Where are the Sweet Spots? A Systematic Approach to Reproducible DASH Player Comparisons

Denny Stohr[○][*], Alexander Frömmgen[‡][*], Amr Rizk[‡],
Michael Zink[◇], Ralf Steinmetz[‡], Wolfgang Effelsberg[○]

[○]DMS / [‡]KOM, TU Darmstadt
[◇]ECE Department, University of Massachusetts Amherst
dstohr@kom.tu-darmstadt.de,alexander.froemmgen@kom.tu-darmstadt.de

ABSTRACT

The current body of research on Dynamic Adaptive Streaming over HTTP (DASH) contributes various adaptation algorithms aiming to optimize performance metrics such as the Quality of Experience. Intuitively, the heterogeneity of the streaming environment and the underlying technologies lead many of the developed approaches to possess clear performance affinities denoted here as *sweet spots*. We observe, however, that systematic comparisons of these algorithms are usually conducted within homogeneous player environments.

In this work, we show the substantial impact of player choice and configuration on the streaming performance. To this end, we systematically examine three established open-source DASH players, i.e., DASH.JS, Google's Shaka Player, and AStream, that implement fundamentally different configurations featuring various adaptation algorithms. By establishing a large scale emulation framework we *(i)* extract player sweet spots and *(ii)* achieve a direct, reproducible comparison of real-world DASH players and algorithms. We present empirical evidence demonstrating that an isolated analysis of DASH player modules is insufficient to capture the player streaming performance. One of the major observations is that the choice of the target buffer size together with the player implementation dominates the choice of the adaptation algorithms.

[*]The two authors contributed equally to this work.

1 INTRODUCTION

With the ongoing and continued success of video streaming in the Internet, it has become a daily used service by billions of users. Facilitated by the wide range of usage scenarios, such as on-demand movies, advertising and user-generated video, a multi-billion dollar industry formed. It is predicted that the high popularity of video streaming applications will keep growing in the near future. According to the latest Sandvine report [19], 71% (forecasted 82% by 2020 [1]) of the downstream Internet traffic at peak hours in North America is generated by live streaming and Video on Demand (VoD) services. Thus, providing reliable video streaming services that meet the Quality of Experience (QoE) demands of the viewers is of paramount importance to VoD service providers, especially, given heterogeneous access networks and devices.

A main technical enabler for VoD streaming is MPEG Dynamic Adaptive Streaming over HTTP (DASH) [21], that is developed to provide constantly high QoE in changing network conditions using regular HTTP requests. It allows the client to request video segments in different qualities based on its current observations of the network condition. The foremost goal of this adaptation approach is the elimination of stalling events during playout that have a strong detrimental impact on the viewer's QoE [20]. The MPEG-DASH standard does not dictate any specific Adaptation Algorithm (AA) fueling the derivation of sophisticated AAs for improved QoE in fluctuating network and environment conditions.

Many works on DASH set the focus on the development of AAs. We notice that empirical evaluations of AAs are typically conducted in a single player, e.g., as in [8, 18, 22, 23]. While this approach is necessary for a systematic comparison, it neglects the impact of the player implementation, its configurations, and its interaction with the networking environment. This reduces the potential to generalize the obtained results. In this work, we provide a systematic comparison of three DASH players (*AStream*, *DASH.JS* and *Shaka*) equipped with different AAs (BOLA, DASH.JS default, Shaka default) under varying network conditions. To substantiate the impact of the player and its configuration Figure 1 shows an aggregated stalling-based QoE metric from [6] for the players and AAs discussed in Sect. 2. Here, we compare the achieved Mean Opinion Score (MOS) for different network volatilities (modeled through the variance of the available bandwidth) and for different video segment lengths. *The figure shows that the choice of the adaptation algorithm is dominated by the choice of the player and its configuration.* Based on this illustrative example, we argue that the isolated analysis of single DASH players or parameters is **not sufficient** to evaluate the streaming performance.

In this paper, we propose an empirical evaluation methodology for DASH players, given their executable implementation, for large configuration spaces in conjunction with extensive network emulation. We provide a systematic approach for reproducible and comparable DASH player performance evaluations. To this end, we implemented a DASH player execution environment, which supports different DASH players in a wide set of Mininet-based network emulations. Our results show the significant impact of the player and its configuration on the streaming performance and

MM'17, October 23–27, 2017, Mountain View, CA, USA.
© 2017 ACM. ISBN 978-1-4503-4906-2/17/10…$15.00
DOI: https://doi.org/10.1145/3123266.3123426

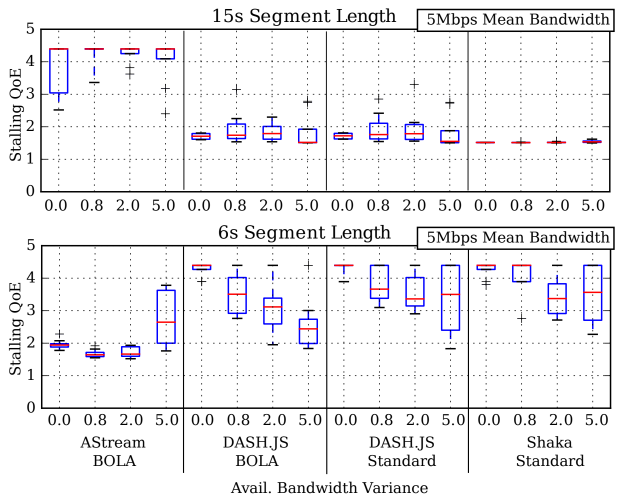

Figure 1: Stalling QoE [6] (higher is better) of DASH Players with various adaptation algorithms for different available bandwidth volatilities and segment length configurations. The player and the segment length have a strong impact on the Stalling QoE whereas the impact of the adaptation algorithm is marginal.

provide evidence that the target buffer size is the main driver for high QoE. Further, we show that DASH players possess many configuration sweet spots depending on the considered performance metrics. In summary,

- we provide a publicly available execution environment for a systematic comparison of DASH players and adaptation algorithms for the community[1]
- we analyze the *individual configuration space* of players and its influence on target performance / QoE metrics
- we provide a multi-objective comparison between three state-of-the art adaptation algorithms and DASH players in a reproducible environment

The remainder of this work is structured as follows: In Sect. 2 we revisit the required background on DASH players and algorithms. In Sect. 3 we introduce the methodology used in this work, before providing our main evaluation results in Sect. 4. We discuss related work in Sect. 5 before concluding the paper in Sect. 6.

2 DISSECTING DASH PLAYERS

In this section, we discuss existing adaptation algorithms and present three DASH players that follow different designs and assumptions: *(i)* the DASH Industry Forum's *DASH.JS*[2], *(ii)* Google's *Shaka Player*[3], *(iii)* an academic DASH simulator denoted *AStream*[4]. These players are a representative choice of existing open-source players with various adaptation algorithms.

[1]https://maci-research.net/DASH
[2]https://github.com/Dash-Industry-Forum/dash.js
[3]https://github.com/google/shaka-player
[4]https://github.com/pari685/AStream

2.1 DASH Adaptation Algorithms

Given that DASH is a standard for client-driven video streaming with multiple media quality representations, the client player requires a logic (often denoted as adaptation algorithm) to decide on the requested media bitrate and quality. DASH adaptation algorithms can be classified into two main categories: *(i) Troughput-based Adaptation (TBA)* and *(ii) Buffer-based Adaptation (BBA).*

TBA algorithms, such as [14], estimate the throughput of the DASH streaming connection through the measured segment download rate $R(n)$ of the nth segment. In its basic form, TBA adapts the quality of the next requested video segment to this throughput estimate. This adaptation is either directly based on $R(n)$ or on aggregated metrics that take the measurement history $\{R_{n-i}, \dots, R_n\}$ for $i \geq 1$ into account. Such aggregated metrics range from simple operations, e.g., averaging or filtering, to fitting the measurements to complex underlying stochastic models.

BBA algorithms choose the quality of the next segment based on the buffer filling information $B(t)$ at time t. Examples for these adaptation algorithms comprise [9, 22], where the quality adaptation behavior is controlled solely through the playout buffer filling. A recent BBA algorithm that gained traction is the *Buffer Occupancy based Lyapunov Algorithm (BOLA)* which assumes, in contrast to TBA, that the playout buffer state is a sufficient statistic to control the quality bitrate adaptation. *BOLA* maximizes the utility of the playback as given by a weighted sum of the average quality bitrate and the average stalling time. *BOLA* uses a Lyapunov optimization technique to solve the segment quality decision problem at each segment request, i.e., either to skip downloading a new segment or to download a utility maximizing new segment. BBA algorithms have the advantage of utilizing a smoothed function of the connection bandwidth estimates which may reduce the impact of connection fluctuations. Some BBA algorithms were even shown to perform strongly in steady state while completely neglecting throughput estimates [8].

Note that a verity of hybrid algorithms exist that combine throughput and buffer information in aggregated models for bitrate adaptation such as in [2, 26]. Both TBA and BBA algorithms may comprise different *aggressiveness* in quality adaptation. For example, cautious adaptation algorithms that try to avoid stalling events choose conservatively low qualities in down scaling direction, i.e., when choosing a lower quality than the previously requested video segment. The same conservatism holds in up scaling direction, i.e., when increasing the quality of requested segments.

2.2 DASH Players and Configurations

In the following, we briefly discuss the open-source DASH players that we analyze and compare in this work. While many other DASH players exist, such as the ones used by Netflix and YouTube, the lack of an open API prevents their direct use in the context of this work. Other commercial players that provide an API, such as Bitmovin's *HTML5 Player*[5], have been omitted due to licensing reasons.

DASH.JS is the reference implementation by the DASH Industry Forum. We used release v2.3.0 in our experiments, which features two types of adaptation algorithms: TBA (default) and BBA. TBA uses a `ThroughputRule` that maintains an up-to-date list of

[5]https://bitmovin.com/html5-player/

the previous four throughput and latency measurement periods that determine upcoming adaptation decisions by passing them on to a `SwitchRequest` class. The BBA algorithm embedded in DASH.JS features *BOLA* as introduced in Sect.2.1.
Configuration: DASH.JS comprises configuration parameters for both BOLA and TBA that include the target buffer-levels depending on the current playback state. If the player selected the top quality, a target buffer size of 30 seconds (denoted `BUFFER_TIME_AT_TOP_QUALITY`) is used. Otherwise, a smaller target buffer size of 12 seconds (denoted `DEFAULT_MIN_BUFFER_TIME`) applies. Thus, DASH.JS adds an additional adaptation layer by changing the target buffer size.

Google's **Shaka Player** (version 2.0.6) features a TBA algorithm that conservatively uses the minimum of two Exponentionally-Weighted Moving Average (EWMA) variables derived from throughput measurements within a period of 2 and 5 seconds, respectively. This aims at faster downscaling in case of bandwidth drops and slower upscaling with increasing bandwidth measurements.
Configuration: Two core settings control the playback buffers in the Shaka Player. First, the `rebufferingGoal`, which is set to 2 seconds, provides the minimal buffered video duration before the playback is started. Second, the `bufferingGoal` of 10 seconds provides the maximum buffer level that is loaded in advance.

AStream is a *headless* Python-based player that is introduced in [11] to simulate DASH players, i.e., it does not natively playout the requested video segments. AStream served as a rapid prototyping environment for multiple TBA and BBA adaptation algorithm such as [11, 25]. In this work, we consider an implementation of BOLA within AStream in comparison to the BOLA implementation within DASH.JS. The rationale behind this comparison is to evaluate the impact of the player while keeping the adaptation algorithm fixed.
Configuration: Like the other introduced players, *AStream-BOLA* has an implementation dependent default target buffer level, which is set to 15 seconds. AStream also comprises additional configuration parameters for which we refer to the source code [11].

2.3 Discussion

This section showed that each player provides a set of similar appearing configuration parameters as well as support for multiple adaptation algorithms. Existing players, such as AStream, DASH.JS, and Shaka Player differ regarding their *adaptation algorithm API* and the provided level of abstraction. The API of DASH.JS, for example, only enables the AA to choose the next quality. While this provides a reasonable abstraction, it limits the expressiveness of the AA, e.g., with regard to the request timing. This limits crucial aspects in adaptation strategies, e.g., *when* the algorithm is called and *how* the algorithm can control parallel requests and request revocation. Also, the quality of network measurements can be strongly affected by the way the player requests new video segments [25]. We note that general models for AA do not capture this impact of player specific components. In contrast to player independent studies of AA, a study on YouTube's DASH player shows a close interaction of requests and adaptation decisions, including a dynamic adaptation of segment sizes, as well as, asynchronous and parallel requests for multiple quality layers [16]. While all presented players provide a set of similar appearing configuration parameters, the player's configuration, e.g., of the target buffer size, directly affects the behaviour of the AA. We leave the question of technical agency for defining target buffer sizes and other configuration parameters in DASH players as well as a design for generalized interfaces up for discussion in future work.

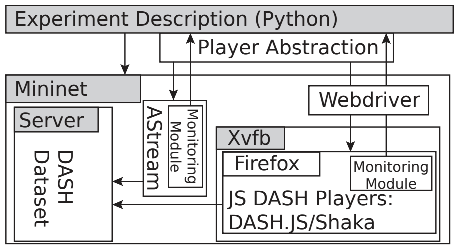

Figure 2: DASH player execution environment architecture.

3 METHODOLOGY

In this section, we provide an overview of the methodology employed in this work. First, we present the player execution environment, its architecture and the monitored key performance metrics. Based on this, we describe our experimental design, including the considered video content and network conditions.

3.1 Player Execution Environment

Driven by the goal to compare DASH-players in a reproducible setup, we build a player execution environment that provides a *player abstraction* interface to *automatically* load, configure and monitor JavaScript, as well as, Python-based players, as illustrated in Figure 2. The first setup layer of the experiment is built using Python, and relies on *Mininet* [5] to configure a virtual network setup consisting of a server, a client, and one or multiple bottleneck links separated by switches and shaped according to Sect. 3.3. This enables a future expansion of the experiment setup to arbitrary topologies and more complex networking conditions.

On the server side, a webserver is launched using the Node.JS *http-server*[6] package. Initial comparisons with an Apache webserver did not show significant differences. The client starts a `Xvfb`[7] session to allow for the execution of the *Firefox 52* browser that supports Media Source Extensions (MSE)[8]. Further, the *Geckodriver*[9] service provides control to the browser by Selenium-API calls from the Python experiment description code. For the JavaScript-based DASH players (*DASH.JS* and *Shaka Player*), the experiment is loaded in the browser by requesting the webpage containing the DASH player's JavaScript application after the setup of the DASH player execution environment. Here, the Webpack[10] loading system is employed to dynamically instantiate the selected DASH player based on the configuration parameters. For each player, a loading script was implemented that maps initialization routines and settings such as buffer levels, the Media Presentation Description (MPD), and

[6]https://www.npmjs.com/package/http-server
[7]https://www.x.org/archive/X11R7.6/doc/man/man1/Xvfb.1.xhtml
[8]http://caniuse.com/#search=mse
[9]https://github.com/mozilla/geckodriver/
[10]https://webpack.js.org/

adaptation algorithms to corresponding function calls. Since AStream, in contrast to DASH.JS and Shaka Player, is built entirely in Python and *does not* playout the video, it is directly called with the corresponding experiment parameters without browser interaction.

3.2 Monitoring Performance Metrics

To retrieve and analyze the system events and data structures which vary depending on the selected player, we implemented a *monitoring module*. It maps the players' monitoring events to a set of callback functions and creates a universal data structure saved in a JSON-format.

We set the playout duration for each experiment to 120 seconds and continuously monitor the playtime. This is to ensure that, even if stalling events occur, players are compared on a playback-time instead of an experiment-time basis. After the given experiment time, we collect and process the experiment metrics.

In this work, we consider two metric types to assess the performance of DASH Players: *(i)* target metrics that are directly *measured* and *(ii)* aggregate metrics that are derived from target metrics.
Target metrics: We consider the following directly measurable metrics: the initial playback delay, the total stalling duration, the initial playback bitrate, the average playback bitrate, the average number of adaptations, the download duration and length of requested segments, and the average amplitude of adaptations.
Aggregate metrics: Here, we use measured target metrics to deduce aggregate performance metrics such as a measure on QoE stalling [6]. This metric combines the duration and frequency of stalls into a Mean Opinion Score (MOS) scale.

3.3 Experimental Design

Our experiments run on the Cartesian product of the networking environment and player configuration vectors given in the experiment setup in Table 1. The control variables are the player and the AA as introduced in Sect. 2. As the default buffer sizes for the investigated players are around 10 seconds, we vary the buffer size to half and double of the default size, i.e., 5 and 20 seconds respectively. The selected settings such as buffer size, segment size, experiment duration, and network connection environment parameters, provide good estimates of common parameters for various streaming scenarios. For an efficient evaluation of the resulting parameter space, we parallelize the experiment execution.
Fluctuating Environment Conditions: For a systematic comparison of player performance we require an evaluation of the impact of fluctuating network environment conditions, which we capture as changes of the available bandwidth of the streaming connection. Replaying captured traces is widely used for the evaluation of adaptation algorithms [4, 7, 22, 23]. While this approach provides meaningful insights, its expressiveness is limited to the scope of the captured traces. Furthermore, it does not allow a parameterizable generation of a wide range of fluctuating environments.

Table 1: Experiment control variables.

	Variable	Values
Configuration	player	DASH.JS, Shaka, AStream
	AA	standard, BOLA
	target buffer size	default, 5, 20 [s]
	segment length	1, 2, 6, 10, 15 [s]
Environment (Avail. BW)	μ_{BW}	0.8, 2, 5, 7.5, 10 [Mbps]
	σ^2_{BW}	0, 0.8, 2, 5 [Mbps^2]

In this work, we characterize the networking conditions through the first two moments of the available bandwidth process. We implemented a control module on top of `tc` [10], the traffic shaper employed underneath Mininet. This module provides parameterizable and reproducible throughput traces. Based on a *target mean available bandwidth* and a *target available bandwidth variance*, the `tc` shaper settings are chosen every 3 seconds, i.e., the available bandwidth burstiness parameter, to ensure regular changes in network conditions. These basic control knobs of network conditions can be straightforwardly extended in our framework to express arbitrary real-world or synthetic traffic traces with given properties. In this work, we concentrate on the available bandwidth parameter, as initial measurements showed a low impact of additional parameters such as network latency regarding a player comparison.
Video Data set: In this work, we use the *Tears of Steel*[11] DASH data set prepared by [13], featuring nine H.264-AVC encoded representations. The attributes of each layer, as shown in Table 2, provide a realistic scenario in terms of bandwidth requirements [12] and match the range of evaluated available bandwidth conditions to enable meaningful results.

Table 2: DASH data set representations

Resulution	Bitrates (Mbps)
1920×1080	10.0, 6.0, 4.0, 3.0
1280×720	2.4, 1.5
640×360	0.807, 0.505
480×270	0.253

4 EVALUATION

In this section, we evaluate the performance of the DASH players introduced in Sect. 2 using the methodology described in Sect. 3. We present an evaluation of key performance metrics and a classification of sweet spot contexts, i.e., environment conditions in which players with certain configurations provide the best performance by analyzing target and aggregate metrics. First, we consider the initial playback delay before going into more detail of video stalling. We then discuss the performance of adaptation algorithms that should eliminate stalling before analyzing the impact of these techniques on the playback bitrate performance. We discuss the general trade-off of playback bitrate, stalling QoE and buffer sizes. Finally, we discuss simulation pitfalls when trying to reproduce video player performance.

4.1 Playback Initiation

The initial delay, i.e., the time until rendering the first frame, is an important performance indicator for DASH streaming sessions since longer initial delays are known to have a negative impact on the viewers' engagement [3]. However, accepting longer initial delays allows for a higher initial playback representation as indicated in Table 3. Thus, it is crucial for players to strike a balance

[11] https://mango.blender.org/

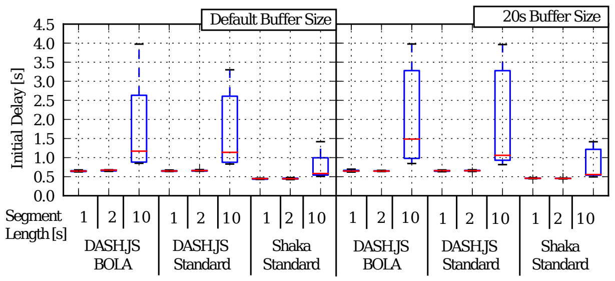

Figure 3: The initial delay depends on the segment length and the used player. Neither the target buffer size nor the adaptation algorithm severely impact the initial delay.

between these opposing factors. Figure 3 gives insights into these design choices by showing box plots of the initial playback delay vs. different combinations of players, adaptation algorithms, buffer sizes and segment lengths. Here it is evident that there are two main factors that impact the initial delay, i.e., *(i)* the used player and *(ii)* the segment length. The impact of the player is apparent when comparing DASH.JS' results with Shaka Player where initial delays vary independent of adaptation algorithms. As intuitively anticipated we find that the target buffer size does not impact the initial delay. This is expected since the target buffer size is usually used to gauge the steady state adaptation behavior of the player while many players, as described in Sect. 2.2, possess minimal buffer level requirements to start playback. A key characteristic that drives the differences in the observed initial playback delays is the selected initial representation. For example, DASH.JS consistently selects the third lowest representation, while both other players begin playback with the lowest representation, leading to smaller download sizes for the initial segments. This is reflected in the first two rows of Table 3. The low initial delay of AStream is an artifact given by a limitation in the implementation as explained in Sect. 4.7. We therefore omitted AStream from Figure 3.

4.2 Stalling

The analysis of the total stalling duration shows distinct patterns for DASH.JS and Shaka Player as depicted in Figure 4. Here, the figure shows box plots of the total stalling duration during the playback of the 120 second video for different players, adaptation algorithms, target buffer sizes and network bandwidths. Note that the AStream simulator fails to handle low available bandwidths

Table 3: Overall comparison of adaptation, video quality, and stalling metrics.

	DASH.JS (standard)		DASH.JS (BOLA)		Shaka (standard)		AStream (BOLA)	
	μ	σ	μ	σ	μ	σ	μ	σ
Init. Rep. [Mbps]	0.8	0.0	0.8	0.0	0.3	0.0	0.3	0.0
Init Delay [sec]	1.6	2.4	1.6	2.3	0.7	0.7	0.1	0.0
Adaptations [#]	8.6	8.5	8.6	8.5	4.8	2.8	18.2	13.7
Amplitude [level]	1.3	1.0	1.3	1.0	1.7	1.9	1.7	2.4
Stalling sum [sec]	8.8	14.9	8.9	15.0	8.3	13.8	18.0	50.3
Stalling avg. [sec]	1.2	1.8	1.2	1.8	1.1	1.8	4.1	11.3
Stalling [#]	4.1	3.8	4.1	3.8	3.7	3.5	3.5	7.9

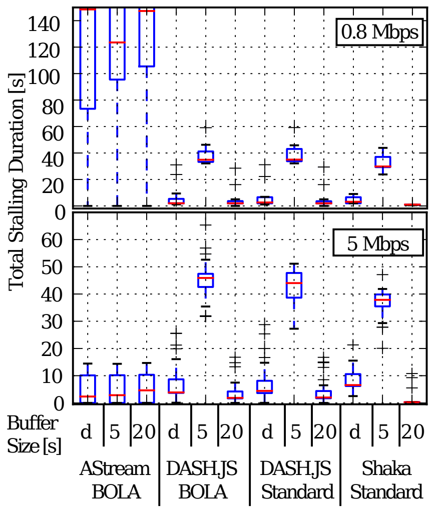

Figure 4: Impact of the buffer size and the avail. bandwidth on the total stalling duration. Segments are 10s long. Players' default buffer size indicated as *d*.

and, in high available bandwidth scenarios, appears to result in stalling durations that are not affected by the player buffer size. For an explanation of this behavior we refer to Sect. 4.7.

The JavaScript players (DASH.JS and Shaka Player) show a distinct and consistent pattern where the total stalling duration is predominantly influenced by the configured target buffer size. The evaluation shows that larger buffers significantly reduce the stalling duration. In general, Shaka Player outperforms the other alternatives with nearly no stalling events for large buffers (see Sect. 3.3 for a discussion of the default buffer sizes). Surprisingly, we note that the adaptation algorithm has a minor impact on the overall stalling duration as compared to the target buffer size.

Note that if the target buffer size is small the player often withholds requests due to a full buffer. DASH players are known for an ON-OFF behavior while downloading video segments which arises due to adaptation algorithms. This behavior is reinforced by a consistently filled buffer due to a small buffer size relative to the segment duration length. Thus, the player is blocked from requesting new segments which is evident in Figure 6, where the player downloading state exceeds 100 seconds for a large target buffer size paired with 2 second segments. In contrast the player has a significantly lower mean downloading time, i.e., 75 seconds for a low target buffer size paired with 15 second long segments.

Since higher quality segments possess larger file sizes we deduce that the time used for fetching segments in the second constellation has a high probability of exceeding the duration of the video content that is currently buffered. Thus, this constellation of small target buffer sizes in combination with relatively long segment lengths can lead to more stalling events as will be discussed in Sect. 4.5.

4.3 Adaptations

DASH Players employ quality adaptation algorithms to provide an overall better QoE, i.e., by improving metrics such as the average quality bitrate and by avoiding stalling. As described in Sect. 2.1

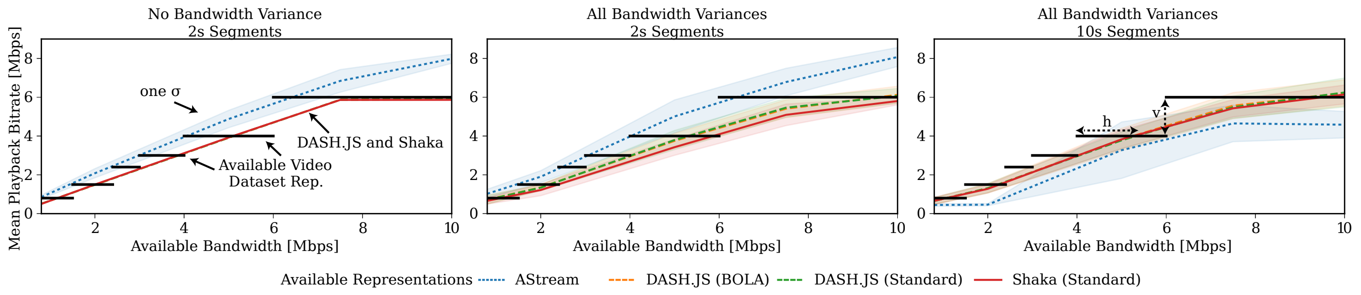

Figure 5: Mean playback bitrates given fixed available bandwidth and default buffer settings. DASH.JS and Shaka Player show consistent behavior while the AStream emulator shows highly sensitive performance wrt. the configuration and the network environment. The transparent pipes show one standard deviation width. Black stairs show the representations available in the data set. Vertical differences v at the beginning of the black stairs indicate QoE degradation given networking conditions. Horizontal deviations h indicate efficiency, i.e., how much available bandwidth is required to sustain a quality bitrate.

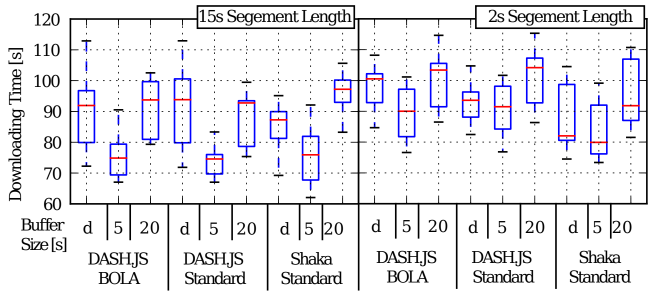

Figure 6: Impact of the buffer and segment length on the total time spent in downloading state. *d:* default buffer size.

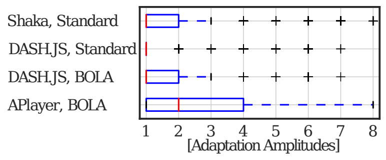

Figure 7: The distribution of quality adaptation amplitudes for combinations of players and adaptation algorithms.

adaptation algorithms take specific information as proxy for network and player states and translate this information into decisions on the streaming bitrate, influenced by models implemented in the player as well as the environment. Various QoE models, as for example discussed in [20], conclude that the number and the magnitude of adaptation events within a stream is detrimental for QoE. Further, stepwise adaptation is favorable compared to large adaptation steps according to [28]. Hence, the streaming performance directly depends on the design of the quality adaptation algorithm within the player. Figure 7 shows the distribution of the adaptation amplitudes for different players and adaptation algorithm for all given environment parameters. Note that AStream, implementing BOLA, oscillates with much higher amplitudes than DASH.JS and Shaka Player. On another note it is surprising that the impact of different adaptation algorithms, e.g., within DASH.JS, is relatively moderate. From Table 3, we find that Shaka Player produces fewer adaptation events at the expense of higher adaptation amplitudes which indicates a less gradual adaptation.

4.4 Playback Bitrate

In Sect. 4.1 to 4.3 we considered stalling performance metrics and adaptation techniques that mitigate the adverse effects of stalling by aligning the requested video qualities with the network and player conditions. These adaptations directly impact the (mean) playback bitrate, i.e., an important factor for QoE. Note that the achievable mean playback bitrate depends directly on the time-average available network bandwidth and the available video representations.

Figure 5 provides a comparison of the considered players showing the average playback bitrate vs. fixed network bandwidths. In the different plots we vary the network conditions and the segment lengths. The black stairs show the representations available in the data set. We use the crossing points of the mean playback bitrates and the representation stairs to express *efficiency* (given as the horizontal deviation h in Figure 5). This indicates the additionally required available bandwidth to sustain a given representation (given stable / volatile networking conditions as depicted). The vertical deviations v of the quality bitrates and the black stairs in Figure 5 denote the following: If the black stairs are higher than the quality bitrate lines, the deviation denotes a loss in QoE in terms of mean quality bitrate given a certain bandwidth condition. If the black stairs are lower than the quality bitrate lines, then the available bandwidth is sufficient to sustain this representation and the excess available bandwidth is used to occasionally fetch a higher than sustainable quality bitrate.

Figure 5 shows that the investigated real-world DASH players achieve comparable performance with respect to the mean playback bitrates given various adaptation algorithms. While variance in the available bandwidth slightly decreases the playback bitrate, increased segment lengths increase the playback bitrate. The DASH simulator AStream shows again inconsistent behavior.

4.5 Performance Metric Trade-offs

Sect. 4.1 to 4.4 presented an analysis of isolated metrics, including both target metrics and efficiency as an aggregate metric. A further aggregate metric, denoted as *stalling QoE* from [6], describes a mapping of the stalling duration and the stalling events to a MOS scale of QoE. In general, however, due to the intrinsic difficulty of constructing a single comprehensive quality metric, the choice of the optimal player and its configuration remains a multidimensional optimization problem.

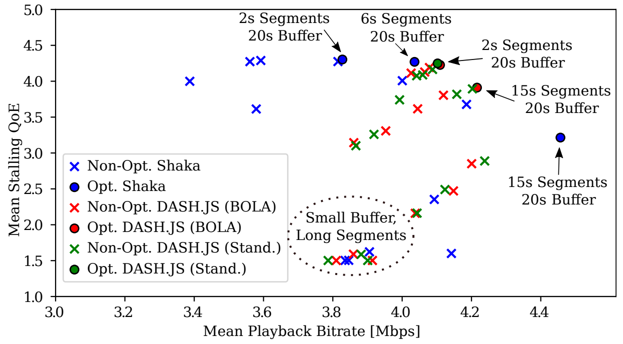

Figure 8: Trade-off between playback bitrate and stalling QoE for different configurations, aggregated over all analyzed environment conditions.

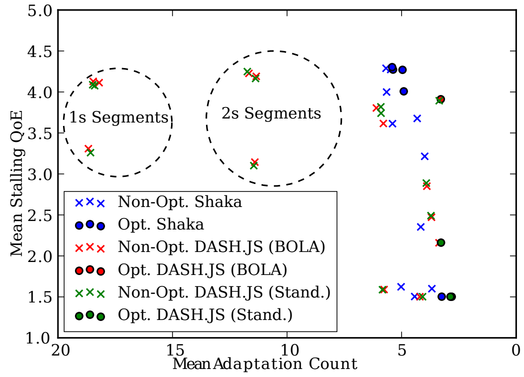

Figure 9: Trade-off between adaptation count and stalling QoE for different configurations, aggregated for all analyzed environment conditions.

Playback Bitrate vs. Stalling QoE: We analyze the trade-offs between two previously discussed aspects, *(i)* the playback bitrate and *(ii)* stalling, captured as stalling QoE. Here, we omit AStream due to the inconsistent stalling results shown in Sect. 4.2. Figure 8 shows a scatter plot of stalling QoE (the higher the better) and the achieved mean playback bitrate, where each entry in the graph denotes the performance of a configuration averaged over all considered network conditions. Interestingly, the depicted Pareto-frontier shows that no single player (configuration) dominates both metrics. In particular, all players and AAs are represented at least once on the Pareto-frontier. Thus, for every player and AA there exists a sweet spot and a weighted aggregate taking stalling QoE and mean playback bitrate that shows that this configuration is superior.

The Pareto-frontier further shows that large buffer sizes dominate the performance for all player configurations for both considered metrics. We also note that moving to higher playback bitrates on the Pareto-frontier corresponds to increasing the segment length of the corresponding configurations. It is also evident that combinations of small buffer sizes and long segments perform bad. Finally, Figure 8 shows the minor impact of different adaptation algorithms within the DASH.JS player, i.e., the green and red marks collate without a consistently superior adaptation algorithm.

Adaptations vs. Stalling QoE: As quality adaptations are used to avoid stalling, we analyze the trade-off between the number of adaptations and the stalling QoE. Figure 9 shows the average stalling QoE (the higher the better) and the average number of quality bitrate adaptations for different player configurations in various networking environments. Here too, we note that all players and adaptation algorithms are represented on the Pareto-frontier such that no single player configuration dominates. The adaptation algorithm choice within DASH.JS shows again nearly no impact.

The figure shows that by allowing a few adaptations a substantial increase in stalling QoE is achieved. We note that the behavior of both players differs with regard to segment length. For DASH.JS, the adaptation count significantly increases for smaller segment lengths. In contrast, Shaka Player does not show such a dependency.

4.6 The Cost of Increased Target Buffer Sizes

So far, the analyzed metrics favored increased buffer sizes. However, depending on the user context large target buffer sizes may lead to resources wasting, for example, due to video session abandonment or partial skipping [17]. This triggers buffer content flushing which wastes resources such as download volume credits in cellular networks or wireless transmission energy on mobile devices.

To incorporate the impact of target buffer sizing, we derive an aggregated streaming performance metric $\mathcal{A}$ that includes the total stalling duration T^{Σ} in addition to the cost of the buffer size. We derive the cost of the buffer size as the expected amount of buffered data to be flushed upon session abandonment, i.e., $\bar{B}_A = \sum_t B(t)p(t)$, where $B(t)$ is the buffer filling at time t and $p(t)$ is the buffer abandonment probability at t. Note that $P_A = \sum_t p(t) \leq 1$ as videos may be well watched until the end. Empirical session abandonment distributions were measured for example in [17]. We calculate $\mathcal{A}$ as follows: $\mathcal{A} = \alpha\bar{B}_A + T^{\Sigma}$, with normalization factor α. Note that increasing the target buffer size reduces the overall stalling duration T^{Σ} but bears a higher risk of resource waste given session abandonment and vice versa.

Figure 10 shows a classification of optimal player configurations with regard to $\mathcal{A}$ depending on the environment, i.e., the available bandwidth, and the user abandonment context, i.e., P_A. For illustration, we use a uniform session abandonment distribution leading to $\bar{B}_A = \bar{B}$, i.e., the average buffer filling and a value $\alpha = 500$. Note that α is a relative weighting parameter of $\bar{B}_A$ vs T^{Σ}. In Figure 10 we observe that Shaka Player with large buffer sizes is superior for low abandonment probabilities P_A while DASH.JS dominates higher P_A. The available bandwidth influences the choice of the adaptation algorithm. From Figure 10 we deduce that a content / video provider can leverage these environment factors to choose the sweet spot configuration given session abandonment statistics.

4.7 Pitfalls in Simulations

Given the measured inconsistencies of the results observed for AStream, e.g., concerning the initial delay and stallings, we reason that the absence of an actual playback buffer[12] in *simulative playback* leads to potentially unrealistic assumptions in AStream's performance. For example, an *immediate* start of the video playback once *any* stream component is fetched in AStream can by explained by the way the `init` segment is processed. Depending

[12]such as the MSE API providing the video playback buffer in web browsers

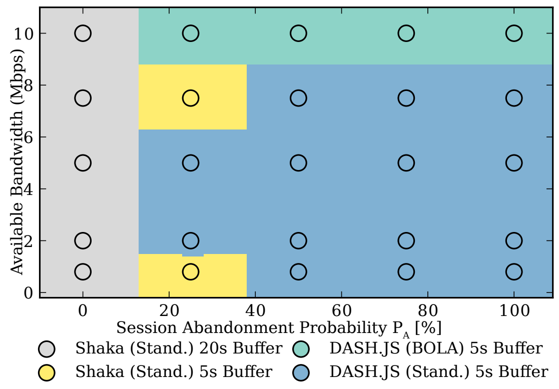

Figure 10: Optimal player configurations depending on the avail. bandwidth and session abandonment probability.

on the data set, the `mp4 init` segment contains the `moov`[13] element that describes stream meta-data, but does not contain video frames. It should not lead to playable content becoming available to the player. However, in our tests, fetching these segments did lead to an increase in AStreams buffer level.[14] Given that these `init` segments are respectively small in file size compared to segments containing frames, as represented by `m4s` segments, the buffer is falsely assumed to be filled very fast when initiating the stream. This obscures segment downloads and impacts initial adaptation decisions as the (false) high buffer filling leads to undesired up-adaptations, as well as stalling at the playback begin. In contrast, both JS players exhibit the expected behavior and begin playout after the first `m4s` segment is downloaded.

5 RELATED WORK

In the following, we focus on studies presenting performance assessments of DASH players using automated player comparisons under various system conditions. Thang et al. [24] analyze TBA and BBA based DASH streaming sessions, considering target buffer sizes and AA as configuration parameters. The experiments are conducted with two bandwidth traces (one replayed trace and one sudden drop trace). Our work differs from this approach by providing a system that allows for an automated player comparison under a large variety of system parameters, including wide range of systematically generated or captured bandwidth traces.

Maki et al. [15] relate DASH playback sessions' QoS factors to a set of configuration parameters, including segment lengths and target buffer sizes, while adapting the network environment using variations in bandwidth and packet loss. Sharing this basic concept and parameter space, our work adds the fundamental aspect of investigating different DASH players and adaptation algorithms, while relying on network emulations for performance analysis.

The work in [18] evaluates the performance of different DASH AA with a focus on investigating diverse Dynamic Adaptive Steaming (DAS) approaches in the contexts of TCP/IP and Information Centric Networking (ICN). The authors consider three classes of adaptation algorithms: BBA, TBA and a combined adaptation concept, each represented by the recent adaptation algorithm, i.e., *BOLA*, *PANDA*, and *AdapTech*, respectively. For their evaluation, each algorithm has been *reimplemented*, if necessary, within the *libdash* framework. In our present work, we take into account the respective DASH players and limit our investigation to BBA and TBA given the concrete implementations provided in those players. Hence, we are able to investigate algorithms given diverse influence factors of browsers and DASH player implementations, seeking to closely imitate the actual usage scenarios. Further our work investigates a broader spectrum of network scenarios and configuration parameters by conducting experiments on the Cartesian product spanning this parameter space. This results in a large experiment design space and allows the identification of configurations that provide the best outcome for given scenarios.

Independently of this work, a recent study also proposed the concept of comparing a wide range of different DASH players [27]. The presented work proposes to analyze and compare Shaka Player, DASH.JS as well as a wide range of other players focusing on QoE measures. In contrast, we provide both a parallelized large-scale evaluation framework and a systematic, detailed analysis and comparison of these players. Along these lines, in a previous work [23], we conducted a limited study using Shaka Player in relation to varying TCP congestion controls algorithms.

6 CONCLUSIONS AND FUTURE WORK

In this work, we provided a systematic study of the impact of the DASH player choice and configuration on the streaming performance. We established an execution environment for reproducible monitoring and evaluation of the performance of real-world and academic DASH players. Our evaluation of player performance affinities, that we denote as sweet spots, shows that suitably configured players can be deemed superior with respect to given QoE performance metrics. A highlight of our contributions is the observation that the choice of the target buffer size together with the player implementation dominates the choice of the adaptation algorithms. This stands in contrast to a majority of research efforts that are being directed towards investigating improvements in adaptation algorithms. Further, our developed methodology allows, e.g., an informed player selection and configuration at the beginning of streaming sessions to maximize QoE. Extensions to this work include the evaluation of commercial players, such as Bitmovin's HTML5 Player and proprietary adaptation algorithms as well as a real world verification of the identified superior configurations.

REPRODUCIBILITY

To enable other researchers to *reproduce* and *extend* our research, we release our evaluation framework as shown in Figure 2 at https://maci-research.net/DASH. The framework consists of:

(1) A web interface for composing and evaluating simulations and a set of scripts to set up simulator instances.
(2) The actual player execution environments and monitoring components for all analyzed players.
(3) The configurations as presented in this paper.

ACKNOWLEDGMENT

This work has been funded by the German Research Foundation (DFG) as part of the projects B4, C2 and C3 in the Collaborative Research Center (SFB) 1053 MAKI. This work was supported by the AWS Cloud Credits for Research program.

[13]http://l.web.umkc.edu/lizhu/teaching/2016sp.video-communication/ref/mp4.pdf
[14]https://github.com/pari685/AStream/blob/master/dist/client/dash_buffer.py#L129

REFERENCES

[1] Cisco. 2016. Cisco Visual Networking Index, 2014-2019 White Paper. (June 2016). http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf

[2] L. De Cicco, S. Mascolo, and V. Palmisano. 2011. Feedback Control for Adaptive Live Video Streaming. In *Proc. of ACM MMSys.* 145–156.

[3] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. 2011. Understanding the Impact of Video Quality on User Engagement. *SIGCOMM Comput. Commun. Rev.* 41, 4 (Aug. 2011), 362–373. DOI: https://doi.org/10.1145/2043164.2018478

[4] Alexander Frömmgen, Denny Stohr, Jan Fornoff, Wolfgang Effelsberg, and Alejandro Buchmann. 2016. Capture and Replay: Reproducible Network Experiments in Mininet. In *In Proc. of ACM SIGCOMM.* ACM, 621–622.

[5] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. 2012. Reproducible Network Experiments Using Container-based Emulation. In *Proc. of CoNEXT.* ACM, 253–264. DOI: https://doi.org/10.1145/2413176.2413206

[6] Tobias Hoßfeld, Raimund Schatz, Ernst Biersack, and Louis Plissonneau. 2013. Internet video delivery in youtube: From traffic measurements to quality of experience. *Lecture Notes in Computer Science* 7754 (2013), 264–301. DOI: https://doi.org/10.1007/978-3-642-36784-7-11

[7] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. 2012. Confused, timid, and unstable. In *Proc. of IMC '12.* ACM Press, 225. DOI: https://doi.org/10.1145/2398776.2398800

[8] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proc. of ACM SIGCOMM (SIGCOMM '14).* ACM, New York, NY, USA, 187–198. DOI: https://doi.org/10.1145/2619239.2626296

[9] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *ACM SIGCOMM Comput. Commun. Rev.* 187–198.

[10] Bert Hubert. 2016. Linux TC. http://linux.die.net/man/8/tc. (2016).

[11] Parikshit Juluri, Venkatesh Tamarapalli, and Deep Medhi. 2016. QoE management in DASH systems using the segment aware rate adaptation algorithm. In *Proc. of IEEE/IFIP NOMS.* IEEE, 129–136. DOI: https://doi.org/10.1109/NOMS.2016.7502805

[12] C. Kreuzberger, B. Rainer, H. Hellwagner, L. Toni, and P. Frossard. 2016. A Comparative Study of DASH Representation Sets Using Real User Characteristics . In *Proc. of NOSSDAV.*

[13] Stefan Lederer, Christopher Müller, and Christian Timmerer. 2012. Dynamic adaptive streaming over HTTP dataset. In *Proc. of ACM MMSys.* ACM Press, New York, NY, USA, 89. DOI: https://doi.org/10.1145/2155555.2155570

[14] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A.C. Begen, and D. Oran. 2014. Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale. *IEEE JSAC* 32, 4 (April 2014), 719–733.

[15] Toni Maki, Martin Varela, and Doreid Ammar. 2015. A Layered Model for Quality Estimation of HTTP Video from QoS Measurements. In *Proc. of SITIS.* 591–598.

[16] Abhijit Mondal, Satadal Sengupta, Bachu Rikith Reddy, M. J.V. Koundinya, Chander Govindarajan, Pradipta De, Niloy Ganguly, and Sandip Chakraborty. 2017. Candid with YouTube. In *In Proce. of NOSSDAV.* ACM Press, New York, New York, USA, 19–24. DOI: https://doi.org/10.1145/3083165.3083177

[17] H. Nam, K. H. Kim, and H. Schulzrinne. 2016. QoE matters more than QoS: Why people stop watching cat videos. In *Proc. of IEEE INFOCOM.* 1–9.

[18] Jacques Samain, Giovanna Carofiglio, Luca Muscariello, Michele Papalini, Mauro Sardara, Michele Tortelli, and Dario Rossi. *Dynamic Adaptive Video Streaming: Towards a systematic comparison of ICN and TCP/IP.* Technical Report.

[19] Sandvine. 2016. Global Internet Phenomena Report 2016: Latin America & North America. (December 2016). https://www.sandvine.com/downloads/general/global-internet-phenomena/2016/global-internet-phenomena-report-latin-america\-and-north-america.pdf

[20] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hossfeld, and P. Tran-Gia. 2015. A Survey on Quality of Experience of HTTP Adaptive Streaming. *IEEE Communications Surveys Tutorials* (2015), 469–492. DOI: https://doi.org/10.1109/COMST.2014.2360940

[21] I. Sodagar. 2011. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE MultiMedia* 18, 4 (April 2011), 62–67. DOI: https://doi.org/10.1109/MMUL.2011.71

[22] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. 2016. BOLA: Near-optimal bitrate adaptation for online videos. In *Proc. of IEEE INFOCOM.* IEEE, 1–9. DOI: https://doi.org/10.1109/INFOCOM.2016.7524428

[23] Denny Stohr, Alexander Frömmgen, Jan Fornoff, Michael Zink, Alejandro Buchmann, and Wolfgang Effelsberg. 2016. QoE Analysis of DASH Cross-Layer Dependencies by Extensive Network Emulation. In *Proc. of Workshop on QoE-based Analysis and Management of Data Communication Networks.* ACM, 25–30.

[24] Truong Cong Thang, Hung T. Le, Anh T. Pham, and Yong Man Ro. 2014. An evaluation of bitrate adaptation methods for HTTP live streaming. *IEEE J. Sel. Areas Commun.* 32, 4 (2014), 693–705. DOI: https://doi.org/10.1109/JSAC.2014.140403

[25] C. Wang, A. Rizk, and M. Zink. 2016. SQUAD: A Spectrum-based Quality Adaptation for Dynamic Adaptive Streaming over HTTP. In *Proc. of ACM MMSys.* ACM, 1:1–1:12.

[26] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. 2015. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proc. of ACM SIGCOMM.* 325–338.

[27] Anatoliy Zabrovskiy, Evgeny Kuzmin, Evgeny Petrov, Christian Timmerer, and Christopher Mueller. 2017. AdViSE. In *Proceedings of the 8th ACM on Multimedia Systems Conference - MMSys'17.* ACM Press, New York, New York, USA, 217–220. DOI: https://doi.org/10.1145/3083187.3083221

[28] Michael Zink, Jens B. Schmitt, and Ralf Steinmetz. 2005. Layer-encoded video in scalable adaptive streaming. *IEEE Trans. Multimedia* 7, 1 (2005), 75–84.