Comparing Fixed and Variable Segment Durations for Adaptive Video Streaming – A Holistic Analysis

Susanna Schwarzmann, Nick Hainke
TU Berlin
{susanna,nick}@inet.tu-berlin.de

Thomas Zinner
NTNU - Norwegian University of Science and Technology
thomas.zinner@ntnu.no

Christian Sieber
TU Munich
c.sieber@tum.de

Werner Robitza, Alexander Raake
TU Ilmenau
{werner.robitza,alexander.raake}@tu-ilmenau.de

ABSTRACT

HTTP Adaptive Streaming (HAS) is the de-facto standard for video delivery over the Internet. It enables dynamic adaptation of video quality by splitting a video into small segments and providing multiple quality levels per segment. So far, HAS services typically utilize a fixed segment duration. This reduces the encoding and streaming variability and thus allows a faster encoding of the video content and a reduced prediction complexity for adaptive bit rate algorithms. Due to the content-agnostic placement of I-frames at the beginning of each segment, additional encoding overhead is introduced. In order to mitigate this overhead, variable segment durations, which take encoder placed I-frames into account, have been proposed recently. Hence, a lower number of I-frames is needed, thus achieving a lower video bitrate without quality degradation. While several proposals exploiting variable segment durations exist, no comparative study highlighting the impact of this technique on coding efficiency and adaptive streaming performance has been conducted yet. This paper conducts such a holistic comparison within the adaptive video streaming eco-system. Firstly, it provides a broad investigation of video encoding efficiency for variable segment durations. Secondly, a measurement study evaluates the impact of segment duration variability on the performance of HAS using three adaptation heuristics and the dash.js reference implementation. Our results show that variable segment durations increased the Quality of Experience for 54% of the evaluated streaming sessions, while reducing the overall bitrate by 7% on average.

CCS CONCEPTS

• **Information systems** → **Multimedia content creation**; **Multimedia streaming**.

KEYWORDS

HAS, Adaptive Streaming, QoE, Video Encoding, Testbed Measurements, Variable Segment Durations, Fixed Segment Durations

MMSys'20, June 8–11, 2020, Istanbul, Turkey

ACM ISBN 978-1-4503-6845-2/20/06…$15.00
https://doi.org/10.1145/3339825.3391858

ACM Reference Format:
Susanna Schwarzmann, Nick Hainke, Thomas Zinner, Christian Sieber, Werner Robitza, and Alexander Raake. 2020. Comparing Fixed and Variable Segment Durations for Adaptive Video Streaming – A Holistic Analysis. In *11th ACM Multimedia Systems Conference (MMSys'20), June 8–11, 2020, Istanbul, Turkey*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3339825.3391858

1 INTRODUCTION

Online video streaming has become the prevalent way of video consumption and constitutes a large and ever-growing fraction of the global Internet traffic [3]. The most applied streaming technology nowadays is HTTP adaptive streaming (HAS). By splitting the video clip into small segments with multiple quality levels per segment and by using typically the HTTP protocol for signaling purposes, HAS allows a client-based dynamic adaptation of the video quality to the available bandwidth. Typically, HAS services rely on segments of 2 to 10 seconds [19], where the duration is fixed throughout the video. Fixed segment durations thus reduce the degrees of freedom of both, the video encoding and the video streaming process. It allows a faster video encoding, because there are less dependencies the encoder has to consider. In terms of video streaming, the prediction complexity for the adaptive bit rate (ABR) algorithm is reduced, since it can rely on a fixed increase of the buffered playtime after a segment download has finished. Fixed segment durations, however, also introduce additional overhead, since keyframes (I-frames) have to be inserted in a content-agnostic manner at the beginning of each segment.

Technically, HAS allows utilizing segments of variable durations, as long as the segment start times, i.e., the I-frame locations, are consistent between the different quality levels. Thus, aligning the segment split positions with I-frames that are needed a-priori, e.g. due to scene-cuts, creates such variable segment durations. Netflix refers to this approach as shot-based encoding [11], and recent papers [15, 25] provide initial insights of its potential to increase the encoding efficiency. However, a large-scale comparative analysis, studying impact factors such as the compression rate, video resolution, or the magnitude of segment duration variability, has not been conducted yet. Besides, the existing works neglect the influence of variable segment durations on the video streaming itself. The variability of the segment durations also increases the segment size variations, which may affect the ABR algorithm and therewith the streaming performance [7]. Hence, to show the applicability of variable segment durations for today's HAS systems,

it is essential to conduct a broad comparative study of the coding efficiency and the streaming behavior for both, variable and fixed segment durations.

This paper addresses this gap by conducting a broad comparative study. In order to evaluate and compare the fixed and the variable approach with regard to encoding-related metrics – such as segment durations, file size, or video quality – we created a large dataset consisting of roughly 2,000 encoded video sequences. For the sake of representativeness of our results, we chose four publicly available video sources with durations between 8 and 12 minutes, all with a resolution of 2160p. We used H.264 and considered variable bitrate encoding (VBR) as well as constant bitrate encoding (CBR) with different constant rate factors (CRF) and target bitrates. Evaluations with this dataset show that variable segment durations can reduce video bitrate by up to 15%, while maintaining a comparable video quality, as shown by the SSIM metric. Furthermore, we reveal the relevant factors influencing the potential for bitrate reduction. To study the impact of variable segment durations on HAS performance, we run testbed measurements with varying network conditions for a subset of the encoded videos and estimate the Quality of Experience (QoE) using the model from ITU-T Recommendation P.1203. The evaluation of more than 7,000 streaming sessions using different ABRs reveals a slight increase in the median QoE of all streaming sessions when using variable segment durations. Additionally, we make our data sets, implementations, and measurement tools available to the public.

The rest of the paper is structured as follows: Section 2 presents related work. Section 3 introduces HAS content preparation and describes the variable approach. Section 4 details the video encoding process and in Section 5 we describe the influence of the proposed approach on the video streaming process. Finally, Section 6 concludes the paper.

2 RELATED WORK

The segment duration is a crucial factor for the performance of adaptive video streaming and the user's satisfaction with the service [17]. Shorter video segments allow a more fine-granular adaptation of the video quality to current network conditions. However, short video segments lower the encoding efficiency [8] and increase the signaling overhead. Flow-level modeling is applied in [9] to develop mathematical models to analyze the performance of HAS for different fixed segment durations in a mobile environment. The results motivate using shorter segments, as they allow higher frequencies for quality adaptations and better properties in terms of video smoothness. To overcome the issue of increased signaling overhead, the authors propose to request several video segments at once. Liu et al. [10] examine how to set the segment duration so as to optimize the client's TCP throughput estimation, which in turn allows an optimized bitrate adaptation for rate-based heuristics.

The selection of segment durations in live-streaming scenarios is discussed in [21]. The paper evaluates the trade-off between high responsiveness in terms of quality adaptation and a higher encoding overhead when selecting a suitable segment duration. The authors show that streaming with segments of a sub-second duration allows to reduce the start-up delay and camera-to-display delay. By using the HTTP/2 push feature, they overcome the issue of increased signaling overhead.

The work presented in [20] proposes to use different segment durations, depending on the current HAS phase. The HAS videos are segmented several times so to obtain several representations in terms of segment duration. During the start-up phase, the client requests very short video segments. With increasing buffer size, longer segments are fetched from the server. Using this method, the authors combine the advantages of a low start-up delay (due to short segments) with encoding efficiency resulting from longer segments during a steady playout phase. The idea of providing the video content not only in different qualities, but additionally split in segments of different durations, is also presented in [22]. A similar approach is presented in [6], which proposes to provide several video representations, some having long and some having short segment durations. Accordingly, an ABR algorithm is proposed which does not only choose the next segment's bitrate, but also the segment's duration. A weakness of this approach is the constrained possibility for switching between representations with longer and shorter durations, as this is only possible where the segments' starting points are synchronized.

Although the works above consider several representations with different segment durations, the durations within these representations are fixed. The idea to improve the alignment of the video segments with the video content has initially been proposed and compared with fixed video segments in [1]. In 2018, Netflix proposed shot-based encoding in their *Dynamic Optimization* approach, which utilizes variable segment durations and thereby allows for improved rate-distortion optimizations for each shot [11]. Similarly, [25] investigates the impact of variable segment durations in terms of video bitrate, quality degradation, and the number of resulting segments for different types of video content. Those evaluations are also performed in [15]. This work goes one step further than the previous ones by also considering the impact of variable segment durations on the video streaming performance. The authors show by means of an analytical model that the bitrate reduction, achieved by using variable segment durations, might also result in a lower stalling probability as compared to fixed segment durations. The theoretical investigations are, however, not backed up with measurements using today's state of the art implementations.

To summarize, the available body of related works indicates an improved coding efficiency of variable segment durations compared to fixed segment durations, but does not provide a large-scale comparative analysis. Hence, it is not yet understood how factors like video resolution, compression rate, or the granted range of segment duration variability, influence the performance. Furthermore, related work misses studying the impact of variable segment durations on video streaming quality from an end-user's perspective. The impact of segment durations is already complex in case of fixed lengths, mainly due to the non-trivial trade-off between quality adaptation frequency and encoding overhead. Variable segments may therefore increase the complexity of the system and result in a higher variability, particularly in a practical streaming scenario. Our paper goes beyond the state-of-the-art by providing a full picture on comparing fixed and variable segment durations, taking into account the implications for video streaming and the resulting QoE.

3 VARIABLE SEGMENT DURATIONS FOR ADAPTIVE STREAMING

This section introduces the state-of-the art mechanisms for preparing HAS video content with fixed segment durations. Afterwards, we explain the major differences of using the variable approach. Finally, we shortly describe requirements and best practices when using variable segment durations for adaptive streaming.

3.1 State-of-the-Art Video Preparation for HAS

In video codecs, intra-frames (I-frames) contain all information required for decoding and do not reference other frames. They typically have a larger size than predicted (P-) or bidirectionally predicted (B-)frames and should be used sparingly as refresh points for the decoder. At scene cuts, however, the placement of I-frames can yield lower file sizes, as predicting from a previous picture would be less efficient, that is, it would require more bits to code the difference than to simply create another I-frame.

In HAS, all encoded video segments must be playable independently, which requires them to start with an I-frame – more specifically, an instantaneous decoder refresh (IDR frame) – which is inserted during the segmentation process. Typically, it is recommended to encode videos for HAS using strictly fixed I-frame intervals. Depending on the used technology and intended encoding latency, these intervals range from 2 to 10 seconds [19, 24]. Choosing fixed intervals has practical reasons, since scene-cut detection can be disabled, and the encoder can work in a "set and forget" mechanism. This approach, however, lowers the encoding efficiency, as more I-frames are needed, particularly in the case of very short segments. Aligning the segment durations with existing I-frames – which are needed anyway due to scene-cuts – could reduce this overhead. This would result in video segments that have different durations. From a technical point of view, variable segment durations can be used for HAS, but there are some practical challenges associated with this method, as we will describe in the following.

3.2 Variable vs. Fixed Segment Durations

Figure 1 represents the first 45 seconds of the *Big Buck Bunny* video with 24 frames per second and highlights the differences between fixed and variable segmentation. In this example, we assume a maximum segment duration of 10 seconds (240 frames) for the variable approach, to avoid segments of too long durations. For the fixed approach, we set a duration of 4.5 seconds, i.e. , 108 frames.[1]

The top box of the figure represents the raw video, where each frame contains the complete image information. The second box illustrates a compressed, but not segmented video. I-frames are inserted for scene-cuts (frames 0, 10, 250, 285, 378, 553, 803) and for the rest of the video, the encoder relies on cheaper P- and B-frames, which are not shown in this figure. The third box illustrates a segmented video using variable segmentation. Similar to the unsegmented video, frames 0 and 10 are I-frames. At frame 240, the maximum duration of 10 seconds of the first segment is reached and a new segment has to start. Consequently, frame 240 must be encoded as an I-frame. This I-frame (240) can then also be used to account for the scene-cut, which is captured in the unsegmented sequence with the I-frame at position 250. Hence, frame 250 can be encoded as a P- or B-frame with the variable approach. This is possible because the encoder has some degree of freedom in terms of where to place an I-frame for an efficient encoding and segmentation. This especially holds, when a scene does not change abruptly, but with fading effects. The next three segment beginnings are aligned with the existing I-frames of the unsegmented video, that is, frame 285, 378, and 553. Finally, segments are split at 793 and 1033 due to the maximum segment duration limit of 240 frames.

The bottom box depicts the fixed segmentation, where all segments have a duration of 4.5 seconds. Two of the frames (250, 803), which used to be I-frames in the unsegmented video, can be replaced by a cheaper frame-type, as an I-frame was inserted nearby. However, due to the strictly fixed segment duration of 108 frames, I-frames are placed at 324 and 432, despite the small differences to their preceding I-frames, which are required because the scene changed. Another I-frame, inserted for the sake of a constant segment duration, is 540. Roughly half a second later, i.e., after 13 more frames, an I-frame is nevertheless needed as the scene changes again. The total number of additionally needed I-frames, due to video segmentation, sums up to 7 in the fixed case, while only 1 additional I-frame is needed in the variable case for the illustrated sequence of 45 seconds.

3.3 Requirements and Best Practices for Variable Segment Durations

The HAS principle of adapting the video quality prior to a segment's download does not hinder the usage of variable segment durations. Nevertheless, some requirements need to be fulfilled in order to implement this approach in a real system: first, the segment boundaries have to be aligned along all available quality representations, that is, video bitrates and resolutions. Even the slightest deviation will provoke a skip or a repetition of frames at quality switches, which may impair the user's experience. Furthermore, the player implementation must be agnostic to changing segment durations, that is, it has to consider each segment's duration individually. During our tests, we found that player implementations often rely on fixed segment play times. The TAPAS player [4], which is intentionally kept simple to ease the integration of own heuristics, only captured the duration of the first segment and assumed all other segments to have this duration, resulting in wrong buffer computations. Another similar issue was found with the `dash.js` reference player version 2.9.3. When the *insufficient buffer rule*[2] rule was triggered, it mapped the current segment's duration to the 10 subsequent segments, to estimate their download duration. As this estimation was based on the wrong segment durations, the player over- or under-estimated the download duration, resulting in a too optimistic or too pessimistic behavior in terms of quality selection. The issue has been addressed with version 3 of `dash.js`.

Finally, a practical maximum segment duration should be defined. The longer a segment's duration becomes, the longer it takes to download it. If the download duration exceeds the buffered time, it will cause video stalling. A dedicated maximum duration furthermore guarantees that an I-frame is placed after a certain time interval, which maintains quality and adaptability of the stream.

[1] Later parts of this work show that these values result in the same number of video segments for the fixed and the variable approach, when considering the entire video.

[2] https://github.com/Dash-Industry-Forum/dash.js/wiki/ABR-Logic

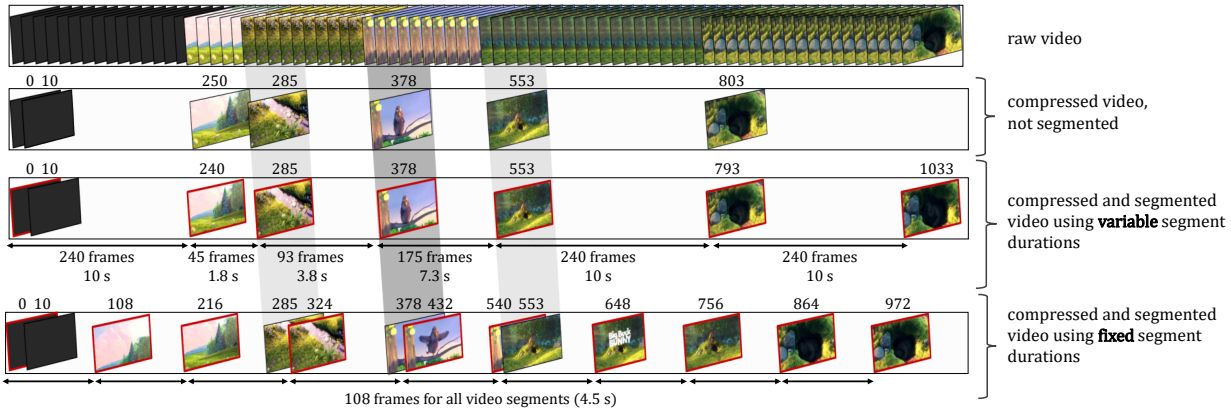


Figure 1: Illustration of I-frame placement for compressed videos without, with variable, and with fixed segmentation. Red frames indicate I-frames at the beginning of a video segment.

Table 1: Parameter settings for video encoding.

Characteristic	Value
Videos	BBB, TOS, MER, ELF
Resolutions	240p, 480p, 720p, 1080p, 2160p
Encoding method	VBR, CBR
CRF values (VBR)	16, 22, 28, 34
Target bitrates (CBR)	Average bitrates resulting from VBR encoding
Segment durations	*VAR* and *EM*: 4s, 6s, 8s, 10s *NA*: Average durations resulting from *VAR*

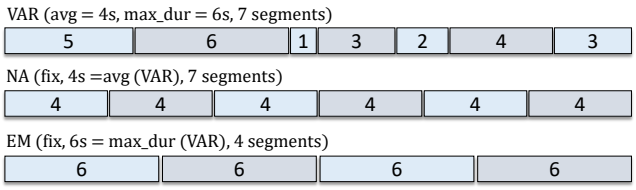


Figure 2: Illustration of the comparison options *NA* and *EM*. Bars denote video segments and their durations in seconds.

4 VIDEO ENCODING AND SEGMENTATION

This section compares the variable approach against the fixed-duration one with respect to encoding efficiency. We first present the methodology, including the encoding process and the parameters used. Afterwards, we focus on different characteristics of the resulting video segments and evaluate the encoding overhead for the fixed and the variable approach.

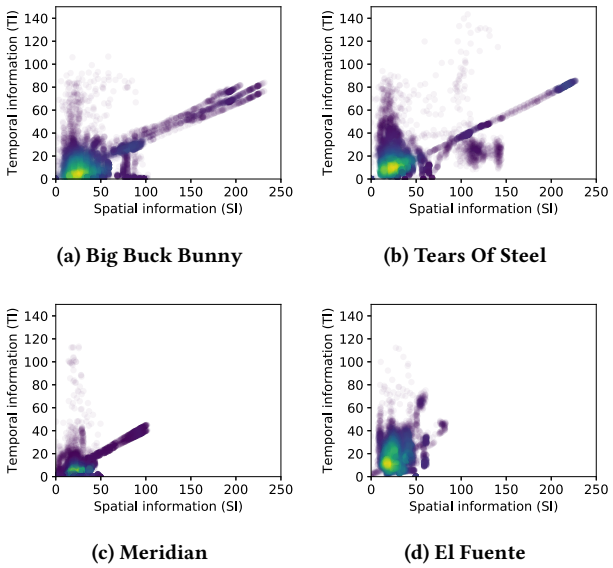
4.1 Methodology

This subsection describes our methodology for the video encoding and segmentation with the variable and the fixed approach. We apply a wide range of encoding options, which are summarized in Table 1. All relevant parameters for the encoding process are discussed in the following.

4.1.1 Terminology for comparing the fixed and the variable approach. When using the variable approach, we define a maximum duration (*max_dur*) for the video segments. The encoder can freely choose a segment's duration within the range from 0 to *max_dur*. In the following, we use two methods to compare the performance of the variable and the fixed approach, as illustrated in Figure 2. In the first approach, we evaluate fixed-segment sequences against those variable-segment sequences which have (nearly) the same *average* segment duration, with a granularity of half a second. For instance, if the variable approach with $max\_dur = 6$ yields an average segment duration of 4.3 seconds, this is compared to the fixed segmentation with a duration of 4.5 seconds, if it yields an average segment duration of 3.9 seconds, this is compared to the fixed segment duration of 4.0 seconds. We refer to this option as *nearest average (NA)*. The second approach consists of comparing the variable-segment encodes against the fixed-segment encodes which have the same duration as the specified *max_dur*. For instance, the variable approach with $max\_dur = 6$ is compared to the fixed approach with a segment duration of 6 seconds. Accordingly, we refer to this approach as *equal max (EM)*. The *NA* comparison yields (nearly) the same number of video segments, and hence has the same signaling overhead when it comes to video streaming. With *EM*, the total number of segments is lower for the fixed duration video. As a consequence, an *EM* video can be streamed with lower signaling overhead, but the quality cannot be adapted as often as for the video with variable segments. In the remainder of this paper, we refer to the approach applying variable segment durations as *VAR*, and use *NA*/*EM* for fixed-duration encoding.

4.1.2 Source videos. For all our tests, we used the freely available videos *Big Buck Bunny (BBB)*[3], *El Fuente (ELF)*, *Meridian (MER)*[4], and *Tears Of Steel (TOS)*[5]. The sources are scaled from their original dimensions to 2160p, 1080p, 720p, 480p, and 240p resolution (using bicubic filtering), with 24 frames per second (using `ffmpeg`'s `fps` filter). The spatial and temporal information (cf. ITU-T Rec. P.910)

[3] https://peach.blender.org/
[4] https://medium.com/netflix-techblog/engineers-making-movies-aka-open-source-test-content-f21363ea3781
[5] https://mango.blender.org/



(a) Big Buck Bunny (b) Tears Of Steel

(c) Meridian (d) El Fuente

Figure 3: Spatial and temporal information of the source videos. Lighter areas indicate higher density.

Table 2: Source video characteristics.

Video	Mean SI	Mean TI	Duration	Category
BBB	49.923	17.575	10:34	Cartoon
ELF	28.994	20.194	07:57	Documentary
MER	28.541	8.732	11:58	Mystery
TOS	45.307	21.027	12:14	Action

for the different videos are illustrated in Figure 3. It shows the ranges for the spatial and temporal complexity of the four videos in the test set. The according average values and further video characteristics are summarized in Table 2. Besides the varying spatio-temporal complexity, these videos have been chosen due to their different categories and due to their durations of at least about 8 minutes.

4.1.3 Encoding methods. We applied the following two different encoding/rate control methods, to either achieve a target quality or a target bitrate for the encoded bitstreams:

- **Variable Bitrate Encoding (VBR):** one-pass, using the x264 Constant Rate Factor (CRF), which results in a roughly constant quality.[6]
- **Constant Bitrate Encoding (CBR):** two-pass, using a target bitrate (br) and Virtual Buffer Verifier (VBV) constraints of $maxrate = 1.25 \cdot br$ and $bufsize = 2 \cdot br$.[7]

VBR encoding has a lower variation in quality over time, but leads to higher bitrate variations, which may impair the streaming performance. CBR, on the other hand, keeps the bitrate static, within constraints, along the video, resulting in possible quality degradations in scenes that are more spatio-temporally complex.

[6] https://slhck.info/video/2017/02/24/crf-guide.html
[7] https://slhck.info/video/2017/03/01/rate-control.html

We use VBR with four CRF settings, i.e., $crf \in \{16, 22, 28, 34\}$. The lower the CRF, the higher is the resulting video quality, whereby a value of 16 can be considered as visually lossless. An increase of the CRF value by 6 will roughly halve the resulting bitrate.[8] To determine the target bitrates for the CBR approach, we first encode all videos in all resolutions and specified segment durations using VBR encoding with the four specified CRF values. For each four-tuple $\{video/resolution/CRF/duration\}$, the average resulting bitrate is then used as the target bitrate for CBR, leading to similar average bitrates of the VBR and CBR videos.

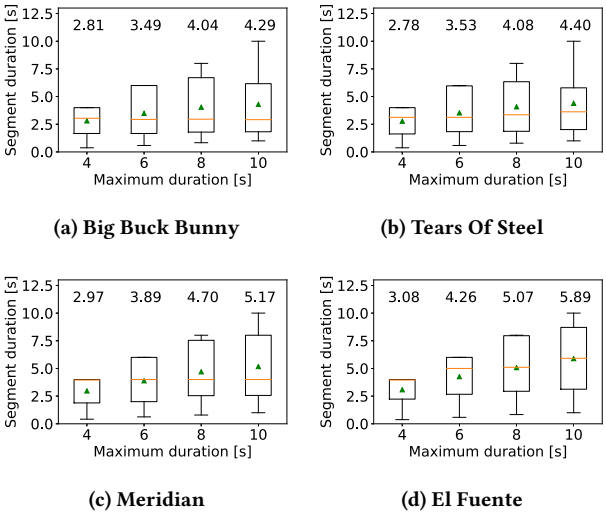
4.1.4 Segment durations. To ensure that segments of variable duration do not become too large, and to also compare the efficiency of different variable segment lengths, we choose four upper bounds for the variable durations, i.e., $max\_dur \in \{4, 6, 8, 10\}$ seconds. For the fixed durations, we used the values according to *NA* and *EM* resulting from the variable segmentation.

For the encoding with fixed segment durations, the `ffmpeg` option `force-key-frames` is used, and scene-cut detection is deactivated. To determine the variable durations for a given maximum duration, we choose for each video its 2160p resolution as the reference. This reference is encoded and segmented with the force keyframe option set to max_dur, i.e., keyframes are only forced if no keyframe was set since max_dur seconds. Furthermore, we do not specify any segment duration with the `seg_duration` option. This allows the encoder to freely choose the segment durations between 0 and max_dur seconds. All frame positions, at which the encoder decides to split the video, are logged during the encoding of the reference. These logged positions are then used as an `ffmpeg` input when encoding and segmenting the remaining video representations, to ensure that we split at exactly the same positions along each resolution and target bitrate or quality. The described procedure turned out to be necessary, as in rare cases, the split positions deviated by a few frames from one resolution to another. This was caused by the scene-cut detection of the encoder treating inter-frame differences differently at lower resolutions.

4.1.5 Encoding architecture and quality calculation. In order to support encoding on any platform and to easily distribute the encoding and evaluation process on several computing nodes, the tasks were encapsulated within a Docker[9] container.[10] Each Docker instance obtains one task, defined by the source video ID and a combination of encoding parameters. These parameters are summarized in a job description that includes the segmentation option (fixed vs. variable), the encoding option (CBR vs. VBR), the (maximum) segment duration, and a target bitrate or CRF value. When the video segmentation and encoding is completed, the container analyzes the resulting video and determines several parameters:

- encoding quality-relevant metrics, such as Structural Similarity (SSIM) [23] metric and Peak Signal-to-Noise Ratio (PSNR), calculated against the 2160p source via `ffmpeg`[11]
- the resulting bitrates and frame characteristics
- a timeline of segment durations and their sizes

[8] https://trac.ffmpeg.org/wiki/Encode/H.264
[9] https://www.docker.com
[10] https://github.com/fg-inet/docker-video-encoding
[11] https://github.com/slhck/ffmpeg-quality-metrics



(a) Big Buck Bunny (b) Tears Of Steel

(c) Meridian (d) El Fuente

Figure 4: Segment durations resulting from *VAR* with different maximum duration settings. Green markers and the numbers on top denote the average segment duration.

4.2 Evaluation

In the following, we present the results from our evaluation of the encoded and segmented videos for variable and fixed segment durations. We consider various characteristics, such as the segment durations themselves, resulting bitrates and quality, as well as factors influencing the performance of variable segment durations.

4.2.1 Resulting segment durations. Figure 4 illustrates the variable segments' durations for several settings of the maximum duration. In the case of Big Buck Bunny (Figure 4a), Tears Of Steel (Figure 4b), and Meridian (Figure 4c), the median durations hardly change between the different maximum durations. However, longer segments are used if allowed, leading to increased average durations when increasing the maximum duration setting. With a maximum duration of 4 seconds, the longest average durations can be observed for Meridian and El Fuente, as for these videos, the median duration corresponds almost to the maximum duration. El Fuente results in the largest average duration among the investigated videos for any of the configured maximum durations. This indicates that there are fewer scene-cuts and consequently less possibilities for the encoder to split at existing I-frames. As a result, I-frame placements and splits, which are required due to the specified duration limit, occur more often for the ELF clip, than for the other clips.

As described above, we compare the variable approach against the fixed one based on the average duration (i.e., *NA*), and based on the same maximum duration (i.e., *EM*). Table 3 summarizes the information retrieved from Figure 4 to allow a quick lookup of the fixed segment durations corresponding to the variable approach according to *VAR* and *NA*.

4.2.2 Reduction of I-frames. In the following, we investigate the number of I-frames that can be reduced by using the variable approach for the different videos. Figure 5a illustrates the number of I-frames needed when using *VAR* with a maximum segment duration of 10 seconds and the number of I-frames needed when considering the respective *NA* fixed segment duration. Note that for the same segment duration setting of a video, the number of I-frames is equal along all resolutions and bitrates, as well as for VBR and CBR encoding. Although both, *VAR* and *NA* segmentation, result in practically the same number of video segments, *VAR* requires less I-frames. For example, for the video BBB, *VAR* reduces the number of I-frames by 130, compared to *NA*. Hence, the *VAR* approach can economize 46% of the expensive frames in this case.

Table 3: Resulting fixed durations based on *EM* and *NA* for variable segmentation with different maximum durations.

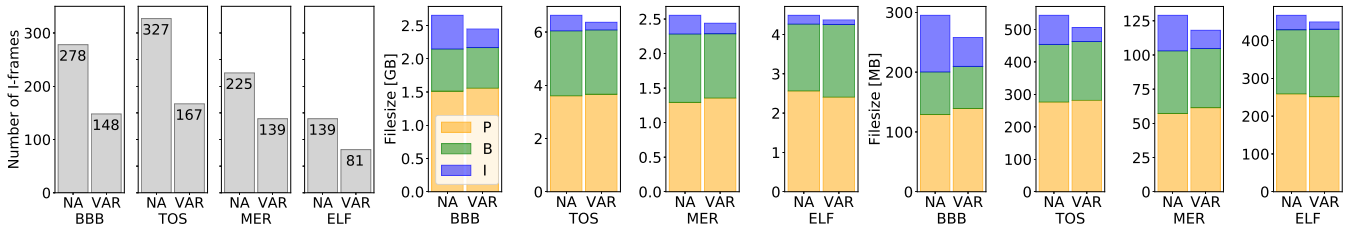
	BBB		TOS		MER		ELF	
VAR	*EM*	*NA*	*EM*	*NA*	*EM*	*NA*	*EM*	*NA*
0-4	4	3	4	3	4	3	4	3
0-6	6	3.5	6	3.5	6	4	6	4.5
0-8	8	4	8	4	8	4.5	8	5
0-10	10	4.5	10	4.5	10	5	10	6

Figure 5b and Figure 5c illustrate the impact of the I-frame reduction on the overall file size for two exemplary encodings with a resolution of 720p and CRF values of 16 and 34, whereby 16 is the highest quality we consider during our encoding evaluations, and 34 the lowest quality, respectively. The different colors denote the total file size made up by a specific type of frame. The height of the blue bars can roughly be halved for all cases by *VAR*, that is, the fraction of file size, which is contributed by I-frames, can roughly be halved.

4.2.3 Reduction of encoding overhead. Figure 6 illustrates the reduced video encoding overhead for the videos resulting from all parameter combinations. The x-axis denotes the percentage reduction that can be achieved with the proposed approach compared to the fixed approach. The y-axis shows the empirical cumulative distribution (ECDF). Figure 6a shows that with variable segments and CBR encoding, the bitrate can be reduced by up to 16%. This saving can be achieved for BBB, while the lowest saving is observed for ELF. While for BBB at least 6% of bitrate can be saved, for ELF, not even 5% can be reached. ELF has fewer scene-cuts and the lowest number of I-frames relative to its duration among all videos, resulting in a lower potential of improvement for *VAR*.

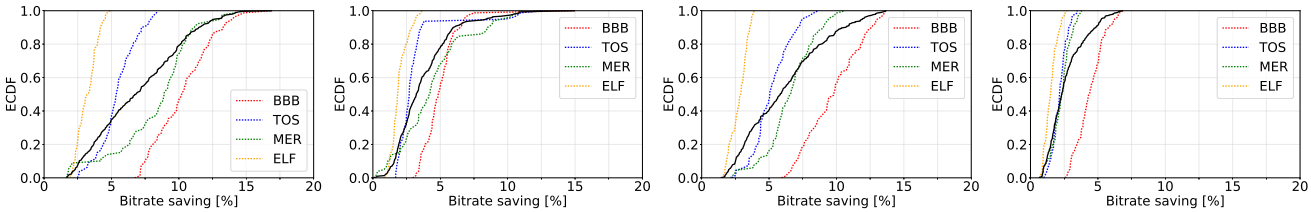
When comparing the variable approach and the fixed approach based on *EM* (Figure 6b), the saving in terms of bitrate is lower than in the *NA* case. This is due to the fact that the variable segmentation results in segments that are shorter compared to the fixed duration segments, and in general, shorter segments imply increased encoding overhead. The respective results for the VBR encoding are illustrated in Figures 6c and 6d. Compared to CBR encoding, the bitrate that can be reduced is lower. Nevertheless, the bitrate can be reduced by up to 13%.

4.2.4 Impact on video quality. In order to fairly compare variable and fixed segment durations, we need to examine whether the I-frame reduction results in a quality degradation for variable segments. Figure 7 illustrates the absolute difference of video quality expressed via the SSIM metric. This metric compares all frames of a



(a) Absolute number of I-frames (b) Filesize for 720p and CRF 16 (c) Filesize for 720p and CRF 34

Figure 5: Number of I-frames and its impact on filesize when using VBR encoding for *VAR* with a maximum duration of 10 seconds and for the respective *NA* segment duration.



(a) Bitrate saving, CBR, *NA* (b) Bitrate saving, CBR, *EM* (c) Bitrate saving, VBR, *NA* (d) Bitrate saving, VBR, *EM*

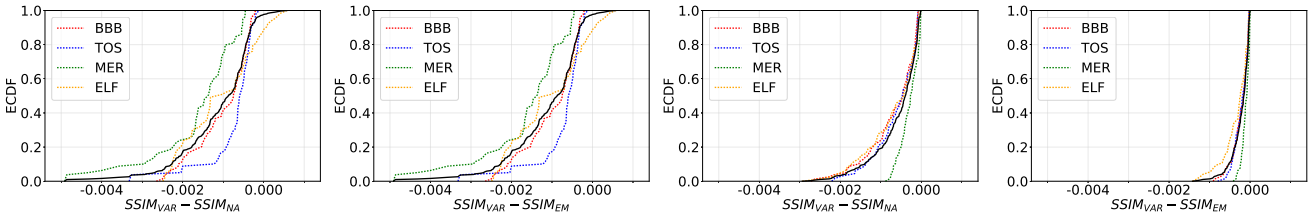
Figure 6: Bitrate saving that can be achieved by the variable approach compared to the fixed approach. Black solid lines denote the overall bitrate saving, colored dotted lines represent the different videos.

compressed video with the respective uncompressed and distortion-free reference frame and yields a value between 0 and 1, where 1 means equality to the uncompressed original content, i.e., highest quality. In our case, we calculated SSIM using the respective `ffmpeg` filter, with bicubic upscaling of the encoded video to the 2160p reference. For CBR encoded videos, we observe a quality degradation for *VAR* videos, with an SSIM reduction of at most 0.005 compared to both *NA* (Figure 7a) and *EM* (Figure 7b).

For VBR encoded videos, where the CRF value specifies a target quality, the differences in terms of SSIM are lower: the maximum difference observed for *NA* (Figure 7c) is below 0.003; for *EM* (Figure 7d) it is less than 0.0015. In general, the relationship between SSIM and perceived quality is not linear [23]. For high qualities, i.e., high SSIM values, already small SSIM disturbances may have a high impact on the MOS, while for lower qualities, i.e., low SSIM values, small disturbances are negligible. However, such effects are generally not visible to humans when they are in the order of magnitude which we observe. Hence, the quality degradations incurred by *VAR* are negligible. In the next subsection, where we perform an in-depth investigation of the factors that influence bitrate reduction and quality decrease, we will see that high quality encodings undergo a much smaller SSIM degradation than 0.005.

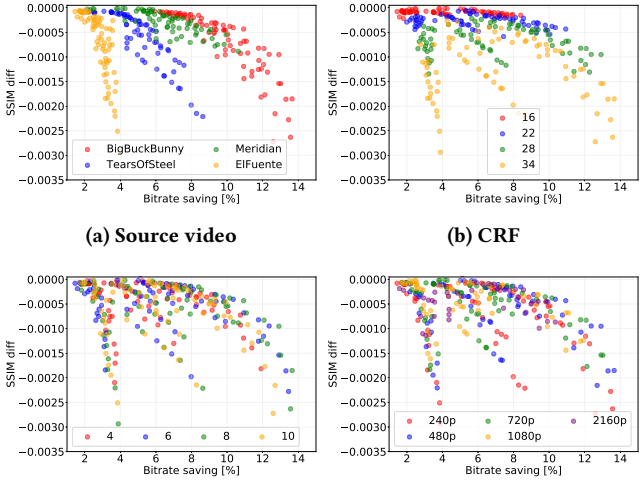
4.2.5 Influence factors. The evaluations above show that when using variable segment durations, bitrate can be saved for a slightly lower video quality. It can also be seen from above that the potential for bitrate saving is highly dependent on the source video, which is in line with the observations in Figure 8a. Further, it shows that for the video Meridian, variable segmentation yields the lowest quality degradation. Figure 8 also illustrates how the remaining factors, that is, CRF value, segment duration, and resolution influence the behavior of variable segment durations on bitrate and quality for VBR encoded videos. Firstly, Figure 8b shows a clear influence of the chosen CRF value. With higher CRF values (i.e., lower video quality), variable segment durations tend to degrade SSIM to a greater extent. Nevertheless, this degradation is still too small to be the reason for the significant bitrate reduction we observe. As shown above, the bitrate reduction can be achieved by eliminating I-frames with the more efficient variable method. For lower CRF values (i.e., higher video quality), the relative bitrate that can be saved is at most 8%, and we also observe a smaller quality degradation. As a third characteristic, we consider the segment duration in Figure 8c. It shows that the chosen maximum duration for variable video segments has no direct effect on the bitrate that can be saved compared to the *NA* fixed segmentation. However, there is a slight trend of lower quality degradation if the variable segments do no exceed a duration of 4 seconds. Finally, Figure 8d shows that the video resolution has no clearly visible influence on how bitrate and quality of variable segmentation behave compared to *NA*. To summarize, CRF and the source video itself highly impact the performance of variable segments in terms of bitrate and quality, while the effects of maximum durations and video resolution are negligible.

4.2.6 Summary. The results show that variable segment durations outperform fixed segment durations with regard to the encoding overhead at the costs of a slightly reduced SSIM value. Even if the bitrate saving is small in certain cases, we want to emphasize that none of the tested configurations results in an increased bitrate with *VAR*. Furthermore, we revealed that the share of bitrate which can be



(a) SSIM diff, CBR, *NA* (b) SSIM diff, CBR, *EM* (c) SSIM diff, VBR, *NA* (d) SSIM diff, VBR, *EM*

Figure 7: Difference in terms of video quality, expressed as SSIM, for the variable and fixed approach. Black solid lines denote the overall bitrate saving, colored dotted lines represent the different videos.



(a) Source video (b) CRF

(c) Segment duration (d) Video resolution

Figure 8: Impact of *VAR* on bitrate and video quality for VBR encoding depending on different video characteristics.

saved mainly depends on the source video and the compression rate, while a direct influence of the video resolution or the configured maximum duration could not be observed.

4.3 Limitations and Future Research Directions

To reveal relevant impact factors on the performance of *VAR*, we encoded the videos with numerous combinations of encoding- (VBR vs. CBR), video- (CRF, resolution), and segmentation-related factors. Consequently, we had to limit the number of source videos in order to reduce the complexity of the factorial design. Despite the varying SI and TI values for the videos in our dataset, it may not be sufficiently representative for a large catalogue that a Video on Demand provider may have. Furthermore, all videos were encoded using H.264 with the `libx264` encoder implementation, and thus, the validity of our results is limited to this codec. However, we assume that the results are generally similar for other codecs or codec implementations, as inefficient segmentation is a general problem for HAS content preparation, independent from the encoder. Nevertheless, the performance of *VAR* should also be studied for other codecs, such as VP9, H.265/HEVC, or AV1. Finally, we note that we used `ffmpeg` in a very basic manner for generating variable segments, and that the split positions (i.e., the scene cuts with maximum segment duration restrictions) were only determined for a single 4K representation of each video. More sophisticated methods to determine the split positions, for example using a deeper analysis of the video content prior to splitting or analyzing all resolutions, could improve the efficiency of *VAR*, and lead to an optimized encoding performance over all resolutions.

5 VIDEO STREAMING

The evaluations in the previous section show that the encoding overhead can be reduced with *VAR*. However, the introduced variability in terms of segment durations results consequently in an enlarged variability of the segments' sizes, which can negatively affect the video streaming performance. It is not clear whether the reduced bitrate can compensate this enlarged variability. Hence, this section evaluates the feasibility of *VAR* for adaptive video streaming based on testbed measurements.

5.1 Methodology

In the following, we describe the methodology for the video streaming experiments. First of all, we present the set of videos chosen for the measurements. Afterwards, we introduce our virtual testbed and describe the network configurations that have been used. Finally, the ITU-T P.1203 model, which is applied to evaluate the QoE, is described.

5.1.1 Videos for streaming evaluation. During the video streaming evaluations, we compare the variable approach against the fixed one based on *NA*, i.e., the number of downloaded segments during a video session is practically equal. Furthermore, we used the videos resulting from the constant bitrate encoding (CBR), as this is a more realistic encoding method for video streaming, since VBR encoding results in an overall higher bitrate variability.

As we showed, the video itself has a strong influence on the performance of variable segment durations during the encoding process (cf. Figure 8a). To account for this influence factor during the video streaming process, we perform testbed measurements with all of the four source videos. For each video, we use the variable video representation with the highest maximum segment duration, i.e. 10 seconds. Note that in terms of encoding efficiency, the effect of the maximum duration is negligible (cf. Figure 8c) compared to the effect of the source video or target quality. However, the

Table 4: Selected bitrates and resolutions for the streaming measurements.

	BBB		TOS		MER		ELF	
L	*Res*	*BR*	*Res*	*BR*	*Res*	*BR*	*Res*	*Br*
0	480p	215 kbps	240p	234 kbps	720p	164 kbps	240p	291 kbps
1	720p	406 kbps	480p	354 kbps	720p	342 kbps	480p	403 kbps
2	1080p	797 kbps	720p	689 kbps	1080p	492 kbps	720p	942 kbps
3	1080p	1.6 Mbps	720p	1.4 Mbps	1080p	2.0 Mbps	720p	2.1 Mpbs
4	1080p	3.4 Mbps	1080p	2.7 Mbps	1080p	12.6 Mbps	1080p	3.5 Mbps

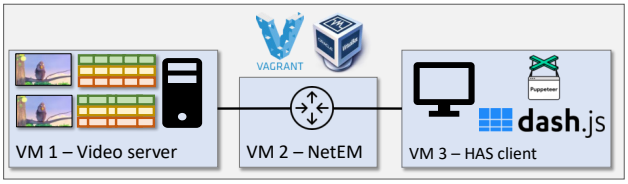


Figure 9: Illustration of the virtual measurement setup.

larger the maximum duration, the higher is the variability of the resulting segments' sizes, which negatively affects video streaming performance [2]. Hence, the evaluations using the variable videos with a maximum duration of 10 seconds can be seen as a "worst case scenario" with respect to the variability of the segment durations and size. The coefficient of variation of the segment sizes is larger with *VAR* for all of the selected video clips. For the BBB clip, it increases from 0.43 (*NA*) to 0.74 (*VAR*), and for TOS from 0.45 to 0.76. In case of MER, the coefficient of variation with *NA* is 0.69 and 0.77 with *VAR*. Finally, *VAR* increases the segment size variability for ELF from 0.52 to 0.55.

To determine the bitrate ladder (i.e., the resolution-bitrate pairs selected for video streaming), we utilize the selection method presented in [5]. For each video, we choose 5 different quality levels, according to the resulting bitrate ladder. Table 4 illustrates the resolutions and bitrates used for the different quality levels for the *VAR* videos. The quality levels for the *NA* videos only differ in the sense that the bitrates are slightly higher on each level, due to the higher encoding overhead. We omit video representations with a resolution of 2160p, as this resolution is not supported by the P.1203 standard.[12]

5.1.2 Measurement environment. Our virtual testbed environment is illustrated in Figure 9. *Vagrant* and *VirtualBox* are used to set up three virtual machines. One of them acts as the server hosting the videos, one as the HAS streaming client, and the third VM acts as a network emulator. The latter connects client and server and allows to emulate different network settings, i.e., rate limiting using the Linux traffic control[13]. The client runs the browser-based DASH reference player `dash.js`[14] in version v3.0.0. We modified the player so to log all relevant metrics for QoE computation, such as playback quality or video stallings. For the sake of scalability and to allow streaming tests without actually playing back the video (e.g., when running on a server where no display is attached), the browser runs in headless mode. To allow the client to request videos in headless mode, we use *Puppeteer*[15], which runs on top of *Node.js*. Our testbed is publicly available on GitHub[16] to allow the research community to use it for further research and to facilitate the reproducibility of our results.

5.1.3 Video player settings. The `dash.js` player implements the three following ABR strategies: a buffer-based solution according to BOLA [18], a throughput-based, and a hybrid solution.[17] We run measurements with each of the available strategies, and refer to them as *BOLA-ABR*, *throughput-ABR*, and *hybrid-ABR*. We set the initial buffer threshold to 12 seconds and the stable buffer time, i.e., the internal buffer target the player tries to reach, to 30 seconds. The maximum buffer time is set to 45 seconds, i.e., the client will pause segment requests when this threshold is reached.

5.1.4 Network settings. We test the feasibility of variable segment durations for adaptive streaming with fluctuating bandwidth capacities, which allows to capture the behavior in a more stressful manner. We use realistic bandwidth traces [13] and scale them so as to achieve an average rate of the $\{1, 2, 4, 6\}$-fold of the lowest quality's bitrate of each of the four *VAR* test videos. We refer to these bandwidth limit settings as *bandwidth provisioning factor* ρ, i.e., $\rho \in \{1, 2, 4, 6\}$. Additionally, we limit the available bandwidth to the 1-fold of the lowest quality's bitrate for each *NA* video. The very low bandwidth settings allow a comparison of *NA* and *VAR* in those scenarios, where hardly any other than the lowest quality can be downloaded and where the heuristics's behavior is negligible for the streaming performance. We confine on $\rho = 6$ as the highest rate, as this bandwidth configuration already triggers the heuristic to choose between different levels that yield a decent video quality. Hence, these scenarios allow to study the impact of *VAR* on the heuristic's behavior and consequently the quality adaptation and resulting video streaming performance.

From the trace dataset[18], we choose three replicas of each of the traces *car*, *ferry*, and *tram*. We furthermore define three different start points for each of the traces, namely from the beginning, i.e., second 0, and two randomly chosen start points. The traces are looped, i.e., if the end of the trace is reached, it starts again from the beginning. For each trace replica and each start point, three measurement runs are performed. This results in 27 streaming session per trace, rate limit, video, and adaptation strategy, resulting in more than 7000 testbed measurement runs in total.

5.1.5 QoE analysis. The QoE of the streamed videos is analyzed with the standardized ITU-T Rec. P.1203 model, using the publicly available software.[19] In contrast to short-term video models, or image quality metrics like PSNR/SSIM, the P.1203 model is well-suited for HAS QoE with longer session durations of several minutes. It has been shown to predict the real streaming QoE with high accuracy [12, 14]. The model includes the typical HAS QoE influence factors, such as stalling during the playout, initial loading delay, or video quality fluctuations over time. The video quality itself can be

[12] Current developments in the ITU-T P.1204 recommendation series will address 4K/UHD video but were not available at the time of writing this paper.
[13] https://linux.die.net/man/8/tc
[14] https://github.com/Dash-Industry-Forum/dash.js
[15] https://github.com/puppeteer/puppeteer
[16] https://github.com/fg-inet/DASH-streaming-setup
[17] https://github.com/Dash-Industry-Forum/dash.js/wiki/ABR-Logic#primary-rules
[18] http://skulddata.cs.umass.edu/traces/mmsys/2013/pathbandwidth/
[19] https://github.com/itu-p1203/itu-p1203

Table 5: Median improvements and confidence intervals (CI) for the different QoE metrics over all runs.

Score	Improvement ± CI		
	hybrid-ABR	*BOLA-ABR*	*throughput-ABR*
O23	0.034 ± 0.010	0.011 ± 0.011	0.008 ± 0.008
O34	0.048 ± 0.005	0.043 ± 0.003	0.046 ± 0.005
O46	0.035 ± 0.012	0.015 ± 0.015	0.011 ± 0.013

estimated in different *modes*. Lower modes require less information at the cost of lower accuracy, but can be computed more easily. In our streaming tests, since we have full access to the streamed segments, we use ITU-T Rec. P.1203.1 Mode 3, which requires decoding of the bitstream. This mode uses video frame-level characteristics, such as the frame types, frame sizes and the quantization parameter (QP) values on a per-macroblock scale. Thus, mode 3 yields the highest QoE estimation accuracy that is possible with the P.1203 model. The model returns one overall quality score and several diagnostic quality scores on a MOS scale between 1 and 5, whereby 1 represents bad, and 5 represents excellent quality. We will use the following scores throughout our evaluations:

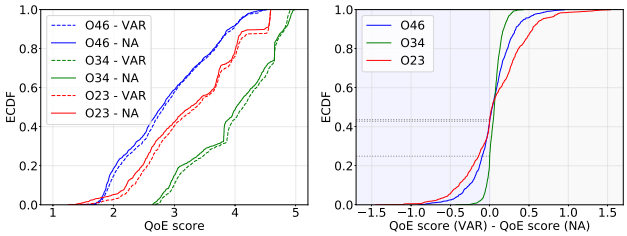
- **O34:** Per-second audiovisual quality score
- **O23:** Stalling quality
- **O46:** Overall quality score, combines audiovisual and stalling quality scores

As we omit the audio track for the videos, the QoE model per default assumes a constant high audio quality when computing the audio-visual quality score (O34). Furthermore, O34 yields a value for each second of the video stream. When we refer to O34 in later parts of this work, we mean the average of all per-second scores of a streaming session.

5.2 Evaluation

In the following, we compare the performance of variable and fixed segment durations for adaptive video streaming. We performed an in-depth analysis of all three ABRs available in the dash.js reference implementation, i.e., *hybrid-ABR*, *BOLA-ABR*, and *throughput-ABR*. For our tested scenarios, we found that there are only slight differences in terms of how *VAR* performs compared to *NA*. Table 5 shows the median improvements for the different QoE scores achieved by *VAR*, i.e., *QoE(VAR)-QoE(NA)*, using the different ABR strategies. We furthermore denote the confidence intervals on a 95% confidence level. The medians of the different QoE scores differ only slightly and the corresponding confidence intervals overlap in most of the cases. Particularly for the overall QoE O46, the confidence intervals for all of the ABRs overlap, showing that the impact of the ABR is not significant. For that reason, we limit the following detailed streaming analysis on *hybrid-ABR*, which is the default configuration of dash.js

5.2.1 QoE scores over all runs. Figure 10a illustrates the different QoE scores obtained for all measurement runs with variable segment durations (*VAR*) and the respective fixed segment durations using *NA* comparison and the *hybrid-ABR* logic. The x-axis represents the values on MOS scale, the y-axis denotes the ECDF. As the



(a) Absolute QoE scores **(b) Difference of QoE scores**

Figure 10: Absolute value and differences of the QoE scores obtained from the measurements using the *hybrid-ABR*

Table 6: Average values for the different QoE scores obtained with *hybrid-ABR*. Bold numbers represent the respective higher value.

	O23		O35		O46	
ρ	*VAR*	*NA*	*VAR*	*NA*	*VAR*	*NA*
1.0	**2.510**	2.370	**3.582**	3.530	**2.178**	2.105
2.0	**3.585**	3.518	**3.963**	3.882	**2.973**	2.931
4.0	**3.873**	3.826	**4.370**	4.309	**3.435**	3.399
6.0	3.923	**3.939**	**4.533**	4.496	3.587	**3.618**

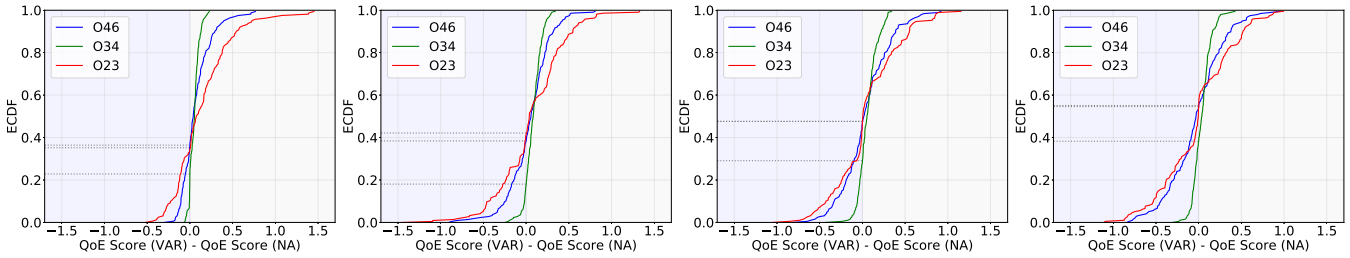
dotted lines, i.e., the values for *VAR* are slightly shifted, *VAR* tends to increase the video streaming QoE. While the median value for O46, i.e., the overall quality, is 2.806 for *NA*, this value can slightly be increased to 2.853 by *VAR*. In terms of stalling quality, denoted as O23, *NA* achieves a median value of 3.346, which also can slightly be improved by *VAR*, which achieves a median value of 3.404. More significant improvements using the variable approach can be seen for O34, i.e. the audio-visual quality score. While the median for *NA* is 4.0, this value increases to 4.12 for *VAR*.

Figure 10b illustrates the absolute differences of the QoE scores obtained for all measurement runs. In terms of audio-visual quality, i.e., O34, *VAR* improves the QoE in 75% of the tested cases. The stalling quality (O23) can be improved for 56% of the runs, while the overall QoE is improved in 57% of the tested cases.

Overall, the median improvement achieved by *VAR* with *hybrid-ABR* is 0.034 ± 0.01 for O23 and 0.048 ± 0.005 for O34. The median improvement of the overall QoE score, i.e., O46, is 0.035 ± 0.012 (cf. Table 5). As none of the denoted confidence intervals includes 0, we can conclude that the improvement by *VAR* is significant for all considered QoE metrics.

5.2.2 QoE scores obtained with different rates. In order to better understand further influence factors, we evaluate the obtained QoE values for different bandwidth capacities. Figure 11 shows the difference $QoE(VAR) - QoE(NA)$ for different rates, while Table 6 denotes the average values of the QoE scores obtained with *VAR* and *NA* using *hybrid-ABR*

For a bandwidth provisioning factor of $\rho = 1$, the overall QoE score can be improved in 64% of the cases. The maximum improvement of O46 that can be observed for this rate is 0.767, while the worst impairment of *VAR* reduces the QoE by 0.297. The differences



(a) $\rho = 1$ **(b)** $\rho = 2$ **(c)** $\rho = 4$ **(d)** $\rho = 6$

Figure 11: Differences in terms of QoE scores for different bandwidth provisioning factors ρ.

in terms of O34 are relatively small, as the small bandwidth capacity hardly leaves room for streaming on any other than the lowest quality level. In contrast, the stalling quality (O23) can be increased by up to 1.452, while it is never worsened by more than 0.52.

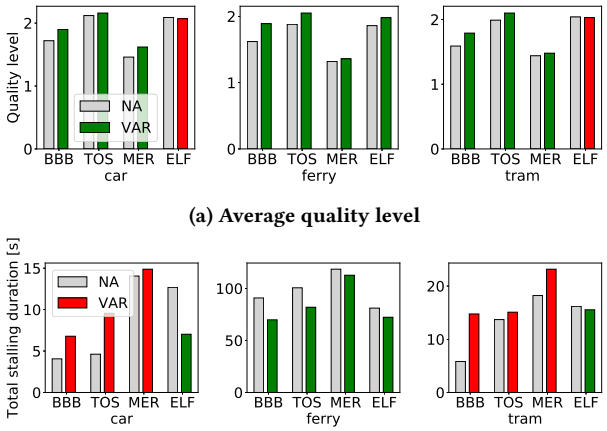
Figure 11b shows the absolute difference of the QoE scores for a bandwidth provisioning factor of $\rho = 2$. At this rate, in 82% of the measurement runs, the visual quality (O35) can be improved, while still improving the stalling quality for 58%. The overall quality score can be improved for 62% of the cases.

For an average rate corresponding to $\rho = 4$, as illustrated in Figure 11c, the overall quality score (O46) is improved by *VAR* for 52% of the test runs, leading to a slightly increased O46 score for *VAR*. Hence, the improvements achieved by *VAR* terms of QoE are more significant than the degradation.

With $\rho = 6$, we observe the first case, where the fixed approach outperforms the variable approach for the majority of the test runs. In 55% of the cases, *NA* yields a higher O46 score than *VAR*. Furthermore, the stalling quality (O23) is in 55% of the cases higher with *NA*, than with *VAR*. However, in 62% of our scenarios, *VAR* still increases the audio-visual quality score. The maximum increase is 0.42, while the worst degradation of O34 is by 0.3.

The overall QoE (O46) is mainly affected by the stallings [16]. This is also noticeable in our evaluations, as O46 and O23 have a similar behavior for the shown cases in Figure 10b and Figure 11. If O23 can be increased by *VAR* for a certain share, the share with which O46 can be increased is similar to that. The degradation of O46 in scenarios with increasing available bandwidth might be due to an increase of stallings, resulting from a too optimistic quality adaptation of the heuristic with *VAR* videos. The resulting higher visual quality, however, cannot compensate the increased number of stallings and hence, the overall QoE decreases for *VAR*. Table 6, which summarizes the average QoE values obtained with *VAR* and *NA* for different settings of ρ, indeed shows a first indication for this behavior: The average O23 score for the $\rho = 6$ scenario is decreased by *VAR*, while this score could be improved in any of the other scenarios. This decrease of the O23 stalling quality results in a lowered average overall QoE O46 with *VAR*.

5.2.3 Detailed investigation for $\rho = 6$. To further investigate this hypothesis, we depict the average quality level and the total stalling duration, i.e., the sum of all video interruptions in Figure 12. As it can be seen from Figure 12a, the average quality level for *VAR* is increased for all videos and all traces, except for two cases. The



(a) Average quality level

(b) Total stalling duration

Figure 12: Quality level and stalling duration for $\rho = 6$. Green bars denote an improvement by *VAR* compared to *NA*, red bars denote an impairment.

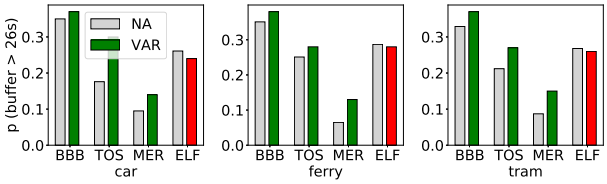


Figure 13: Probability for buffer levels nearby the target buffer.

first one is the ELF clip for the *car* trace, the second case is again the clip ELF, but for the *tram* trace.

Figure 12b shows that for the *car* trace, the total stalling duration is increased for all videos, except ELF, which is the only one where the average quality is not increased, but slightly decreased using *VAR*. The same holds for the *tram* trace, where the stalling duration is increased for all those videos, where the average quality is increased using *VAR*.

The *hybrid-ABR* selects the next segment's quality based on the measured throughput and the buffer level. As we configured the target buffer level as 30 seconds, quality switches are likely to happen shortly before this level is reached. Figure 13 illustrates the probability of a buffer level above 26 seconds. In all cases, the probability

of having a buffer level near the target buffer is higher for *VAR* than for *NA*, except for the ELF clip. This strengthens our hypothesis that with *VAR*, the overall QoE score (O46) decreases for higher rates due to a too aggressive quality adaptation. As the average bitrate for *VAR* is lower compared to *NA*, higher buffer values can be reached faster. This triggers the heuristic to switch to a higher quality, which can lead to stalling, especially in environments with varying bandwidth capacity and if the downloaded segment is of comparably long duration. Please note the increased variability in terms of segment sizes for the *VAR* approach. A possible solution to overcome this drawback is to set higher buffer thresholds for quality switches, when segments of variable durations are used.

5.2.4 Summary. In scenarios with low bandwidth capacity, the approach using variable segment durations can clearly outperform the conventional approach for HAS with fixed segment durations. In these cases, the impact of the adaptation logic and the buffer-based quality switching thresholds are negligible, as hardly any other than the lowest quality level can be played back. In scenarios with sufficient bandwidth for a decent streaming quality, the variable approach performs worse than the fixed approach for roughly half of the measured cases. Due to the decreased bitrate for variably segmented videos, the probability of high buffer values – and hence the probability to switch to a higher quality level – is increased. Hence, with variable segment durations, slightly higher buffer-thresholds for quality switches should be set.

5.3 Limitations and Future Research Directions

The BOLA algorithm does not simply switch quality as soon as a certain buffer threshold is reached, but implements more sophisticated decision criteria, which might not sufficiently comply with variable segment durations. These criteria have not been addressed in this paper, and have also not been investigated more deeply in terms of how they interact with variable segment durations. However, the goal of this paper is to test the performance of variable segment duration using a state-of-the-art reference implementation for HAS. By tuning the relevant parameters and using heuristics with a higher potential of supporting variable segment durations, similar to [7], which accounts for segment size variability, it can be assumed that the streaming performance of *VAR* in terms of QoE can further be increased.

The results might be biased, as we set the initial buffer threshold to 12 seconds for any video. However, as the segment durations vary (between *VAR* and *NA*), the number of downloaded segments prior to video playback may differ, and consequently also the buffer levels when the playback actually starts. As a result, this can impact the initial delay and the probability of video stallings, the latter one especially during the first phases of playback. Furthermore, the experiments were conducted with relatively low bandwidth capacities, as compared to the highest quality levels available. This could bias the results in favor for *VAR*, which outperforms *NA* in low bandwidth scenarios.

Future research needs to examine how the comparison with *EM* performs. *EM* results in an increased number of video segments and yields a lower bitrate saving than *NA*, as fewer I-frames can be economized. On the other hand, *VAR* has no increased segment durations when compared with *EM* and as a consequence, quality adaptation can be done on the same, or even on a more fine-granular scale. Furthermore, the maximum duration for variable video segments, which has been limited to 10 seconds in this work, should be studied. Finally, factors such as the number of provided quality levels or concurrent video streams can have an impact on the performance and should be investigated in the context of variable segment durations.

6 CONCLUSION

Using video segments of variable durations for HAS videos can reduce the encoding overhead compared to the conventional approach which uses fixed segment durations. In this work, we encoded a small set of videos using numerous encoding options, that is, different segment durations, CRF values, resolutions, and CBR and VBR encoding. Based on the resulting dataset, consisting of about 2,000 encoded video sequences, we quantified the bitrate saving and showed that using variable segment duration can be up to 15% more efficient than the fixed approach.

To evaluate the performance of variable segment durations for adaptive streaming, we performed more than 7,000 testbed measurement runs with different network settings. Here as well, the testbed is made publicly available. It uses the standardized ITU-T P.1203 QoE model in order to quantify the impact of the encoding decisions on user-experienced quality. In scenarios with low bandwidth capacity, the variable approach clearly outperforms the fixed approach in terms of QoE, which is attributed to the reduced bitrate, whilst maintaining a comparable visual quality. In scenarios with high bandwidth, we observed several cases where the QoE was degraded by the variable approach. We found that in these cases, the player switched more aggressively to a higher quality level when streaming the video with variable segment durations. In turn, stallings occurred more often. In order to eliminate such drawbacks, one might consider to tune player parameters such as buffer thresholds, or to design a heuristic which is dedicated for variable segment durations. For example such a heuristic would decide not to download the next segment in a higher quality, although the current buffer level allows it, if the next segment has a comparably large duration.

To conclude, due to the reduced bitrate, HAS can profit from variable segment durations with a similar to slightly better QoE. However, further research is required on the design of heuristics and buffer thresholds that do not suppress the benefits of variable segment durations, while eliminating the drawback of a too aggressive quality adaptation.

ACKNOWLEDGMENTS

We thank our shepherd Hermann Hellwagner and all anonymous reviewers for their valuable feedback, as well as Bruno Péjac and Hendrik von Kiedrowski for their continuous support with the streaming setup.

REFERENCES

[1] Velibor Adzic, Hari Kalva, and Borko Furht. 2012. Optimizing video encoding for adaptive streaming over HTTP. *IEEE Transactions on Consumer Electronics* 58, 2 (2012), 397–403.

[2] Valentin Burger, Thomas Zinner, Lam Dinh-Xuan, Florian Wamser, and Phuoc Tran-Gia. 2018. A generic approach to video buffer modeling using discrete-time analysis. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 14, 2s (2018), 33.

[3] Cisco. 2018. Cisco Visual Networking Index: Forecast and Trends, 2017–2022. White Paper.

[4] Luca De Cicco, Vito Caldaralo, Vittorio Palmisano, and Saverio Mascolo. 2014. TAPAS: a Tool for rApid Prototyping of Adaptive Streaming algorithms. In *Proceedings of the 2014 Workshop on Design, Quality and Deployment of Adaptive Video Streaming*. ACM, 1–6.

[5] Jan De Cock, Zhi Li, Megha Manohara, and Anne Aaron. 2016. Complexity-based consistent-quality encoding in the cloud. In *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, Phoenix, AZ, USA, 1484–1488.

[6] Iheanyi Irondi, Qi Wang, and Christos Grecos. 2016. Optimized adaptation algorithm for HEVC/H.265 dynamic adaptive streaming over HTTP using variable segment duration. In *Real-Time Image and Video Processing 2016*, Vol. 9897. International Society for Optics and Photonics, Brussels, Belgium, 98970O.

[7] P. Juluri, V. Tamarapalli, and D. Medhi. 2015. SARA: Segment aware rate adaptation algorithm for dynamic adaptive streaming over HTTP. In *2015 IEEE International Conference on Communication Workshop (ICCW)*. IEEE, London, UK, 1765–1770. https://doi.org/10.1109/ICCW.2015.7247436

[8] Stefan Lederer, Christopher Müller, and Christian Timmerer. 2012. Dynamic adaptive streaming over HTTP dataset. In *Proceedings of the 3rd Multimedia Systems Conference*. ACM, Chapel Hill, North Carolina, USA, 89–94.

[9] Yu-Ting Lin, Thomas Bonald, and Salah Eddine Elayoubi. 2016. Impact of chunk duration on adaptive streaming performance in mobile networks. In *2016 IEEE Wireless Communications and Networking Conference*. IEEE, Doha, Qatar, 1–6.

[10] Chenghao Liu, Imed Bouazizi, and Moncef Gabbouj. 2011. Segment duration for rate adaptation of adaptive HTTP streaming. In *2011 IEEE International Conference on Multimedia and Expo*. IEEE, Barcelona, Spain, 1–4.

[11] Megha Manohara, Anush Moorthy, Jan De Cock, Ioannis Katsavounidis, and Anne Aaron. 2018. Optimized shot-based encodes: Now Streaming! https://medium.com/netflix-techblog/optimized-shot-based-encodes-now-streaming-4b9464204830

[12] Alexander Raake, Marie-Neige Garcia, Werner Robitza, Peter List, Steve Göring, and Bernhard Feiten. 2017. A bitstream-based, scalable video-quality model for HTTP adaptive streaming: ITU-T P.1203.1. In *Ninth International Conference on Quality of Multimedia Experience (QoMEX)*. IEEE, Erfurt, 1–6. https://doi.org/10.1109/QoMEX.2017.7965631

[13] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. 2013. Commute path bandwidth traces from 3G networks: analysis and applications. In *Proceedings of the 4th ACM Multimedia Systems Conference*. ACM, Oslo, Norway, 114–118.

[14] Werner Robitza, Steve Göring, Alexander Raake, David Lindegren, Gunnar Heikkilä, Jörgen Gustafsson, Peter List, Bernhard Feiten, Ulf Wüstenhagen, Marie-Neige Garcia, Kazuhisa Yamagishi, and Simon Broom. 2018. HTTP Adaptive Streaming QoE Estimation with ITU-T Rec. P.1203 – Open Databases and Software. In *9th ACM Multimedia Systems Conference*. ACM, Amsterdam, The Netherlands, 466–471. https://doi.org/10.1145/3204949.3208124

[15] Susanna Schwarzmann, Thomas Zinner, Stefan Geissler, and Christian Sieber. 2018. Evaluation of the benefits of variable segment durations for adaptive streaming. In *2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX)*. IEEE, Sardinia, Italy, 1–6.

[16] Michael Seufert, Nikolas Wehner, and Pedro Casas. 2018. Studying the Impact of HAS QoE Factors on the standardized QoE model P.1203. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1636–1641.

[17] Anargyros Sideris, E Markakis, Nikos Zotos, Evangelos Pallis, and Charalabos Skianis. 2015. MPEG-DASH users' QoE: The segment duration effect. In *2015 Seventh International Workshop on Quality of Multimedia Experience (QoMEX)*. IEEE, Costa Navarino, Messinia, Greece, 1–6.

[18] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. 2016. BOLA: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, San Francisco, CA, USA, 1–9.

[19] Thomas Stockhammer. 2011. Dynamic adaptive streaming over HTTP–: standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, San Jose, California, USA, 133–144.

[20] Jeroen van der Hooft, Dries Pauwels, Cedric De Boom, Stefano Petrangeli, Tim Wauters, and Filip De Turck. 2018. Low-latency delivery of news-based video content. In *Proceedings of the 9th ACM Multimedia Systems Conference*. ACM, Amsterdam, The Netherlands, 537–540.

[21] Jeroen Van Der Hooft, Stefano Petrangeli, Tim Wauters, Rafael Huysegems, Tom Bostoen, and Filip De Turck. 2018. An HTTP/2 push-based approach for low-latency live streaming with super-short segments. *Journal of Network and Systems Management* 26, 1 (2018), 51–78.

[22] Bjørn J Villa and Poul E Heegaard. 2013. Group based traffic shaping for adaptive HTTP video streaming by segment duration control. In *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*. IEEE, Barcelona, Spain, 830–837.

[23] Zhou Wang, Alan C Bovik, Hamid R Sheikh, Eero P Simoncelli, et al. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4 (2004), 600–612.

[24] Anatoliy Zabrovskiy, Christian Feldmann, and Christian Timmerer. 2018. A practical evaluation of video codecs for large-scale HTTP adaptive streaming services. In *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, 998–1002.

[25] Ondrej Zach and Martin Slanina. 2018. Content aware segment length optimization for adaptive streaming over HTTP. *Radioengineering* 27, 3 (2018), 819.

A APPENDIX

This appendix shows how both, the video encoding and the video streaming experiments described in the paper, can be reproduced. The following page gives an overview to the provided artifacts: https://fg-inet.github.io/acm-mmsys-2020 It links to the GitHub repositories of the tools used in the paper and provides the Zenodo link for the dataset.

A.1 Overview of Artifacts

Our artifacts include:

- **Video Encoding Results** – The statistics for all encoded videos used for comparing the variable and the fixed segment durations approach.
- **Videos Used for Streaming Measurement** – We provide all encoded video sequences that have been used for the video streaming experiments.
- **Video Encoding Environment** – The code for the setup for scalable video encoding and analysis.
- **Video Streaming Setup** – The code for the setup for controlled and monitored experiments with the reference player dash.js.

Please note: For the paper, more than 6,000 streaming measurement runs have been performed and roughly 2,000 videos (including 4K) have been encoded and analyzed. For both the encoding environment and the streaming setup we give a step-by-step introduction how to use them with a light, exemplary setting. However, all data and information needed to reproduce the whole range of experiments are made available.

A.2 Dataset

In the following, we describe the dataset, which can be found here: https://zenodo.org/record/3732206.

A.2.1 Video Encoding Results. This data set contains two .csv files, one for the encoding with constant bitrate (CBR) and one for the encoding with variable bitrate (VBR). For each encoded video, the result files contain the necessary source video information, such as name, frame rate, or resolution. Furthermore, we denote the used encoding settings, including CRF value, target bitrate, segment duration, as well as whether a the fixed or the variable approach was used. Finally, we provide the metrics obtained from analyzing the encoded sequences. Among others, minimum, maximum, average, and standard deviation are given for the resulting bitrate, segment duration, and SSIM.

A.2.2 Videos Used for Streaming Measurements. For the streaming measurement, the CBR encoded videos have been used. The video directories are named as follows:
`<encoding>_<title>_<segmentation>_<max_dur>`
For example, `CBR_BBB_VAR_10` denotes the CBR encoded version of the BBB clip with variable segments of a maximum duration of 10 seconds. Each video folder contains the video playlist and one subfolder for each available quality level. The subfolder's name represent the quality level's resolution and CRF, i.e., quality:
`<resolution>_<CRF>`.

We use the following structure for a video directory:

```
CBR_BBB_VAR_10
├── playlist.mpd
├── 1080_22
│   ├── init-stream0.m4s
│   ├── segment1.m4s
│   ├── segment2.m4s
│   └── ...
└── 720_34
    ├── init-stream0.m4s
    ├── segment1.m4s
    ├── segment2.m4s
    └── ...
```

A.3 Docker Container for Video Encoding

This part details on the Docker-based encoding environment used in this paper and available on GitHub: https://github.com/fg-inet/docker-video-encoding. We first give a detailed description and present the steps for a quick testing afterwards.

A.3.1 Description. In the following present the most relevant components of the setup. Then, we describe in more detail their interplay and the sequence of encoding a video with this setup.

- `Dockerfile` – This file is needed to create the Docker image. It installs all dependencies and provides all necessary scripts to the container. Furthermore, it defines the so-called entrypoint, which is in our case the file `video_encode.py`
- `video_encode.py` – Using this script, the actual video encoding is performed, based on a number of given parameters:
  - `video` – The raw video to be encoded.
  - `reference_video` – The reference video for calculating the quality metrics (the reference's resolution can be higher than the resolution of `video`).
  - `CRF` – The value for the constant rate factor. The CRF may range from 0 to 51, whereby lower values result in higher video quality.[20]
  - `min_length` – The minimum segment duration, specified in seconds. This parameter is only relevant with variable segments. (We always set the parameter to 0 in this work)
  - `max_length` – The maximum segment duration, specified in seconds. This parameter is only relevant with variable segments. (We used values of 4, 6, 8, and 10 in this paper)
  - `target_seg_length` – The segment duration, specified in seconds, when segments of fixed duration should be used. This parameter needs to be set to "var", if variable segments should be used.
  - `encoder` – The encoder to be used. Please note, the current version supports x264 only.

[20] https://slhck.info/video/2017/02/24/crf-guide.html

  - `timestamps` – Timestamps where segments should be split.
  - `cst_bitrate` – The target bitrate for the resulting video. This is only needed for CBR encoding.
- `worker.py` – A script for setting up a so-called worker. A worker is one running Docker instance with a dedicated job, i.e, a set of encoding parameters.
- `run_workers_mmsys.sh` – A script starting the workers. It determines the number of available CPU cores, and starts one worker per CPU core.
- `dirjobs.py` – A script for managing all encoding jobs.

In the following, we describe in detail the interplay of `worker.py`, `run_workers_mmsys.sh`, and `dirjobs.py`. The setup is designed so as to make video encoding scalable across many CPU cores. Hence, it allows to run several encoding jobs in parallel (one per CPU core) and maintains a job queue. A job description stores all relevant encoding parameters as shown below.

```
{"video": "bigbuckbunny480p24x30s.y4m",
 "reference_video": "bigbuckbunny480p24x30s.y4m",
 "crf": 16,
 "min_length": 4,
 "max_length": 4,
 "target_seg_length": 4,
 "encoder": "x264",
 "cst_bitrate": 123456
}
```

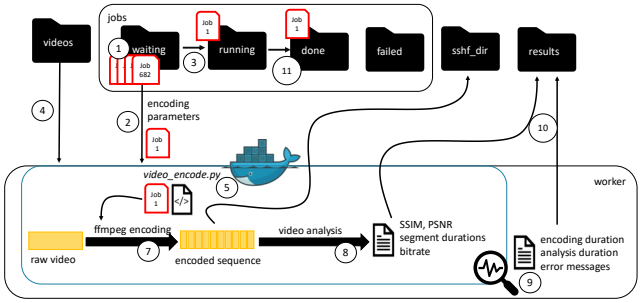


Figure 14: Illustration of the video encoding setup.

As shown in Figure 14, all pending jobs are stored in the folder `jobs/waiting` (①). A free worker fetches a job from this job queue and can so obtain all relevant encoding parameters (②). This triggers `dirjobs.py` to move the respective job from `jobs/waiting` to `jobs/running` (③). The worker gets the raw video specified in the job description from the `videos` directory (④). Then, the worker starts the Docker container (⑤), which triggers the execution of `video_encode.py`. Hence, the video is encoded using `ffmpeg` along with the parameters specified in the job description (⑦). As soon as the encoding process has finished, the encoded and segmented video is analyzed (⑧). This includes statistics regarding the segment durations and bitrate, but also quality metrics, such as SSIM and PSNR. The ladder one are obtained using `ffmpeg_quality_metrics`[21]. The worker monitors the container during running, and logs the video encoding duration, the time it took for analyzing the resulting sequence, and possible error messages (⑨). The collected logs and statistics are then stored in the `results` folder. Finally, the job description is moved to the `jobs/done` folder (⑪). If any of the step were not successful, the job description would be move to the `jobs/failed` folder.

A.3.2 Exemplary Encoding / Quick Testing. To perform a quick testing of the setup, please follow the next steps. It starts the procedure as described above with four representative jobs. The jobs include VBR and CBR encoding, as well as segmentation with fixed and with variable segment durations. The chosen source video is a 30 seconds snippet of the BBB clip.

(1) In the main directory, i.e., `docker-video-encoding`, run `bash init_mmsys.sh`. This will download the raw video sequence and create all necessary directories.
(2) Run `bash run_workers_mmsys.sh`. This will initiate the encoding process and take some time. The waiting, running, and finished jobs can be found in the respective folders (`00_waiting`, `01_running`, and `99_done`).
(3) The encoding and video statistics are stored in the subfolder `results`, the encoded videos can be found in `sshfs_dir`.

A.3.3 Log Files. Upon others, the following log files are produced for each encoding job:

- `video_statistics.json` – Contains the size, duration, and bitrate for each video segment. For these metrics, it furthermore provides statistics, such as min, max, standard deviation, and average values.
- `timings.json` – Lists the durations for encoding the video, analyzing SSIM and PSNR, and for gathering the remaining statistics.
- `stats.json` – Contains information about the used worker, summarizes the encoding parameters, and lists the used directories for storing the encoded sequences and statistics.
- `segments.json` – Frame-level statistics for each video segment. This includes height and width, frame type (I,P,B), key frame, etc.
- `psnr_ssim_vmaf.csv` – Quality statistics on frame-level.

A.4 Virtual Environment for Streaming Measurements

The testbed environment is publicly available on Github: https://github.com/fg-inet/DASH-streaming-setup. It allows to run measurements using the dash.js player in headless mode for different network settings. The setup requires Vagrant, at least 8GB RAM and must be run bare-metal hardware, as Virtualbox does not provide the setup of virtual interfaces in an already virtualized environment.

A.4.1 Description. The setup consists of three virtual machines: a server, a client, and a network emulator (netem). The client uses the Chrome browser in headless mode to do video streaming using the dash.js player. The client's requests in headless mode are realized using Puppeteer 16, which runs on top of Node.js. All relevant files on client-side can be found in the folder `DASH-setup/client`, all relevant files on server-side in the folder `DASH-setup/server`,

[21] https://github.com/slhck/ffmpeg-quality-metrics

respectively. We give a brief overview on the most relevant files and folders.

- `DASH-setup/client/logs` – As soon as video has been played back completely, its log files are stored in .json format in this folder. This folder is shared between the client VM and the host machine.
- `DASH-setup/server/public/videos` – All videos placed in this directory will be available at the server VM after provisioning and can be requested by the client.
- `DASH-setup/server/public/javascripts/player.js` – The javascript file for the video player. The function called in line 20, `player.updateSettings`, allows to customize streaming settings, such as buffer thresholds, or the applied ABR.[22]

All necessary files for booting the VMs can be found in the folder `DASH-setup/vagrant_files`. In the following, we give a short description on the files relevant for booting the VMs.

- `Vagrantfile` – This file configures the VMs. It assigns IP addresses to the machine's interfaces, allocates resources, provides all necessary files to the machines and defines shared folders between the VMs and the host machine.
- `setup_server.sh`, `setup_client.sh`, `setup_netem.sh` – These are the provisioning scripts to install required packages and to define additional settings, e.g. IP routes.

Finally, we provide some scripts to allow automatized measurements and setting bandwidth limits at the netem. They are as well located in `DASH-setup/vagrant_files`.

- `netem_start_trace.sh` – This script takes a bandwidth trace as input parameter and throttles the bandwidth on the netem's outgoing interface according to the values specified in the trace. The bandwidth values are updated each second.
- `trace_killer.sh` – A cleanup script which stops bandwidth being throttled at the netem.
- `experiment_startup.sh` – The script for measurement automation. Via SSH, it triggers the netem to start and stop limiting the interface. Similarly, it triggers the client to request videos. It allows to specify the number of repetitions to be performed, the location of the trace files to be used, and the video which should be streamed. An overview is given in Figure 15.

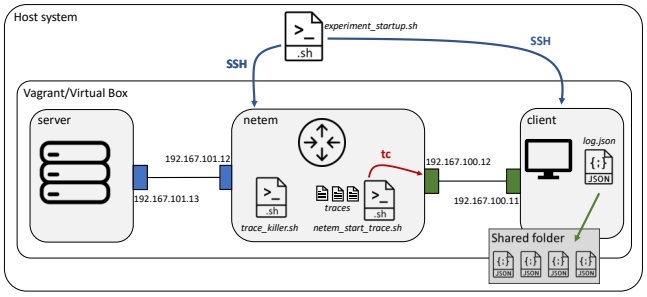


Figure 15: Illustration of the streaming measurement setup.

[22]Please find all possible settings for the dash.js player here: http://cdn.dashjs.org/latest/jsdoc/module-Settings.html#~StreamingSettings__anchor

A.4.2 Exemplary Measurement Runs. In order to run exemplary measurements, the repository provides two versions of the BBB clip as sample sequences. One version using fixed segment durations, one version using variable segment durations. The automation script `experiment_startup.sh` is configured so to use the two traces given in the folder `test_traces`, to stream both versions of the BBB clip, and to repeat each configuration two times. This results in 8 runs in total. To start the measurements, please follow the next steps:

(1) Open a terminal and navigate into the following directory: `DASH-streaming-setup/vagrant_files`.
(2) Run `vagrant up` in order to boot all virtual machines. This will take a few minutes. All VMs are completely provisioned when the following output is given: `[nodemon] starting node ./bin/www`.
(3) Open another terminal window and navigate into the directory `DASH-streaming-setup/vagrant_files`.
(4) There, run `bash experiment_startup.sh` to initiate the measurements. This will take some time.
(5) As soon as a run has finished, its log files are stored on the host machine in `DASH-setup/client/logs`

A.4.3 Log Files. Two log files are produced for each successful measurement run:

- `segmentLog.json` – For each downloaded video segment, the following information is given: segment index, segment size, timestamp of segment download start, download duration.
- `qualityLog.json` – Each 100ms, the following information is logged: timestamp, buffer level, quality level, overall played back time.