Automated Performance Evaluation of Adaptive HTML5 Player Deployments

Anatoliy Zabrovskiy, Evgeny Petrov,
Evgeny Kuzmin
Petrozavodsk State University
Petrozavodsk, Russia
{z_anatoliy, johnp, kuzmin}@petrsu.ru

Christian Timmerer
Alpen-Adria-Universität Klagenfurt
Klagenfurt, Austria
christian.timmerer@itec.aau.at

***Abstract*—Adaptive video streaming over HTTP is becoming omnipresent in our daily life. In the past, dozens of research papers have proposed novel approaches to address different aspects of adaptive streaming and a decent amount of player implementations (commercial and open source) are available. However, state of the art evaluations are sometimes superficial as many proposals only investigate a certain aspect of the problem or focus on a specific platform – player implementations used in actual services are rarely considered. HTML5 is now available on many platforms and foster the deployment of adaptive media streaming applications. We propose a common evaluation framework for adaptive HTML5 players and demonstrate its applicability by evaluating eight different players which are actually deployed in real-world services.**

I. Introduction

Using adaptive streaming techniques over HTTP is nowadays state of the art and massively deployed on the Internet adopting the over-the-top (OTT) paradigm, i.e., these services are deployed on top of existing infrastructures. For example, Netflix and YouTube alone account for more than 50% of the traffic at peak periods [13]. Although Internet capacity is constantly increasing for both fixed and mobile networks, the adoption of new streaming services will continue as well as new applications and services will emerge. Major formats in this domain are MPEG-DASH and Apple's HLS which both have the same underlying principles.

However, the current effort in MPEG referred to as Common Media Application Format (CMAF) [5] aims at harmonizing at least segment formats and it is expected that soon DASH and HLS will support ISO base media file format (ISOBMFF) segments which are compatible with each other. In such a situation the most interest aspect – at least from a research perspective – is the rate adaption logic of players, because it is not defined in the standard and left open for competition.

In the past, many studies for such a rate adaptation logic were proposed and/or evaluated, e.g., [14], [15]. However, most of them focus on the development of new adaptation algorithms and compare it only with a limited subset of existing approaches [8], [4], [16], [19]. Evaluations of real-world deployments are very rare [12]. Additionally, it is also difficult to come up with a comprehensive evaluation of existing approaches due to the lack of the appropriate tools. Hence, the main contributions of this paper are as follows: *(i)* we developed an adaptive video streaming evaluation framework for the automated testing of different players and, consequently rate adaptation logics; *(ii)* we identified eight well-known adaptive HTML5 players (commercial and open source) and integrated them in our framework; *(iii)* we conducted a series of experiments to prove the suitability and usefulness of our framework for the comparison of players and rate adaptation logics. In general, in this paper we present a novel approach and framework for the automated evaluation of adaptive HTML5 players. A full-length version of this paper can be found here [18].

The rest of the paper is organized as follow. Section II introduces the general architecture of our evaluation framework. The setup for the evaluation is described in Section III. Results are presented and discussed in Section IV. Section V concludes the paper and highlights future work.

II. System architecture

In this section we describe our system architecture enabling the automated evaluation of adaptive streaming systems within a controlled environment. Our proposed system comprises the following components:

- Web server with standard HTTP hosting.
- Network emulation server.
- Selenium server.
- Web management interface.
- Adaptive HTML5 players.

The system architecture is depicted in Fig. 1 and consists of the three servers running Ubuntu OS (version 16.04 LTS) and connected using Gigabit Ethernet switches. It defines a flexible system that allows adding new adaptive HTML5 players easily. There is an algorithm which describes the sequence of the steps of embedding a new player and its API.

The *Web server* hosts the video content for adaptive streaming over HTTP. For our experiments we adopted the MPEG-DASH format. Therefore, the video sequence will be provided in multiple configurations (e.g., bitrates, resolutions) which are referred to as representations. Each video sequence will be divided in segments of equal length measured in seconds of video content. Multiple versions of the same content, each version segmented in multiple smaller files. This enables the dynamic adaptation at segment boundaries according to the given context. It also hosts a MySQL database for collecting

ISSN 2305-7254

all the performance measurements and the *Web management interfaces* for configuring and conducting the experiments. It is accessible from outside the controlled environment, everything else is within a controlled environment in order to avoid any cross-traffic that may influence the experiments.

The *Selenium server* [2] is an open source software testing framework for Web applications which is used to automatically conduct our experiments with adaptive HTML5 players running within a Web browser. In our case we adopted the Google Chrome browser but it is also possible to use other browsers on various platforms (desktop, mobile, operating systems). The Selenium server is activated through the Web management interface to run the various experiments automatically according to a given configuration.

For the *Network emulation server* we have adopted the Mininet emulator [1]. Although this emulator is basically used for emulating Software Defined Network (SDN) environments, it has been also used for streaming environments [17]. It provides a straightforward and extensible Python API for network creation and prototyping. We have utilized that functionality to create a virtual link with changeable network throughput characteristics. Our Network emulation server comprises two network interfaces (eth0, eth1). A python script is used to create a virtual network which consists of one switch connected to the real network using a TCLink [3]. The TCLink is a Mininet performance-modeling link. We setup our TCLink to change its characteristics at the specified moments of time. The schedule is stored within a file using JSON format. Finally, we made this schedule configurable through our Web management interface.

The *Web management interface* provides two functions, *(i)* one for configuring and conducting the experiments and *(ii)* one which includes the player and provides real-time information about the currently conducted experiment. Thus, the proposed framework in this paper provides means for comprehensive end-to-end evaluations of adaptive streaming services over HTTP including the possibility for subjective quality testing. The interface allows to define the following items and parameters:

- configuration of network emulation profiles including the bandwidth trajectory, packet loss, and packet delay;
- specification of the number of runs of an experiment; and
- selection of the adaptive HTML5 player (or rate adaptation logics) and the utilized adaptive streaming protocol (MPEG-DASH or HLS).

The result page provides a list of conducted experiments and the analytics section contains various metrics of the conducted experiments. It is possible to generate graphs of the results and export the raw values for further offline analysis. The following quality parameters and metrics are currently available: download video bitrate; video buffer length; video startup time; stalls (or buffer underruns); number of quality switches; average video bitrate; instability and inefficiency [6]; simple QoE models specially designed for the adaptive streaming solutions [9], [10].

Before starting the experiment we need to create a bandwidth trajectory profile. For each profile we can define duration

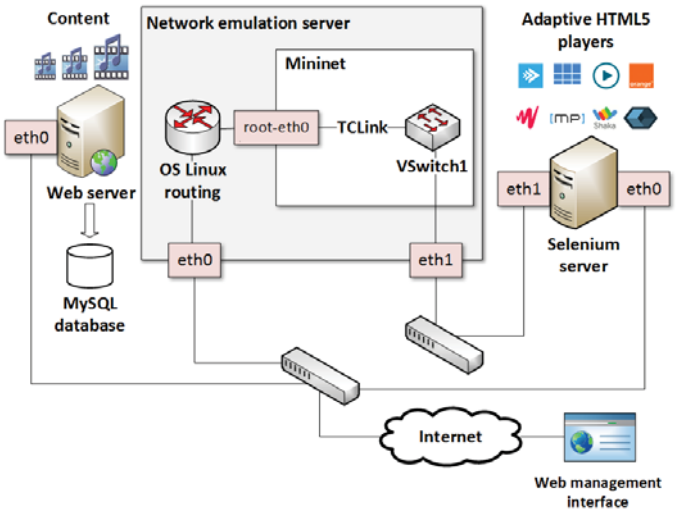

Fig. 1. System architecture

TABLE I. OVERVIEW OF THE ADAPTIVE HTML5 PLAYERS.

Media player	Version	Web site (last access: May 27, 2017)
Bitmovin Player	7.0	https://bitmovin.com
dash.js	2.4.0	http://dashif.org
Flow Player	6.0.5	https://flowplayer.org
HAS Player	1.7	https://github.com/Orange-OpenSource/hasplayer.js
JW Player	7.6.1	https://www.jwplayer.com
Radiant MP	3.10.8	https://www.radiantmediaplayer.com
Shaka Player	2.0.3	https://github.com/google/shaka-player
VideoJS Player	5.9.2	http://videojs.com

of each stage, bandwidth, delay, and packet loss. As soon as we start an experiment within the Web management interface, the Google Chrome browser (version 55.0.2883.87 64 bit) is automatically launched on the Selenium server and the selected network profile including the link parameters is sent to Network emulation server. The actual requests for the video content towards the Web server goes through the Network emulation server. When running an experiment it is possible to display the currently selected adaptive HTML5 player (including the video streaming) and real-time information about the currently conducted experiment. Details about the actual evaluation setup including all parameters and metrics are described in Section III. All of the selected *Adaptive HTML5 players* (cf. Table I in alphabetic order) have been available at no or relatively low costs for evaluation purposes.

III. EVALUATION SETUP

In this section, we define the setup for evaluating and comparing the adaptive HTML5 players. We describe what content is used and how it has been encoded, details about the network configuration, and the metrics used for the comparison. The MPEG-DASH content and a MPD file for our experiments have been produced using Bitmovin Cloud Encoding Service by encoding the Big Buck Bunny animation movie which is also a part of the DASH dataset [7]. Note that our focus is primarily on the streaming performance, not a visual quality and, thus, we believe that one test sequence is sufficient. We have encoded and prepared two different profiles

as it is used in industry deployments. The first comprises a *FullHD* profile with five different representations: 426x238 pixels (400kbps), 640x360 (800), 854x480 (1200), 1280x720 (2400), and 1920x1080 (4800). For the second configuration we reverse-engineered the *Amazon* Prime video service which offers 15 different representations: 400x224 (100), 400x224 (150), 512x288 (200), 512x288 (300), 512x288 (500), 640x360 (800), 704x396 (1200), 704x396 (1800), 720x404 (2400), 720x404 (2500), 960x540 (2995), 1280x720 (3000), 1280x720 (4500), 1920x1080 (8000), and 1920x1080 (15000). In both cases we have adopted a segment length of four seconds as it provides a good trade-off regarding streaming performance and coding efficiency [7] which is also used in commercial deployments like Netflix. The bitrate of audio stream for both sets was defined as 128 Kbps.

The *network configuration* comprises a bandwidth trajectory adopted from [11] providing both step-wise and abrupt adjustments in the available bandwidth to properly test all adaptive HTML5 players and its adaptation behavior under different conditions. The predefined bandwidth trajectory scheme adjusts using the following sequence: 750 kbps (65 seconds), 350 kbps (90), 2500 kbps (120), 500 kbps (90), 700 kbps (30), 1500 kbps (30), 2500 kbps (30), 3500 kbps (30), 2000 kbps (30), 1000kbps (30) and 500 kbps (85). The network delay parameter was set to 70 milliseconds which corresponds to what can be observed within long-distance fixed line connections or reasonable mobile networks and, thus, is representative for a broad range of application scenarios.

IV. Evaluation results

In this section, we present some results of our evaluation and discuss the key aspects. Each experiment was conducted five times and the average is presented here. We noted that the variance is quite low and, thus, we believe that five runs per experiment is sufficient. In total we conducted 80 experiments, each with a duration of 630 sec resulting in a total duration of 14 hours. In the figures of this section we present the results for both content configurations. The red bars refer to the $Amazon$ content configuration and the blue ones refer to the $FullHD$.

Figure 3 shows that the Bitmovin player has the highest download video bitrate for both content configurations. The results of Flowplayer, Radiant MP, and VideoJS are very similar to dash.js as those players are based on dash.js and most likely adopt the same rate adaptation logic.

The video startup time is shown in Fig. 4. The results show that the HAS Player has the lowest video startup time and JW Player has the highest. Video startup time of Flowplayer, Radiant MP, and VideoJS is very similar to dash.js with minor differences. Interestingly, the number of stalls for the $Amazon$ profile is much lower than for the $FullHD$ profile as shown in Fig. 5 due to the fact that the former has a much higher number of content representations providing a better match to the bandwidth trajectory. The bandwidth trajectory has a reduction of the available bandwidth at the very beginning which does not match the lowest bitrate representation of the $FullHD$ profile and, thus, results in many stalls, at least for some players.

The total time of stalls exposes additional findings about the adaptation behaviour of the different players. As shown in Fig. 6 the Bitmovin player has the lowest total time of stalls for both profiles. Looking at the result for the $FullHD$ profile with five different content representations, the total time of stalls is $>$20 sec for all other players which may enormously affect the user perception. The results for the $Amazon$ profile are better due to the availability of more content representations. Nevertheless, the total time of stalls is still $>$20 sec for Radiant MP and VideoJS.

TABLE II. Instability and Inefficiency. Blue indicates best and red indicates worst.

Players	Instability		Inefficiency	
	FullHD	Amazon	FullHD	Amazon
Bitmovin	0.012	0.0145	0.388	0.3984
dash.js	0.0157	0.0185	0.3656	0.371
Flowplayer	0.0151	0.0188	**0.3502**	0.3726
HAS Player	**0.0106**	**0.0143**	**0.5564**	0.4836
JW Player	0.011	0.0144	0.4854	**0.7015**
Radiant	**0.0232**	0.0261	0.3729	0.3647
Shaka	0.016	**0.0282**	0.4342	0.6369
Video JS	0.0144	0.0198	0.3521	**0.358**

The number of quality switches is an important factor for the smoothness of the video streaming behaviour. As we can see in Fig. 7 it is twice as much for the $Amazon$ profile as more content representations are available. JW Player has a very low number of quality switches for the $Amazon$ profile but we also noticed that this player is inefficient with respect to bandwidth utilization (cf. download video bitrate).

The instability and inefficiency metrics for the $FullHD$ and the $Amazon$ profiles are shown in Table II. Lower values of the instability metric reflect smoother video quality adaptation to the changing network characteristics. Lower values of the inefficiency metric indicate that the player rate adaptation algorithm more efficiently utilize the available network throughput in order to deliver the media content to the application. All tested players have fairly low values of instability indicating a smooth adaptation behaviour with a low number of quality switches. Only Radiant player has a bit higher instability and the Shaka Player has a higher instability but only for the $Amazon$ profile. In general, the number of quality switches using the $Amazon$ profile is twice as much as with the $FullHD$ profile which can be explained by the fact that the $Amazon$ profile has much more representations than the $FullHD$ profile. Inefficiency is comparable for the $FullHD$ profile except for the HAS Player which has a slightly higher inefficiency than the rest. Interestingly, JW Player and Shaka Player has a much higher inefficiency for the $Amazon$ profile – both have the lowest download video bitrate – and we again notice that Flowplayer, Radiant MP, and VideoJS provides a similar result as dash.js.

In general, we observe quite a different behaviour of all adaptive HTML5 players leading to different performance results. A summary of the results is provided in Fig. 2. All players adopt – more or less – a conservative approach according to the achieved download video bitrate. It seems that the Bitmovin player shows superior performance but also has the highest video buffer level (up to 40 sec) compared to

Quality metrics	Bitmovin		dash.js		Flowplayer		HAS		JW Player		Radiant		Shaka		Video JS	
	Amazon	FullHD	Amazon	FullHD	Amazon	FullHD	Amazon	FullHD	Amazon	FullHD	Amazon	FullHD	Amazon	FullHD	Amazon	FullHD
Download video bitrate (kbps)	**936**	**845**	891	782	905	792	705	657	244	536	892	791	499	627	884	794
Video startup time (sec)	2	2.8	3.5	3.4	3.2	3.3	**1.4**	**2.2**	7.9	9.6	2.6	2.6	4.7	7.3	2.8	2.8
Number of stalls	**0**	**1**	4	12	7	13	2	14	0	2	10	7	0	4	8	12
Total time of stalls (sec)	**0**	**2.4**	5.4	20	14.2	28.4	4.2	49.8	0	21.8	32.3	25.8	2	35.1	23.1	30.1
Number of quality switches	18	8	29	15	23	11	20	8	**6**	**7**	28	14	32	10	24	10

Fig. 2. Average results for Amazon and FullHD. Blue color indicates the best results and yellow indicates they might have the same adaptation logic

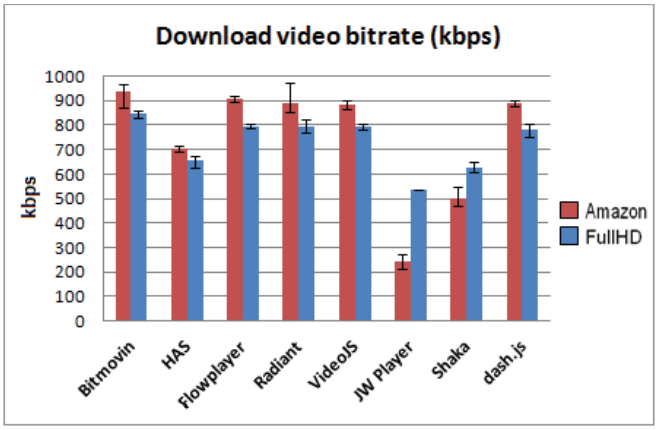

Fig. 3. Download video bitrate

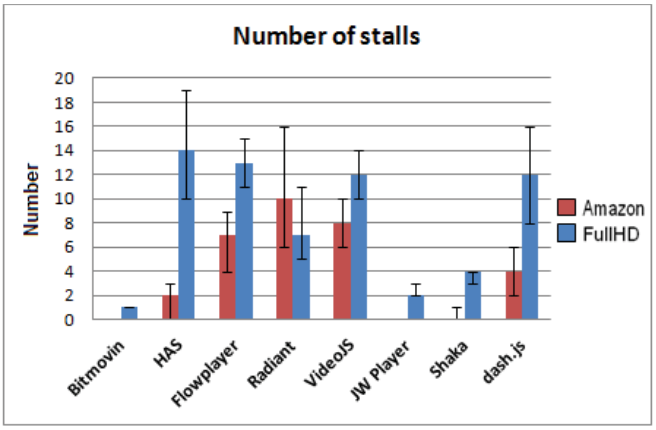

Fig. 5. Number of stalls

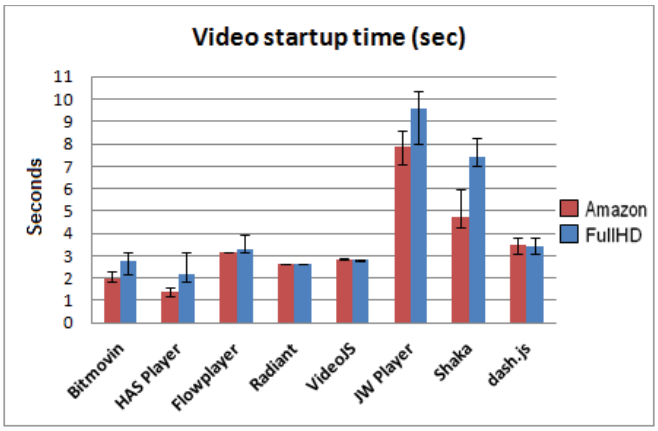

Fig. 4. Video startup time

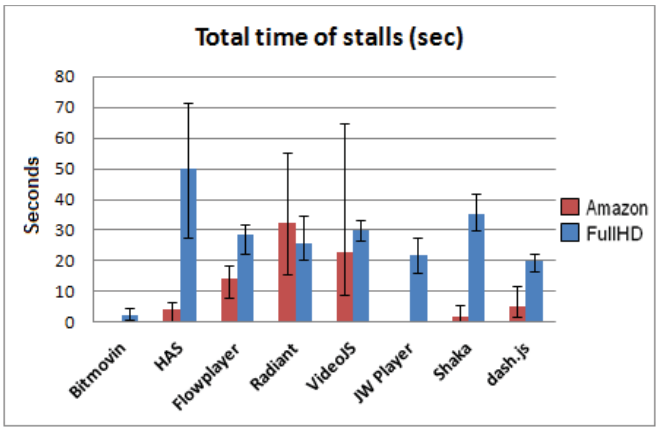

Fig. 6. Total time of stalls

all others (approx. 12-20 sec). Various user studies suggest that stalls should be avoided at all as they decrease the Quality of Experience (QoE) significantly. Looking at the results for the *Amazon* profile, only the Bitmovin player, the JW Player, and the Shaka Player (with some outlier) achieve this goal.

The evidence from these results suggests the following: *(i)* in networks with bandwidth fluctuations, the playback quality significantly depends on the selected adaptive HTML5 player; *(ii)*it is reasonable to use suitable rate adaptation algorithms for different groups of users depending on the state of their network connections (e.g., dynamic switching of rate adaptation algorithms can be applied); *(iii)* the number of stalls depends on how many representations are available. The more bitrates exist, the lower the number of stalls.

V. Conclusions and Future Work

In this paper we have evaluated eight adaptive HTML5 players which are actually and – some of them – massively deployed in the real-world services. In order to conduct the performance evaluations we developed an adaptive video streaming evaluation framework. With this framework it is possible to conduct a high number of experiments in a relatively short amount of time providing reliable results. The results of the experiments clearly show that the players have a different behavior depending on the status of the network characteristics and available content representations. Future work will include adding new players (as they emerge on the market), investigating how different adaptive HTML5 players compete for the available bandwidth in a shared network and

Fig. 7. Number of quality switches

optimizing different adaptation algorithms depending on the network characteristics/conditions and the client devices.

REFERENCES

[1] Mininet. http://mininet.org/, (Online: accessed May 24, 2017).

[2] Selenium server. http://www.seleniumhq.org/, (Online: accessed May 24, 2017).

[3] Tclink. http://mininet.org/api/classmininet_1_1link_1_1TCLink.html, (Online: accessed May 24, 2017).

[4] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 187–198, 2014.

[5] K. Hughes and D. Singer. ISO/IEC DIS 23000-19 Part 19: Common media application format (CMAF). Draft International Standard, ISO/IEC JTC 1/SC 29/WG 11, Oct. 2016. Work in Progress.

[6] J. Jiang, V. Sekar, and H. Zhang. Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming With Festive. *IEEE/ACM Trans. Netw.*, 22(1):326–340, Feb. 2014.

[7] S. Lederer, C. Müller, and C. Timmerer. Dynamic Adaptive Streaming over HTTP Dataset. In *Proceedings of the 3rd Multimedia Systems Conference*, MMSys '12, pages 89–94, 2012.

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

[9] T. Mäki, M. Varela, and D. Ammar. A Layered Model for Quality Estimation of HTTP Video from QoS Measurements. In *2015 11th International Conference on Signal-Image Technology Internet-Based Systems (SITIS)*, pages 591–598, Nov 2015.

[10] R. K. P. Mok, E. W. W. Chan, and R. K. C. Chang. Measuring the Quality of Experience of HTTP Video Streaming. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, pages 485–492, May 2011.

[11] C. Müller, S. Lederer, and C. Timmerer. An Evaluation of Dynamic Adaptive Streaming over HTTP in Vehicular Environments. In *Proceedings of the 4th Workshop on Mobile Video*, MoVid '12, pages 37–42, 2012.

[12] R. Roverso, S. El-Ansary, and M. Högqvist. On HTTP Live Streaming in Large Enterprises. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 489–490, New York, NY, USA, 2013. ACM.

[13] Sandvine. 2016 Global Internet Phenomena Report: Latin America & North America, 2016. Online: *http://sandvine.com/*.

[14] T. C. Thang, H. T. Le, A. T. Pham, and Y. M. Ro. An Evaluation of Bitrate Adaptation Methods for HTTP Live Streaming. *IEEE Journal on Selected Areas in Comm.*, 32(4):693–705, April 2014.

[15] C. Timmerer, M. Maiero, and B. Rainer. Which Adaptation Logic? An Objective and Subjective Performance Evaluation of HTTP-based Adaptive Media Streaming Systems. *arXiv.org [cs.MM]*, abs/1606.00341:11, Jun 2016.

[16] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 325–338, 2015.

[17] A. Zabrovskiy, E. Kuzmin, E. Petrov, and M. Fomichev. Emulation of dynamic adaptive streaming over http with mininet. In *2016 18th Conference of Open Innovations Association and Seminar on Information Security and Protection of Information Technology (FRUCT-ISPIT)*, pages 391–396, April 2016.

[18] A. Zabrovskiy, E. Petrov, E. Kuzmin, and C. Timmerer. Evaluation of the Performance of Adaptive HTTP Streaming Systems. *ArXiv e-prints*, Oct. 2017.

[19] S. Zhao, Z. Li, D. Medhi, P. Lai, and S. Liu. Study of user QoE improvement for dynamic adaptive streaming over HTTP (MPEG-DASH). In *2017 International Conference on Computing, Networking and Communications (ICNC)*, pages 566–570, Jan 2017.