COBIRAS: offering a continuous bit rate slide to maximize DASH streaming bandwidth utilization

Michael Seufert, Marius Spangenberger, Fabian Poignée, Florian Wamser, Werner Robitza, Christian Timmerer, Tobias Hoßfeld

Angaben zur Veröffentlichung / Publication details:

Seufert, Michael, Marius Spangenberger, Fabian Poignée, Florian Wamser, Werner Robitza, Christian Timmerer, and Tobias Hoßfeld. 2024. "COBIRAS: offering a continuous bit rate slide to maximize DASH streaming bandwidth utilization." ACM Transactions on Multimedia Computing, Communications, and Applications 20 (10): 311. https://doi.org/10.1145/3677379.

COBIRAS: Offering a Continuous Bit Rate Slide to Maximize DASH Streaming Bandwidth Utilization

MICHAEL SEUFERT, University of Augsburg, Augsburg, Germany
MARIUS SPANGENBERGER and FABIAN POIGNÉE, University of Würzburg, Würzburg, Germany
FLORIAN WAMSER, Lucerne University of Applied Sciences and Arts, Lucerne, Switzerland
WERNER ROBITZA, AVEQ GmbH, Vienna, Austria
CHRISTIAN TIMMERER, Christian Doppler-Labor ATHENA, Alpen-Adria-Universität, Klagenfurt, Austria
TOBIAS HOßFELD, University of Würzburg, Würzburg, Germany

Reaching close-to-optimal bandwidth utilization in dynamic adaptive streaming over HTTP (DASH) systems can, in theory, be achieved with a small discrete set of bit rate representations. This includes typical bit rate ladders used in state-of-the-art DASH systems. In practice, however, we demonstrate that bandwidth utilization, and consequently the quality of experience (QoE), can be improved by offering a continuous set of bit rate representations, i.e., a continuous bit rate slide (COBIRAS). Moreover, we find that the buffer fill behavior of different standard adaptive bit rate (ABR) algorithms is sub-optimal in terms of bandwidth utilization. To overcome this issue, we leverage COBIRAS' flexibility to request segments with any arbitrary bit rate and propose a novel ABR algorithm *MinOff*, which helps maximizing bandwidth utilization by minimizing download off-phases during streaming. To avoid extensive storage requirements with COBIRAS and to demonstrate the feasibility of our approach, we design and implement a proof-of-concept DASH system for video streaming that relies on just-in-time encoding (*JITE*), which reduces storage consumption on the DASH server. Finally, we conduct a performance evaluation on our testbed and compare a state-of-the-art DASH system with few bit rate representations and our *JITE* DASH system, which can offer a COBIRAS, in terms of bandwidth utilization and video QoE for different ABR algorithms.

CCS Concepts: • **Information systems** → **Multimedia streaming**; • **Software and its engineering** → **Software performance**;

Michael Seufert did a substantial part of the work for this article while he was at University of Würzburg, Würzburg, Germany.

This work was partly funded by Deutsche Forschungsgemeinschaft (DFG) under grant SE 3163/3-1, project number: 500105691. The financial support of the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development, and the Christian Doppler Research Association, is gratefully acknowledged. The authors alone are responsible for the content.

Authors' Contact Information: Michael Seufert (corresponding author), University of Augsburg, Augsburg, Germany; e-mail: michael.seufert@uni-a.de; Marius Spangenberger, University of Würzburg, Würzburg, Germany; e-mail: marius.spangenberger@stud-mail.uni-wuerzburg.de; Fabian Poignée, University of Würzburg, Würzburg, Germany; e-mail: fabian.poignee@uni-wuerzburg.de; Florian Wamser, Lucerne University of Applied Sciences and Arts, Lucerne, Switzerland; e-mail: florian.wamser@hslu.ch; Werner Robitza, AVEQ GmbH, Vienna, Austria; e-mail: werner.robitza@aveq.info; Christian Timmerer, Christian Doppler-Labor ATHENA, Alpen-Adria-Universität, Klagenfurt, Austria; e-mail: christian.timmerer@aau.at; Tobias Hoßfeld, University of Würzburg, Würzburg, Germany; e-mail: tobias.hossfeld@uni-wuerzburg.de.

ACM 1551-6865/2024/10-ART311
https://doi.org/10.1145/3677379

Additional Key Words and Phrases: Dynamic adaptive streaming over HTTP, DASH, HTTP adaptive streaming, HAS, encoding, bit rate representations, adaptive bit rate, ABR, bandwidth utilization, quality of experience, QoE

ACM Reference format:
Michael Seufert, Marius Spangenberger, Fabian Poignée, Florian Wamser, Werner Robitza, Christian Timmerer, and Tobias Hoßfeld. 2024. COBIRAS: Offering a Continuous Bit Rate Slide to Maximize DASH Streaming Bandwidth Utilization. *ACM Trans. Multimedia Comput. Commun. Appl.* 20, 10, Article 311 (October 2024), 24 pages.
https://doi.org/10.1145/3677379

1 Introduction and Motivation

Video streaming services, which concurrently download and playback media files, are among the most popular and most challenging applications in today's Internet. End users expect a high **quality of experience (QoE)** when consuming video streams, which generally can be reached by minimizing initial delay and playback interruptions (stalling, rebuffering) [16, 46, 65], while maximizing the visual quality of the streamed content in terms of high resolution, high frame rate, and low visible compression artifacts [46, 47, 55, 63]. However, increasing the visual quality typically also increases the bit rate, and thus, the required bandwidth for video streaming, which may not always be available in communication networks.

Adaptive bit rate (ABR) streaming is the current dominant streaming technology [4]. It allows mitigating possible QoE degradation by adapting the video bit rate to the network conditions. It typically utilizes standard web protocols, such as **Hypertext Transfer Protocol (HTTP)** over Transmission Control Protocol or QUIC, to promote simple service implementation and high availability. Such HTTP adaptive streaming is implemented in many commercial solutions and was standardized as Moving Picture Expert Group **dynamic adaptive streaming over HTTP (DASH)** [28]. To enable adaptation to the current network conditions, the server stores the video content encoded in different representations, i.e., in different bit rates. The representations are split in temporal dimension into segments (also referred to as chunks), each containing a fixed amount of playback time, such that the bit rate can be seamlessly switched after each segment.

The ABR algorithm (or adaptation logic) at the client is an algorithm that selects which segment from which representation to download next from the list of available segments and representations into the local playout buffer at the client. These decisions usually take into account characteristics of the representations (e.g., bit rate), current network conditions (e.g., bandwidth measurements or estimations), and current playout statistics (e.g., buffer fill level) [4]. Note that DASH systems typically operate on a limited playout buffer to avoid downloading many video segments that will not be watched if the user quits early, and also to be able to quickly react to network fluctuations, e.g., by switching to a higher representation if throughput increases. If the buffer is full, the ABR algorithm thus will transition from an on-phase (download ongoing) to an off-phase (no download) until the buffer fill level drops below a certain threshold. Consequently, it is the ABR algorithm that predominantly influences the network demands of the streaming and the resulting QoE.

However, the ABR algorithm is also restricted to only select between the bit rate representations that are available on the server. As state-of-the-art DASH systems typically offer only a small set of representations, a so called bit rate ladder, the resulting utilization of the available bandwidth, and consequently the resulting QoE, might not be optimal. To provide a simplified numerical example, consider a 1080p video, which may be available in six resolutions (144p, 240p, 360p, 480p, 720p, 1080p), e.g., [64], which correspond to six bit rate representations for a given video codec. According to [18], the recommended YouTube upload bit rate is 12 Mbps for 1080p and 7.5 Mbps for 720p

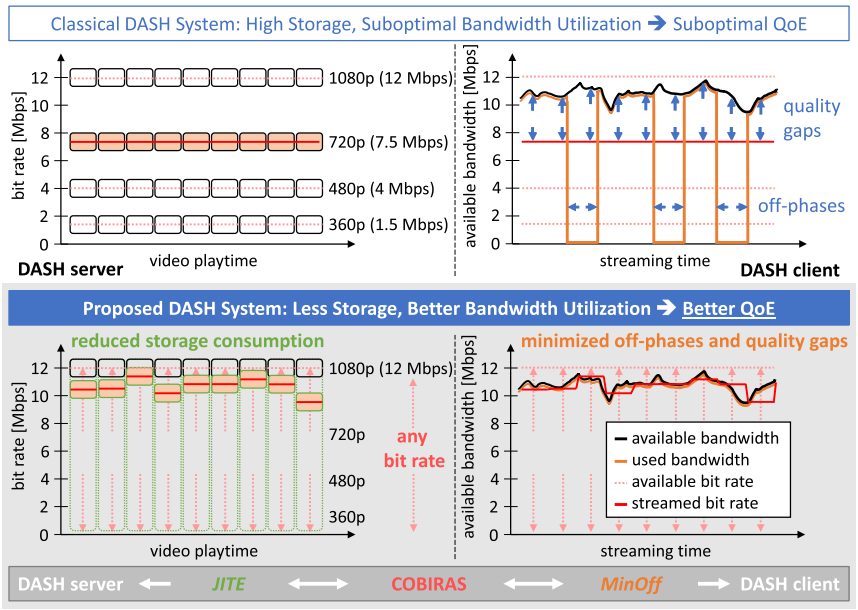


Fig. 1. Comparison of classical DASH system to proposed DASH system with continuous bit rate slide (COBIRAS), just-in-time encoding (*JITE*) at the DASH server, and *MinOff* at the DASH client.

content, which—for the sake of this numerical example—we will also consider to be the target bit rates of the corresponding representations.[1] The upper left part of Figure 1 shows this situation for a classical DASH server with representations from 360p to 1080p.

Now, if a client is provided a throughput of around 11 Mbps by the network provider, as shown by the solid black line in the upper right part of Figure 1, a conservative ABR algorithm needs to download the 7.5 Mbps representation, indicated by the solid red line, which leaves 3.5 Mbps of the available bandwidth unused. In practice, this results in a quality gap on the user side (vertical blue arrows) as the available bandwidth in the network would support streaming a higher video bit rate. Moreover, this leads to a faster download of the smaller segment and an earlier start of the download of the next segment. Due to the limited buffer size, however, the buffer will eventually run full earlier, and the DASH client will go to an off-phase, indicated by the blue horizontal arrows, in which available bandwidth cannot be used (solid orange line). This suboptimal bandwidth utilization of current DASH systems and the accompanying off-phases and quality gaps can lead to a suboptimal QoE of the end users. Although this behavior is in line with most research activities, which aim to minimize the usage of network resources while maintaining acceptable QoE, *we deliberately design our proposed DASH system to maximize bandwidth utilization, as there are many situations in which users require the best possible QoE and might even be willing to pay a premium to video streaming service and network providers.*

To maximize bandwidth utilization and enable QoE improvements in DASH systems, two possibilities arise. First, we can move from a small, discrete set of bit rate representations (bit rate ladder) to a continuous set of bit rate representations, referred to as **continuous bit rate slide (COBIRAS)**. As depicted in the lower left part of Figure 1, the flexibility of COBIRAS allows the DASH server to offer such a COBIRAS, which means that DASH clients can request and download segments

[1] In practice, the streamed bit rates may be much lower since YouTube recommends higher upload bit rates to avoid generation loss after re-encoding the input source.

in any arbitrary bit rate. This enables the streaming service to optimally align requested bit rate representations and available bandwidth for each client individually, which decreases the quality gaps and also enables a fine-grained reaction to changing network conditions. The result on the user side is depicted in the lower right part of Figure 1, where the streamed bit rate (solid red line) aligns closely with the available bandwidth (solid black line). The second possibility is that the DASH client stays longer in on-phases to make the most out of the available bandwidth. We tackle this by proposing *MinOff*, an ABR that *Min*imizes *Off*-phases during streaming. *MinOff* allows to keep the buffer fill around a certain target level by requesting appropriate video bit rates. In particular, to avoid off-phases, it can slow down the buffer filling by requesting segments with higher bit rates. In comparison to, e.g., buffer-based ABRs which aim to maintain a high buffer level near the maximum, this increases the duration of on-phases. Further, we also demonstrate the synergies of both possibilities, i.e., *MinOff* leveraging COBIRAS' flexibility to request segments in any arbitrary bit rate to jointly minimize off-phases and better align the requested bit rate representations and available bandwidth. Thus, compared to a classical DASH system, our proposed DASH system not only has the potential for an overall better bandwidth utilization, but also for a better QoE.

Considering a practical implementation and deployment of a DASH system with COBIRAS, we face the challenge that increasing the set of representations also increases the storage requirements, not only at the video server, but also in **content delivery networks (CDNs)** or caching infrastructure deployed inside the networks. Thus, in this work, we also design and implement a proof-of-concept DASH system with COBIRAS named **just-in-time encoding (JITE)** to demonstrate the feasibility of our approach. *JITE* is a DASH server, which can offer a continuous set of bit rate representations by **JITE** the requested representations from the source video. This means that, as shown in the lower left part of Figure 1, the source video is the only representation that is physically stored on the video server, while all segments can be requested and encoded in an arbitrary bit rate representation upon request from the client. This is also in line with the increasing deployment and availability of **graphics processing unit (GPU)** resources, which could perfectly be utilized for the encoding tasks. Considering the numerical example in Figure 1, the *JITE* DASH system, which stores only the 1080p representation, reduces the storage consumption by more than 50% compared to the classical DASH system with four pre-encoded representations.

The proposed system assumes a certain use case in which it may offer benefits over classical DASH streaming. First, our proposed DASH system requires fast compute resources for JITE of segments with minimal delay. Second, the resulting individually encoded segments have little value for caching, as they were created for a particular bitrate request. Therefore, the system is not suitable for the most popular content on large video platforms. In that case, pre-encoding segments with a discrete bit rate ladder and relying on state-of-the-art video distribution via CDNs is more cost-efficient. However, we envision the usage of COBIRAS and *JITE* in situations where the user base is small, such that sufficient compute resources can be deployed. Considering that content popularity in large video platforms follows a power law [8, 9, 17], this includes a vast amount of unpopular video content. For those, streaming providers currently pre-encode and store several bit rate representations, although there are few or no requests by end users. In this situation, deploying *JITE* for the unpopular content substantially reduces the traffic volume and storage requirements for video streaming service providers, CDN providers, and operators of backbone networks compared to current DASH systems.

Moreover, we envision the usage of *JITE* in situations where sufficient compute resources are available to support the user base, but storage is very limited, and when no caching infrastructure or CDN nodes are further down the road toward the client. In addition, there might be a high demand for excellent video streaming QoE. For example, a possible usage scenario is at the network edge, e.g., a base station within a mobile or campus network, or an edge node serving a train, plane, or

cruise ship. Here, the bandwidth between that node and the vehicle may undergo strong bandwidth fluctuations, and content may be encoded on-demand when it is requested by passengers.

To realize DASH with a COBIRAS at the network edge, only the source videos need to be distributed, and *JITE* could be deployed as a service function using the available compute resources to individually encode the requested representations for the connected end users. Considering the deployment at edge nodes serving a train, plane, or cruise ship, as storage consumption is substantially reduced by *JITE*, this would effectively allow offering larger video catalogues and more flexibility in onboard entertainment systems. At the same time, COBIRAS and *MinOff* would allow to deliver the best possible QoE by maximizing bandwidth utilization and minimizing quality gaps, e.g., to users willing to pay a premium for excellent quality entertainment.

In summary, the contributions of this article are as follows:

(1) Presentation of a DASH system with COBIRAS, including investigation of the impact of the size of the representation set on the bandwidth utilization, both theoretically, using an **integer linear program (ILP)**, and practically, using a testbed implementation.
(2) Design and performance evaluation of *MinOff*, a novel ABR algorithm, which maximizes bandwidth utilization by minimizing off-phases during streaming.
(3) Design and implementation of *JITE*, a proof-of-concept DASH system for video streaming, which can offer a COBIRAS and is realized by JITE, as well as its evaluation with respect to bandwidth utilization and video QoE of single video streams. For the sake of reproducibility and as an additional contribution, we make our testbed implementation publicly available at [48].

2 Background and Related Work

ABR algorithms take over the pivotal role of selecting an appropriate representation/bit rate for the next video segment from the available representations at the server in order to optimize the users' experience under current network conditions. However, optimal bit rate selection is an NP-hard problem [27] and an optimal solution can only be determined retrospectively, as during streaming the future evolution of the bandwidth is uncertain. Many ABR algorithms have been proposed in the literature [4], which mainly aim to deliver high video quality, but differ greatly in their approaches. For example, while Bola [52] does not even require a prediction of network bandwidth, the *Throughput* ABR algorithm relies on estimating the current throughput, the *Dynamic* ABR algorithm [51] combines both. Other algorithms were designed for specific scenarios, such as *L2A* [31], which considers low-latency streaming, or are based on more complex models, such as Pensieve [35], which uses reinforcement learning. Moreover, ABR algorithms can consider additional goals, such as saving data while targeting a specific video quality [41] or optimizing energy consumption [57]. However, while many ABR algorithms exist, which consider and want to maximize the client's throughput, the existing literature lacks in ABR algorithms that tackle maximizing the bandwidth utilization by minimizing off-phases during streaming. Note that the DASH Industry Forum provides a reference implementation [11] of an ABR client with existing algorithms that focus on QoE optimization, including Bola, Throughput, Dynamic, and L2A, which we will use as a benchmark.

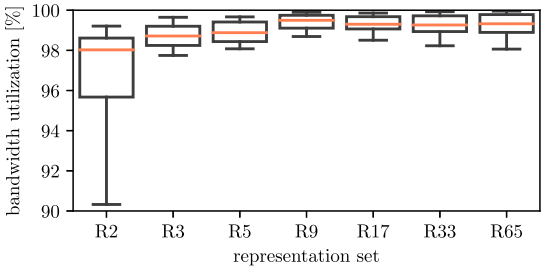
For content provisioning, a streaming service provider usually pre-encodes and provides multiple representations of the same video for the ABR algorithm to choose from, which is typically referred to as bit rate ladder. However, for unpopular videos, not all generated representations will be requested, resulting in unnecessary encoding, storage, and network costs for these representations [8, 9, 17, 56]. Although transcoding techniques for re-encoding an existing video source have long been around [10, 61], these challenges for modern DASH systems have motivated recent research on

the application of transcoding for generating different bit rate representations. These works often focus on video transcoding at the network edge as a means of cost-efficient content distribution, aggregating user requests [15], or trading-off storage cost for computation cost [14, 32]. [13] even proposed transcoding on end devices and P2P distribution with the aim of data saving and network offloading. In contrast to these works, our proposal *JITE* explores JITE or transcoding of a single source video representation at the streaming server to realize a COBIRAS. Note that the concept of JITE or transcoding has already been considered previously to reduce storage or bandwidth requirements and to ensure format compatibility [6, 10, 32]. However, we believe this contribution to be the first to focus on maximizing bandwidth utilization by offering a COBIRAS. The idea of real-time transcoding and streaming for file-hosting services has been briefly investigated by VSync [2], which is closest to our work. For this, the authors focused on accurately estimating transcoding times and resulting file sizes using machine learning models. In contrast to this work, we emphasize optimizing QoE by maximizing bandwidth utilization during streaming. Here, we rely on standard DASH components and demonstrate a practical implementation for real DASH deployments. Our system *JITE* is a generic approach, which could be integrated into existing streaming systems, and allows streaming providers to tradeoff storage for computational cost. This means that no other video representation needs to be pre-encoded and stored at the server, but all representations are dynamically generated upon the client's request.

For *JITE* to work, encoding must be performed as fast as possible, which ultimately depends on both the available computing resources at the streaming server and the selected video codec. Similar challenges on the encoding duration are well-known for live streaming. The difference is that standard live streaming DASH systems can encode immediately into the pre-defined discrete bit rate latter, while *JITE* DASH systems would only encode to the highest bit rate representation. When clients request a segment, further encoding processes would be triggered to be able to deliver the desired bit rate according to current network conditions. Nevertheless, *JITE* can potentially be integrated into a live streaming system, although this might require more computational resources relative to the number of subscribed clients. It is important to note that the additional encoding duration after a request may make the system unsuitable in scenarios where low latency live streaming is crucial [30, 58]. With respect to minimizing encoding duration, it is mandatory to select an appropriate video codec. [7] showed that VP9 and H.265 codecs are not suitable for transcoding at the edge due to their long encoding duration, while H.264 is well-suited for this task. [3] came to similar conclusions when comparing codecs for live gaming video streaming. Therefore, we built and evaluated *JITE* using H.264 as codec. Recently, [14] proposed to store information of the encoding process to speed up and reduce the cost of transcoding, which could potentially be considered in *JITE* in the future. Furthermore, [40] analyzes encodings in order to prioritize certain video frames during transmission, and potentially drops frames with minimal impact on QoE. While this could be used to improve our fallback strategy, more insights into the impact of these frame drops on QoE are necessary.

3 Impact of Size of Representation Set on Bandwidth Utilization

Theoretical Considerations. To theoretically investigate the impact of the size of the bit rate representation set on the bandwidth utilization of single video streams, we employ an ILP to find an optimal segment download strategy, which maximizes the download volume of a DASH stream for a given network trace. We define the ILP to maximize the utilized bandwidth while avoiding stalling, which is generally agreed to be the worst possible QoE degradation. Similar work has been shown in [25, 26, 37] while [54] focused on the streaming provider's perspective. However, in contrast to the aforementioned works, we increase the realism of our system model. To make the modeled system behave as close as possible to a real DASH system, we limit the streaming system



Set	Lowest Bit Rate	Second Bit Rate	Step	Highest Bit Rate
R2	314	-	19686	20000
R3	314	10157	9843	20000
R5	314	5236	4922	20000
R9	314	2775	2461	20000
R17	314	1544	1230	20000
R33	314	929	615	20000
R65	314	622	308	20000

Fig. 2. Bandwidth utilization for each representation set.

to a strictly sequential download of segments and a fixed buffer size, which has not been done in previous works. The resulting ILP and a detailed description of its formulation can be found in Appendix A. We solve the ILP using the `Gurobi Optimizer` [22].

As solving the ILP for constant bandwidth is trivial—the optimal set of representations would be the one containing the closest representation level to the bandwidth—for this analysis, we employ traces measured from an operational 4G/LTE mobile network [60]. These traces describe the available throughput as experienced by a single user in a realistic scenario, in particular, reflecting that the user is moving between cells and sharing the available cell capacity with other users. Thus, these traces are well suited for our evaluations. As the provided **Long Term Evolution (LTE)** traces have a higher network throughput than required for 4K video streaming, i.e., a resolution of 2160p, which would make the adaptation problem too simple, we divided them by a factor of 3. In total, we use 40 network traces covering different means of mobility, namely bicycle, bus, car, train, tram, and walking. Traces are between 165 s and 757 s long, and the network throughput values are provided every second. We concatenate each trace multiple times with itself and cut after 900 s, which gives an equal length of 15 min for all traces. The resulting traces average from 4.9 to 19.1 Mbps with throughput peaks to more than 30 Mbps. Two thirds of the traces have at least one second of outage, i.e., zero throughput, the average outages per trace are 10.05 s, and the maximum outages per trace are 62 s.

For our tests, we stream the *Tears of Steel* video [5]. To align the video to the the used LTE network traces, we define the maximum rate of our top quality 2160p representation level at 20 Mbps. After conducting simple encoding experiments using `FFmpeg`, we found the rate of the minimal quality 360p representation level at 314 kbps. We consider different representation sets, which exponentially increase their number of representations. Starting from R2, which is the set with just the top and minimal quality representations ({314, 20,000} kbps), we repeatedly insert bit rates in the middle of the range between two representations. In this way, we obtain sets R3 ({314, 10,157, 20,000} kbps), R5 ({314, 5,236, 10,157, 15,079, 20,000} kbps), and so on for R9, R17, R33, R65.

For the performance evaluation, we use a segment duration of $D = 4s$ and a buffer size of 20 s, which corresponds to $B = 5$ time slots. In addition, the initial delay was set to $T_0 = 4s$. We solved the optimization problem for all 40 network traces for the different representation sets. While most problems could be solved within 5 min on a 16-core `Intel Xeon Gold 5218` processor, some solutions took a very long time. Thus, we introduced a time limit of 60 min for solving a problem, which always left us with a feasible solution less than 0.05% below the approximated upper bound solution given by `Gurobi` in the pre-solve process [1]. Note that for one network trace, due to a low bandwidth section, no solution could be found that avoids stalling for any representation set.

Figure 2 depicts the distribution of the optimal bandwidth utilization in percent over all network traces for each representation set as a box plot. Each box extends from first to third quartile, whiskers from 10th to 90th percentile, and the median is highlighted in orange. The first box for the smallest set with only two representation levels shows that, in theory, not many representations

would be needed to achieve very high utilization. The median utilization of R2 is 98.03%, but results are much worse for some network traces, as indicated by the extended box and whiskers. An improved median utilization at 98.71% and highly reduced variance can be observed for R3 and this trend continues until R9, which has a median utilization of 99.50%. Afterwards, we observe a slight decline in utilization until R65. This behavior is counter-intuitive, given that R17, R35, and R65 are supersets of R9, meaning that they also contain all representations of R9. Thus, any solution for R9 is also a feasible solution for R17, R33, and R65, and adding additional representations cannot result in a worse solution, but can only improve the result. The reason for the counter-intuitive behavior is the time limit introduced for solving the ILP. Adding more representations makes the problem combinatorially harder, as the ILP now has more choices when selecting which segments to download in which representation. Thus, the ILP runs into the time limit more often for R17, R33, and R65, which gives results slightly below the optimal result. Nevertheless, the optimal solutions for R17–R65 actually have to be at least as good as solutions for R9.

In summary, our ILP results show that in theory a very small number of representation levels can already be sufficient to reach an excellent bandwidth utilization in DASH video streaming. Adding more representation levels just allows for a marginal improvement, which may not justify the additional encoding and storage effort required to provide these representation levels. These results are in line with [37], which found no difference in the optimal bandwidth utilization when streaming with 6 or 14 representations, although their ILP was based on a more unrealistic DASH system as discussed above. Nevertheless, the results from our ILP are theoretical results requiring significant knowledge of the future bandwidth evolution, which is hard to predict, and thus, cannot be assumed in a practical DASH system. Thus, next, we will investigate the impact of the representation set using a measurement study in a realistic testbed.

Testbed Evaluation. We base our testbed on a `Vagrant VirtualBox` **virtual machine (VM)** setup published in [45], which has been created for the analysis of the influence of different segment durations in DASH streaming [44]. As the `Vagrant` VM setup was quite complex, we recreated it as a multi-container setup in a `Docker` environment to utilize advantages of the `Docker` ecosystem, namely, a smaller and more flexible footprint in terms of CPU cores, memory, and disk space, and a reduced time to implement and run measurements. The resulting testbed is able to configure the DASH system, control network conditions, and run measurements in a fully automated way. It consists of three containers, i.e., the *server* hosting the video segments, the *client* using standard `dash.js` to request and play the video, and a *network emulator* regulating the available bandwidth between server and client. All technical details are given in Appendix B.

We compare the performance for two representation sets using the same LTE traces as for the ILP. The first representation set *FewReps* contains six representations, one for each of the six resolutions, with target bit rates of 20,000 kbps for 2160p, 9,000 kbps for 1440p, 4,600 kbps for 1080p, 2,150 kbps for 720p, 1,050 kbps for 480p, and 570 kbps for 360p. For each resolution reduction, the bit rate approximately halves. To clearly see the impact of increasing the number of offered representations, the second set *ManyReps* approaches a continuous set of bit rate representations with a high number of representations. For this, we encode different *crf* values for each resolution except for the 2160p source representation, which is again available only with bit rate 20,000 kbps. For the other resolutions, we perform the encodings, measure the average resulting file sizes, and set maximum bit rate levels at 103% and 110% for each combination, which gives the final bit rate levels. They range from 15 180-7,768 kbps for 1440p, from 6,906 to 3,350 kbps for 1080p, from 3,064 to 1,645 kbps for 720p, from 1,568 to 759 kbps for 480p, and from 708 to 314 kbps for 360p. The lowest bit rate is 314 kbps, which is why it was already used in the ILP. This process results in 49 representations with an average jump of 9% in bit rate between levels. Note that an actually COBIRAS would allow to request a segment in any bit rate, but we consider our set of 49 representations with only small

Table 1. Impact of Offering More Representations in DASH
(*FewReps*: 6 reprs., *ManyReps*: 49 reprs.)

ABR Algo.	Repr. Set	Bandw. Util. [%]	Qual. Swit. [−]	Stall. Dur. [s]	Downl. Dur. [s]	Avg. Buffer [s]
Bola	FewReps	61.32	12.78	2.85	485	17.84
	ManyReps	74.18	**50.92**	8.43	573	**15.99**
	Δ	+12.86	+38.14	+5.58	+88	−1.85
Throughput	FewReps	61.72	10.25	6.38	487	17.51
	ManyReps	85.64	54.00	16.55	640	12.75
	Δ	+23.92	+43.75	+10.17	+153	−4.76
Dynamic	FewReps	62.59	**9.90**	6.62	494	17.47
	ManyReps	75.67	52.20	14.46	583	15.09
	Δ	+13.08	+42.30	+7.84	+89	−2.38
L2A	FewReps	57.03	30.18	**1.73**	446	**18.67**
	ManyReps	85.56	100.45	7.28	637	14.08
	Δ	+28.53	+70.27	+5.55	+191	−4.59
MinOff	FewReps	**90.66**	85.00	2.69	**671**	13.41
	ManyReps	**91.55**	120.72	**4.50**	**675**	10.70
	Δ	+0.89	+35.72	+1.81	+4	−2.71

The bold values indicate the best value in each column for FewReps and for ManyReps. Algo., algorithms; Avg. Buffer, average buffer; Bandw. Util., bandwidth utilization; Downl. Dur., download duration; Qual. Swit., quality switches; Repr., representations; Stall. Dur., stalling duration.

differences in bit rates to be a sufficient approximation for the purpose of this study. We also use this approximation for practical reasons, as it is not possible to implement a fully COBIRAS without major modifications to standard `dash.js` and DASH manifests.

We conduct measurement runs streaming the *Tears of Steel* video for all 40 LTE traces and all ABR algorithms. Although we just perform evaluations for the single *Tears of Steel* video in this work, we want to note here that our results should generalize and should apply also when other videos are used. The reason is that, depending on the available throughput, all videos would in the end be encoded into representations that have (within certain limits) very similar bit rates. Still, dedicated studies with other videos have to be conducted in future work to confirm this assumption. Table 1 lists the average results of each ABR algorithm for the most relevant metrics, namely, bandwidth utilization, number of **quality switches (QS)**, stalling duration, download duration, and average buffer size. To exclude any impact from buffer depletion at the end of the video, we limit our analysis to the first 700 s of media playback time. We can clearly see consistent trends for all ABRs when the representation set changes from *FewReps* with 6 representations to *ManyReps* with 49 representations. First, all standard ABRs can increase the bandwidth utilization by many percent points with more representations, which is challenging the results from the ILP, but is more in line with the intuition that having a larger set of representations to select from allows to better align the bit rate to the available bandwidth. However, this additional flexibility comes at the cost of having a higher number of QS, which is expected. Therefore, it has to be investigated how these QS affect the QoE of the end user, which we will do below for our *JITE* system. When we look at the standard ABR algorithms in detail, Dynamic reaches the best bandwidth utilization for *FewReps*, however, a utilization of 62.59% is mediocre, especially considering the ILP results. For *ManyReps*, Throughput reaches the best bandwidth utilization of 85.64%, however, it is still far below the ILP results, and also gives the largest amount of stalling. The live streaming ABR algorithm L2A benefits the most from having a larger representation set. It is able to increase its

bandwidth utilization by 28.53 percent points from 57.03% for *FewReps* to 85.56% for *ManyReps*, however, also at the cost of more than four times longer stalling.

Generally, we see a highly increased stalling duration when more representation levels are available. This is a surprising result at first sight, which might be due to the fact that requesting bit rates too close to the current bandwidth does not leave a safety margin for sudden bandwidth drops. In fact, clients request bit rates closer to the current bandwidth, such that the download times of segments grow, which can be seen in the sixth column. This leads to a reduced average buffer size, as can be seen in the last column, and thus, the streaming becomes more susceptible to stalling. This is a severe problem, which will negatively influence the QoE of end users. On the other hand, the results also suggest that a longer download duration and a smaller average buffer size are associated with a higher bandwidth utilization. The reason is that clients whose buffer is full will enter an off-phase in which nothing is downloaded, and thus, no bandwidth is utilized. This means, from a bandwidth utilization perspective, it seems desirable that ABR algorithms avoid entering off-phases by avoiding to fill up the buffer entirely, although this might lead to a higher risk of stalling.

Overall, we see that the theoretically possible bandwidth utilization that was found by the ILP cannot be reached by the ABR algorithms in the testbed. Instead, the resulting bandwidth utilization for the tested DASH systems using a set of six representations was mediocre. Nevertheless, we saw that, in practice, the bandwidth utilization of the DASH systems can be substantially increased when using 49 representation levels, as the requested bit rate can be better aligned to the current bandwidth. However, this comes at the cost of a higher number of QS and a higher risk for stalling, which can negatively influence the QoE. Finally, we identified the ABR algorithms' tendency to entirely fill up the buffer and enter off-phases as being detrimental with respect to maximizing the bandwidth utilization. Thus, in the following, we will propose a novel ABR algorithm, *MinOff*, which will minimize the off-phases during streaming.

4 *MinOff*: Novel ABR for Minimizing Off-Phases during Streaming

Motivated by the results presented in Section 3, we design a novel ABR algorithm, *MinOff*, which tries to maximize the requested and downloaded video bit rate and to avoid stalling, but at the same time does not completely fill the buffer to minimize off-phases, and thus, to increase bandwidth utilization. We also leverage the envisioned deployment scenario in a DASH system, which offers a COBIRAS, such that *MinOff* will output the bit rate of the next requested segment as a continuous value. Note that when using *MinOff* with a discrete bit rate ladder instead, the continuous output needs be matched at the server to the available representation, which has the highest bit rate below the output.

We start off with the Dynamic ABR algorithm and its baseline for selecting the bit rate of the next segment, which is $tp_{s-4,\dots,s-1}$, i.e., the average download throughput of the last four segments. In our case, this would consider the last 16 s, which is rather long for rapidly changing mobile connections. In addition, a bandwidth increase could lead to the rapid download of intermediate representations, as Dynamic up-switches rather conservatively. The buffer would be filled with said representations rather fast and the client would enter an off-phase with zero bandwidth utilization.

Thus, our proposed *MinOff* algorithm modifies Dynamic to additionally factor in a throughput ratio $tpr = \frac{tp_{s-1}}{tp_{s-4,\dots,s-1}}$, i.e., the throughput of the last segment divided by the the throughput of the last four segments, to allow for a faster up-switching when network conditions increase. However, to avoid requesting too high bit rates when a sudden bandwidth spike appears, we use the dampening function $f(tpr) = 2 \cdot (1 - 0.5^{tpr})$. It holds $f(tpr) \approx tpr$ for $tpr \in [0, 1]$ and $f(1) = 1$, as throughput reductions or stable throughput should not alter the factor, while $tpr >> 1$ are substantially reduced.

Table 2. Desired Impact of Buffer on Representation Selection

	Buffer Size		
	Low	**Medium**	**High**
Objective	Avoid stalling	Carry on as before	Reach higher quality level
Up-switch	conservative	moderate	aggressive
Down-switch	aggressive	moderate	conservative

Our testbed measurements showed tpr values mostly below 3, which would be dampened to a factor of $f(3) = 1.75$.

In addition, we add a second factor based on the current buffer size bs in seconds. Depending on the buffer size, we target a different objective. As shown in Table 2, while having a low buffer size, stalling should be avoided at all costs to avoid bad QoE. In contrast, having a high buffer size should lead to downloading better quality levels to fill the buffer slower or even reduce its level. To consider this behavior in the decision for the next representation request, we propose a continuous piecewise function g with parameters $a_1 > a_2 \geq 0, a_3 > 0$ to map bs into a meaningful factor:

$$g(bs) = \begin{cases} (1 + \exp(-a_1 \cdot \frac{bs}{tb} + a_2))^{-1} & \text{for } bs \leq tb, \\ a_3 \cdot (bs - tb)^2 + g(tb) & \text{for } bs > tb. \end{cases}$$

From the definition of g, it is clear that we use a target buffer level tb at which the resulting factor $g(tb)$ should be close to 1, and thus, does not substantially alter the requested bit rate. The first part of g is a sigmoid of a linear function used for $bs \leq tb$. For small values of bs, it returns a factor below 1, and even close to 0 for a very small buffer, as a low bit rate for the next request is desired to reduce the risk of stalling. The second part of g is a quadratic function used for $bs > tb$ to avoid a full buffer and resulting off-phases for large bs. The quadratic increase of the next segment's bit rate will not only result in a higher visual quality for the end user, but also enlarge the download duration of the segment, such that the buffer will grow slower or even reduce toward the target buffer level. The function has a constant additive offset $g(tb) = (1 + \exp(-a_1 + a_2))^{-1} \approx 1$ to ensure that g is a continuous function. Note that, when a segment is requested, the last segment download has just been completed, so in our scenario, the buffer cannot be below 4 s. On the other hand, the client will not request a segment when the buffer is full at 20 s. Thus, the relevant domain of our function g is essentially 4–20 s. We will consider a target buffer level tb =11 s, which is not too low to avoid a high risk for stalling and not too close to the buffer size to avoid off-phases. We experimentally found that the best parameters for our scenario are $a_1 = 9.9$, $a_2 = 6.3$, and $a_3 = 0.02$, resulting in a factor ranging from 0.0630 for $bs = 4$ s to 0.9736 for $bs = tb = 11$ s to 2.5934 for $bs = 20$ s. This contributes to very low bit rates if the buffer is low, fast increase to the plateau around tb, but also not too aggressive up-switching if the buffer fills above tb. Note that the parameters can be adjusted if other buffer sizes and target buffer levels are used or a different ABR behavior is desired.

We multiply the baseline bit rate, i.e., the average throughput of the last four segments, with $f(tpr)$ and $g(bs)$ to yield the final bit rate for the next segment request: $br_s = tp_{s-4,\ldots,s-1} \cdot f(tpr) \cdot g(bs)$ Figure 3 shows the resulting modification of the baseline bit rate by each factor as a contour plot. The color map colors the resulting bit rate factor on a scale from 0 (dark blue) to 1 (white) to 4.5 (dark red), thus, the baseline bit rate is reduced in blue areas, while it is increased in red areas. The black lines are isolines, which have the same resulting bit rate factor, plotted in steps of 0.5. It can be seen that the buffer size has a larger impact on the resulting modification than the throughput factor. For buffer sizes below 5 s, it dictates to reduce the requested bit rate to avoid stalling, and above 15 s, it greatly increases the requested bit rate to avoid off-phases. The impact of the throughput factor is most visible between 0.5 and 1.5 for buffer sizes around or above tb.

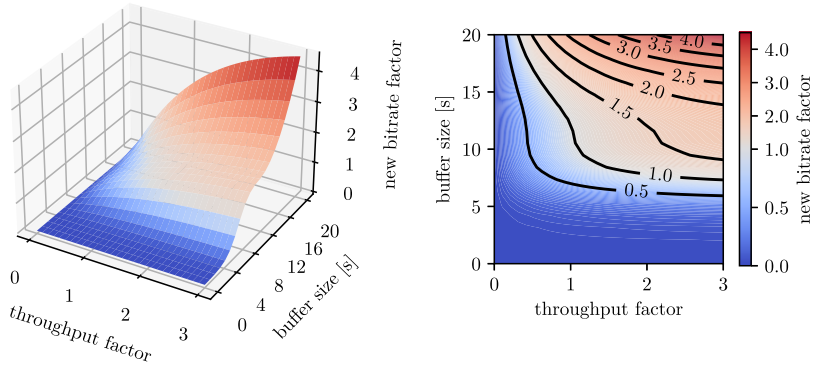
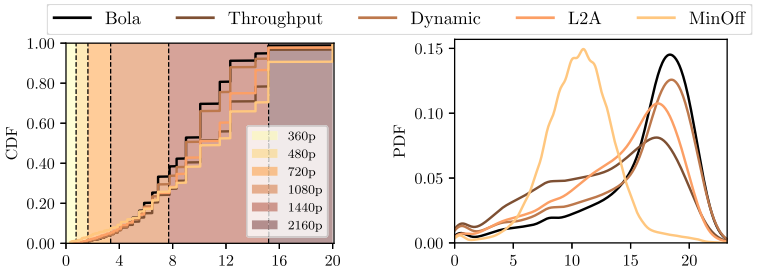


Fig. 3. Impact of *MinOff* factors on baseline bit rate.



(a) Segment bit rate [Mbps] (b) Buffer size [s]

Fig. 4. Performance comparison of *MinOff* for *ManyReps*.

We can see the testbed results of *MinOff* compared to the other ABR algorithms evaluated in the last row of Table 1, where *MinOff* reaches the highest average bandwidth utilization above 90% with both representation sets. For *FewReps*, this means an increase of over 28 percent points compared to Dynamic, and for *ManyReps*, it is still roughly 6 percent points higher than second place L2A. This is due to successfully fulfilling our objective of minimizing off-phases by downloading higher bit rate representations, which can be seen when looking at the download durations and at the **cumulative distribution function (CDF)** of the segment bit rates in Figure 4(a). We can also observe that the average buffer size is much lower than for the other ABR algorithms, and that it is closer to our target buffer level of 11 s. In Figure 4(b), the PDFs of the buffer size show that *MinOff* effectively is able to keep the buffer size around the target buffer level, while the other algorithms often fill the buffer, and thus, trigger off-phases.

Although operating on a lower buffer size, *MinOff* maintains a low level of stalling. As can be seen in Table 1, for *FewReps*, it has an average stalling duration of 2.69 s, which is second place behind L2A, but for *ManyReps*, it reaches the lowest stalling duration of 4.50 s, 40% lower than second place L2A. This shows that our ABR algorithm design is able to achieve its second goal of avoiding stalling. Finally, we see that *MinOff* has the highest number of QS compared to the other algorithm. While this indicates a very good alignment of requested bit rate and currently available bandwidth, it can also have a negative effect on QoE if too many QS are visible to the users. We expect that QS will be more noticeable for *FewReps*, and will become less visible for *ManyReps* or generally in cases with a COBIRAS as the algorithm might switch between close representation levels. However, we will evaluate the QoE impact of the additional QS introduced in Section 6.

To conclude, we created *MinOff*, a custom ABR algorithm, which not only considers the average throughput of the last segments as it is done by dash.js' Dynamic ABR algorithm, but additionally takes the latest throughput and the current buffer size into account. Since the baseline bit rate is multiplied by two continuous factors, *MinOff* benefits from a setup with a COBIRAS, where it can precisely request the desired bit rate on a continuous scale. The testbed evaluations show that *MinOff* could avoid off-phases, which resulted in the highest bandwidth utilization and the highest video bit rate, while stalling could be minimized.

5 Design of *JITE* DASH system

To demonstrate the feasibility of rolling out a DASH system with COBIRAS, we present the design of the *JITE* proof-of-concept DASH system, which requires only segments of the highest representation of the source video at the server. The client can request a representation of any bit rate from the COBIRAS, according to the used ABR. This means that instead of having a fixed discrete set of representation levels, any continuous value is possible. The server will then carry out the encoding of the video segment from the corresponding segment of the source video at runtime, i.e., while the streaming is ongoing.

The simplest procedure might consist of the server receiving a request, then encoding the requested segment, and finally sending it to the client. However, this would lead to a huge encoding delay in which no data is transferred. This would negatively affect bandwidth utilization and might cause stalling if not enough buffer was built up at the client. Although the encoding delay could be reduced to some extent by loosening encoding settings, this would result in poorer visual quality.

To fully utilize the bandwidth and hide the encoding delay from the client, our *JITE* system is designed to have a segment ready for download immediately upon receipt of a client's request. To achieve this, we implement a predictive encoding strategy anticipating that the bit rate requested for the subsequent segment will be the same as for the last requested segment, i.e., assuming short-term throughput stability. Hence, we already predictively encode the next segment with the bit rate requested for the last segment, except when the highest quality level is requested (this source representation is available at the server for all segments). However, a sudden drop in network throughput would increase the download duration, and thus, the risk of stalling. To counteract this behavior, for every request our system encodes an additional "safety" representation at one third of the bit rate of the predictively encoded segment. Although only one representation will be sent to the client for each segment, the additional resource consumption of simultaneously encoding a second representation should be marginal compared to having an increased risk of stalling and resulting bad QoE. Moreover, the resolution of the safety representation is reduced, thereby consuming even fewer resources. There are three different cases depending on whether the requested bit rate (rbr) is (1) the same or a higher or (2) lower than the last requested bit rate ($lrbr$):

(1) $rbr \geq lrbr$**:** We send the predictively encoded segment even though its bit rate is lower than the requested bit rate. The bit rate of the downloaded segment ($lrbr$) might be suboptimal with respect to the throughput (rbr) which could negatively affect the QoE. However, given that the bit rate is lower than the throughput, buffer depletion will be avoided.

(2a) $0.5 \cdot lrbr \leq rbr < lrbr$**:** We nevertheless send the predictively encoded segment. The download might take longer than expected, and thus, might deplete the buffer by up to one segment length assuming the worst case that the throughput (rbr) is half of the segment bit rate ($lrbr$). This is not an issue if the system is operating normally around the target buffer.

(2b) $rbr < 0.5 \cdot lrbr$**:** We send the safety representation as the observed throughput (rbr) has severely deteriorated compared to the bit rate of the predictively encoded segment ($lrbr$). The download of the predictively encoded segment would substantially increase the risk of stalling.

As we cannot apply our predictive encoding strategy for the very first segment of a video, we fall back to the simple procedure described above, i.e., we start the encoding after the first request is received. While the first segment is downloaded, the JITE of the second segment takes place, and the *JITE* system continues smoothly from there, as described above. In on-demand video streaming, the additional encoding delay of the first segment only slightly increases the initial delay, which does not have a severe impact on QoE [12, 24, 46]. For live streaming, we would need to account for a live delay of one encoding duration and one segment length, which confirms that our system might be impractical in scenarios which require very low latency live streaming.

As discussed above, we approximate a fully COBIRAS by implementing the *ManyReps JITE* DASH system in our testbed, which is *JITE* limited to the *ManyReps* representation set. For this, we only need to modify the *server* container and the manifest file. Nevertheless, we expect all performance results to be similar to a *JITE* system with a truly COBIRAS.

When implementing a *JITE* DASH system, the encoding duration has to be constrained to be at most as long as the segment duration. The reason is that when having a perfect alignment of video bit rate and current bandwidth, the download of one segment will take exactly one segment duration, and thus, the next segment has to be encoded and available at the latest by then. To have some margin for sudden bandwidth spikes, we aim at limiting the encoding duration to half of the segment duration. To find suitable `libx264` presets for JITE with `FFmpeg` in our *server* container, we measured the encoding speed for all presets and resolutions in a pre-study. We determined different preset values, which meet the time constraint and provide a reasonable balance of compression and encoding speed. The resulting mean encoding time of a segment is 2.3 s, with the 95th percentile being at 2.6 s. For creating DASH segments just-in-time, we only create one segment instead of the whole video, which requires to adjust some metadata in the segment headers. Besides that, we use default `MP4Box` settings, as we try to keep the DASH setup as common as possible. In our testbed, creating a DASH segment takes 140 ms to 160 ms, which adds a small extra delay to the encoding duration.

6 Performance Evaluation of *JITE*

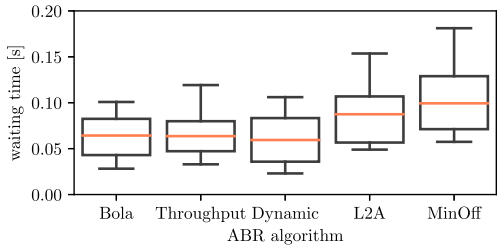
Impact of JITE. Table 3 compares the results of our testbed measurements for *ManyReps* and *ManyReps JITE* using the same LTE traces as above. This comparison is sensible as *ManyReps* can be seen as a perfect JITE, i.e., it contains no disadvantages introduced by JITE. Note that to have a comparable streaming start in both DASH systems, *ManyReps JITE* also delivers pre-encoded representations of the first segment and JITE is used only from the second segment. The results show that JITE in our *JITE* system has only a small negative impact on the performance compared to *ManyReps*. Thus, the bandwidth utilization of *JITE* is still much higher for all ABR algorithms compared to *FewReps*, i.e., to classical DASH systems.

When looking in detail at the negative impact of encoding representations just-in-time, in particular, we see that all ABR algorithms show a small decrease in bandwidth utilization and an increase in stalling. Only L2A is able to slightly reduce stalling by 0.71 s on average, but it also has the largest decrease in bandwidth utilization of 11.50 percent points. Our ABR algorithm *MinOff* still has the highest bandwidth utilization, the highest download duration, and the least amount of stalling. However, the number of QS increases again for *MinOff*, while Bola still has the least. Still, the number of switches is much higher than for *FewReps*.

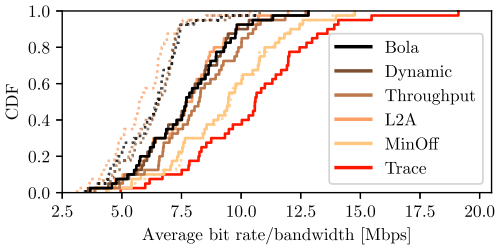
We now take a closer look at the performance of the *JITE* DASH system. First, we investigate the waiting time for segment encodings, which occurs when a segment encoding is not finished before the next segment is requested. This waiting time increases the risk of stalling and can be considered unused bandwidth. The results show that the client has to wait an additional 0.08 s on average for the encoding of the segment to complete with *JITE*. In addition, Figure 5(a) shows

Table 3. Impact of Just-in-Time Encoding in *JITE*

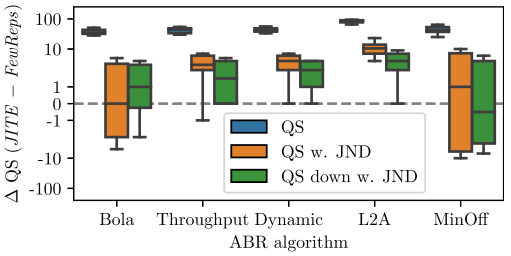
ABR Algo.	Repr. Set	Bandw. Util. [%]	Qual. Swit. [−]	Stall. Dur. [s]	Downl. Dur. [s]	Avg. Buffer [s]
Bola	ManyReps *JITE*	73.36	**48.20**	12.86	567	15.47
	Δ to ManyReps	−0.82	−2.72	+4.43	−6	−0.52
Throughput	ManyReps *JITE*	77.28	51.10	20.32	591	14.48
	Δ to ManyReps	−8.36	−2.90	+3.77	−49	+1.73
Dynamic	ManyReps *JITE*	73.04	51.25	17.03	565	15.28
	Δ to ManyReps	−2.63	−0.95	+2.57	−18	+0.19
L2A	ManyReps *JITE*	74.06	111.35	6.57	562	**16.20**
	Δ to ManyReps	−11.50	+10.90	−0.71	−75	+2.12
MinOff	ManyReps *JITE*	**89.63**	123.83	**5.64**	**660**	11.86
	Δ to ManyReps	−1.92	+3.11	+1.14	−15	+1.16



(a) Waiting time for encoding end per segment



(b) Average video bit rate per session



(c) Difference in quality switch events

Fig. 5. Performance evaluation of *ManyReps JITE*.

the boxplots of the distributions for all ABR algorithms. We can see that L2A and *MinOff* have a longer waiting duration compared to the other ABR algorithms. Tracing the testbed log files, we see that both ABR algorithms perform larger QS compared to the other ABR algorithms. When the network throughput shows a large increase, the client requests a large quality up-switch, but our *JITE* DASH system sends the segment in the previously predictively encoded lower bit rate, which can be downloaded very fast. Following the requested large quality up-switch, the next segment has to be encoded in a higher bit rate. If the encoding of the larger segment is not finished when the download of the smaller previous segment is finished, the waiting time occurs. Such a situation might also occur when the safety representation is used. Nevertheless, looking at the average stalling durations in Table 3, we can see that the additional waiting time just slightly increased stalling and slightly reduced bandwidth utilization.

Impact on QoE. To investigate the resulting video streaming QoE in our LTE scenario, we again consider *ManyReps JITE* as an approximation of a DASH system with a truly COBIRAS. First, we evaluate the **mean opinion scores (MOS)** as predicted by the standardized QoE model ITU-T Rec. P.1203.1 Mode 0 [29] for *JITE* and compare them to *FewReps*. Specifically, we use code from the publicly available GitHub repository [23, 42, 43] to receive an estimate of the MOS on a scale from 1 (bad) to 5 (excellent) for each measurement run. Note that our streaming setup operates outside the resolution (higher than 1080p) and the video length (longer than 300 s) supported by P.1203. Nevertheless, we present results for the first 300 s of streaming considering a display resolution of 2160p and assuming perfect audio quality, since P.1203 is the only standardized model to jointly consider the most important QoE factors, i.e., initial delay, frequency and length of stalling, as well as level and switches of bit rate/visual quality [49]. Overall, the results show that MOS values for *FewReps* are slightly higher than for *JITE*, however, the absolute differences are marginal. Moreover, we do not find any substantial differences with respect to the ABR algorithms neither for *FewReps* nor for *JITE*. For *FewReps*, Dynamic has the highest average score of 4.05, while Bola achieves the highest average score of 3.95 for *JITE*. We conduct a paired t-test on the difference between the MOS for *FewReps* and *JITE*, which shows no significant difference for Bola and *MinOff* at the 5% significance level. For the other algorithms, there is a significant but very small MOS difference ranging on average from −0.08 (L2A) to −0.20 (Dynamic). As P.1203 is very sensitive to stalling [49], from these results, we can at least rule out a major negative impact of *JITE* on the QoE despite slightly increased stalling. However, since we assess resolutions above the supported range of P.1203 and obtain similar numerical results for both DASH systems and all ABR algorithms, we cannot see a clear benefit of the increased bandwidth utilization, and thus, increased visual quality with a COBIRAS, or a clear degradation by the increased number of QS.

To provide at least a quantitative evaluation of the resulting visual quality, Figure 5(b) shows the CDF of the average bit rate of the streamed video sessions, solid lines for *JITE* and dotted lines for *FewReps*. The results show that using *JITE* results in a substantial increase in video bit rate compared to a classical DASH system with few representations. For the standard ABR algorithms, the average increase ranges from 1.31 Mbps for Dynamic to 1.89 Mbps for Throughput. We expect that such large increases of the video bit rate will have a large positive impact on the QoE, despite the P.1203 results discussed above. Our ABR algorithm *MinOff* shows very good results for both DASH systems with overlapping CDFs and is clearly superior to standard ABR algorithms. Its achievable bit rate is 9.37 Mbps on average, by far higher than the best standard ABR algorithms, which achieve only 6.42 Mbps (Dynamic) for *FewReps* and 8.21 Mbps (Throughput) for *JITE*. Moreover, its distribution aligns much better with the corresponding trace bandwidth distribution depicted in red, which confirms its strong performance.

Finally, we investigate the additionally introduced QS in detail by applying the **video multi-method assessment fusion (VMAF)** [34] method, which assesses the visual quality of the video. We compute the VMAF scores for each segment streamed in our testbed. In contrast to classical DASH systems with few representations where there is a noticeable gap in bit rate and quality between the representations of a segment, changes in bit rate from segment to segment can be very small with continuous representations, and QS may not even be noticeable to the user. The smallest unit which can be perceived by a user—the so-called **just noticeable difference (JND)**—can be assumed for a change in VMAF of ± 6 [39], and the larger the absolute VMAF difference, the more noticeable a quality switch is. We now compare the paired difference in QS (Δ *QS*) from *FewReps* to *JITE* for each network trace. The boxplot in Figure 5(c) shows the distribution of this difference for each ABR algorithm. In blue, we see that the absolute number of QS increases when using *JITE* instead of *FewReps*, which we already saw in Table 1. This is expected, as there are more representations to choose from for the ABR algorithms. We now consider only noticeable QS, i.e.,

QS where the VMAF difference is at least one JND, which is shown in orange. For Bola, Throughput, Dynamic, and *MinOff*, the median difference in noticeable QS is between zero and four QS per run, while L2A has a median increase of 11 QS. Although direction and amplitude of the quality switch have the biggest impact on QoE [33, 38, 46], an abrupt up-switch might even improve QoE [19, 53]. Thus, the distribution depicted in green considers only down-switches, which have a negative impact on QoE. While the median difference between *FewReps* and *JITE* slightly increases for Bola from zero to one, for the other ABR algorithms it reduces to values ranging from one to four. For *MinOff*, the distribution spans similar positive values, but extends more toward negative values. We can even see a negative median of −1, which means that *JITE* can even reduce the number of noticeable quality down-switches in most situations.

Lessons learned. We conclude that although using a COBIRAS leads to an increase in QS as expected, most of the additional switches cannot be perceived as negative by users, so their impact on the QoE should be low. Thus, the increased bandwidth utilization and video bit rate, which can be achieved in DASH systems offering COBIRASs, should dominate and overall result in an improved QoE for end users compared to DASH systems with few representations.

7 Conclusion

In this work, we targeted the goal to maximize bandwidth utilization in DASH systems by minimizing download off-phases during streaming and by offering a COBIRAS, i.e., a continuous set of bit rate representations. For this, we first investigated the impact of adding more representations on the bandwidth utilization of DASH systems both theoretically using an ILP, and practically using a containerized testbed, which we published in [48]. We found that while few representation levels are sufficient in theory to achieve an excellent alignment of video bit rate and fluctuating available bandwidth in mobile networks, in practice, DASH systems can benefit a lot from a larger number of bit rate representations, which provides a strong argument for offering continuous bit rate scales in DASH systems.

Moreover, we identified the ABR algorithms' tendency to entirely fill up the buffer and enter off-phases as detrimental to maximizing bandwidth utilization. Thus, we proposed a novel ABR algorithm, *MinOff*, which minimizes off-phases by keeping the video buffer at a target buffer level below the maximum buffer size. The testbed results showed that *MinOff* is capable of increasing bandwidth utilization from slightly above 60% for standard ABR algorithms to more than 90% in a classical DASH system, resulting in the highest bit rates. At the same time, *MinOff* was able to effectively avoid stalling. This comes at the cost of an increased number of QS, which could potentially degrade the QoE in DASH systems with few bit rate representations, where QS might be more noticeable, but would not pose a QoE degradation for DASH systems with a continuous bit rate slide.

In addition, we designed a proof-of-concept DASH system for video streaming, *JITE*, which can offer a COBIRAS by JITE the requested representations from the source video. This allows offering arbitrary bit rate representations while only storing a single source video representation on the video server, and thus, is well suited for situations where compute resources are available but storage is very limited, e.g., at the mobile network edge. We implemented *JITE* using standard tools (`dash.js`, `FFmpeg`, `MP4Box`) and found that JITE has only a negligible negative impact on the performance.

Finally, we evaluated the impact of offering a COBIRAS in our testbed using a set of LTE traces. For this, we compared *JITE* offering an approximated bit rate slide with a large set of bit rate representations (*ManyReps JITE*) to an ideal system without any negative impact of JITE (*ManyReps*) and to a state-of-the-art DASH system with a small set of bit rate representations, i.e., a bit rate ladder (*FewReps*). We found for all investigated standard ABR algorithms that the

bandwidth utilization could be increased by 13–29 percent points when many representations were offered. This also led to a substantial increase in the average bit rate of the video session by 1.31–1.89 Mbps on average, which is expected to improve the visual quality, and thus, the resulting QoE. This demonstrates the advantages of COBIRAS to DASH systems. However, we observed an increase in stalling and QS, which required further investigation.

For this, we evaluated the resulting QoE of *JITE* in terms of P.1203 and concluded that the slightly increased stalling does not have a major negative impact on the MOS. With respect to QS, we analyzed VMAF scores and found that, despite the drastically increased number of QS in *JITE* compared to DASH systems with few representations, the number of actually noticeable quality down-switches, which constitute a QoE degradation, at most increased slightly for standard ABR algorithms. Considering *MinOff*, we even observed reductions in noticeable quality down-switches, such that at least half of the video streaming sessions experience at least one noticeable down-switch less. This shows that the increased number of QS should not have a major negative impact on the QoE. Thus, overall, the interplay of *MinOff* and a COBIRAS improved DASH bandwidth utilization without any major QoE degradation introduced by slightly increased stalling and a higher number of QS, such that QoE can be expected to improve due to the substantially increased video bit rate and consequently improved visual quality.

In future work, we plan to optimize our ABR algorithm *MinOff* to limit the amplitude of QS, and thus, to particularly avoid noticeable quality down-switches, in order to further improve the QoE. Considering our *JITE* DASH system and given the promising results in this work, we plan to implement a truly COBIRAS, to experiment with higher safety representations, to implement and evaluate live streaming use cases, to investigate the interactions, performance, and fairness when multiple users compete for bandwidth on the same link, and to optimize the encoding process to reduce waiting times and improve video quality. For this, we want to investigate the integration of two-pass live encoding [36] and GPU-offloading. We also plan to evaluate the *JITE* DASH system in dedicated subjective QoE studies on large display sizes, and we want to combine our approach of offering a COBIRAS with variable segment lengths as proposed in [44] to create a DASH system with full flexibility in both dimensions, i.e., temporal and spatial dimension.

Appendices

A ILP Formulation

Table 4 provides the notation required to formulate the ILP. The considered video is divided into N segments of duration D. Each segment is available in R representation levels, such that $S_{i,j}$ contains the size of segment $i \in N$ in representation $j \in R$. The function $V(t)$ represents the volume of the considered network trace from the beginning until point in time t. T_0 is the initial delay, denoting the playback deadline for the first segment, and B is the size of the client's buffer. To reduce the size and complexity of the ILP, we introduce time slots of length D, i.e., one segment duration, as D is the greatest common divisor for the problem at hand. Consequently, the buffer size B is denoted in time slots, and thus, provides the number of segments, which fit into the buffer. Note that initial delay is considered as time slot 0 with length T_0.

In addition to these constants, the ILP needs to solve for the variables $x_{i,j}$, which indicates if the client downloads representation j of segment i ($x = 1$) or not ($x = 0$), and $y_{i,s}$, which indicates the used bandwidth volume for each segment i during a time slot s. Variable $w_{i,s}$ denotes if segment i is downloaded in time slot s ($w = 1$) or not ($w = 0$). Finally, b_i and e_i mark the beginning/end of time slots in which the download of segment i begins/ends.

Table 4. Notation of ILP

Notation	Definition	Unit
N	Number of video segments	scalar
D	Duration of video segment → length of a time slot	second
R	Bit rate representation levels	scalar
$S_{i,j}$	Size of segment i with representation j	kbit
$V(t)$	Trace volume until time t, i.e., in interval $[0,t]$	kbit
T_0	Initial delay	second
B	Buffer size in segments/time slots	scalar
$x_{i,j}$	Indicator if client downloads representation j of segment i ($x_{i,j}=1$)	$\{0,1\}$
$y_{i,s}$	Used bandwidth volume for each segment i in time slot s	kbit
$w_{i,s}$	Indicator if segment i is downloaded in time slot s	$\{0,1\}$
b_i	Time slot, in which download of segment i begins	scalar
e_i	Time slot, in which download of segment i ends	scalar
IV	Index vector $[0,1,2,\ldots,N]^T$ containing indices for all time slots $0,\ldots,N$	scalar

In the following, the ILP is formulated:

$$\textit{Maximize } BW = \sum_{i=1}^{N}\sum_{j=1}^{R} S_{i,j}x_{i,j} \tag{1}$$

Subject to:

$$\sum_{j=1}^{R} x_{i,j} = 1 \qquad \forall i = 1,\ldots,N \tag{2}$$

$$\sum_{i=1}^{s}\sum_{j=1}^{R} S_{i,j}x_{i,j} \le V(T_0 + (s-1)\cdot D) \qquad \forall s = 1,\ldots,N \tag{3}$$

$$\sum_{i=1}^{N} y_{i,0} \le V(T_0) \tag{4}$$

$$\sum_{i=1}^{N} y_{i,s} \le V(T_0 + s\cdot D) - V(T_0 + (s-1)\cdot D) \qquad \forall s = 1,\ldots,N \tag{5}$$

$$\sum_{s=0}^{i-B} y_{i,s} = 0 \qquad \forall i = B+1,\ldots,N \tag{6}$$

$$\sum_{s=i}^{N} y_{i,s} = 0 \qquad \forall i = 1,\ldots,N \tag{7}$$

$$\sum_{s=i-B+1}^{i-1} y_{i,s} = \sum_{j=1}^{R} x_{i,j}\cdot S_{i,j} \qquad \forall i = 1,\ldots,N \tag{8}$$

$$b_i \le e_i \qquad \forall i = 1,\ldots,N \tag{9}$$

$$e_i \le b_{i+1} \qquad \forall i = 1,\ldots,N \tag{10}$$

$$y_{i,s} \le M\cdot w_{i,s} \qquad \forall i = 1,\ldots,N,\ \forall s = 0,\ldots,N \tag{11}$$

$$e_i = \max(IV\cdot w_{i,s}) \qquad \forall i = 1,\ldots,N \tag{12}$$

$$b_i = N - \max((N-IV)\cdot w_{i,s}) \qquad \forall i = 1,\ldots,N. \tag{13}$$

The ILP maximizes the accumulated size for all downloaded video representations for all segments, see Equation (1). Note that we implicitly assume that maximizing the bandwidth utilization, and thus, the video bit rate, maximizes the QoE due to an improved visual quality, which does not take into account potential negative effects on the QoE when too many QS occur. However, the number of QS could be minimized by another ILP in a second step as it was done in [25, 26, 37], which is out of scope here.

Our optimization is subject to the following constraints: The first constraint in Equation (2) ensures that every segment of the video needs to be downloaded in exactly one representation. Constraint (3) checks that the cumulative size of downloaded segments up to segment s is smaller than the cumulative bandwidth volume of the network trace until time slot s. This means, all segments can completely be downloaded before their respective playback deadlines to avoid stalling. Obviously, multiple consecutive segments can share the network bandwidth in a time slot. Constraints (4) and (5) ensure that the amount of downloaded volume for each time slot s is less or equal than the trace's bandwidth volume, thus, prohibiting that the the maximum network throughput for a given network trace is exceeded at any point in time.

Constraint (6) prohibits a segment being downloaded B time slots before its playback deadline. This constraint follows from the property of having a maximum video buffer size in real DASH systems. Consider, for example, having a DASH system with a 20 s video buffer. This buffer limitation implies that a segment can only be downloaded at most 20 s before its playout deadline, hence, justifying the constraint formulation. Constraint (7) prevents a segment being downloaded after its playback deadline and Constraint (8) assures that the downloaded bytes of a segment are equal to the size of the downloaded representation, i.e., all bytes of a segment are actually downloaded.

The next two constraints check that segments are downloaded in order by ensuring that the begin of a segment download is before or in the same time slot as its completion (9), and the completion is before or in the same time slot as the begin of the next segment's download (10).

While it is trivial for a human to determine begin and end of the download of a segment i—they are the first and last slots of $y_{i,s}$ not being zero—the optimizer requires an additional indicator variable and additional constraints which expose b_i and e_i. For this, Constraint (11) binarizes $y_{i,s}$ into $w_{i,s}$, such that for each segment i at time slot s the variable is 0 if $y_{i,s} = 0$, or 1 if $y_{i,s} > 0$. Here, we employ the big-M method [20], which introduces the artificial constant M, which is larger than the biggest value y can attain, to calculate the binary $w_{i,s}$. As `Gurobi` advises to choose M parameters as small as possible [21], we set M for each y separately with $(\sum_{k=1}^{N} y_{k,s} + 1)$ for each time slot s. To set e_i to the time slot at which the download of segment i completes, i.e., the last time slot in which at least one byte is downloaded, we multiply $w_{i,s}$ with an auxiliary index vector $IV = [1, 2, \ldots, N]^T$. The resulting vector contains all zeros at indices/time slots where nothing was download, and the actual time slot numbers at the indices/time slots in which the download was ongoing. Thus, e_i can be computed as the maximum value of this vector as shown in Constraint (12). Note that this is possible as the used `Gurobi Optimizer` offers a method to obtain the maximum value of a vector. Since we also require the smallest value/index larger than zero for b_i, the inverted vector was subtracted from its size N, as shown in Constraint (13).

B Testbed Setup

We implemented a testbed, which is able to configure the DASH system, control network conditions, and run measurements in a fully automated. It consists of three `Docker` containers, i.e., the *server* hosting the video segments, the *client* using standard `dash.js` to request and play the video, and a *network emulator* regulating the available bandwidth between server and client.

The *server* container runs a `Node.js` web server, `FFmpeg` for video encoding, and `MP4Box` for DASH manifest and segment creation. We decided to use the H.264 codec, which is supported by most devices and provides a good tradeoff between compression efficiency and encoding duration [3, 59], which will become relevant later for our proposed *JITE* DASH system. We use `libx264` as encoder, and consider a segment length of 4 s for *Tears of Steel* as in the ILP above. As encoder, `libx264` is used. As above in the ILP, we use a segment length of 4 s and, considering the *Tears of Steel* frame rate of 24 fps, we set our GOP size to 96. Furthermore, we force *no-scenecut* as *x264 option* in `FFmpeg`, use *film* for the *tune* parameter, and set *B-frames* to 3. We support six resolutions, namely, 360p, 480p, 720p, 1080p, 1440p, and 2160p. They are encoded using constant rate factor (*crf*) mode, in which we apply a bit rate constraint with *maxrate* and set *bufsize* to twice *maxrate* to avoid bit rate spikes, and set the *libx264 preset* to *slow*. To reduce the size of the manifest file, we use the *dashavc264:live* profile in `MP4Box`, which presents each representation without listing individual segments.

The *server* runs a website with an embedded `dash.js` v4.2.1 video player [11]. The video player can be configured to use different ABR algorithms, namely, the standard ABR algorithms Bola, Throughput, Dynamic, and L2A, which are implemented in `dash.js`. Note that in contrast to the former three ABR algorithms, L2A is designed for achieving low latency in live streaming scenarios [31], and thus, could potentially have advantages in highly fluctuating network conditions. To allow for a better comparability between the ABR algorithms and a better reproducibility of the measurement results, we decided to adjust a few `dash.js` player settings. First, we set the *stable buffer time*, i.e., the buffer size, to 20 s, exactly as for the ILP, and the *initial buffer level* to 12 s, i.e., three times the segment size, which is higher than for the ILP to help the streaming cope with high bandwidth fluctuations especially at the beginning of the streaming. Next, we set the *bandwidth safety factor* to 1.0, which makes the ABR algorithms decide on the actually measured throughput instead of using slightly reduced values, and we disable the *extra top level buffer* to keep the buffer size fixed. We also disable *fast switch* to avoid that already buffered segments are discarded and replaced by higher quality versions, and disable the *dropped frames rule*, such that requested resolution and bit rate are not reduced in case the testbed browser drops frames during video playback. Finally, we enable the dispatching of events for the ABR logics, so that we can trace all decisions made by the algorithms. In addition, we save an extensive `Node.js` server log including all configurations and durations for encoding, as well as timestamps, sizes, and quality levels of requested and delivered segments.

The *client* consists of a `Node.js` runtime environment and a headless `Google Chrome` web browser v97.0 controlled through `puppeteer` v13.1.1. For each measurement, the `Node.js` environment first starts a logging process, collecting `dash.js` metrics and events on the client side. This includes information about the requested and downloaded segments, and a periodic logging of current playback time, buffer size, and video quality every 100 ms, which allows for fine-granular analyses of the streaming [50, 62]. Then, it starts a video stream by requesting a DASH manifest from the server. The *network emulator* uses `tc` to adjust the network throughput each second according to a given network trace.

For the sake of reproducibility and as an additional contribution, we make our testbed implementation publicly available at [48].

References

[1] Tobias Achterberg, Robert E. Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. 2016. *Presolve Reductions in Mixed Integer Programming*. Technical Report 16-44. ZIB, Berlin.

[2] Eilwoo Baik, Amit Pande, Zizhan Zheng, and Prasant Mohapatra. 2016. VSync: Cloud based video streaming service for mobile devices. In *Proceedings of the IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 1–9.

[3] Nabajeet Barman and Maria G. Martini. 2017. H.264/MPEG-AVC, H.265/MPEG-HEVC and VP9 codec comparison for live gaming video streaming. In *Proceedings of the 9th International Conference on Quality of Multimedia Experience (QoMEX '17)*. 1–6.

[4] Abdelhak Bentaleb, Bayan Taani, Ali C. Begen, Christian Timmerer, and Roger Zimmermann. 2019. A survey on bitrate adaptation schemes for streaming media over HTTP. *IEEE Communications Surveys & Tutorials* 21, 1 (2019), 562–585.

[5] Blender Foundation. 2012. Tears of Steel | Mango Open Movie Project. Retrieved from https://mango.blender.org/.

[6] David F. Brueck, C. Ryan Owen, Tyler Bye, Nathan J. Edwards, and Ken Brueck. 2014. Just-in-time (JIT) encoding for streaming media content. Retrieved from https://patents.google.com/patent/US20140247887A1/en

[7] Syed Muhammad Ammar Hassan Bukhari, Kashif Bilal, Aiman Erbad, Amr Mohamed, and Mohsen Guizani. 2023. Video transcoding at the edge: Cost and feasibility perspective. *Cluster Computing* 26, 1 (2023), 157–180.

[8] Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. 2007. I Tube, You Tube, everybody tubes: Analyzing the world's largest user generated content video system. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*. 1–14.

[9] Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. 2009. Analyzing the video popularity characteristics of large-scale user generated content systems. *IEEE/ACM Transactions on Networking* 17, 5 (2009), 1357–1370.

[10] Shih-Fu Chang and Anthony Vetro. 2005. Video adaptation: Concepts, technologies, and open issues. *Proceedings of the IEEE* 93, 1 (2005), 148–158.

[11] Dash Industry Forum. 2013. dash.js. Retrieved from https://github.com/Dash-Industry-Forum/dash.js

[12] Toon De Pessemier, Katrien De Moor, Wout Joseph, Lieven De Marez, and Luc Martens. 2013. Quantifying the influence of rebuffering interruptions on the user's quality of experience during mobile video watching. *IEEE Transactions on Broadcasting* 59, 1 (2013), 47–61.

[13] Pradeep Dogga, Sandip Chakraborty, Subrata Mitra, and Ravi Netravali. 2019. Edge-based transcoding for adaptive live video streaming. In *Proceedings of the 2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*. 1–7.

[14] Alireza Erfanian, Hadi Amirpour, Farzad Tashtarian, Christian Timmerer, and Hermann Hellwagner. 2021. LwTE: Light-weight transcoding at the edge. *IEEE Access* 9 (2021), 112276–112289.

[15] Guanyu Gao and Yonggang Wen. 2021. Video transcoding for adaptive bitrate streaming over edge-cloud continuum. *Digital Communications and Networks* 7, 4 (2021), 598–604.

[16] Deepti Ghadiyaram, Janice Pan, and Alan C. Bovik. 2015. A time-varying subjective quality model for mobile streaming videos with stalling events. In *Proceedings of SPIE Applications of Digital Image Processing XXXVIII*. 348–355. Retrieved from https://patents.google.com/patent/US20140247887A1/en

[17] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. 2007. YouTube traffic characterization: A view from the edge. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*. 15–28.

[18] Google. 2022. YouTube Recommended Upload Encoding Settings - YouTube Help. Retrieved from https://support.google.com/youtube/answer/1722171#zippy=%2Cbitrate

[19] Michael Grafl and Christian Timmerer. 2013. Representation switch smoothing for adaptive HTTP streaming. In *Proceedings of the 4th International Workshop on Perceptual Quality of Systems (PQS)*. 178–183.

[20] Igor Griva, Stephen G. Nash, and Ariela Sofer. 2009. *Linear and Nonlinear Optimization* (2nd. ed.). Society for Industrial and Applied Mathematics.

[21] Gurobi. 2022. Dealing with Big-M Constraints. Retrieved from https://www.gurobi.com/documentation/9.5/refman/dealing_with_big_m_constra.html

[22] Gurobi. 2023. Gurobi Optimizer Reference Manual. Retrieved from https://www.gurobi.com

[23] Steve Göring and Werner Robitza. 2017. ITU-T Rec. P.1203 Standalone Implementation. Retrieved from https://github.com/itu-p1203/itu-p1203/

[24] Tobias Hoßfeld, Sebastian Egger, Raimund Schatz, Markus Fiedler, Kathrin Masuch, and Charlott Lorentzen. 2012. Initial delay vs. interruptions: Between the devil and the deep blue sea. In *Proceedings of the 4th International Workshop on Quality of Multimedia Experience (QoMEX)*. 1–6.

[25] Tobias Hoßfeld, Michael Seufert, Christian Sieber, Thomas Zinner, and Phuoc Tran-Gia. 2014. Close to optimum? User-centric evaluation of adaptation logics for HTTP adaptive streaming. *PIK - Praxis der Informationsverarbeitung und Kommunikation* 37 (2014), 275–285.

[26] Tobias Hoßfeld, Michael Seufert, Christian Sieber, Thomas Zinner, and Phuoc Tran-Gia. 2015. Identifying QoE optimal adaptation of HTTP adaptive streaming based on subjective studies. *Computer Networks* 81 (2015), 320–332.

[27] Te-Yuan Huang, Chaitanya Ekanadham, Andrew J. Berglund, and Zhi Li. 2019. Hindsight: Evaluate video bitrate adaptation at scale. In *Proceedings of the 10th ACM Multimedia Systems Conference*. 86–97.

[28] International Standards Organization/International Electrotechnical Commission (ISO/IEC). 2012. 23009-1:2012 Information Technology – Dynamic Adaptive Streaming over HTTP (DASH) – Part 1: Media Presentation Description and Segment Formats.

[29] International Telecommunication Union. 2016. ITU-T Recommendation P.1203: Parametric Bitstream-Based Quality Assessment of Progressive Download and Adaptive Audiovisual Streaming Services over Reliable Transport. Retrieved from https://www.itu.int/rec/T-REC-P.1203/en

[30] Mark Kalman, Geraint Davies, Michael Hill, and Benjamin Pracht. 2017. Introducing LHLS Media Streaming. Retrieved from https://medium.com/@periscopecode/introducing-lhls-media-streaming-eb6212948bef

[31] Theo Karagkioules, Rufael Mekuria, Dirk Griffioen, and Arjen Wagenaar. 2020. Online Learning for Low-Latency Adaptive Streaming. In *Proceedings of the 11th ACM Multimedia Systems Conference*. 315–320.

[32] Dilip Kumar Krishnappa, Michael Zink, and Ramesh K. Sitaraman. 2015. Optimizing the video transcoding workflow in content delivery networks. In *Proceedings of the 6th ACM Multimedia Systems Conference*. 37–48.

[33] Blazej Lewcio, Benjamin Belmudez, Amir Mehmood, Marcel Wältermann, and Sebastian Möller. 2011. Video quality in next generation mobile networks – Perception of time-varying transmission. In *Proceedings of the IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR)*. 1–6.

[34] Zhi Li, Anne Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara. 2016. Toward a practical perceptual video quality metric. *The Netflix Tech Blog* 6, 2 (2016).

[35] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 197–210.

[36] Vignesh V. Menon, Hadi Amirpour, Mohammad Ghanbari, and Christian Timmerer. 2022. ETPS: Efficient two-pass encoding scheme for adaptive live streaming. In *Proceedings of the IEEE International Conference on Image Processing (ICIP '22)*. IEEE, 1516–1520.

[37] Konstantin Miller. 2016. *Adaptation Algorithms for HTTP-Based Video Streaming*. Technische Universitaet Berlin.

[38] Pengpeng Ni, Ragnhild Eg, Alexander Eichhorn, Carsten Griwodz, and Pål Halvorsen. 2011. Flicker effects in adaptive video streaming to handheld devices. In *Proceedings of the 19th ACM International Conference on Multimedia (MM)*. 463–472.

[39] Jan Ozer. 2017. *Finding the Just Noticeable Difference with Netflix VMAF*. Streaming Learning Center.

[40] Mirko Palmer, Malte Appel, Kevin Spiteri, Balakrishnan Chandrasekaran, Anja Feldmann, and Ramesh K. Sitaraman. 2021. VOXEL: Cross-layer optimization for video streaming with imperfect transmission. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*. 359–374.

[41] Yanyuan Qin, Shuai Hao, Krishna R Pattipati, Feng Qian, Subhabrata Sen, Bing Wang, and Chaoqun Yue. 2019. Quality-aware strategies for optimizing ABR video streaming QoE and reducing data usage. In *Proceedings of the 10th ACM Multimedia Systems Conference*. 189–200.

[42] 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 *Proceedings of the 9th International Conference on Quality of Multimedia Experience (QoMEX)*. IEEE, 1–6.

[43] 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 *Proceedings of the 9th ACM Multimedia Systems Conference*. 466–471.

[44] Susanna Schwarzmann, Nick Hainke, Thomas Zinner, Christian Sieber, Werner Robitza, and Alexander Raake. 2020a. Comparing fixed and variable segment durations for adaptive video streaming: A holistic analysis. In *Proceedings of the 11th ACM Multimedia Systems Conference*. 38–53.

[45] Susanna Schwarzmann, Nick Hainke, Thomas Zinner, Christian Sieber, Werner Robitza, and Alexander Raake. 2020b. DASH-Streaming-Setup. Retrieved from https://github.com/fg-inet/DASH-streaming-setup

[46] Michael Seufert, Sebastian Egger, Martin Slanina, Thomas Zinner, Tobias Hoßfeld, and Phuoc Tran-Gia. 2015a. A survey on quality of experience of HTTP adaptive streaming. *IEEE Communications Surveys & Tutorials* 17, 1 (2015), 469–492.

[47] Michael Seufert, Tobias Hoßfeld, and Christian Sieber. 2015b. Impact of intermediate layer on quality of experience of HTTP adaptive streaming. In *Proceedings of the 11th International Conference on Network and Service Management (CNSM)*. 256–260.

[48] Michael Seufert, Marius Spangenberger, Fabian Poignée, Florian Wamser, Werner Robitza, Christian Timmerer, and Tobias Hoßfeld. 2024. COBIRAS GitHub Repository. Retrieved from https://github.com/netcom-augsburg/cobiras

[49] Michael Seufert, Nikolas Wehner, and Pedro Casas. 2018. Studying the impact of HAS QoE factors on the standardized QoE model P. 1203. In *Proceedings of the 3rd Workshop on QoE-based Analysis and Management of Data Communication Networks (Internet-QoE)*. 1636–1641.

[50] Michael Seufert, Nikolas Wehner, Florian Wamser, Pedro Casas, Alessandro D'Alconzo, and Phuoc Tran-Gia. 2017. Unsupervised QoE field study for mobile YouTube video streaming with YoMoApp. In *Proceedings of the 9th International Conference on Quality of Multimedia Experience (QoMEX)*. 1–6.

[51] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. 2019. From theory to practice: Improving bitrate adaptation in the DASH reference player. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 15, 2s (2019), 1–29.

[52] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. 2020. BOLA: Near-optimal bitrate adaptation for online videos. *IEEE/ACM Transactions on Networking* 28, 4 (2020), 1698–1711.

[53] Samira Tavakoli, Sebastian Egger, Michael Seufert, Raimund Schatz, Kjell Brunnström, and Narciso García. 2016. Perceptual quality of HTTP adaptive streaming strategies: Cross-experimental analysis of multi-laboratory and crowdsourced subjective studies. *IEEE Journal on Selected Areas in Communications* 34, 8 (2016), 2141–2153.

[54] Laura Toni, Ramon Aparicio-Pardo, Gwendal Simon, Alberto Blanc, and Pascal Frossard. 2014. Optimal set of video representations in adaptive streaming. In *Proceedings of the 5th ACM Multimedia Systems Conference (MMSys '14)*. 271–282.

[55] Huyen T. T. Tran, Thang Vu, Nam Pham Ngoc, and Truong Cong Thang. 2016. A novel quality model for HTTP adaptive streaming. In *Proceedings of the 6th IEEE International Conference on Communications and Electronics (ICCE)*. 423–428.

[56] Armin Trattnig, Christian Timmerer, and Christopher Mueller. 2018. Investigation of YouTube regarding content provisioning for HTTP adaptive streaming. In *Proceedings of the 23rd Packet Video Workshop (PV '18)*. ACM, New York, NY, 60–65.

[57] Bekir Oguzhan Turkkan, Ting Dai, Adithya Raman, Tevfik Kosar, Changyou Chen, Muhammed Fatih Bulut, Jaroslaw Zola, and Daby Sow. 2022. GreenABR: Energy-aware adaptive bitrate streaming with deep reinforcement learning. In *Proceedings of the 13th ACM Multimedia Systems Conference*. 150–163.

[58] Twitch. 2020. Twitch Invites You to Take on Our ACM MMSys 2020 Grand Challenge. Retrieved from https://blog.twitch.tv/en/2020/01/15/twitch-invites-you-to-take-on-our-acm-mmsys-2020-grand-challenge/

[59] Tadeus Uhl, Christian Hoppe, and Janusz Klink. 2020. Modern codecs by video streaming under Use DASH technique: An objective comparison study. In *Proceedings of the International Conference on Software, Telecommunications and Computer Networks (SoftCOM '20)*.

[60] Jeroen Van Der Hooft, Stefano Petrangeli, Tim Wauters, Rafael Huysegems, Patrice Rondao Alface, Tom Bostoen, and Filip De Turck. 2016. HTTP/2-based adaptive streaming of HEVC video over 4G/LTE networks. *IEEE Communications Letters* 20, 11 (2016), 2177–2180.

[61] Anthony Vetro, Charilaos Christopoulos, and Huifang Sun. 2003. Video transcoding architectures and techniques: An overview. *IEEE Signal Processing Magazine* 20, 2 (2003), 18–29.

[62] Florian Wamser, Michael Seufert, Pedro Casas, Ralf Irmer, Phuoc Tran-Gia, and Raimund Schatz. 2015. YoMoApp: A tool for analyzing QoE of YouTube HTTP adaptive streaming in mobile networks. In *Proceedings of the European Conference on Networks and Communications (EuCNC)*. 239–243.

[63] Fei Wang, Zesong Fei, Jing Wang, Yifan Liu, and Zhikun Wu. 2017. HAS QoE prediction based on dynamic video features with data mining in LTE network. *Science China Information Sciences* 60, 4 (2017), Article 042404.

[64] Nikolas Wehner, Michael Seufert, Viktoria Wieser, Pedro Casas, and Germán Capdehourat. 2021. Quality that matters: QoE Monitoring in education service provider (ESP) networks. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM '21)*. IEEE, 830–835.

[65] Kai Zeng, Hojatollah Yeganeh, and Zhou Wang. 2016. Quality-of-experience of streaming video: Interactions between presentation quality and playback stalling. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*. 2405–2409.

Received 6 November 2023; revised 12 June 2024; accepted 29 June 2024