Performance of Low-Latency DASH and HLS Streaming in Mobile Networks

By Bo Zhang, Thiago Teixeira, and Yuriy Reznik

Abstract

Reducing end-to-end streaming latency is critical to Hypertext Transfer Protocol (HTTP)-based live video streaming. There are currently two technologies in this domain: 1) Low-Latency HTTP Live Streaming (LL-HLS) and 2) Low-Latency Dynamic Adaptive Streaming over HTTP (LL-DASH). The latter is sometimes also referred to as Low-Latency Common Media Application Format (LL-CMAF), 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, and so on. In this article, 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 4G LTE mobile networks from two major operators. We capture several performance metrics, such as average stream bitrate, the amounts of downloaded media data, streaming latency, buffering, frequency of stream switching, and so on. Subsequently, we analyze the captured data and describe the observed differences in the performance of LL-HLS and LL-DASH-based systems.

Keywords

Dynamic Adaptive Streaming over HTTP (DASH), Hypertext Transfer Protocol (HTTP) Adaptive Streaming, HTTP Live Streaming (HLS), low-latency live streaming, performance evaluation, video players

Digital Object Identifier 10.5594/JMI.2022.3180777
Date of publication: 5 August 2022

Introduction

In the past few years, the video streaming industry has seen immense interest in low-latency streaming protocols, targeting about 5-sec 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, and so on.

As is well known, the delay in the conventional live streaming technologies such as Hypertext Transfer Protocol (HTTP) Live Streaming (HLS)[1] and Dynamic Adaptive Streaming over HTTP (DASH)[2] is much longer. It is caused by relatively long (4–10 sec) 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 10–30 sec, or even longer.

Low-Latency HLS (LL-HLS) and Low-Latency DASH (LL-DASH) are the recent evolutions of the HLS and DASH standards, designed to reduce 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.

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 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] and DASH.js,[10] as well as modifications of DASH.js, including machine learning-based adaptation methods such as Low on Latency (LoL)[11] and

Learn2Adapt-LowLatency (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 article 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 (ABR) 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 quality of experience (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, Ayad et al.[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. Midoglu et al.,[32] Taraghi et al.,[33] and Zabrovskiy et al.[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 do not capture the essential fine-grain dynamics of such networks. Additional related studies can be found in Refs. 36–42.

To ensure a more accurate and fair evaluation of different players, in this article, 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]

Experimental Setup

In this section, we describe the overall setup of our experiments, including encoding and packaging toolchains.

The overall diagrams of our systems built for LL-HLS and LL-DASH streaming appear in the left and right subfigures of **Fig. 1**. To generate LL-HLS streams, we used Apple's HLS reference tools[13] and FFmpeg.[14] To generate LL-DASH streams, we used Open Broadcast Software (OBS) studio,[47] FFmpeg,[14] and node-gpac-dash.[15] Additional details about our setups can be found in Refs. 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]

As shown in **Fig. 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

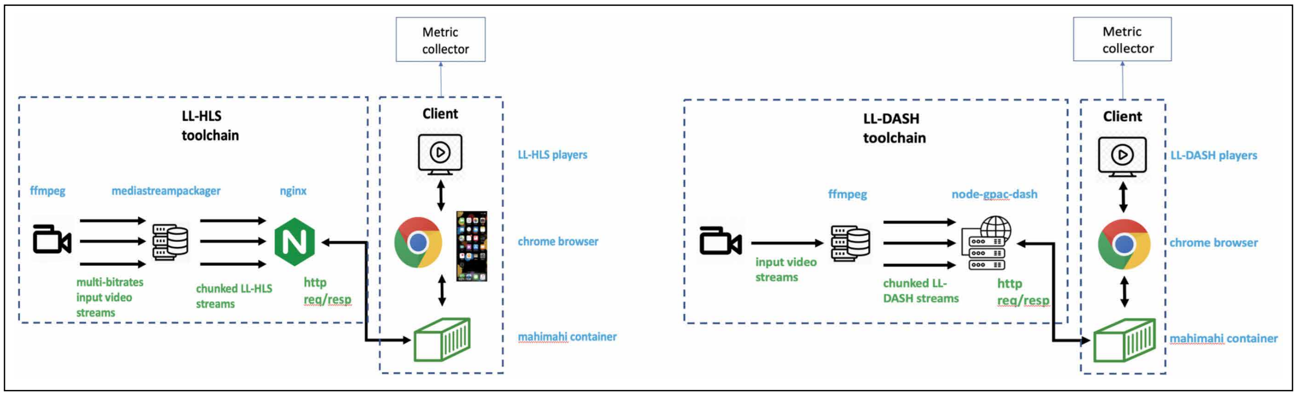


FIGURE 1. Architectures LL-HLS (left) and LL-DASH (right) streaming systems used for testing.

(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/30 frames/sec version of the Big Buck Bunny video.[51] This sequence is sufficiently long and exhibits a broad range of spatial and temporal statistics, making it a popular choice for testing video applications. To enable continuous live streaming, this sequence was looped by FFmpeg. For adaptive streaming, this sequence was transcoded into three variant streams with parameters listed in **Table 1**. The resolution and bitrate parameters have been selected for this video by using the Brightcove Context Aware Encoding tool.[52–54] The top bitrate was limited to match bitrates attainable by mobile networks.

To minimize fluctuations of encoding bitrates from their declared targets, a constant bitrate (CBR) encoding mode has been utilized. H.264 encoder operating in Baseline profile has been used. Lookahead processing is disabled. The segment lengths and fragment durations were set to 4 and 1 sec, respectively, matching the default values used in Apple's streaming tools for LL-HLS.[13] The same encoding profile 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 six 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: 1) Dash.js original,[10] 2) Dash.js with LoL algorithm,[11] and 3) Dash.js with L2ALL algorithm.[12] We have implemented simple test applications for all the players. The applications were built using the latest player Software Development Kit (SDK) releases as available in December 2020.

The reporting of metrics indicative of live streaming latency, playback speed, and rebuffering 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 Ref. 6, which is

Table 1. Encoding profile parameters used for both LL-HLS and LL-DASH systems.

Parameter	Rendition 1	Rendition 2	Rendition 3
Bitrate (kbits/s)	279	925	1,253
Frame rate (frames/sec)	30	30	30
Video resolution (pixels)	320 × 180	640 × 360	768 × 432
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 2. Performance metrics collected in our experiments.

Metrics	Impact domain(s)
Streaming bitrate (kbits/s)	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

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

$$\mathrm{PL} = (\mathrm{WC} - \mathrm{WCA}) - (\mathrm{PT} - \mathrm{PTA})/\mathrm{TS} \qquad (1)$$

where PL represents the live presentation latency, and 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. TS represents the timescale used for reporting presentation time.

For LL-DASH, the above values have been obtained from the ProducerReferenceTime[6] element embedded in a media presentation description (MPD) file, and W3C HTML5 video currentTime application programming interface (API),[55] 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 rebuffering events and the players' playback speed have been obtained by using the waiting event API[55] and the playbackRate API,[55] 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)

Table 3. Bandwidth statistics of network traces used for tests.

Bandwidth statistics	T-Mobile	Verizon
Average bitrate (kbits/s)	12,258	10,565
St. deviation of bitrate (kbits/s)	9,314	8,619
Minimum bitrate (kbits/s)	12	12
Maximum bitrate (kbits/s)	59,460	42,804

$$\text{playbackSpeedVariation} = 1n\sum i = 1Nsi - 12 \qquad (2)$$

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, and a 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**.

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

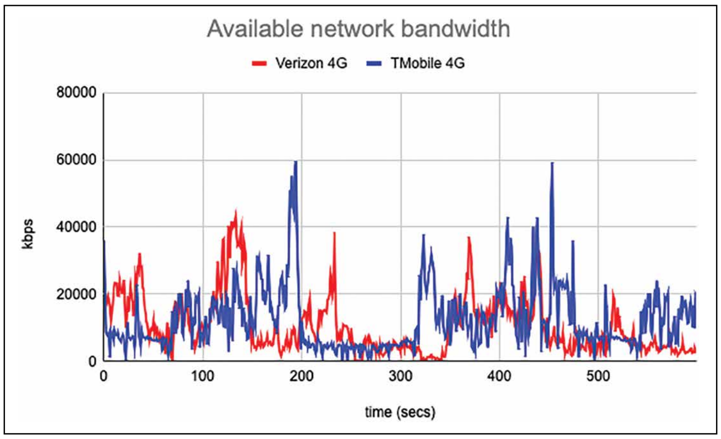


FIGURE 2. Visualizations of network bandwidth traces used for tests.

and Verizon, respectively.[43] We provide visualizations of these traces in **Fig. 2**. In **Table 3**, we list several basic statistics associated with them.

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 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.

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 3** shows the dynamics of bitrate changes in LL-HLS and LL-DASH systems. **Figure 4** shows the dynamics of playback latencies achieved by both systems.

Based on **Table 4** and **Fig. 4**, we first note that the latencies achieved by LL-DASH players and their variations were considerably lower than the 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 the 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 runs in a large number of buffering events as a result. The AVplayer's behavior was interesting: it started to operate in about 4-sec latency mode, but then, by the middle of the session, it increased latency to 12 sec, and then increased it again to 16 sec, and never recovered to a 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 two times, while DASH.js buffered five times. But we also noticed that many players have been switching across streams very often. For example, AVplayer has made 130 switches in 600 sec—a 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 the best among LL-HLS. AVplayer was a close next. The average consumed bitrate and resolutions delivered by the best players for LL-HLS and LL-DASH systems were comparable. But we also noted the number

Table 4. Summary of performance metrics obtained for Verizon 4G LTE network.

Metrics	LL-HLS players			LL-DASH players		
	HLS.js	Shaka	AVplayer	DASH.js	LoL	L2ALL
Avg. bitrate (kbits/s)	849	1,228	1,136	1,165	595	1,073
Avg. height (pixels)	328	426	404	410	262	387
Avg. latency (sec)	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 Mbits/s	85	90	99	88	45	81
Downloaded objects (chunks + segments)	673 (662 + 11)	587 (587 + 0)	669 (611 + 58)	152	151	152

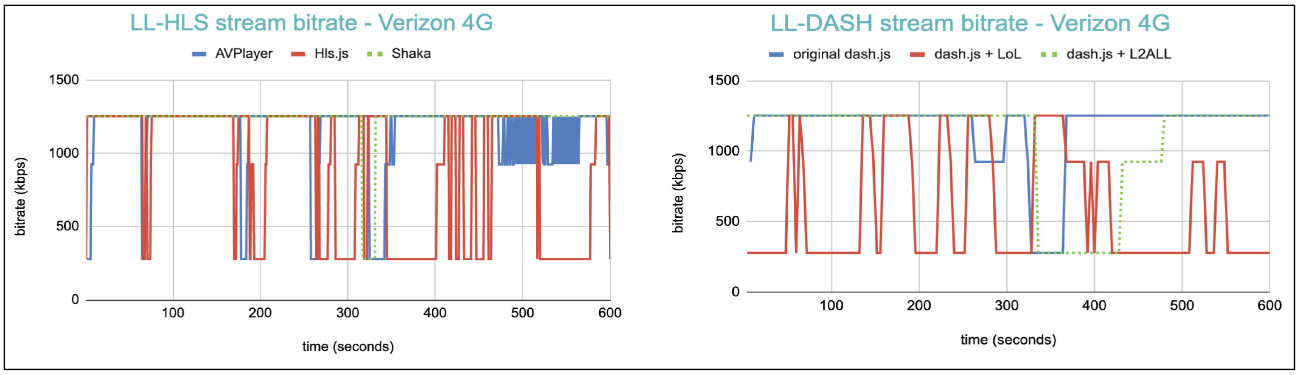


FIGURE 3. Bitrate variation over time—Verizon 4G LTE.

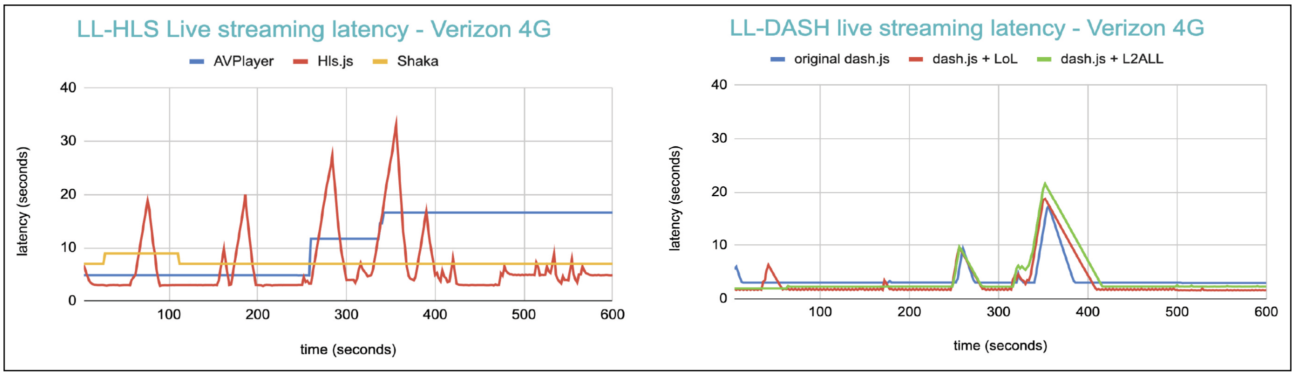


FIGURE 4. Latency variation over time—Verizon 4G LTE.

Table 5. Summary of performance metrics obtained for T-Mobile 4G LTE network.

Metrics	LL-HLS players			LL-DASH players		
	HLS.js	Shaka	AVplayer	DASH.js	LoL	L2ALL
Avg. bitrate (kbits/s)	783	1,043	1,037	1,225	537	1,251
Avg. height (pixels)	311	378	378	426	248	432
Avg. latency (sec)	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 Mits/s	156	81	92	93	42	94
Downloaded objects (chunks + segments)	965 (743 + 222)	621 (621 + 0)	703 (698 + 5)	151	152	151

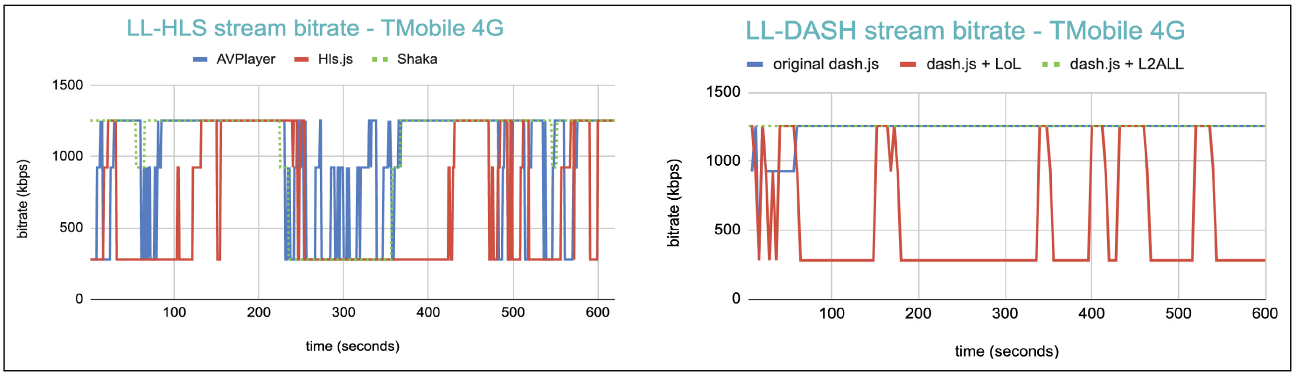


FIGURE 5. Bitrate variation over time—T-Mobile 4G LTE.

of objects (chunks or whole segments) downloaded by LL-HLS systems was significantly higher than in LL-DASH. This relates to the differences in the 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 5** shows the dynamics of bitrate changes in LL-HLS and LL-DASH systems. **Figure 6** shows the dynamics of playback latencies achieved by both systems.

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 lower 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 the 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.

Conclusion

In this study, we evaluated the LL-HLS and LL-DASH streaming systems under identical network conditions

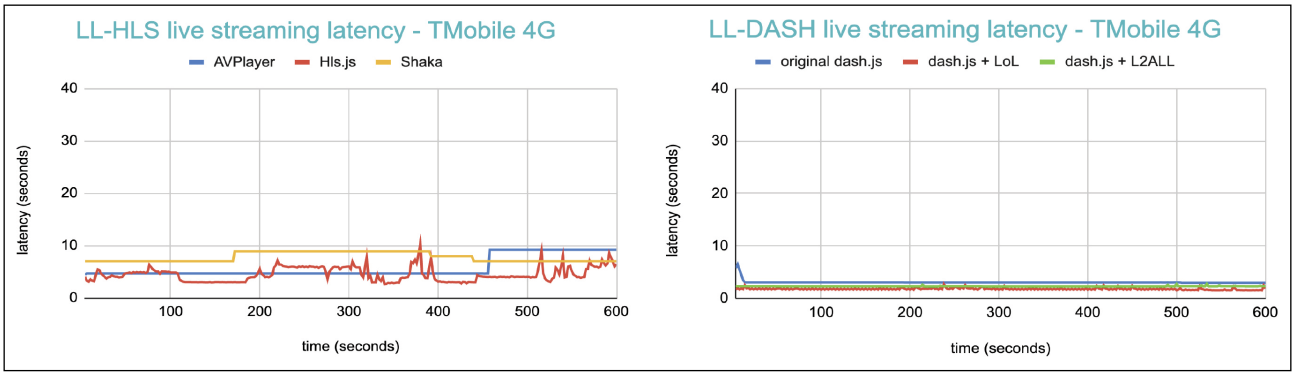


FIGURE 6. Latency variation over time—T-Mobile 4G LTE.

and by using several available implementations of streaming players for both systems.

Based on our experiments, we 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 observed latencies in the range of 3-4 sec, except for a few segments when bandwidth was insufficient to maintain live playback. For LL-HLS players, we observed a broader variation in streaming latencies across different player implementations, but with most data points fitting in the 4-10 sec range.

However, we 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); and
- the inability of some players to maintain low delay.

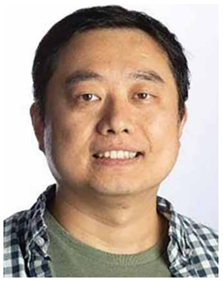
Based on these observations, we believe that while promising, both LL-HLS and LL-DASH systems still have some room for improvement. This is especially true when operating under challenging network conditions, such as mobile networks with significant load, handoffs, poor connectivity, and other effects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

References

1. Internet Engineering Task Force (IETF), RFC 8216, "HTTP Live Streaming." Accessed: May 31, 2022. [Online]. Available: https://tools.ietf.org/html/rfc8216, 2017
2. International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) 23009-1:2012, "Information Technology—Dynamic Adaptive Streaming Over HTTP (DASH)—Part 1: Media Presentation Description and Segment Formats," Feb. 2012.
3. Internet Engineering Task Force (IETF), RFC 8216, "HTTP Live Streaming, 2nd Edition," 2019. Accessed: May 31, 2022. [Online]. Available: https://tools.ietf.org/html/draft-pantos-hls-rfc8216bis-08
4. Apple, "Enabling Low-Latency HLS." Accessed: May 31, 2022. [Online]. Available: https://developer.apple.com/documentation/http_live_streaming/enabling_low-latency_hls
5. European Telecommunication Standards Institute (ETSI) Technical Specification, "MPEG-DASH Profile for Transport of ISO-BMFF Based DVB Services Over IP Based Networks." Accessed: July 8, 2022. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/103200_103299/103285/01.03.01_60/ts_103285v010301p.pdf
6. DASH Industry Forum, "Low-Latency Modes for DASH." Accessed: May 31, 2022. [Online]. Available: https://dashif.org/docs/CR-Low-Latency-Live-r8.pdf
7. AVFoundation, Accessed: May 31, 2022. [Online]. Available: https://developer.apple.com/av-foundation/
8. Hls.js Player, Accessed: May 31, 2022. [Online]. Available: https://github.com/video-dev/hls.js/
9. Shaka Player, Accessed: May 31, 2022. [Online]. Available: https://github.com/google/shaka-player
10. Dash.js Player, Accessed: May 31, 2022. [Online]. Available: https://github.com/Dash-Industry-Forum/dash.js
11. M. Lim et al., "When They Go High, We Go Low: Low-Latency Live Streaming in dash.js with LoL," *ACM Multimedia Syst. Conf.*, Online, June 8–11, 2020.
12. T. Karagkioules et al., "Online Learning for Low-Latency Adaptive Streaming," *ACM Multimedia Syst. Conf.*, Online, June 8–11, 2020.
13. HLS Tools, Accessed: May 31, 2022. [Online]. Available: https://developer.apple.com/documentation/http_live_streaming/about_apple_s_http_live_streaming_tools
14. FFmpeg, Accessed: May 31, 2022. [Online]. Available: https://www.ffmpeg.org/
15. DASH Low Latency Server, Accessed: May 31, 2022. [Online]. Available: https://github.com/maxutility2011/node-gpac-dash
16. D. Wu et al., "Streaming Video Over the Internet: Approaches and Directions," *IEEE Trans. CSVT*, 11(3):282–300, 2001.
17. G. Conklin et al., "Video Coding for Streaming Media Delivery on the Internet," *IEEE Trans. CSVT*, 11(3):269–281, 2001.
18. B. Girod, et al., "Advances in Channel-Adaptive Video Streaming," *Wireless Comm. Mobile Comp.*, 2(6):573–584, 2002.
19. A. Bentaleb et al., "A Survey on Bitrate Adaptation Schemes for Streaming Media Over HTTP," *IEEE Commun. Surv. Tutor.*, 21(1):562–585, 2019.
20. 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.
21. Z. Li et al., "Probe and Adapt: Rate-Adaptation for HTTP Video Streaming at Scale," *IEEE J Sel. Areas Commun.*, 32(4):719–733, April 2014.
22. K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-Optimal Bitrate Adaptation for Online Videos," *Annu. IEEE Int. Conf. Comput. Commun.*, pp. 1–9, 2016.
23. T. Huang et al., "A Buffer-Based Approach to Rate Adaptation: Evidence From a Large Video Streaming Service," *ACM SIGCOMM*, pp. 187–198, 2014.
24. K. Spiteri, R. Sitaraman, and D. Sparacio, "From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player," *ACM Trans. Multimedia Comput. Commun. Appl.*, 15(2s), Article 67, 2019.
25. S. Hesse, "Design of Scheduling and Rate-Adaptation Algorithms for Adaptive HTTP Streaming," *SPIE 8856, Appl. of Digital Image Process. XXXVI, 88560M*, 2013.
26. X. Yin et al., "A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP," *SIGCOMM Comput. Commun.*, Rev. no. 45(4) 325–338, 2015.
27. C. Zhou et al., "A Control-Theoretic Approach to Rate Adaptation for Dynamic HTTP Streaming," *Visual Comm. Image Processing, San Diego, CA*, pp. 1–6, 2012.
28. D. Talon et al., "Comparing DASH Adaptation Algorithms in a Real Network Environment," *Eur. Wireless 2019; 25th Eur. Wireless Conf.*, 2019.
29. C. Storck and F. Figueiredo, "A Performance Analysis of Adaptive Streaming Algorithms in 5G Vehicular Communications in Urban Scenarios," *IEEE Symp. Comput. Commun.*, pp. 1–7, 2020.
30. D. Raca et al., "DASHbed: A Testbed Framework for Large Scale Empirical Evaluation of Real-Time DASH in Wireless Scenarios," *ACM Multimedia Syst. Conf.*, Amherst, MA, pp. 285–290, June 18–21, 2019.
31. I. Ayad et al., "A Practical Evaluation of Rate Adaptation Algorithms in HTTP-based Adaptive Streaming," *Elsevier Comput Netw.*, 133:90–103, 2018.
32. C. Midoglu, et al., "Docker-Based Evaluation Framework for Video Streaming QoE in Broadband Networks," *ACM Int. Conf. on Multimedia*, Nice, France, pp. 2288–2291, Oct. 21–25, 2019.
33. B. Taraghi et al., "CAdViSE: Cloud-based Adaptive Video Streaming Evaluation Framework for the Automated Testing of Media Players," *ACM Multimedia Syst. Conf.*, Online, June 8–11, 2020.
34. A. Zabrovskiy et al., "AdViSE: Adaptive Video Streaming Evaluation Framework for the Automated Testing of Media Players," *ACM Multimedia Syst. Conf.*, Taipei, Taiwan, June 20–23, 2017.
35. D. Raca et al. "Beyond Throughput: A 4G LTE Dataset With Channel and Context Metrics," *ACM Multimedia Syst. Conf.*, New York, NY, pp. 460–465, 2018.
36. A. Bentaleb et al., "Data-Driven Bandwidth Prediction Models and Automated Model Selection for Low Latency," *IEEE Trans. on Multimedia*, Aug. 2020.
37. A. Bentaleb et al., "Bandwidth Prediction in Low-Latency Chunked Streaming," *ACM Workshop on Network and Operating Syst. Support for Digital Audio and Video*, Amherst, MA, pp. 7–13, June 21, 2019.
38. A. Bentaleb et al., "Performance Analysis of ACTE: A Bandwidth Prediction Method for Low-latency Chunked Streaming," *ACM Trans. Multimedia Comp., Comm., Appl.*, 16(2s):1–24, 2020.
39. I. Ozcelik and C. Ersoy, "Low-Latency Live Streaming Over HTTP in Bandwidth-Limited Networks," *IEEE Commun. Lett.*, 25(2):450–454, 2021.
40. K. Durak et al., "Evaluating the Performance of Apple's Low-Latency HLS," *IEEE Int. Workshop on Multimedia Signal Process.*, pp. 1–6, Sep. 21–24, 2020.
41. T. Huang et al., "Hindsight: Evaluate Video Bitrate Adaptation at Scale," *ACM Multimedia Syst. Conf.*, Amherst, MA, pp. 86–97, June 18–21, 2019.
42. B. Zhang, T. Teixeira, and Y. Reznik, "Performance of Low-Latency HTTP-based Streaming Players," *ACM Multimedia Syst. Conf.*, Istanbul, Turkey, Sept. 28–Oct. 1, 2021.
43. R. Netravali et al., "Mahimahi: Accurate Record-and-Replay for HTTP," *USENIX Annual Tech. Conf.*, Santa Clara, CA, July 8–10, 2015.
44. A. Mondal et al., "EnDASH—A Mobility Adapted Energy Efficient ABR Video Streaming for Cellular Networks," *IFIP Networking Conf.*, pp. 127–135, 2020.
45. G. Ribezzo et al., "A DASH 360° Immersive Video Streaming Control System," *Internet Tech. Lett.*, 3(5), 2020.
46. S. Sengupta et al., "HotDASH: Hotspot Aware Adaptive Video Streaming Using Deep Reinforcement Learning," *IEEE Int. Conf. on Network Protocols*, pp.165–175, 2018.
47. Open Broadcast Software. Accessed: May 31, 2022. [Online]. Available: https://obsproject.com/
48. B. Zhang, "Setting Up Your Own Low-Latency HLS Server to Stream from any Source Inputs." Accessed: May 31, 2022. [Online]. Available: 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." Accessed: May 31, 2022. [Online]. Available: https://bozhang-26963.medium.com/low-latency-dash-streaming-using-open-source-tools-f93142ece69d
50. Nginx Web Server. Accessed: May 31, 2022. [Online]. Available: https://www.nginx.com/
51. Blender Foundation, "Big Buck Bunny Video." Accessed: May 31, 2022. [Online]. Available: http://bbb3d.renderfarming.net/download.html
52. Y. Reznik et al., "Optimal Design of Encoding Profiles for ABR Streaming," *Proc. Packet Video Workshop*, Amsterdam, The Netherlands, June 12, 2018. Accessed: July 8, 2022. [Online], Available: https://www.w3.org/TR/2011/WD-html5-20110113/video.html
53. Y. Reznik et al., "Optimizing Mass-Scale Multiscreen Video Delivery," *SMPTE Motion Imaging J.*, 129(3):26–38, 2020.
54. Brightcove Context Aware Encoding. [Online]. Available: https://www.brightcove.com/en/products/online-video-platform/context-aware-encoding/
55. HTML5 Video. Accessed: May 31, 2022. [Online]. Available: https://www.w3.org/TR/2011/WD-html5-20110113/ video.html

About the Authors



Bo Zhang is a video systems engineer at Brightcove, Inc., Boston, MA. He researches video delivery and playback technologies and builds high-quality, smooth, scalable, and low-latency video streaming software. He has published several research articles in the domains of video streaming and wireless communications. He received a PhD degree in computer science from George Mason University, Fairfax, VA, an MS degree in computer science from the University of Cincinnati, Cincinnati, OH, and a BS degree in computer science from the Huazhong University of Science and Technology, Wuhan, China. He was a recipient of the Best Paper Award from the Association for Computing Machinery (ACM) MSWiM 2011.

Thiago Teixeira is a software engineer at Brightcove, Inc., Boston, MA. He works across teams to improve the company's internal design systems and contributes to innovations and new technology research initiatives. He received a PhD degree in computer engineering from the University of Massachusetts at Amherst, Amherst, MA, in 2019, and a BE degree in electrical engineering from Unisinos University, São Leopoldo, Brazil, in 2013. His research interests include computer networks, new internet architectures, and wireless communications.

Yuriy Reznik is a technology fellow and vice president of research at Brightcove Inc., Boston, MA. Previously, he held engineering and management positions with InterDigital, San Diego, CA, from 2011 to 2016, Qualcomm, San Diego, CA, from 2005 to 2011, and RealNetworks, Seattle, WA, from 1998 to 2005. In 2008, he was a visiting scholar at Stanford University, Stanford, CA. Since 2001, he has been involved in the work of ITU-T SG16 and MPEG standards committees and made contributions to several multimedia coding and delivery standards, including ITU-T H.264/MPEG-4 AVC, MPEG-4 ALS, ITU-T G.718, ITU-T H.265/MPEG-HEVC, and MPEG-DASH. Several technologies, standards, and products that he has helped to develop (RealAudio/RealVideo, ITU-T H.264/MPEG-4 AVC, Zencoder, Brightcove CAE, and MPEG-DASH) have been recognized by the National Academy of Television Arts and Sciences (NATAS) Technology and Engineering Emmy Awards. He received a PhD degree in computer science from Kyiv University, Kyiv, Ukraine. He is a Senior Member of IEEE and the International Society for Optics and Photonics (SPIE), and a member of the Association for Computer Machinary (ACM), Audio Engineering Society (AES), and SMPTE. He is a coauthor of over 150 conference papers and journal articles, and a coinventor of over 70 granted U.S. patents.

SMPTE
MEDIA TECHNOLOGY SUMMIT
2022
24-27 October
We're back at the Loews Hollywood Hotel in the heart of Hollywood 24-27 October