Server and Content Selection for MPEG DASH Video Streaming with Client Information

Florian Wamser, Steffen Höfner, Michael Seufert, Phuoc Tran-Gia
Chair of Communication Networks, University of Würzburg, Würzburg, Germany
{wamser,hoefner,seufert,trangia}@informatik.uni-wuerzburg.de

ABSTRACT

In HTTP adaptive streaming (HAS), such as MPEG DASH, the video is split into chunks and is available in different quality levels. If the video chunks are stored or cached on different servers to deal with the high load in the network and the Quality of Experience (QoE) requirements of the users, the problem of content selection arises. In this paper, we evaluate client-side algorithms for dynamically selecting an appropriate content server during DASH video streaming. We present three algorithms with which the DASH client itself can determine the most appropriate server based on client-specific metrics, like actual latency or bandwidth to the content servers. We evaluate and discuss the proposed algorithms with respect to the resulting DASH streaming behavior in terms of buffer levels and quality level selection.

ACM Reference format:
Florian Wamser, Steffen Höfner, Michael Seufert, Phuoc Tran-Gia. 2017. Server and Content Selection for MPEG DASH Video Streaming with Client Information. In *Proceedings of Internet QoE, USA, 2017,* 6 pages.
DOI: 10.1145/3098603.3098607

1 INTRODUCTION

HTTP adaptive streaming (HAS), standardized by MPEG as Dynamic Adaptive Streaming over HTTP (DASH), is one of the most important streaming approaches in today's Internet. The market leaders YouTube and Netflix decided to go for DASH in early 2013 [8] and late 2015 [9], respectively. The reason for this is plain obvious: the heavy and proprietary Flash browser plug-in can be avoided and the vendor-independent HTML5, coupled with adaptive and client specific video streaming, offers itself as a future-proof, open-source, and widely supported solution.

Internet QoE, USA

DOI: 10.1145/3098603.3098607

MPEG DASH is gaining momentum, however, the high popularity and the rising user demands pose severe challenges for DASH and its underlying architecture. With DASH, the video files are split up in a sequence of small file chunks, that are downloaded on demand from the streaming server. The chunks are stored in various quality levels, e.g., different resolutions and formats, on the content server and are described in a manifest file, which is downloaded by the client at the beginning of the streaming session.

In today's approaches, the video chunks are almost always stored on the same content server, which is usually only fully replicated for load balancing purposes. Due to today's load situation and the required proximity of the video content to the user, full replication of all DASH video chunks on all servers is impractical and not useful from the energy and resource consumptions' point of view. Instead, it is desirable to distribute the individual video chunks on different servers, so that, for example, low resolution chunks of short videos are stored in servers in mobile networks (as many smartphone users access short clips of medium quality), while high resolution video content is cached within fixed "eyeball" networks.

If content is stored redundantly and can be requested in different qualities on different servers, the problem of content selection arises. *Client-based server selection* is one possible solution to counteract the problem [10]. The idea behind this is that the client determines and utilizes the best available connection to the content servers in order to reach the highest Quality of Experience (QoE). A so-called entrance server provides a list of available hosts and quality levels to the client, using a DASH manifest file, i.e., a Media Presentation Description (MPD) file. The client then uses an individual server selection algorithm to determine which content server will be requested for content. The advantage is that the server selection can be based on client-specific metrics to reach a high QoE, e.g., client properties, latency, bandwidth, and availability.

In this paper, we evaluate algorithms for selecting the streaming content server for DASH video streaming. We present three algorithms that determine the content server due to different metrics, namely (1) a latency-based algorithm, which takes into account the current client-server

latency, (2) a bandwidth-based approach, which probabilistically selects the server with the highest bandwidth, and (3) a weighted bandwidth-based algorithm, which emphasizes the probability of the best server. Eventually, we compare the algorithms to a baseline streaming, and thus, provide insights into the performance of HAS with client-side server selection algorithms. Based on the obtained results from the performance evaluation, the advantages and disadvantages of the different algorithms are discussed.

The rest of the work is structured as follows. Related work referring to video streaming is described in Section 2. The concept for client-based server selection is presented in Section 3 as well as the different selection algorithms. In Section 4, the results are discussed and evaluated. Conclusions are finally given in Section 5.

2 RELATED WORK

In section, we present work that is related to adaptive video streaming and try to highlight similarities and differences to our work. The *DASH Industry Forum* (DASH-IF) is working on implementation guidelines for DASH with H.264/AVC and H.265/HEVC [2]. In order to improve DASH, the authors of [6] show the benefits of using Scalable Video Coding (SVC) for a DASH environment. They conclude that using SVC would lead to a better experience for users and a higher number of satisfied users. Comparisons of different streaming solutions can be found in [1, 4, 7]. In [7], Seufert et al. present a survey on the quality of experience of HTTP Adaptive Streaming and compare the different solutions to adaptive video streaming. They also give an overview of the current state of the art and recent developments. In [1], the authors focus on the rate-adaptation mechanisms of adaptive streaming and evaluate two commercial players (i.e. Smooth Streaming by Microsoft and Netflix) and one open source player (OSMF). In their work, they identify differences between these players.In [4], Hoßfeld et al. present a user-centric evaluation of adaptation logics for HTTP Adaptive Streaming. Therefore, they compare existing algorithms to their own SVC based solution, regarding user achieved quality of experience. They are able to show that their own mechanism outperforms the other algorithms in terms of video quality, switching frequency, and utilization of the available resources for the investigated network scenario. While adaptive streaming is related to the client side, often mechanisms such as load balancing are active on the server side. For this purpose, data and content are distributed in a CDN network. By DNS name resolution, the requests can be distributed to the various CDN servers to avoid overloaded servers. Our algorithms proposed in the paper go further than simple CDN load balancing solutions since they take into account the nature of the content. Thus a fine-granular

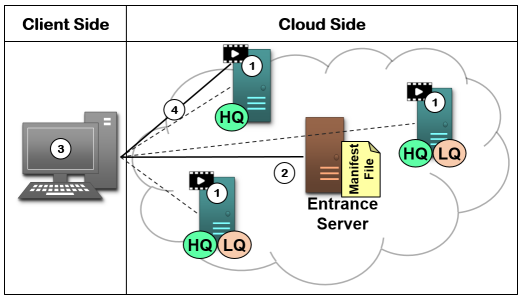


Figure 1: Concept of client side server selection.

decision can be made. The disadvantage of our solution is that it is more complex to implement.

3 CONCEPT FOR CLIENT-BASED SERVER SELECTION

We now briefly cover the concept of client side server selection that is featured in this work [10]. We depict a typical server selection scenario in Figure 1. New content chunks are registered at a master or entrance server so that this server always owns a list of all available content (1). This server then lists the chunks in a manifest file. This manifest file contains the IP addresses as well as optional information regarding the instances, e.g. available quality levels of the provided content. The master or entrace server is the only new instance that must be added to the server side. It can be implemented as a logical instance on several physical hosts. It knows all content servers and builds the manifest files for the clients. Once a client connects to the service, it will first request the manifest file from the entrance server (2). Thereafter, the client can utilize several client side metrics (3), e.g. latency, available bandwidth or display resolution, to select the most appropriate content server. In the last step (4), the client establishes a connection to the most appropriate host based on its own measurements. To ensure that the optimal host is selected at all times, the client can repeat steps (3) and (4) periodically.

We now introduce three client side server selection algorithms. These algorithms distribute requests to all content servers to measure and dynamically select the best server. They complement the quality adaptation logic, which additionally runs after the server selection, and is explicitly not in the focus of this work.

Latency-based. The first algorithm utilizes the latency between client and content servers as a metric for the selection of the content server. This approach enables the client to detect server outage or congestion on the current network link. As input, the *Bandwidth* algorithm requires a manifest file, which contains the IP addresses of the available content servers. For the latency estimation, we make use of the *ping* tool, which is standard for most Unix distributions. With

this tool, we periodically measure the latency to each of the content servers available in the manifest file and set the current host for the video downloads accordingly. In order to verify the correctness of latency estimation functionality, we performed an adequate amount of measurements with our latency algorithm. The algorithm proved to estimate the correct latency with only small deviations for higher latencies.

Bandwidth-based. The idea behind this server selection algorithm is to passively measure the current bandwidth of the available content servers, whenever a video chunk is downloaded, to select the most appropriate content server. The algorithm is called whenever a new video chunk is requested by the DASH player and determines which content server is to be used as host for the current chunk download. The behavior of the algorithm can be divided in two phases. In the first phase, the algorithm simply returns the addresses of content servers, which have not been used by the client as host yet. This ensures that the available bandwidth of each of the content servers is measured once. If the available bandwidth of all content servers has been measured, the second phase starts. In the second phase, whenever the algorithm is called, it assigns each of the content servers a selection probability, which is equal to the content server's individual bandwidth divided by the sum of the bandwidths of all available content servers. The actual server for the currently requested video chunk is then selected randomly according to the previously assigned selection probability. After the download of a video chunk has ended successfully, the current bandwidth of the associated content server is updated to the newly measured value.

Weighted Bandwidth-based. The weighted bandwidth-based algorithm was developed as an improvement to the standard bandwidth algorithm. Therefore, we altered the previously introduced bandwidth algorithm in such a manner that the currently best content server is preferred with a specific probability. This probability will be referred to as *weight* in the following. For this purpose, a uniform random number Y with $0 \leq Y < 1$ is additionally drawn. If it is smaller than the weight, the currently best content server is selected, else the normal algorithms is applied. The idea behind this is to prevent the algorithm from selecting an unfavorable content server too frequently. Moreover, this solution is supposed to be suited to deal with a higher number of available content servers since the participation of the best available server in the overall bandwidth decreases as the number of content servers increases.

4 RESULTS

In this section, the scenario to evaluate the different server selection algorithms is described. We chose the well known open-source movie *"Big Buck Bunny"* (https://peach.blender.org/) as video content. This video clip was converted in seven different quality levels and divided in 158 video chunks with a playtime of around four seconds each. The bit rate ranges from 600 Kbps for Quality 0 to 8000 Kbps for Quality 6:
(0): 640x360, 600 Kbps, (1): 640x360, 1000 Kbps, (2): 854x480, 1400 Kbps, (3): 1280x720, 2000 Kbps, (4): 1920x1080, 3500 Kbps, (5): 2560x1440, 4500 Kbps, (6): 3840x2160, 8000 Kbps.

In the evaluation scenario, we instantiated two independent content servers with a limited upload bandwidth of 8 Mbps each. Additionally, we used multiple download clients to continuously generate background traffic between 1 Mbps and 7 Mbps, in order to simulate server load. Every 20 s, the total background traffic of each server is increased or decreased by 1 Mbps in an anti-cyclical pattern, which is depicted in Figure 2. Server 1 (blue) starts to offer 1 Mbps and its available bandwidth increases to 7 Mbps, which is reached after 120 s. Afterwards, the available bandwidth is decreased and reaches 1 Mbps at 240 s. In the same interval, Server 2 (red) starts with an available bandwidth of 7 Mbps, which is decreased to 1 Mbps at 120s, and again increased to 7 Mbps at 240 s. This pattern is periodically repeated until the end of the video streaming. The fluctuations of the ping times were additionally measured. The measured latency is very low up to a load of 8 Mbps, since the bandwidth limitation does not come to bear. From 8 Mbps, the ping goes to an average value of 807ms in our scenario. The video is available on both servers, and we use a single client with a TAPAS [3] player to stream the video. As the focus of this work is not on the adaptation logic of the video streaming player, but rather on the server selection mechanism, we use the *native* TAPAS algorithm as adaptation logic. It continuously monitors the buffered playtime of the video and the available bandwidth of the content server [5]. These values are then compared to the available quality levels and the most appropriate quality level is chosen in order to maintain a high level of buffered playtime.

4.1 Baseline Results

In the baseline scenario, the DASH client only requests chunks from a single content server. We selected Server 1 and evaluate 50 streaming sessions in this scenario. Figure 3 presents the playtime of the video on the x-axis, the buffered playtime in seconds on the left y-axis and the level of the video resolution on the right y-axis. The presented results show the mean buffered playtime (green) and mean quality level (pink) over the 50 streaming sessions including the 95% confidence intervals. Generally, we can divide the streaming in three main sections. The first section is an initial buffering phase, in which the buffered playtime is quickly filling up to a maximum of 50 s. This buffering phase approximately covers the first 80 s of playtime. In the next section until about 580 s of playtime, the buffer level fluctuates closely

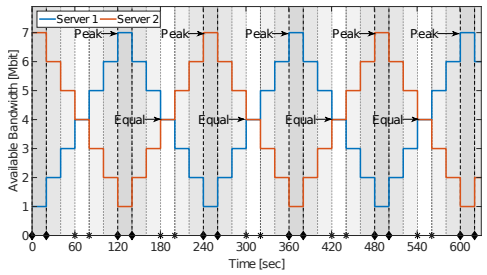


Figure 2: Bandwidth pattern of each content server at the evaluation scenario.

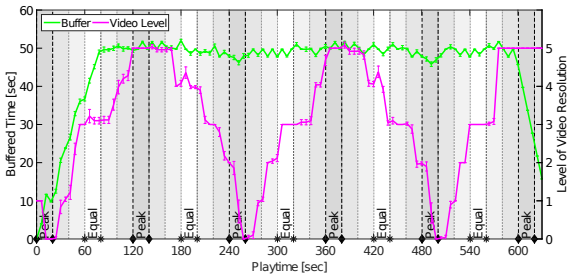


Figure 3: Video playback time versus buffered playtime and quality level with the native TAPAS algorithm.

around its maximum value of 50 s. In the last section, all video chunks have been downloaded successfully and the buffer level is consequently decreasing to zero.

In contrast to the buffer level, which is mostly stable in the second phase, the quality level is highly depending on the available bandwidth of Server 1. It can be seen that the video resolution reaches its maximum whenever Server 1 has a peak in the available bandwidth, i.e., at 120 s, 360 s, and 600 s of playtime. Analogously, the resolution is lowest, whenever the available bandwidth reaches a trough, i.e. at 0 s, 240 s, and 480 s of playtime. Note that the peaks and troughs of the quality level are slightly shifted to the right compared to the corresponding available bandwidth of Server 1. This is caused by the fact that the adaptation is based on past measurements, and the adaption logic aims to maintain a stable video resolution as long as possible and avoid frequent quality adaptation, which could negatively influence the QoE.

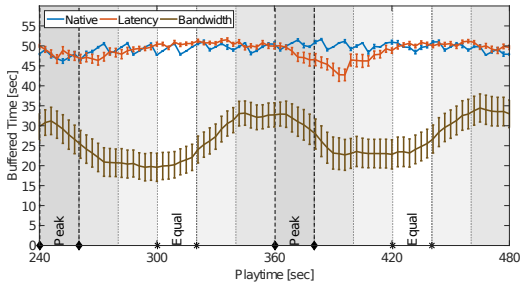
4.2 Results for Latency- and Bandwidth-based Server Selection

We now compare the baseline results to the results, in which a latency- or bandwidth-based server selection is enabled. As the initial buffering phase and the final de-buffering phase are identical for each algorithm, we will only focus on the specific interval between 240 s and 480 s of playtime, which covers a complete bandwidth cycle for each of the two content servers.

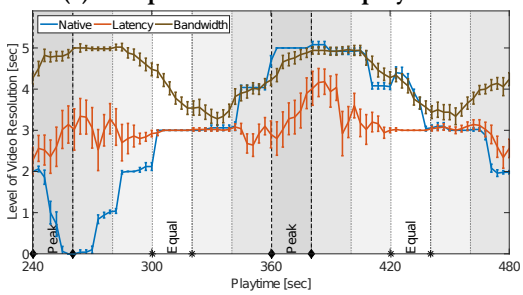
Figure 4 shows the average buffered playtime and 95% confidence intervals for 50 independent streaming runs for each of the following algorithms: Native TAPAS (i.e., baseline) adaptation logic (blue), latency-based server selection (red), bandwidth-based server selection (brown). It can be seen, that the latency-based server selection results in a similar buffer behavior around 45 s to 50 s as in the baseline scenario. In contrast, the buffer level of the bandwidth-based server selection oscillates between 15 s and 37 s. Thereby, the buffer level reaches its maximum in the 20 s interval right before the peaks in the available bandwidth, and generally mimics the course of the maximum available bandwidth with a shift of 20 seconds to the left.

Figure 4b shows the average quality level and 95% confidence intervals for the same period. It can be seen that the average quality level of the latency-based server selection lies between 2 and 4 in this period with small confidence intervals when the available bandwidth of the content servers is equal. During the bandwidth peaks, the confidence intervals notably increase. This shows that the latency-based algorithm is not able to consistently select the most appropriate content server, which results in a high variance of the selected video resolution. The quality levels of the bandwidth-based server selection periodically alternates between 3 and 5, similar to the corresponding course of the buffered playtime. However, the peak and trough intervals of the video quality are shifted to the right by around 20 s in comparison to the maximum available bandwidth, just like in the baseline scenario with the native TAPAS algorithm.

In the following, we will discuss the behavior of the bandwidth-based server strategy in detail for the interval from 320 s to 440 s. During this interval, the average quality level is close to the naive TAPAS algorithm because Server 1 has the highest available bandwidth in this time period. From 320 s to 340 s, the average buffered playtime increases from 24 s to 31 s. In the same interval, the average quality level reaches a trough at around 3.3, just as expected because both servers offered equal bandwidth at 320 s. In the next interval of 20 s, we would expect both the buffered playtime and the quality level to increase because the available bandwidth on one of the servers increases. However, only the average quality level increases while the average buffered playtime does not change at all. This effect is caused by the fact, that the bandwidth algorithm is based on a randomized access strategy. In this interval, the probability to choose the content server with the lower available bandwidth is equal to 25 %, which results in high download times for the current video chunk. At the same time, the quality level is only adjusted when the best available content server is chosen. This intensifies the negative effect on download time and the buffer level.



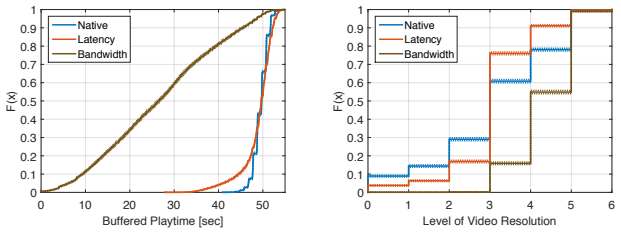
(a) Comparison of buffered playtime.



(b) Comparison of video quality level.

Figure 4: Comparison of the latency-based algorithm and bandwidth-based algorithm.

This becomes even more visible in the interval from 360 s to 380 s, in which the difference in the available bandwidth of the content servers is at its maximum, i.e., 7 Mbps at Server 1 and 1 Mbps at Server 2, while the average quality level reaches the peak at 5. As we can see, the average buffered playtime starts to decrease in this period, which is caused by an unfavorable server selection.



(a) CDF of buffered playtime. **(b) CDF of the quality level.**

Figure 5: CDFs of streaming parameters for the native, latency-, and bandwidth-based algorithms.

Over the whole course of the streaming, the bandwidth-based server selection results in a much lover average buffer level than the other two algorithms. However, the average quality level converges to the ideal curve resulting from the available bandwidth in the system. This is confirmed by the CDFs of the buffered playtime and quality level, which are presented in Figure 5a and 5b, respectively.

As shown in Figure 5a, the probability of the buffered playtime being at a certain level between 0 s and 50 s is almost linear for the bandwidth algorithm. Also, the probability of the buffer level being below 10 s is at around 11%, which leads to frequent occurrences of unwanted playback stalling. The buffered playtime of the other two algorithms however, is higher than 27 s with a probability of 100% (the start and end of the streaming are excluded). Thus, both algorithms are unaffected by stalling. However, if we have a look at the CDF of the quality level in Figure 5b, we can see that the video level of the bandwidth algorithm is much better than the quality level of the other algorithms. The quality level of the bandwidth-based algorithm never drops below a value of 3, while the probability of the other algorithms dropping below a video resolution of 3 is at 18 % for the latency-based server selection and 30% for the native algorithm. Also, the probabilities of the quality level being at a high value is much higher for the bandwidth algorithm. Thereby, the probability of the quality level being equal to 4 or greater is at 82% for the bandwidth-based server selection, while the best of the two remaining algorithms only reaches a probability of 40%.

Although the latency-based server selection is able to maintain a stable, high buffer level like in the baseline scenario, it is not able to significantly improve the streamed video resolution. In contrast, the bandwidth-based server selection reaches generally higher quality levels, which nicely reflect the available bandwidth in the system. However, the buffer levels of this algorithm are much lower, which increases the risk of stalling. This is caused by a poor selection of the most beneficial content server due to the random selection algorithm, which is tied to the content servers' participation in the overall available bandwidth. The weighted bandwidth-based server selection aims to overcome this drawback by adding additional weight to the content server with the highest available bandwidth. The corresponding results will be discussed in the following section.

4.3 Results of the Weighted Bandwidth-based Server Selection

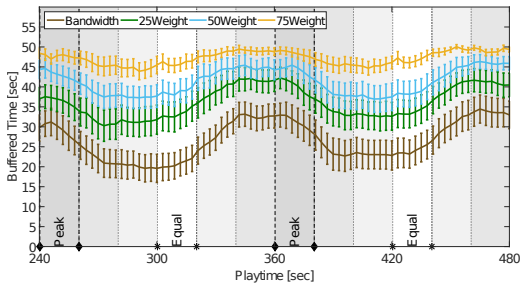
Figure 6 compares the average temporal courses of buffered playtime and the quality of the bandwidth-based algorithm and the weighted bandwidth-based algorithm in the interval from 240 s to 480 s. Three different weights of 25 %, 50 %, 75 % are considered. Figure 6a shows that the average buffer level increases in correlation with the chosen weight. Furthermore, for higher weights, also the amplitude of the buffer level fluctuation decreases. This can be explained by the fact that the probability of probing a server with less available bandwidth is decreased drastically by adding weight to the current best server. For the temporal course of the quality level in Figure 6b, we can observe that the courses of the average quality levels for the different weights are very close to each other with overlapping confidence intervals. This

suggests that the resulting quality is not affected by the used weight. However, it is essential that the weight is not set too high in order to ensure a constant probing of the different content servers.

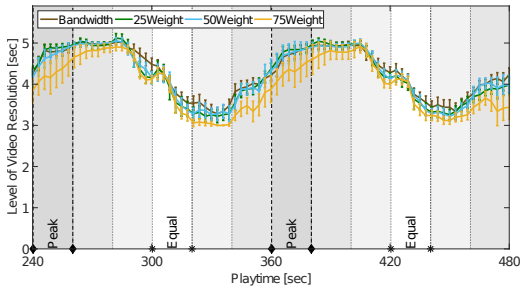
To sum up, the weighted bandwidth algorithm with a weight of 50 % showed the best overall results for the relation between buffer level and video resolution among all server selection algorithms in the current scenario. To further increase the consistency of the buffer level, it is possible to increase the weight at cost of the video resolution. Thus, the performance of this algorithm highly depends on the optimal selection of the weight parameter, such that in other scenarios a different weight might be more appropriate.

5 CONCLUSION

In this paper, distributed DASH video streaming is evaluated. The idea is to develop a client side server selection mechanism, which reliably selects the current best content server, in order to optimally utilize the available bandwidth to ensure a high quality of the video playback. Therefore, a latency based and a bandwidth based server selection strategy have been developed. The results show that the latency based algorithm is capable of maintaining a high buffer level even if the available bandwidth of the content servers is continuously changing. However, it does not choose the content servers efficient enough to achieve a high video quality. In contrast to this, the bandwidth based algorithm always chooses a video quality that matches the highest available bandwidth among the content servers. However, the buffer level of the bandwidth algorithm was relatively low during the video playback, which may lead to frequent playback interruptions. This is caused by frequent probing of disadvantageous content servers. Thus, the client receives less bandwidth than required. As a consequence, the bandwidth algorithm was extended by a weighted server probing system, which reduces the probing frequency of disadvantageous content servers. The evaluation of the improved bandwidth based selection algorithm with a high weight shows that it is capable of maintaining decent buffer values and a high video quality in scenarios with high differences in the available bandwidth of the content servers. For future work, we plan to enhance our measurement scenario by adding multiple clients, different bandwidth patterns and more servers.



(a) Comparison of buffered playtime.



(b) Comparison of video quality level.

Figure 6: Comparison of the bandwidth-based algorithm and the weighted bandwidth-based algorithm with 25%, 50% and 75% weight.

This work was partialy supported by German Research Foundation (DFG) under Grant No. KE 1863/6-1, TR 257/41-1 and the H2020 INPUT (H2020-ICT-2014-1, Grant No. 644672).

REFERENCES

[1] Saamer Akhshabi, Ali C. Begen, and Constantine Dovrolis. 2011. An Experimental Evaluation of Rate-adaptation Algorithms in Adaptive Streaming over HTTP. In *Second Annual ACM Conference on Multimedia Systems (MMSys '11)*. ACM, USA, 157–168.

[2] DASH Industry Forum. 2015. Guidelines for Implementation: DASH-IF Interoperability Points. (2015).

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

[4] 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).

[5] Zhi Li, Xiaoqing Zhu, Joshua Gahm, Rong Pan, Hao Hu, Ali C Begen, and David Oran. 2014. Probe and adapt: Rate adaptation for HTTP video streaming at scale. *IEEE Journal on Selected Areas in Communications* 32, 4 (2014), 719–733.

[6] Yago Sánchez de la Fuente, Thomas Schierl, Cornelius Hellge, Thomas Wiegand, Dohy Hong, Danny De Vleeschauwer, Werner Van Leekwijck, and Yannick Le Louédec. 2011. iDASH: improved dynamic adaptive streaming over HTTP using scalable video coding. In *Second Annual ACM conference on Multimedia systems*. ACM, 257–264.

[7] Michael Seufert, Sebastian Egger, Martin Slanina, Thomas Zinner, Tobias Hoßfeld, and Phuoc Tran-Gia. 2015. A Survey on Quality of Experience of HTTP Adaptive Streaming. *IEEE Communications Surveys & Tutorials* 17 (2015).

[8] Rajeev Tiwari. 2013. MPEG-DASH Support in Youtube. (2013). http://streamingcodecs.blogspot.de/2013/01/mpeg-dash-support-in-youtube.html

[9] Matt Trunnell. 2015. HTML5 Video is now supported in Firefox. (2015). http://techblog.netflix.com/2015/12/html5-video-is-now-supported-in-firefox.html

[10] Florian Wamser, Michael Seufert, Steffen Höfner, and Phuoc Tran-Gia. 2016. Concept for Client-initiated Selection of Cloud Instances for Improving QoE of Distributed Cloud Services. In *ACM SIGCOMM Workshop, Internet-QoE*. Florianópolis, Brazil.