design of BETA involves modification to the encoding process to produce video segments with frames in BETA-order (to

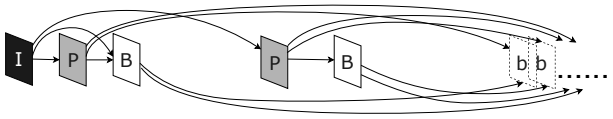

Figure 2: An example of GOP structure in BETA-order

be detailed in Section 4.1). We then discuss an interesting opportunity for storage savings with BETA (presented in Section 4.2). Based on the observations in Section 4.1 and Section 4.2, we derive the per-segment *VQ threshold* in Section 4.3. Finally, we address a challenge that comes with dropped frames in Section 4.4.

4.1 Redefining transmission order

In HAS, although delivery is reliable, delay may still occur and halt the client's decoding process. Any missing I-, P-, or (reference) B-frames will make a GOP non-decodable since they are needed to decode other frames. However, the loss of any of the (non-reference) b-frames will not impact the decodability of a GOP, but may cause degradation in visual quality (more precisely, its temporal quality). This coding dependency inspired us to redefine the transmission order so that all I-, P-, and B-frames are delivered first *to the application*, and b-frames last, as per in-order reliable transports used by the most HAS protocols (e.g., DASH [39] and HLS [8] are implemented over TCP or QUIC).

To achieve the intended importance-based frame delivery to the application layer, BETA uses this new transmission order for the frames within each HAS segment, as shown in Figure 2: (1) I-frames first, (2) P-frames and B-frames follow, and (3) b-frames last. When the client receives frames in a segment in BETA-order, it could decide to simply drop the tail of the segment and decode the GOP using a fraction of received segments. Re-ordering of frames should not prevent decoding and playback at the client end since each frame has a unique identifier (known as Picture Order Count or POC) that indicates its position in the display order.

By dropping the tail of a segment, we only compromise the temporal quality. It has been confirmed that among the three properties of video compression (temporal, spatial, and amplitude), varying temporal property has no impact on the other two, while spatial and amplitude features are inter-dependent [35]. Moreover, altering temporal property has less negative impact on video quality than altering spatial and amplitude properties do [16, 19, 35].

To verify the idea and assess the trade-off in visual quality, we examined the decodability of a video and video quality in relation to the percentage of b-frames being absent at the decoding time. We use two common metrics, Peak Signal to Noise Ratio (PSNR) and Structural SIMilarity (SSIM), for full-reference[2] VQ assessment [14, 26, 35, 44]. We remove a percentage of b-frames and replace them with the previous existing frames in the display order in the degraded video. Figure 3 shows the PSNR and SSIM of the first nine segments from one of our test videos – "Aspen". Higher values of PSNR and SSIM imply better quality. There is no theoretical upper bound for PSNR, and SSIM values are expected to be between 0 and 1.

[2]Full-reference measurement is done by comparing corresponding frames between the reference and degraded videos.

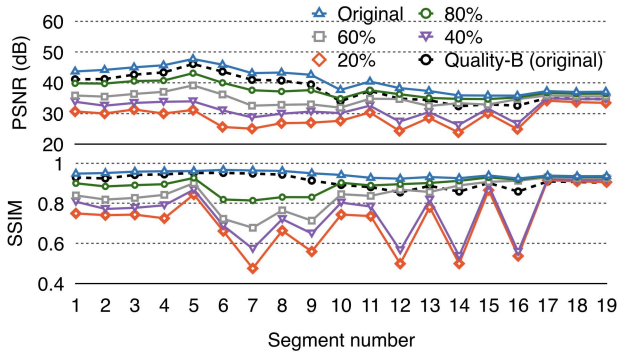

Figure 3: Impact of removing b-frames on per-segment VQ for Quality-A

We observe that removing b-frames does not affect the decodability of the video, but does leads to various levels of impact on VQ. The *impact level varies significantly between segments*. For instance, in the case of segment 3, PSNR drops from 45.02 to 31.28 as more b-frames are missing, signifying a large degradation in quality, whereas for segments 17-19, PSNR does not change much. The SSIM also shows a similar trend. These observations imply that the percentage of b-frames that can be dropped without affecting VQ should be computed per segment. Table 3 provides the average PSNR and SSIM values and the resulting overall bitrates across all segments of "Aspen" video, as different percentages of b-frames are retained in the segment. For this video, removing all b-frames reduces the top bitrate by 53% (from 6.8 to 3.2 *Mbps*), suggesting that substantial bandwidth loss could potentially be tolerated (by dropping b-frames) if VQ cost is carefully managed.

4.2 Opportunity: Reducing the bitrate set

To further analyze the impact of BETA-order on bandwith demand, we compare the average bitrate of "Aspen" video with different percentages of b-frames retained over all five quality levels, as shown in Table 4. We observe that by retaining different percentages of b-frames, the bitrate range from 6.8 *Mbps* to 0.8 *Mbps* could be effectively covered by only three quality levels (Quality-A, Quality-B and Quality-C) instead of all five, as indicated by highlighted values. Similar trend is observed across other videos as well. This implies that BETA would require less storage (by 37% for our test videos) in origin servers, CDNs and cloud DVR libraries, as well as less processing power to transcode content, while still being able to serve a wide spectrum of quality levels.

The selection of which particular levels to encode would likely be guided by the content providers' target network conditions and device capabilities, since not all levels are intended to be regularly played. Some intermediate levels are there only to offer transitions and satisfy guidelines [8]. Furthermore, this opens a possibility

Table 3: VQ and bitrate of "Aspen" video at Quality-A across different percentages of retained b-frames

	Original	80%	60%	40%	20%	0%
PSNR (dB)	40.72	37.58	34.74	31.41	28.86	27.08
SSIM	0.94	0.89	0.84	0.76	0.71	0.68
Bitrate (Mbps)	6.8	6.3	5.5	4.7	3.9	3.2

Table 4: Average bitrates (*Mbps*) resulting from different percentages of retained b-frames ("Aspen" video)

Quality levels	Original	80%	60%	40%	20%	0%
A (5.3 *Mbps*)	**6.8**	6.3	5.5	4.7	3.9	**3.2**
A' (3.5 *Mbps*)	4.6	4.2	3.7	3.2	2.7	2.3
B (2.3 *Mbps*)	**3.0**	2.8	2.5	2.2	1.9	**1.6**
B' (1.5 *Mbps*)	1.9	1.8	1.6	1.4	1.3	1.1
C (1 *Mbps*)	**1.3**	1.2	1.1	1.0	0.9	**0.8**

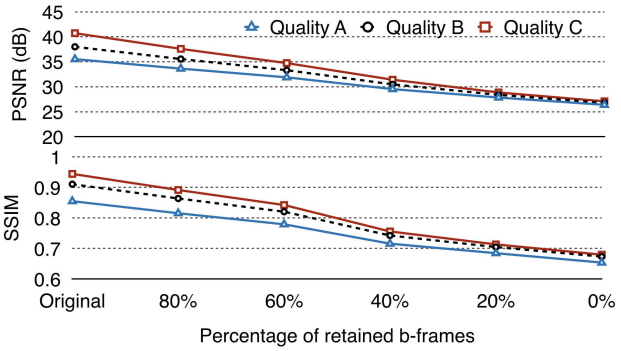

Figure 4: Average PSNR and average SSIM over different percentages of missing b-frames ("Aspen" video)

to consider creating a continuous quality ladder, as opposed to the current discrete design, simply by dynamically adjusting the number of dropped b-frames.

In Figure 4, for illustrative purposes, we analyze the average VQ coverage of Quality levels A, B, and C, as percentage of retained b-frames changes, for the whole "Aspen" video, with the simplifying assumption that VQ loss is similar across segments. The indication we get is that both Quality A and Quality B may often need 80% of b-frames to ensure higher or similar VQ than their respective next lower quality levels. Therefore, reducing the set of quality levels should be done very carefully.

4.3 *VQ threshold*

With BETA-order and the reduced set of quality levels, we need to be cautious with the VQ trade off. For example, for segment 3 of the "Aspen" video at Quality A with 6.8 *Mbps*, its PSNR measure may drop below what Quality B offers at 2.3 *Mbps* if any b-frames are dropped (Figure 3). In this case, it would be better off to stream at Quality B. In contrast, for segment 16, its PSNR measure is better than what Quality B offers even if 60% or more b-frames are received. Since scarification in visual quality can vary significantly among segments, we need to determine the acceptable VQ loss to balance visual quality (i.e., higher PSNR and SSIM) and stability (i.e., fewer stalls and switches). Since the tolerance level may vary from segment to segment, we define per-segment *VQ threshold*, f_i^{VQ} as the *percentage of b-frames that must be received to ensure equal or better visual quality than the next lower quality level.*

We conduct detailed analysis of all segments from all videos to mine correlation between visual features (MA, CSD, HTD, and EHD) and the *VQ threshold*, based on which we propose a model for deriving the appropriate f_i^{VQ} value based on video properties. We examine PSNR and SSIM over various percentages of b-frames

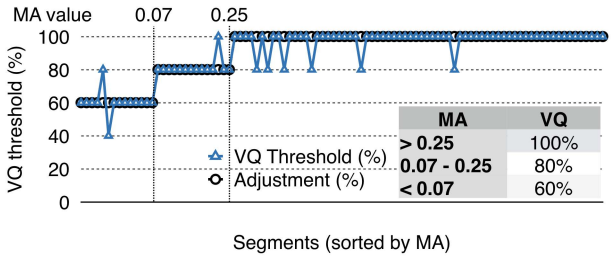

Figure 5: Per-segment *VQ threshold* vs. motion activity

being removed from a segment. We find that MA has a strong correlation with f_i^{VQ}. Figure 5 shows the per-segment *VQ threshold* from all four test videos sorted by MA values (0.002 - 3.49). We observe clustering of the f_i^{VQ} values with very few noisy data points. After manually adjusting most noisy data points to a higher level (i.e., more stringent tolerance to losses), we arrive at a more clear clustering. The conclusion is that at least 60% of b-frames should be retained to preserve VQ across a range of MA values, and that for most segments with high MA, any drop of frames impacts VQ. In addition, the clustering suggests that a simple mapping function can be applied to deciding the *VQ threshold* per segment based on its MA value, as shown in the table in Figure 5.

The per-segment f_i^{VQ} values are included in the modified MPD file, causing no harm to the standard operation of ABR logic, while providing the information to the client for deciding how to trade off VQ with stall prevention.

4.4 Challenge: Dropping sequences of b-frames

As shown in Figure 1, in each PBb* sequence, there may be up to 16 consecutive (non-reference) b-frames, and as shown in Figure 2, BETA-order has all sequences of consecutive b-frames at the end of the segment in their decoding order. Though the playback order of P- and B-frames is different from the decoding order, the b-frames are displayed in the decoding order. Hence, dropping the tail-end of a segment implies dropping up to 16 consecutive b-frames in the playback order. Thus, the *maximum time* a replicated previous frame will be displayed is $16/R$ seconds, where R is the frame rate. This is 0.67 s for 24 fps videos, 0.27 s for 60 fps videos, and only 0.13 s for 120 fps videos. In other words, the higher the frame rate, the less noticeable the "pause", even if all b-frames are dropped.

However, to more precisely characterize and mitigate the risk of having perceivable pauses or mini-stalls, we measure the sequence length of consecutive b-frames in playback order for each segment of our testing videos. The empirical CDF of the distribution of b* sequence lengths is shown in Figure 6. Across all segments of all videos, more than 60% of the b-frame sequences have the length of 14 frames or less, which means the expected length of pauses or mini-stalls is much less than $14/24 = 0.58$ s for 24 fps videos. To further reduce the length of such pauses, BETA server side can randomly shuffle the b-frames at the end of the BETA-ordered segment to reduce the chance of consecutive b-frames being dropped in the segment tail. The intuition for the random shuffle is to scatter the missing frames throughout the playback order of the segment, allowing better motion continuity.

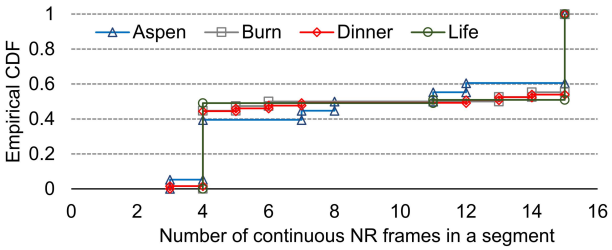

Figure 6: CDF of b-frame sequence length across all videos

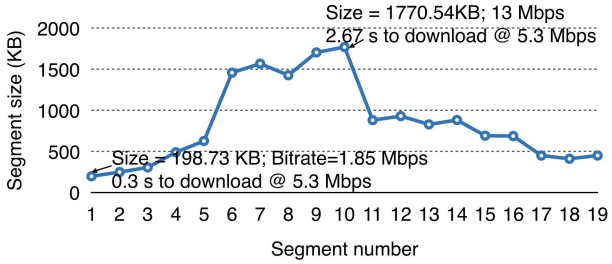

Figure 7: Variable segment size as a result of VBR encoding

5 BETA CLIENT SIDE

With BETA-order, the HAS system is less dependent on bandwidth estimation accuracy. BETA clients actively monitor the progress of the transmission of a segment, which is already a part of some players [4], and perform agile tuning according to the progress, buffer level, and bandwidth estimation. BETA is more robust to unforeseeable network disturbances and mitigates stalls without sacrificing overall VQ and sometimes even improves it. To take advantage of BETA-order and to balance visual quality and stability, clients need to: (1) know the *VQ threshold* for each segment, (2) have the BETA quality management algorithm that controls the appropriate actions at sub-segment granularity, (3) monitor the progress of the downloads, so it can drop and replace the current segment as a last resort, and (4) be able to play the partial segment.

The mechanism to monitor, drop and replace the segment is a client-side feature that can be implemented in the HAS player independently, or as a part of BETA algorithm. Moreover, BETA algorithm can be implemented as an addition to any ABR logic in the video player to add robustness at per-segment level. For the rest of this section, we introduce the per-segment timeout to exploit the variable segment size in VBR coding in Section 5.1, detail the BETA quality management algorithm in Section 5.2, and then discuss the mechanism for playing partial segments in Section 5.3.

5.1 Per-segment timeout

With VBR, the size of each segment varies according to visual complexity, even though all segments represent the same playback duration. Figure 7 illustrates VBR behavior for "Aspen" video, coded at target bitrate of 5.3 *Mbps* with segment duration of 1 *s*. The per-segment bitrate varies between 1.85 and 13 *Mbps*, and the transmission time varies from 0.3 to 2.67 seconds if 5.3 *Mbps* is the average available bandwidth. Similar trend is observed for other videos, though the extent of variation differs across videos. This pattern is also seen in publicly available HAS datasets [3, 39].

Hence, we define the per-segment timeout t_i that is calculated based on the segment size (s_i), which can be obtained from the HTTP response header or MPD file, and the estimated bandwidth (b_i) at the time the segment is to be transmitted, e.g., $t_i = \frac{s_i}{b_i}$. The t_i indicates when the segment transmission is past the expected completion time, according to the recent throughput measurement.

5.2 BETA quality management algorithm

The BETA client algorithm is a mechanism to take advantage of the importance-based transmission order in the video player and it operates as an additional feature to the existing ABR logic running in that player. In our design, BETA algorithm does not determine or influence the bitrate of the next segment – that is the job of the ABR logic. However, an alternative design where two algorithms interact is certainly possible, but left for future work. The main purpose of the BETA algorithm is to decide if there is a need to drop the tail of the segment and whether it is possible to do so. This is achieved by comparing the download progress with timeout parameters and *VQ threshold* for possible quality tuning.

As illustrated in Figure 8, the algorithm requests each segment i at the quality level q_i, as decided by the ABR logic. If the segment is received before the per-segment timeout t_i, no action is taken by the BETA algorithm and the player simply moves on to request the next segment in the usual manner. Otherwise, the actual bandwidth was lower than the estimated bandwidth, causing partial delivery of the segment by timeout t_i. There are three possible cases to consider depending on the number of received frames (denoted by f_i), and their relation to the number of essential reference frames (I-, P-, and B-frames) in the segment (denoted by f_i^{min}) and the per-segment *VQ threshold* (f_i^{VQ}) defined in Section 4.3.

- $f_i < f_i^{min}$: In case the number of frames received is less than the number of reference frames, the segment is not decodable. We have two choices here. We can wait if the buffer level l is healthy, i.e., $l > l^{min}$, where l^{min} is the minimum buffer level. Otherwise, we abort the transmission (if the segment being downloaded is not of the lowest quality level available) and re-transmit the *same* segment at the lowest available quality level, as a last resort decision.
- $f_i^{min} \leq f_i < f_i^{VQ}$: In case where all reference frames are received but some b-frames are still missing, the segment is decodable but with a noticeable loss in visual quality. We have two choices here as well. We can wait if the buffer level is healthy. Otherwise, we abort the transmission (and accept the segment) and allow the player to move on to request the *next* segment at the quality level according to its ABR logic.
- $f_i \geq f_i^{VQ}$: In case that sufficient number of frames are received to display with the expected visual quality, we abort the transmission (and accept the segment) and move on to request the *next* segment.

For the first two cases, there is a maximum timeout t^{max} for the wait. When t^{max} is triggered, if the number of frames successfully received is still less than the number of reference frames (*i.e.*, $f_i < f_i^{min}$), we abort the transmission and re-transmit the *same* segment at the lowest available quality level (unless the segment downloaded is already of the lowest quality). Otherwise, we simply accept the current segment, and move on to request the next segment as per

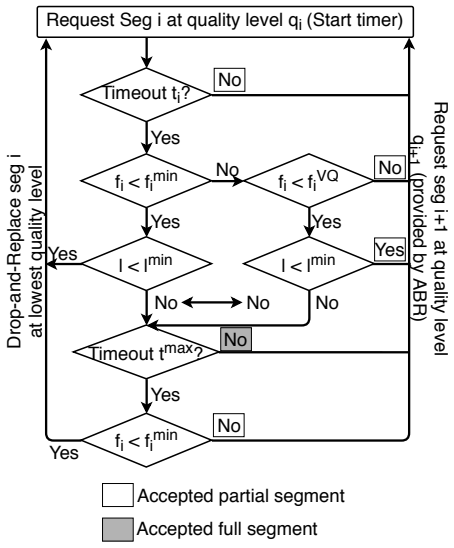

Figure 8: BETA **quality management algorithm**

player's ABR logic. Compared to the conventional behavior of HAS clients, which typically persist in downloading a full segment even if that will result in a stall, BETA offers the ability to the player to move along between consecutive segments faster, in case of delayed delivery. With this design, it is expected that BETA will improve visual quality and stability throughout the streaming session.

5.3 Playback of partial segments

Partial segments accepted by the player have fewer frames than the original segments, thus, the segment is shorter and would play out sooner, since each frame represents the same duration. Therefore, we need to reconstruct the original segment duration by filling in the missing frames. From our empirical analysis and `ExoPlayer` implementation, we discovered that some of today's decoders for both AVC and HEVC as well as Android's renderer have the capability to restore segments with missing frames. For example, `ExoPlayer` plays each frame according to the release time provided by the decoder. Thus, for the duration of each missing b-frame, the preceding received frame is displayed to fill the time.

6 IMPLEMENTATION OF BETA

We implement BETA using the real server and client. For the server side, we use `FFmpeg` [6] with x265 HEVC encoder [11] for encoding videos with frames in BETA transmission order, and `MP4box` [10] for encapsulating the encoded video in a container format (e.g., a series of MP4 segments) and generating the MPD file. We implement the client side in `ExoPlayer` 2.4.2, the de-facto standard media player for Android [5]. The rest of this section details the modification made to realize BETA. We implement BETA for both AVC and HEVC to verify that BETA can support both current codec standards. In this paper, we present results using HEVC only.

6.1 Creating and serving BETA-ordered videos

The videos are encoded into HEVC format using x265 encoder by providing appropriate run-time arguments to ensure no decoding dependency among segments (i.e., all closed GOPs), fixed GOP size in terms of playback time, and no more than 16 non-reference b-frames to produce GOP structure defined in Figure 1.

To generate segments with frames in BETA order, we modify the `raw.cpp` file that is part of the `output` module in x265 encoder. There are some practical issues to address to make the modification work in our test setup, related to the bit depth limit and meta-data update for BETA order.

Bit depth limit. The x265 hardware decoder in Samsung J3 phone does not support videos with bit depth greater than 8. Since the original videos are all using 12-bit depth (I422), this causes playback disturbances in `ExoPlayer`. Hence, all source videos are first converted to 8-bit depth (I420) using `FFmpeg` before further processing.

Meta-data update. Some meta-data attributes in the Sequence Parameter Set (SPS) and Video Parameter Set (VPS) of the video stream[3] provide information about the nature of re-ordering of frames, buffering requirement, and initial output delay for the decoder at the client (player) end to successfully arrange frames in a segment back to the playback order. These attributes must be updated to reflect the BETA order to ensure proper decoding and playback of the video. In the case of HEVC encoder, this means changing the values of `numReorderPics`, `maxDecPicBufferring` and `maxLatencyIncrease` in the SPS and VPS according to the video frame rate and segment size. The `numReorderPics` specifies the number of frames that will appear before a frame in the receiving order, but have to be displayed after the frame in the playback order. The `maxDecPicBufferring` specifies the size of the decoding buffer, called Decoder Picture Buffer (DPB), which should be large enough to reorder frames that arrive in BETA order. The `maxLatencyIncrease` specifies the initial output delay for the DPB. For example, for the 24 fps video and segment size of 1 second, the values are 5, 16 and 30, respectively.

The encoder also computes the *VQ threshold* for each segment. The video is then passed through `MP4box` to encapsulate the encoded video segments in a standard container format (MP4) and create the MPD file. Since HAS uses multiple quality levels (bitrates), the video encoding process is repeated for each quality level.

BETA-ordered videos, as any typical HAS content, are served using a commodity HTTP server. The HTTP server provides the MPD file (in XML format) and video segments to clients, which obtain MPD URL using application-specific mechanisms. The key requirement in BETA is that the MPD file include *VQ threshold* for each segment, so that the client-side BETA algorithm can work. The size of the segment is obtained from the `Content-Length` HTTP response header. Therefore, we develop a module that inserts per-segment *VQ threshold* into the `SegmentURL` elements of the MPD file produced by `MP4box`. *VQ threshold* is calculated for each segment based on the MA value from the modified encoder and reference table in Figure 5. A sample entry for this MPD element is as follows:

```
<SegmentURL media="segment_1.m4s" VQThreshold="0.8" />
```

The Android's `XmlPullParser` interface used by `ExoPlayer` reads attributes of each element in an XML file by specifying the attribute name (e.g., `VQThreshold`). Therefore, a default `ExoPlayer` that does not read `VQThreshold` could also successfully parse the modified MPD file. Moreover, for our implementation, we also observe that available decoders are able to reorder frames from

[3] Transmitted as Non-Video Coding Layer (non-VCL) NAL units

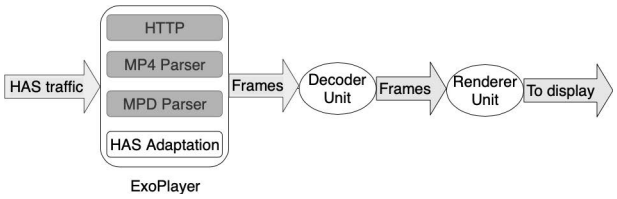

Figure 9: `ExoPlayer` on the client side. Modules in grey boxes are modified for BETA

BETA order to playback order without any issues. Hence, we do not foresee any adoption barrier for BETA as players which are not BETA-capable could still read the modified MPD file, stream and play BETA-enabled HAS videos. Further, with active research in the direction of faster feature extraction [28], we do not expect a bottleneck in *VQ threshold* computation at the encoding stage.

6.2 BETA in `ExoPlayer`

As shown in Figure 9, there are four main modules in `ExoPlayer` for HAS streaming, namely HTTP, MPD parser, MP4 parser and HAS adaptation. The HTTP module handles the HTTP connection between the player and the server, and downloads each segment, whose quality level is determined by the ABR logic. The ABR logic uses an internal bandwidth estimator to compute expected throughput, and decide on the quality of the next segment, taking into account the buffer level. The MPD parser processes the MPD file at the beginning of a streaming session (or periodically for live streaming) and records the properties of each segment for all supported quality levels. The MP4 parser unpacks the container for each segment and passes individual segments to the decoding unit. The segment is then decoded, rendered, and eventually displayed at playback time. The decoding, rendering and playback units are parts of the Android operating system.

To extend `ExoPlayer` to support BETA, we modify the MPD parser, MP4 parser, and HTTP module. The key `ExoPlayer` classes for these are `DashManifestParser`, `FragmentedMp4Extractor` and `HttpDataSource`, respectively, though we also made appropriate changes in a wider set of other classes to incorporate the BETA extensions. The MPD parser is modified to extract the *VQ threshold* for each segment. The HTTP module is extended to implement the robust BETA quality management algorithm presented in Figure 8, using supplied *VQ threshold*. When required to drop the tail of a segment, this module *terminates the connection* and delivers the segment upstream as long as at least the minimum number of frames required to successfully decode the segment are received. In order to do this, we extend the MP4 Parser module to obtain information about the size of each VCL NAL unit,[4] type of the VCL NAL unit (i.e., whether it carries a reference frame or a non-reference frame) and to track the number of b-frames read so that the algorithm knows when it is possible to cut off the segment if required. Further, we enable segment drop in the HTTP module so that segments which timed out during transmission could be dropped and replaced with the lowest quality available[5].

We inspect several devices and Android versions for handling decoding and rendering of BETA-ordered frames. We find that in the case of HEVC, BETA-ordered videos could be played on Android 7.0 and above, while for AVC, they could be played on Android 6.0 and above[6]. This implies that the underlying decoder[7] is able to successfully reorder BETA-ordered frames to playback order, and the video renderer unit, which is a part of Android, is able to replace missing frames with previous frames. Upon detailed inspection, we find that the decoder supplies display time of each decoded frame that it outputs to the renderer, and the renderer uses this information to display a frame on the screen. So, if there is a missing frame, renderer simply does not refresh the screen with the next frame until its display time.

However, we also observe that due to underlying Android and/or hardware limitations, we can only test videos with segment size 1 *s*. For longer segments, more buffering and processing power (for reordering frames) is required. For example, for 2 *s* segments, some frames are dropped by the renderer since the decoder took too much time to process and output these frames. We also observe that some of the software AVC decoders on Android 6.x and above regenerate missing frames depending on how much time is left to display them. To investigate the impact of frame regeneration, we also develop a *frame writing* module that sits between the *decoder* and *renderer* units (Figure 9) by making modifications to the `ExoPlayerMediaCodecVideoRenderer` class. This module collects the frames that are output by the decoder, writes them to a file, and then passes them to the renderer for display. The analysis (omitted for lack of space) of the saved decoded frames shows higher video quality (in terms of PSNR and SSIM) for segments with regenerated frames compared to those where missing frames are simply replaced by previous frames. Hence, we are confident that with more powerful hardware, performance improvements of BETA in Section 7 will represent a lower bound and BETA-ordered videos with longer segments could eventually be played on mobile devices.

7 PERFORMANCE EVALUATION OF BETA

In this section, we present the performance analysis of BETA based on the real implementation described in Section 6. To verify and understand the advantages of BETA, we deploy the implementation in an emulated network – a controlled environment that allows us to tune the bandwidth and record the performance of BETA without interference from background traffic. In this setting, the server and client communicate through a WIFI router with no other devices connected to the router. The bandwidth available between the server and client is controlled by Linux `tc`. To emulate the bandwidth variability found on dynamic wireless links, such as in cellular LTE networks, we use a synthetic trace where bandwidth changes every second following a normal distribution with the mean of 4 *Mbps*. This mean is chosen because it is below the highest bitrate of all test videos. We chose 2 *Mbps* and 4 *Mbps* as the standard deviations of the normal distribution to simulate moderate and extreme bandwidth fluctuation, respectively, according to recent work [21]. The goal is that bandwidth fluctuation crosses all quality levels and challenges the ABR logic with possible stalls.

[4] NAL unit that contains the encoded video information.
[5] This feature applies to any ABR logic in `ExoPlayer` and can be enabled as desired.
[6] Unmodified `ExoPlayer` version is found to successfully play videos (AVC/HEVC) only on Android 6.x and above, and is found not supportive on Android 5.x and below.
[7] All experiments in Section 7 use hardware decoder – default selection by `ExoPlayer`.

Table 5: Target and actual bitrates (*Kbps*) for the full set

Quality	A	A'	B	B'	C	C'	D	D'	E
Target	*5300*	*3500*	*2300*	*1500*	*1000*	*670*	*450*	*300*	*200*
Aspen	6787	4563	2987	1947	1316	886	594	413	345
Burn	5512	3553	2295	1506	980	632	412	288	288
Dinner	4781	3149	2048	1325	890	611	418	281	194
Life	5166	3427	2284	1513	1008	675	439	342	352

The four test videos (Table 1) are available as the *full set* of bitrates, as listed in Table 5. We also introduce the *reduced set* of bitrates, including only alternate ones from the full set – Quality A, B, C, D, and E, to evaluate this opportunity with BETA. We evaluate BETA with the default `ExoPlayer` 2.4.2 ABR logic on Samsung J3 phone with Android 7.0. We refer to the basic `ExoPlayer` as the *Baseline*, and compare it to the BETA-enabled `ExoPlayer` referred to as *BETA*.

The player is configured with the buffer size of 6 seconds (i.e., six 1-second segments at most), and videos are streamed in VoD fashion. The minimum buffer level l^{min} is set to 2 seconds (i.e., at least two 1-second segments). The segment transmissions time out when the buffer level is below the minimum threshold l^{min}. We set t^{max} to trigger when minimum buffer level is reached. If the transmission of a segment takes longer than the maximum timeout t^{max}, both players (*Baseline* and *BETA*) abort the transmission and re-transmit the segment at the lowest quality level. If the segment is already transmitted at the lowest quality level, the timeout has no effect and the transmission continues until the playback time. The per-segment VQ threshold f_i^{VQ} is set according to the results presented in Figure 5.

The performance analysis is carried out by measuring the common QoE metrics: VQ using PSNR and SSIM, stability using number and duration of stalls, as well as the efficiency using the number of segment replacements and amount of data wasted due to it.

We first illustrate the differences in how *Baseline* and *BETA* players manage segments of the "Dinner" video (for the first 20 segments) under moderate bandwidth fluctuation, and then present overall results. Figure 10 illustrates a how *BETA* can improve VQ. For segments 4, 5, 6, 8 and 9, *BETA* accepts partial segments and maintains higher average segment quality than *Baseline*, which would drop and replace segments with the lowest quality in these cases. Although *BETA* tries to maintain the quality threshold, it has to accept a partial segment below *VQ threshold* if the buffer level is low. However, *BETA* still offers better quality than *Baseline* for most of the partial segments, as well as the future ones.

In Figure 11, we see which corresponding bitrates were originally requested by *Baseline* and *BETA* in the same scenario. BETA avoids drop-and-replace by accepting partial segments in all cases, and keeps requesting the same bitrate of 3.5 *Mbps* (which is just below the mean bandwidth of 4 *Mbps*), while *Baseline* has to drop and replace with the lowest quality.

Overall results that follow are presented as percentages of improvement when using BETA. The actual QoE metrics are in the range of 32-38 for PSNR, 0.70-0.91 for SSIM, and 5.4-10.5 seconds of stall duration with majority of segments requiring replacement or BETA action, across videos under moderate bandwidth fluctuation. While stall values may seem uncommonly high, we stress that our test setup is intentionally very challenging. This includes

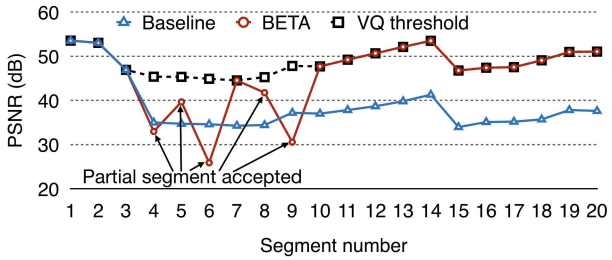

Figure 10: Per-segment VQ – *Baseline* vs. *BETA*

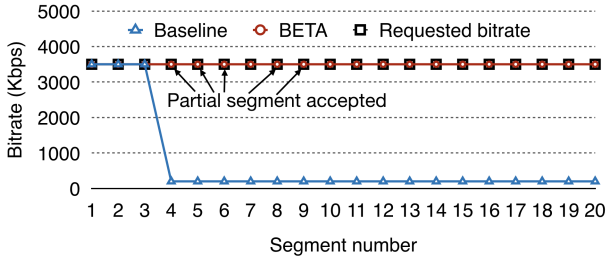

Figure 11: Per-segment bitrate – *Baseline* vs. *BETA*

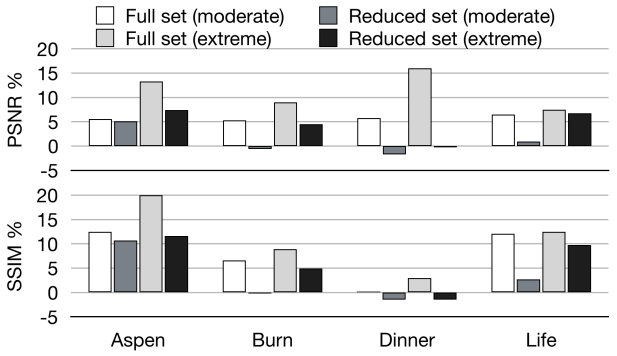

Figure 12: Improvement in VQ where both *Baseline* and *BETA* use the same bitrate set and bandwidth setting

the extremely short buffer settings and segment size on a dynamic network, to match low-latency live streaming primarily. However, the approach also works well for VoD, especially where very fast startup is desired, or a shallow buffer is used by the player.

7.1 Visual Quality

The PSNR/SSIM computation as described in Section 4.1 is used to create a look-up table for each video with per-segment PSNR/SSIM values for different percentage of retained b-frames. During playback, if *BETA* has to accept a partial segment, the percentage of b-frames received is recorded and PSNR/SSIM of the segment is computed offline to measure VQ. Overall, VQ is improved up to 20% with *BETA* compared to *Baseline* across all videos, bandwidth conditions (moderate and extreme fluctuation) and both bitrate sets (full and reduced). Figure 12 shows the comparison where both *Baseline* and *BETA* use the same bitrate set. Around 88% of cases result in VQ improvement with BETA, with a small number of cases resulting in the same VQ or negligible reduction (up to -1.7%) due to accepting partial segments. In most cases, improvement is in the range of 5% to 20%.

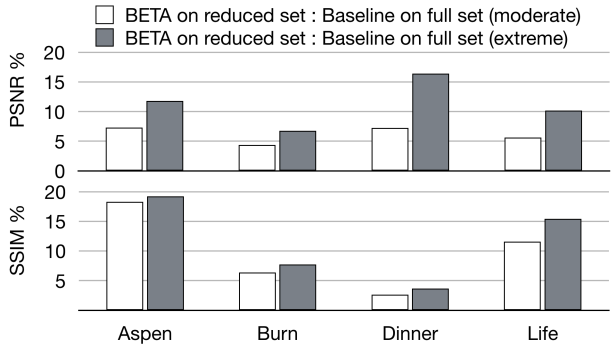

Figure 13: Improvement in PSNR and SSIM when *BETA* uses the reduced set and *Baseline* uses the full set

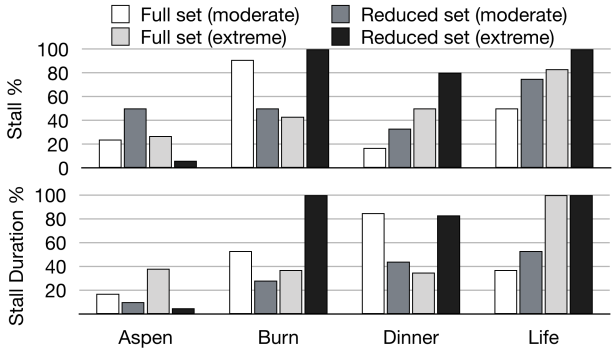

Figure 14: Improvement in visual stability (stall reduction) where both *Baseline* and *BETA* use the same bitrate set

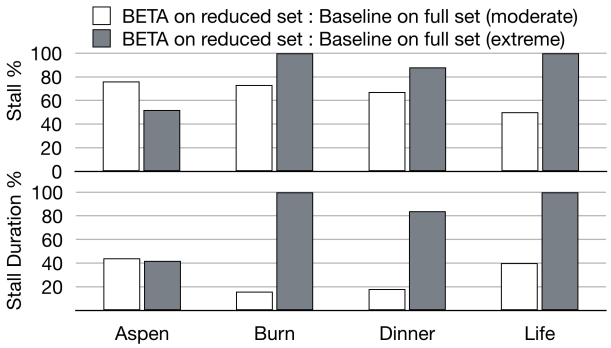

Figure 15: Improvement in visual stability (stall reduction) when *BETA* uses the reduced set and *Baseline* uses the full set

In the case of the reduced quality set, there are fewer instances of partial segments being accepted since the bitrates are further apart, i.e., the next bitrate below the mean value (4 *Mbps*) is 2.3 *Mbps* while for the full quality set it is 3.5 *Mbps*. Hence, in the former case if the segment times out, there is a greater chance that the minimum number of frames required for proper decoding of the segment is not received, resulting in the segment drop and re-transmission for both *Baseline* and *BETA*. This results in smaller VQ improvements for BETA with the reduced quality set.

We also examine the improvement when *BETA* uses the reduced set and *Baseline* uses the full set, as a possible deployment configuration. Figure 13 shows improvement in all cases between 3% to 19%, indicating that *BETA* is equally good for full and reduced sets of quality levels. In all configurations, improvement is similar or higher with extreme bandwidth fluctuation, as desired of BETA.

7.2 Visual Stability

Visual stability (or smoothness) of a playback is reflected primarily by the number of stalls and also by the number of quality switches. We focus on stall reduction as the critical metric that typically always outweighs switching, and we measure the number of stalls as well as the duration of each stall. As expected of BETA, significant improvement of 6% to 100% is seen across all cases, where 100% indicates elimination of stalls. Figure 14 shows that stall reduction exceeds 20% in 92% of cases for number of stalls, and in 79% of cases for stall duration. While enabling BETA intuitively results in lower number of stalls (due to acceptance of partial segments), the time saved by avoiding full download is utilized for subsequent segments during the video session. This contributes to the high improvement with the reduced bitrate set under extreme bandwidth variation for most videos, as the bitrate difference between the mean bandwidth (4 *Mbps*) and the closest lower quality level (2.3 *Mbps*) is higher than for the full quality set (3.5 *Mbps*).

In the specific case of *BETA* using the reduced set and *Baseline* using the full set, improvements are generally higher, with number of stalls reduced from 50% to 100% (Figure 15). Stall duration is also significantly reduced. The number of bitrate switches, other than segment replacements (content of a segment is played out only after its successful download), remains roughly the same for *Baseline* and *BETA*, and hence is not reported here.

7.3 Bandwidth Efficiency

The bandwidth efficiency is measured by the number of segment replacements and resulting volume of data wasted. Since BETA minimizes replacements by tolerating frame losses, it effectively reduces the data wasted due to abandoned segments. The improvement levels are shown in Figure 16 and range from 22% to 100%. BETA minimizes segment re-transmissions in two ways; (1) in the most obvious manner by accepting partial segment, as in the case of segments 4, 5, 6, 8 and 9 in Figure 11, and (2) by saving time to download future segments due to acceptance of partial past segments, as in the case of segments 7 and 10–20 in Figure 11. Finally, in the special case of *BETA* using the reduced bitrate set and *Baseline* using the full set, efficiency is improved in the range from 68% to 100% (Figure 17).

8 CONCLUSION

We presented the design, implementation, and evaluation of BETA, a bandwidth-efficient system that dynamically tunes quality and mitigates stalls in HAS using temporal adaptation. We outlined several practical and backward-compatible changes to the server (encoder and packager) and client (video player) sides of a HAS system to realize BETA, and demonstrated its uses for QoE improvement, efficiency and reducing storage and processing requirements. We leave several aspects of BETA design for future work, such as exploring interactions with different adaptation algorithms and in real networks, generating optimal bitrate sets, decoder improvements and other system and algorithmic aspects.

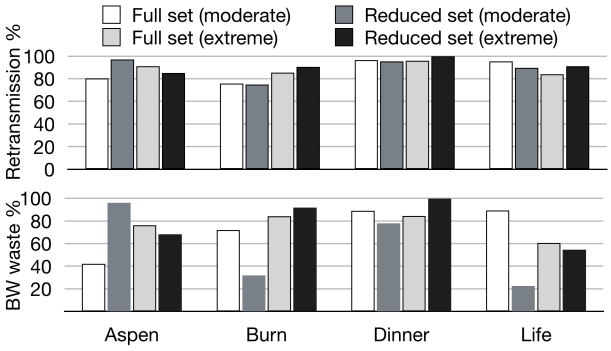

Figure 16: Improvement in bandwidth efficiency (waste reduction) where both *Baseline* and *BETA* use the same bitrate set

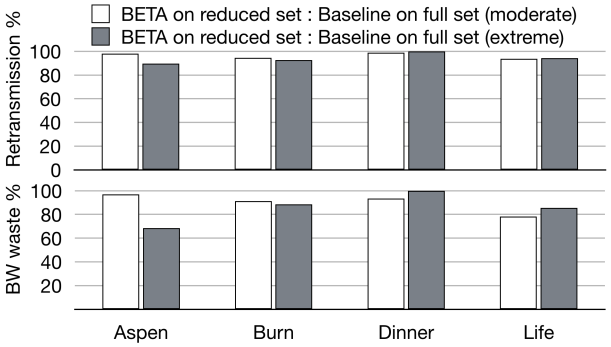

Figure 17: Improvement in bandwidth efficiency (waste reduction) when *BETA* uses the reduced set and *Baseline* uses the full set

Acknowledgment. This work is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] 2018. CMAF. https://www.theoplayer.com/blog/low-latency-chunked-cmaf. (Nov. 2018).
[2] 2018. Conviva Viewer Experience Report. https://www.conviva.com/research/. (Nov. 2018).
[3] 2018. DASH Dataset. http://www-itec.uni-klu.ac.at/dash/?page_id=207. (Nov. 2018).
[4] 2018. dash.js. https://github.com/Dash-Industry-Forum/dash.js. (Nov. 2018).
[5] 2018. ExoPlayer. http://google.github.io/ExoPlayer/. (Nov. 2018).
[6] 2018. FFmpeg: Multimedia framework to decode, encode, transcode, mux, demux, stream, filter and play. https://www.ffmpeg.org/. (Nov. 2018).
[7] 2018. Google Recommended Upload Encode Settings. https://support.google.com/youtube/answer/1722171?hl=en. (Nov. 2018).
[8] 2018. HTTP Live Streaming. https://developer.apple.com/streaming/. (Nov. 2018).
[9] 2018. Limelight Networks: Multiple Solutions for Low-Latency Live Video Streaming. https://www.limelight.com/blog/multiple-solutions-for-low-latency-live-video-streaming/. (Nov. 2018).
[10] 2018. MP4Box: A Multimedia Packager. https://gpac.wp.imt.fr/mp4box/. (Nov. 2018).
[11] 2018. x265 HEVC Video Codec. http://x265.org/. (Nov. 2018).
[12] 2018. Xiph Dataset (Derf's collection). https://media.xiph.org/. (Nov. 2018).
[13] S. Ahsan, S. McQuistin, C. Perkins, and J. Ott. 2018. DASHing towards Hollywood. In *MMSys '18*.
[14] L. Anegekuh, L. Sun, E. Jammeh, I. Mkwawa, and E. Ifeachor. 2015. Content-Based Video Quality Prediction for HEVC Encoded Videos Streamed Over Packet Networks. *IEEE Transactions on Multimedia* 10, 8 (June 2015), 1323–1334.
[15] I. Ayad, Y. Im, E. Keller, and S. Ha. 2018. A Practical Evaluation of Rate Adaptation Algorithms in HTTP-based Adaptive Streaming. *Computer Networks* 133 (March 2018), 90–103.
[16] S. Buchinger and H. Hlavacs. 2006. Subjective quality of mobile MPEG-4 videos with different frame rates. *Journal of Mobile Multimedia* 1, 4 (2006), 327–341.
[17] L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo. 2013. ELASTIC: A Client-Side Controller for Dynamic Adaptive Streaming over HTTP (DASH). In *20th International PV Workshop*. 1–8.
[18] X. Corbillon, R. Aparicio-Pardo, N. Kuhn, G. Texier, and G. Simon. 2016. Cross-layer Scheduler for Video Streaming over MPTCP. In *MMSys '16*.
[19] S. Deshpande. 2013. Adaptive HTTP Streaming Utilizing Temporal Sub-layers of High Efficiency Vide Coding (HEVC). In *IEEE ISM*. 384–390.
[20] F. Dobrian et al. 2011. Understanding the impact of video quality on user engagement. In *ACM SIGCOMM*. 362–373.
[21] H. Du, Q. Zheng, W. Zhang, and X. Gao. 2017. A Bandwidth Variation Pattern-Differentiated Rate Adaptation for HTTP Adaptive Streaming Over an LTE Cellular Network. *IEEE Access* 6 (Dec. 2017), 9554 – 9569.
[22] D. L. Gall. 1991. MPEG: A Video Compression Standard for Multimedia Applications. *ACM Communications* 34, 4 (1991), 46–58.
[23] J. He, M. A. Qureshi, L. Qiu, J. Li, F. Li, and L. Han. 2018. Favor: Fine-Grained Video Rate Adaptation. In *ACM MMSys*. 64–75.
[24] P. I. Hosur and K.-K. Ma. 1999. Motion Vector Field Adaptive Fast Motion Estimation. In *ICICS '99*. 234–246.
[25] D. Jarnikov and T. Ozcelebi. 2010. Client Intelligence for Adaptive Streaming Solutions. In *IEEE ICME*. 1499–1504.
[26] A. Javdtalab, M. Omidyeganeh, S. Shirmohammandi, and M. Hosseini. 2011. A rate control algorithm for X264 high definition video conferencing. In *IEEE ICME*.
[27] J. Jiang, V. Sekar, and H. Zhang. 2014. Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming With FESTIVE. *IEEE/ACM Transactions on Networking* 22, 1 (2014).
[28] V. Kantorov and I. Laptev. 2014. Efficient feature extraction, encoding and classification for action recognition. In *IEEE CVPR*.
[29] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, et al. 2017. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *ACM SIGCOMM*. 183–196.
[30] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran. 2014. Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale. *IEEE Journal on Selected Areas in Communications* 32, 4 (April 2014), 719–733.
[31] J. Lievens, S. Satti, N. Deligiannis, P. Schelkens, and A. Munteanu. 2013. Optimized Segmentation of H.264/AVC Video for HTTP Adaptive Streaming. In *IFIP/IEEE IM*. 1312–1317.
[32] B. S. Manjunath, J. R. Ohm, V. V. Vasudevan, and A. Yamada. 2001. Color and Texture Descriptors. *IEEE Trans. Cir. and Sys. for Video Technol.* 11, 6 (June 2001), 703–715.
[33] C. Muller, S. Lederer, and C. Timmerer. 2012. An Evaluation of Dynamic Adaptive Streaming over HTTP in Vehicular Environments. In *ACM MoVid*. 37–42.
[34] C. Muller, D. Renzi, S. Lederer, S. Battista, and C. Timmerer. 2012. Using Scalable Video Coding For Dynamic Adaptive Streaming Over HTTP in Mobile Environment. In *EUSIPCO*. 2208–2212.
[35] Y. Ou, Y. Xue, and Y. Wang. 2014. Q-STAR: A perceptual video quality model considering impact of spatial, temporal, and amplitude resolutions. *IEEE Transactions of Image Processing* 23, 6 (2014), 2473–2486.
[36] Y. Sani, A. Mauthe, and C. Edwards. 2017. Adaptive Bitrate Selection: A Survey. *IEEE Communications Surveys and Tutorials* 19, 4 (July 2017), 2985–3014.
[37] T. Schierl, T. Stockhammer, and T. Wiegand. 2007. Mobile Video Transmission Using Scalable Video Coding. *IEEE Transactions on Circuits and Systems for Video Technology* 17, 9 (2007), 1204–1217.
[38] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. 2016. BOLA: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM*.
[39] T. Stockhammer. 2011. Dynamic adaptive streaming over HTTP –: standards and design principles. In *ACM MMSys*. 133–144.
[40] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand. 2012. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuit and Systems for Video Technology* 22, 12 (2012).
[41] S. Wee, W. Tan, Apostolopoulos J., and M. Etoh. 2002. Optimized video streaming for networks with varying delay. In *IEEE ICME*.
[42] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. 2003. Overview of the H.264/AVC Video Coding Standard. *IEE Transactions on Circuit and Systems for Video Technology* 13, 7 (2003).
[43] A. H. Zahran, J. Quinlan, D. Raca, C. J. Sreenan, E. Halepovic, R. K. Sinha, R. Jana, and V. Gopalakrishnan. 2016. OSCAR: An Optimized Stall-cautious Adaptive Bitrate Streaming Algorithm for Mobile Networks. In *8th MoVid Workshop*.
[44] T. Zinner, O. Hohlfeld, O. Abboud, and T. Hossfeld. 2010. Impact of frame rate and resolution on objective QoE metrics. In *IEEE QoMEX Workshop*. 29–34.
[45] X. K. Zou, J. Erman, V. Gopalakrishnan, E. Halepovic, R. Jana, X. Jin, J. Rexford, and R. K. Sinha. 2015. Can Accurate Predictions Improve Video Streaming in Cellular Networks?. In *16th HotMobile Workshop*. 57–62.