Performance of Low-Latency DASH/CMAF and Low-Latency HLS Streaming Systems

Yuriy Reznik, Thiago Teixeira, Bo Zhang
Brightcove, Inc.
Boston, MA, USA

Abstract – Reducing end-to-end streaming latency is critical to HTTP-based live video streaming. There are currently two technologies in this domain: Low-Latency HLS (LL-HLS) and Low-Latency DASH (LL-DASH). The latter is sometimes also referred to as Low-Latency CMAF (LL-CM.AF), but effectively it is the same architecture. Several existing implementations of streaming players, as well as encoding and packaging tools, support both technologies. Well-known examples include Apple's AVplayer, Shaka player, HLS.js, DASH.js, FFmpeg, etc.
In this paper, we conduct a performance analysis of such streaming systems. We perform a series of live streaming experiments, repeated using identical video content, encoders, encoding profiles, and network conditions, emulated by using traces of real-world networks. We capture several performance metrics, such as average stream bitrate, the amounts of downloaded media data, streaming latency, buffering, frequency of stream switching, etc. Subsequently, we analyze the captured data and describe the observed differences in the performance of LL-HLS and LL-DASH-based systems.

Introduction

In the past few years, the video streaming industry has seen immense interest in Low-Latency streaming protocols, targeting about 5 seconds end-to-end delay, comparable with the delay in live broadcast TV systems. Attaining such low delay is considered critical for streaming live sports, gaming, online learning, interactive video applications, etc.

As well known, the delay in the conventional live streaming technologies such as HLS [1] and DASH [2] is much longer. It is caused by relatively long (4-10 seconds) segments and a segment-based delivery model, requiring complete delivery of each media segment before playback. Combined with buffering strategies used by the HLS or DASH streaming clients, this typically produces delays of 10 to 30sec, or even longer.

Low-Latency HLS (LL-HLS) [3,4] and Low-Latency DASH (LL-DASH) [2,5,6] are the recent evolutions of the HLS and DASH standards, designed to reduce the latency. They employ a new encoding and transmission process, effectively splitting each segment into several (typically 4-10) chunks and then using such "chunks" for transmission. Since each "chunk" is significantly shorter than a segment, this reduces the delay in the streaming system.

Several existing implementations of streaming players, encoding and packaging tools support LL-DASH and LL-HLS technologies. The available player implementations include Apple's AVPlayer [7], HLS.js [8], Shaka player [9], DASH.js [10], as well modifications of DASH.js, including machine learning-based adaptation methods such as LoL [11] and L2All [12]. The available encoding and packaging tools include Apple's HLS reference tools [13], FFmpeg [14], node-gpac-dash [15], and others. Many of these technologies have demonstrated lower streaming delay and promising performance when operated over high-speed network connections or tested using simple in-browser bandwidth throttling tools [11,12]. However, the actual performance of such systems under more challenging and more realistic deployment environments has not (to the best of authors' knowledge) been well-studied yet.

This paper aims to perform a practical evaluation and comparison of such available implementations of LL-HLS and LL-DASH players and systems in more realistic and challenging environments, such as delivery over mobile networks.

Related Work and Adopted Evaluation Methodology

The operation under unknown or changing network conditions has been one of the most fundamental challenges that adaptive bitrate streaming systems have been trying to solve since their birth in the 1990s [16-18]. This challenge still exists today, although in a somewhat simplified setting, allowed by using HTTP-based Adaptive Streaming (HAS) architectures [1-3,19]. In such architectures, the network adaptation logic resides in streaming clients, effectively driving the selection and loading of segments of media streams.

In the past decade, many methods have been proposed for the design of stream selection algorithms. These include throughput-based methods [20-21], buffer-level-based heuristics [22-25], control-theoretic approaches [26-27], as well as machine-learning algorithms [11-12]. However, the methodologies used by different researchers for comparison of such bandwidth adaptation algorithms have varied, and in some cases, employed very basic bandwidth throttling tools in web browsers. Such tools can only control video players' download bandwidth at the application layer and have no means for accurately simulating highly fluctuating network bandwidth changes or packet loss statistics present, for example, in mobile networks.

References [28-33] proposed testbeds/frameworks for evaluating video streaming QoE using real networks or fine-controlled network links to evaluate HAS systems. For instance, Talon et al. [28] have implemented several HAS players and assessed them in a campus network from different performance perspectives. Ayad et al. [31] took a similar approach and conducted a practical and in-depth evaluation of HAS players. Notably, the authors of [31] have built an experimental framework emulating wired network links using Netem and Linux Traffic Control (TC). Their experiments and code-level analysis revealed how different HAS players operate in detail. This study was limited to the use of wired networks, however. References [32-34] have proposed a framework for automating video streaming testing and QoE evaluation. The framework integrates with the Mobile Broadband Networks in Europe (MONROE) project. The players run in docker containers with managed network connections and the environment metadata collection functionalities built into MONROE nodes. The framework enables running experiments on a cloud infrastructure. These proposed frameworks, however, focus more on automation and simplification of player evaluation, but they do not ensure a fair comparison of different players because there is no guarantee that different players experience the same network conditions. Raca et al. [30] have proposed DASHbed, a framework for simulating large-scale empirical evaluation of DASH players. However, the mobile network traces it relies upon [35] have limited sampling granularity and thus don't capture the essential fine-grain dynamics of such networks.

To ensure a more accurate and fair evaluation of different players, in this paper, we introduce a custom-built evaluation framework incorporating the Mahimahi network emulator [43-46]. Our framework guarantees a fair comparison of different players by replaying the same network traces across playback sessions. Such an approach allows us to compare multiple players side by side under the same network condition. The Mahimahi network simulator can accurately emulate mobile network links using the physical network traces recorded from different mobile operators. Specifically, we will use network traces from T-Mobile and Verizon 4G LTE networks [43].

Experiment Setup

In this section, we describe the overall setup of our experiments, including encoding and packaging tool-chains, the

The overall diagrams of our systems built for LL-HLS and LL-DASH streaming appear in the left and right sub-figures of Figure 1. To generate LL-HLS streams, we used Apple's HLS reference tools [13] and FFmpeg [14]. To generate LL-DASH streams we used OBS studio [47], FFmpeg [14], and node-gpac-dash [15]. Additional details about our setups can be found in [48-49]. The LL-HLS stream was served dynamically by the NGINX web server [50]. The LL-DASH stream was served dynamically by node-gpac-dash [15].

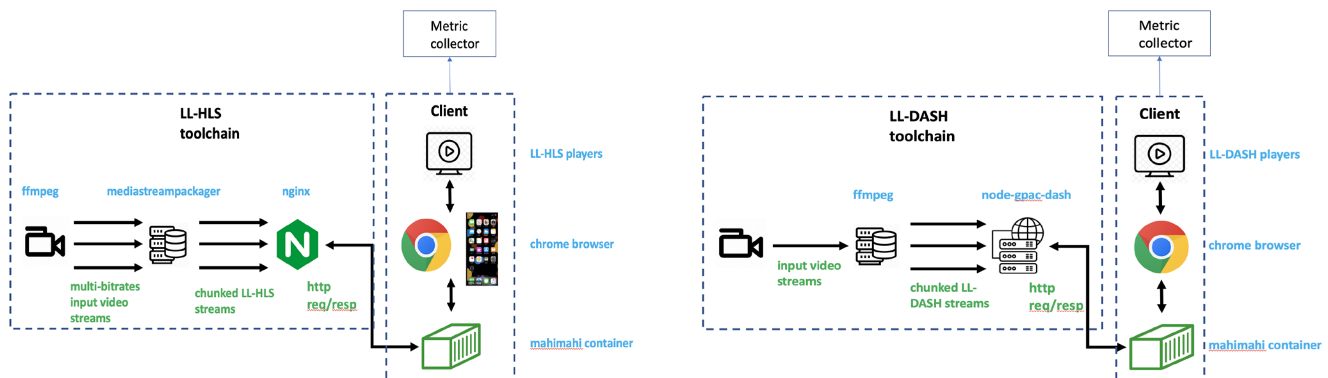

FIGURE 1: ARCHITECTURES LL-HLS (LEFT) AND LL-DASH (RIGHT) STREAMING SYSTEMS USED FOR TESTING.

As shown in Figure 1, the encoded input video streams are subsequently processed by the low-latency packagers (mediastreamsegmenter [13] for LL-HLS, and FFmpeg [14] for LL-DASH). The outputs of low-latency packagers are the chunked video segments and manifest files informing the players on how to consume the streams

in low-latency mode. Next, the output stream files are served by the low-latency media servers (lowLatencyHLS.php [13] for LL-HLS, node-gpac-dash [15] for LL-DASH) to players in a chunked manner. On the player side, the web-based players run on the Chrome web browser, and the iOS native player (HLS) runs on the AVPlayer framework on iOS. The Chrome browser and the AVPlayer run inside the Mahimahi container [43] and connect to the media server via an emulated virtual network interface.

As a test video sequence, we used a 1080p version of the Big Buck Bunny video [51]. This sequence was looped to enable continuous testing. For streaming, 3 live transcoded variant streams have been subsequently generated, with parameters listed in Table 1.

Parameter	Rendition 1	Rendition 2	Rendition 3
Bitrate (kbps)	279	925	1253
Frame rate (fps)	30	30	30
Video resolution (pixels)	320x180	640x360	768x432
Seg. duration (sec)	4	4	4
Chunk duration (sec)	1	1	1
Video codec	H.264	H.264	H.264
Video codec profile	Baseline	Baseline	Baseline
Media format	ISOBMFF	ISOBMFF	ISOBMFF

TABLE 1: ENCODING PROFILE PARAMETERS USED FOR BOTH LL-HLS AND LL-DASH SYSTEMS.

To minimize fluctuations of encoding bitrates from their declared targets, constant bitrate (CBR) encoding mode has been utilized. H.264 encoder operating in Baseline profile has been used. Lookahead processing disabled. The segment lengths and fragment durations were set to 4 sec and 1 sec, respectively, matching the default values used in Apple's streaming tools for LL-HLS [13]. The same encoding parameters have been used for the generation of both LL-DASH and LL-HLS streams.

The overall session duration that we used to test each player's performance under each network was 10 minutes. Given selected chunk and fragment durations, this has allowed about 600 chunks or equivalently 150 segments to be downloaded per session.

We have evaluated 6 implementations of low-latency streaming players. For LL-HLS, we used Apple's AVPlayer [7], HLS.js [8], and Shaka player [9]. For LL-DASH, we used Dash.js with three different low-latency ABR algorithms: Dash.js original [10], Dash.js with LoL algorithm [11], and Dash.js with L2All algorithm [12]. We have implemented simple test applications for all the players. The applications were built using the latest player SDK releases as available in December 2020.

The reporting of metrics indicative of live streaming latency, playback speed, and re-buffering events has been instrumented in the video player applications. Other metrics such as stream bitrate, video resolution, and media data downloaded have been derived from the streaming servers' access logs. The processing of all collected metrics was done offline.

The player's streaming latency was calculated by following the method described in [6], which is common for both LL-DASH and LL-HLS. Essentially, at any time point, we take the difference between the elapsed presentation time and the elapsed wall clock time, from the beginning of a streaming session:

$$PL = (WC - WCA) - (PT - PTA)/TS \quad (1)$$

where PL represents the live Presentation Latency, WC and PT represent the current Wall Clock time and the current Presentation Time, respectively. WCA and PTA represent the beginning wall clock time and the beginning presentation time, respectively. For LL-DASH, the above values have been obtained from the ProducerReferenceTime [6] element embedded in an MPD file, and W3C HTML5 video currentTime API [52], and/or a DASH MPD file. For LL-HLS, these values have been derived from the HLS m3u8 file and currentTime API.

The number of re-buffering events and the players' playback speed has been obtained by using the waiting event API [52] and the playbackRate API [52] respectively.

The playback speed variation was calculated as the Euclidean distance of all the measured playback speeds relative to the native speed (which equals 1):

$$playbackSpeedVariation = \frac{1}{n}\sqrt{\sum_{i=1}^{N}(s_i - 1)^2} \quad (2)$$

The parameter N used in this formula denotes the number of playback speed measurements conducted during the session. All other metrics including stream bitrate, video resolution, media data downloaded, number of bitrate switches have been derived from the server logs. The full list of metrics collected in our test system is summarized in Table 2.

Metrics	Impact domain(s)
Streaming bitrate (kbps)	Efficiency, QoE
Video resolution (height)	QoE
Streaming latency (sec)	Latency, QoE
Variation of playback speed	QoE
Frequency of stream switches	QoE
Frequency of rebuffering events	QoE
Downloaded media data (Mbytes)	Efficiency
Media objects (chunks or segments) downloaded	Efficiency

TABLE 2: PERFORMANCE METRICS COLLECTED IN OUR EXPERIMENTS.

We used the Mahimahi network emulator [43] to emulate network conditions at the network interface level. Mahimahi is essentially a Linux container that can run an application inside of it. An application inside Mahimahi connects to the outside world through a virtual network interface that sends and receives bytes according to the running downlink and uplink traces. This way, the capacity of the network interface is limited by the running trace. We used traces that have been recorded from real-world mobile networks. When we run the test players inside Mahimahi, the player download speed is limited by the capacity of the virtual interface. Unlike using bandwidth throttling features in web browsers, Mahimahi provides more faithful network emulation by using real-world traces and throttling bandwidth at the network interface level. Additionally, the same network traces are replayed for all the test sessions. This allows a fair and realistic comparison of different players.

We have evaluated LL-HLS and LL-DASH players using two 4G-LTE network traces from T-Mobile and Verizon respectively [43]. We provide visualizations of these traces in Figure 3. In Table 3 we list several basic statistics associated with them.

Bandwidth statistics	T-Mobile	Verizon
Average bitrate (kbps)	1607.43	1323.97
St. deviation of bitrate (kbps)	1147.60	1075.80
Minimum bitrate (kbps)	148.5	1.178
Maximum bitrate (kbps)	7545	5433

TABLE 3: BANDWIDTH STATISTICS OF NETWORK TRACES USED FOR TESTS.

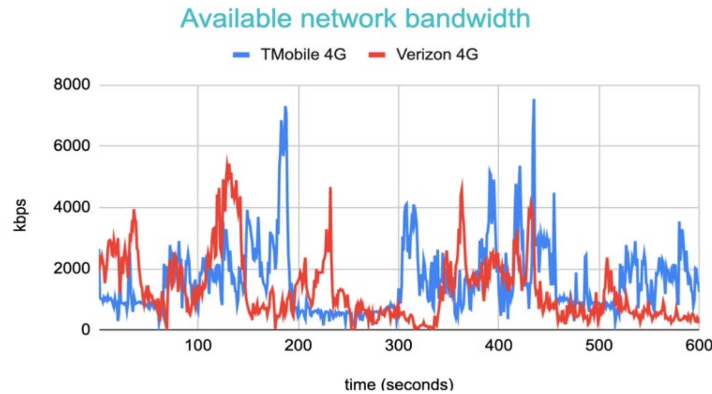

FIGURE 3: VISUALIZATIONS OF NETWORK BANDWIDTH TRACES USED FOR TESTS.

We note that the traces that we have selected for testing are pretty challenging, capturing situations with mobile handoffs and other forms of impairments that may happen in practice. In fact, with the selected traces, we should expect streaming players to enter in a buffering state at least once or twice throughout the session. On the other hand, we also note that the effective average bitrate supported by both networks is higher than the bitrate used by the top rendition in our encoding profiles. This should enable players to use all renditions for network adaptation.

The Results

In this section, we present the results of our tests of LL-DASH and LL-HLS systems using different networks.

Results for Tests using Verizon 4G LTE Network

First, we review the results obtained by using traces of the Verizon 4G LTE network. Table 4 offers summary metrics. Figure 4 shows the dynamics of bitrate changes in LL-HLS and LL-DASH systems. Figure 5 shows dynamics of playback latencies achieved by both systems.

Metrics	LL-HLS players			LL-DASH players		
	HLS.js	Shaka	AVplayer	DASH.js	LoL	L2All
Avg. bitrate (kbps)	849	1228	1136	1165	595	1073
Avg. height (pixels)	328	426	404	410	262	387
Avg. latency (secs)	4.32	7.28	15.96	3.71	3.2	3.9
Var. playback speed	3.97	0	0	0.19	0.39	0.44
# of switches	48	2	130	6	29	3
# of rebufferings	36	12	2	5	79	56
Downloaded MBs	85	90	99	88	45	81
Downloaded objects (chunks + segments)	673 (662+11)	587 (587+0)	669 (611+58)	152	151	152

TABLE 4: SUMMARY OF PERFORMANCE METRICS OBTAINED FOR VERIZON 4G LTE NETWORK.

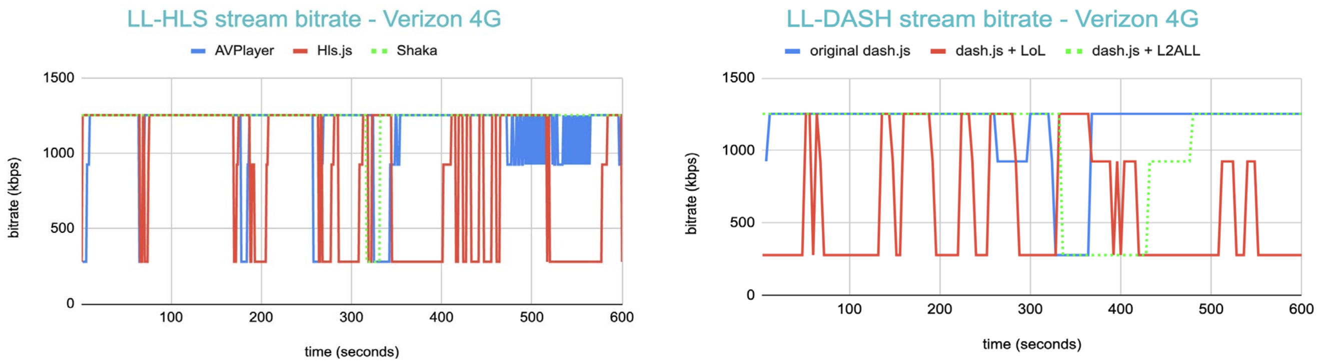

FIGURE 4: BITRATE VARIATION OVER TIME – VERIZON 4G LTE.

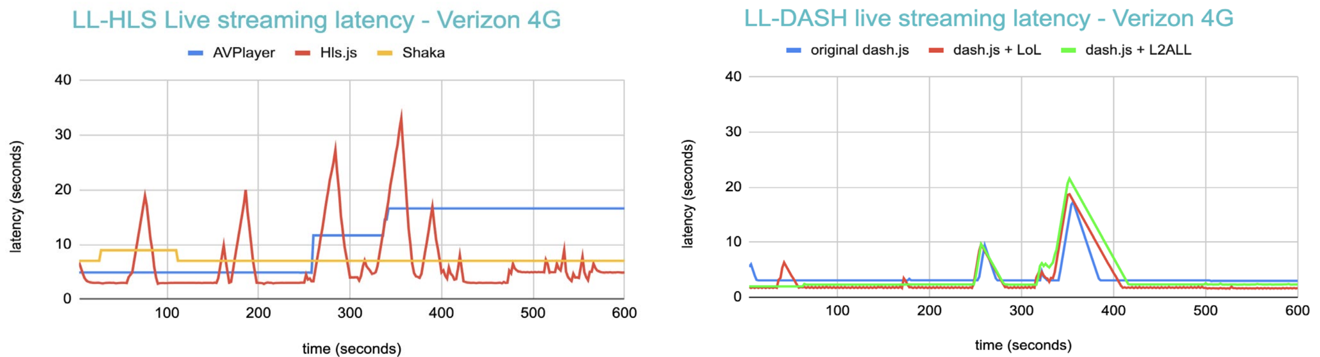

FIGURE 5: LATENCY VARIATION OVER TIME – VERIZON 4G LTE.

Based on Table 4 and Figure 5, we first note that the latencies achieved by LL-DASH players and their variations were considerably lower than ones achieved by LL-HLS. Except for a couple of segments where bandwidth drops significantly, the latencies of LL-DASH players have been in the range of 3-4 sec. Among LL-HLS players, only Shaka player was able to stay at latency in the range of 7-9 sec. The HLS.js has also tried to keep latency low, but run in a large number of buffering events as a result. The AVplayer's behavior was interesting: it starter to operate in about 4-seconds latency mode, but then, by the middle of the session, it increased latency to 12 sec, and the increased it again to 16 sec, and never recovered to low-latency mode.

In terms of playback stability / prevention of rebuffering events, we noted that AVplayer was most robust among LL-HLS players, and DASH.js among LL-DASH players. AVplayer buffered only 2 times, while DASH.js buffered 5

times. But we also noticed that many players have been switching across streams very often. E.g., AVplayer has made 130 switches in 600sec – long session. A switch at almost every segment boundary.

In terms of data usage and the ability to deliver high-resolution videos, we noted that DASH.js was the best among LL-DASH systems, and Shaka player was best among LL-HLS. AVplayer was a close next. The average consumed bitrate and resolutions delivered by best players for LL-HLS and LL-DASH systems was comparable. But we also noted the number of objects (chunks or whole segments) downloaded by LL-HLS systems was significantly higher than in LL-DASH. This relates to the differences in implementation of transfer protocols employed by both systems.

Results for Tests using T-Mobile 4G LTE Network

First, we review the results obtained by using traces of the T-Mobile 4G LTE network. Table 5 offers summary metrics. Figure 6 shows the dynamics of bitrate changes in LL-HLS and LL-DASH systems. Figure 7 shows dynamics of playback latencies achieved by both systems.

Metrics	LL-HLS players			LL-DASH players		
	HLS.js	**Shaka**	**AVplayer**	**DASH.js**	**LoL**	**L2All**
Avg. bitrate (kbps)	783	1043	1037	1225	537	1251
Avg. height (pixels)	311	378	378	426	248	432
Avg. latency (secs)	5.82	4.48	7.78	3.06	1.78	2.28
Var. playback speed	3.62	0	0	0.23	1.62	0.42
# of switches	50	8	72	4	28	0
# of rebufferings	43	18	1	1	69	13
Downloaded MBs	156	81	92	93	42	94
Downloaded objects (chunks + segments)	965 (743+222)	621 (621+0)	703 (698 +5)	151	152	151

TABLE 5: SUMMARY OF PERFORMANCE METRICS OBTAINED FOR T-MOBILE 4G LTE NETWORK.

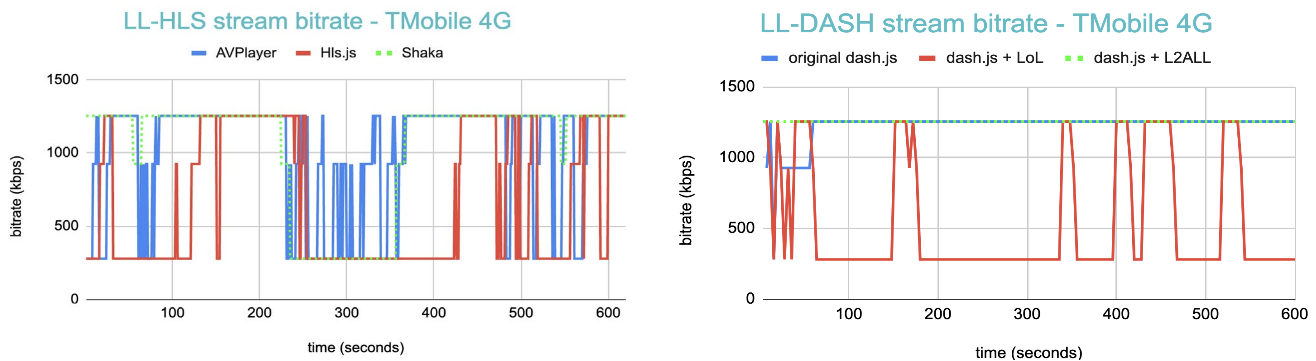

FIGURE 6: BITRATE VARIATION OVER TIME – T-MOBILE 4G LTE.

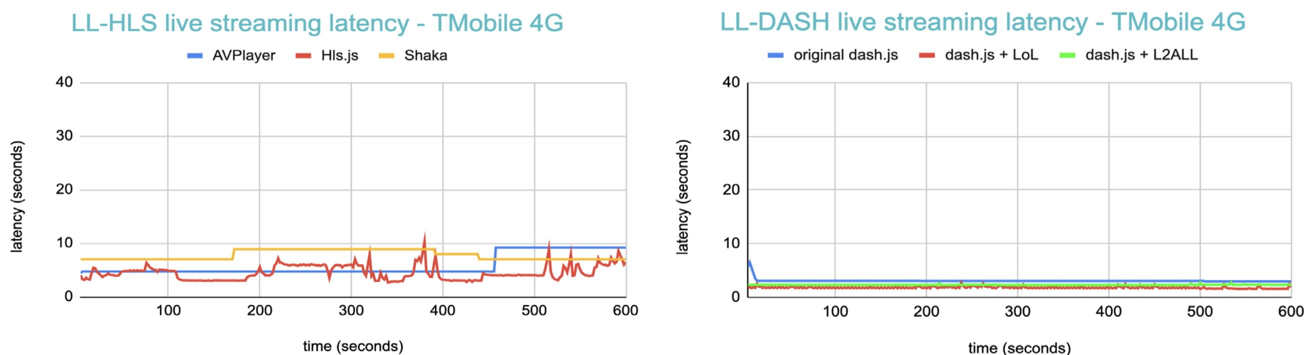

FIGURE 7: LATENCY VARIATION OVER TIME – T-MOBILE 4G LTE.

In the above table and plots, we notice many of the same effects as we reported earlier. LL-DASH players deliver lower latency, with much lover variation among player implementations. The AVplayer starts in low-latency mode but then increases latency by the end of the session. The AVplayer and DASH.js are best in terms of buffering. While overall network conditions, in this case, appear to be better, most players still perform a high number of switches and run into at least one buffering situation. The observations regarding data loads are the same as reported earlier.

Conclusions

In this study, we have evaluated the LL-HLS and LL-DASH streaming systems under identical network conditions and by using several available implementations of streaming players for both systems.

Based on our experiments, we've confirmed that both LL-HLS and LL-DASH can deliver significantly lower latencies compared to the traditional HLS and DASH streaming systems. Specifically, for LL-DASH players, we've observed latencies in the range of 3-4 seconds, except for a few segments when bandwidth was insufficient to maintain live playback. For LL-HLS players, we have observed a broader variation in streaming latencies across different player implementations, but with most data points fitting in the 4-10 second range.

However, we have also noticed that in trying to maintain such a low delay, both LL-DASH and LL-HLS players frequently make decisions impacting the QoE in many other dimensions. Such observed effects include:

- high stream switching and buffering rates,
- the inability of some players to select high renditions,
- the inability of some players to maintain playback speed,
- more requests sent to the CDNs (particularly for LL-HLS),
- the inability of some players to maintain low delay, etc.

Based on these observations, we believe that while promising, both LL-HLS and LL-DASH systems still have some room for improvements. Especially when operating under challenging network conditions, such as mobile networks with significant load, handoffs, poor connectivity, and other effects occurring in practice. What is needed the most is additional tuning of the player's ABR rate selection algorithms. They need to made more robust. However, with much work in this direction already ongoing, including trying advanced machine-learning-based rate selection techniques (see e.g. [10-12]), we hope that these technologies will soon mature and will be ready for deployment at scale.

References

[1] IETF RFC 8216, "HTTP Live Streaming", https://tools.ietf.org/html/rfc8216, 2017.
[2] ISO/IEC 23009-1:2012, "Information technology - Dynamic adaptive streaming over HTTP (DASH) - Part 1: Media presentation description and segment formats," February 2012.
[3] IETF RFC 8216, HTTP Live Streaming, 2nd Edition, https://tools.ietf.org/html/draft-pantos-hls-rfc8216bis-08, 2019.
[4] Apple, Enabling Low-Latency HLS, https://developer.apple.com/documentation/http_live_streaming/enabling_low-latency_hls
[5] ETSI technical specification, "MPEG-DASH Profile for Transport of ISO-BMFF Based DVB Services over IP Based Networks", https://www.etsi.org/deliver/etsi_ts/103200_103299/103285/01.03.01_60/ts_10328v010301p.pdf
[6] DASH Industry Forum, "Low-Latency Modes for DASH", https://dashif.org/docs/CR-Low-Latency-Live-r8.pdf
[7] AVFoundation, https://developer.apple.com/av-foundation/
[8] Hls.js player, https://github.com/video-dev/hls.js/
[9] Shaka player, https://github.com/google/shaka-player
[10] Dash.js player, https://github.com/Dash-Industry-Forum/dash.js
[11] M. Lim, M. N. Akcay, A. Bentalab, A. C. Begen, R. Zimmermann, "When they go high, we go low: low-latency live streaming in dash.js with LoL," ACM Multimedia Systems Conference, Online, June 8-11, 2020.
[12] T. Karagkioules, R. Mekuria, D. Griffioen, A. Wagenaar, "Online Learning for Low-Latency Adaptive Streaming," ACM Multimedia Systems Conference, Online, June 8-11, 2020.
[13] HLS tools, https://developer.apple.com/ documentation/http_live_streaming/about_apple_s_http_live_streaming_tools
[14] FFmpeg, https://www.ffmpeg.org/
[15] DASH Low Latency Server, https://github.com/maxutility2011/node-gpac-dash
[16] D. Wu, Y. Hou, W. Zhu, Y-Q. Zhang, and J. Peha, "Streaming video over the internet: approaches and directions," IEEE Trans. CSVT, vol. 11, no. 3, 2001, pp. 282-300.
[17] G. Conklin, G. Greenbaum, K. Lillevold, A. Lippman, and Y. Reznik, "Video coding for streaming media delivery on the internet," IEEE Trans. CSVT, vol. 11, no. 3, 2001, pp. 269-281.
[18] B. Girod, M. Kalman, Y.J. Liang, and R. Zhang, "Advances in channel-adaptive video streaming," Wireless Comm. and Mobile Comp., vol. 2, no. 6, 2002, pp. 573-584.

[19] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, R. Zimmermann, "A Survey on Bitrate Adaptation Schemes for Streaming Media Over HTTP," in IEEE Communications Surveys & Tutorials, vol. 21, no. 1, 2019, pp. 562-585.
[20] J. Jiang, V. Sekar, H. Zhang, "Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE," in IEEE/ACM Transactions on Networking, vol. 22, no. 1, Feb. 2014, pp. 326-340.
[21] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, D. Oran, "Probe and Adapt: Rate-Adaptation for HTTP Video Streaming at Scale," IEEE Journal on Selected Areas in Communications, Vol. 32, No. 4, April 2014, pp. 719-733.
[22] K. Spiteri, R. Urgaonkar, R. K. Sitaraman, "BOLA: Near-Optimal Bitrate Adaptation for Online Videos," Annual IEEE International Conference on Computer Communications, 2016, pp. 1-9.
[23] T. Huang, R. Johari, N. Mckeown, M. Trunnell, M. Watson, "A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service," ACM SIGCOMM, 2014, pp. 187–198.
[24] K. Spiteri, R. Sitaraman, D. Sparacio, "From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player," ACMTrans. Multimedia Comput. Commun. Appl., vol. 15, no. 2s, Article 67, 2019.
[25] S. Hesse, "Design of scheduling and rate-adaptation algorithms for adaptive HTTP streaming," SPIE 8856, Applications of Digital Image Processing XXXVI, 88560M, 2013.
[26] X. Yin, A. Jindal, V. Sekar, B. Sinopoli, "A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP," SIGCOMM Comput. Commun. Rev. no. 45, vol. 4, 2015, pp. 325–338.
[27] C. Zhou, X. Zhang, L. Huo, and Z. Guo, "A control-theoretic approach to rate adaptation for dynamic HTTP streaming," Visual Comm. Image Processing, San Diego, CA, 2012, pp. 1-6.
[28] D. Talon, L. Attanasio, F. Chiariotti, M. Gadaleta, A. Zanella, M. Rossi, "Comparing DASH Adaptation Algorithms in a Real Network Environment," in European Wireless 2019; 25th European Wireless Conference, 2019.
[29] C. Storck, F. Figueiredo, "A Performance Analysis of Adaptive Streaming Algorithms in 5G Vehicular Communications in Urban Scenarios," of IEEE Symposium on Computers and Communications, 2020, pp.1-7.
[30] D. Raca, Y. Sani, C. J. Sreenan, J. J. Quinlan, "DASHbed: a testbed Framework for Large Scale Empirical Evaluation of Real-Time DASH in Wireless Scenarios," ACM Multimedia Syst. Conference, Amherst, MA, June 18-21, 2019, pp. 285-290.
[31] I. Ayad, Y. Im, E. Keller, S. Ha, "A Practical Evaluation of Rate Adaptation Algorithms in HTTP-based Adaptive Streaming", Elsevier Computer Networks vol. 133, 2018, pp. 90-103.
[32] C. Midoglu, A. Zabrovskiy, O. Alay, D. Holbling-Inzko, C. Griwodz, C. Timmerer, "Docker-Based Evaluation Framework for Video Streaming QoE in Broadband Networks," ACM International Conference on Multimedia, Nice, France, 21 - 25 October 2019, pp. 2288–2291.
[33] B. Taraghi, A. Zabrovskiy, C. Timmerer, H. Hellwagner, "CAdViSE: Cloud-based Adaptive Video Streaming Evaluation Framework for the Automated Testing of Media Players," ACM Multimedia Systems Conference, Online, June 8-11, 2020.
[34] A. Zabrovskiy, E. Kuzmin, E. Petrov, C. Timmerer, C. Mueller, "AdViSE: Adaptive Video Streaming Evaluation Framework for the Automated Testing of Media Players," ACM Multimedia Systems Conference, Taipei, Taiwan, June 20-23, 2017.
[35] D. Raca, J. Quinlan, A. Zahran, and C. Sreenan. "Beyond throughput: A 4G LTE dataset with channel and context metrics," ACM Multimedia Systems Conference, New York, NY, USA, 2018, pp 460–465.
[36] A. Bentaleb, A. C. Begen, S. Harous, R. Zimmermann, "Data-Driven Bandwidth Prediction Models and Automated Model Selection for Low Latency," IEEE Transactions on Multimedia, August 2020.
[37] A. Bentaleb, C. Timmerer, A. C. Begen, R. Zimmermann, "Bandwidth prediction in low-latency chunked streaming," ACM Workshop on Network and Operating Systems Support for Digital Audio and Video, Amherst, MA, June 21, 2019, pp. 7–13.
[38] A. Bentaleb, C. Timmerer, A. C. Begen, R. Zimmermann, "Performance Analysis of ACTE: A Bandwidth Prediction Method for Low-latency Chunked Streaming," ACM Trans. Multimedia Comp., Comm., and Applications, vol 16, no. 2s, 2020.
[39] I. Ozcelik, C. Ersoy, "Low-Latency Live Streaming Over HTTP in Bandwidth-Limited Networks," IEEE Communications Letters, vol. 25, no. 2, 2021, pp. 450-454.
[40] K. Durak, M. N. Akcay, Y. K. Erinc, B. Pekel, A. C. Begen, "Evaluating the Performance of Apple's Low-Latency HLS," IEEE Int. Workshop on Multimedia Signal Processing, September 21-24, 2020, pp.1-6.
[41] T. Huang, C Ekanadham, A. Berglund, Z. Li, "Hindsight: evaluate video bitrate adaptation at scale," ACM Multimedia Systems Conference, Amherst, MA, June 18 - 21, 2019, pp 86–97.
[42] B. Zhang, T. Teixeira, Y. Reznik, "Performance of Low-Latency HTTP-based Streaming Players", ACM Multimedia Systems Conference, Istanbul, Turkey, Sept. 28 - Oct 1, 2021.
[43] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, H. Balakrishnan, "Mahimahi: accurate record-and-replay for HTTP," USENIX Annual Technical Conference, Santa Clara, CA, July 8-10, 2015.
[44] A. Mondal, B. Palit, S. Khandelia, N. Pal, J. Jayatheerthan, K. Paul, N. Ganguly, Sandip Chakraborty, "EnDASH - A Mobility Adapted Energy Efficient ABR Video Streaming for Cellular Networks," IFIP Networking Conference, 2020, pp. 127-135.
[45] G. Ribezzo, L. D. Cicco, V. Palmisano, S. Mascolo, “A DASH 360° immersive video streaming control system,” in Internet Tech. Letters, vol. 3, no. 5, 2020.
[46] S. Sengupta, N. Ganguly, S. Chakraborty, P. De, "HotDASH: Hotspot Aware Adaptive Video Streaming using Deep Reinforcement Learning," IEEE International Conference on Network Protocols, 2018, pp.165-175.
[47] Open Broadcast Software, https://obsproject.com/
[48] B. Zhang, "Setting up your Own Low-Latency HLS Server to Stream from any Source Inputs", https://bozhang-26963.medium.com/setting-up-your-low-latency-hls-server-to-stream-from-any-source-inputs-de1e757a6688
[49] B. Zhang, "Low-Latency DASH Streaming Using Open-Source Tools", https://bozhang-26963.medium.com/low-latency-dash-streaming-using-open-source-tools-f93142ece69d
[50] NGINX web server, https://www.nginx.com/
[51] Blender Foundation, Big Buck Bunny video, https://download.blender.org/

[52] HTML5 Video, https://www.w3.org/TR/2011/WD-html5-20110113/ video.html