Received 23 February 2023, accepted 10 March 2023, date of publication 14 March 2023, date of current version 17 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3257099

RESEARCH ARTICLE

LLL-CAdViSE: Live Low-Latency Cloud-Based Adaptive Video Streaming Evaluation Framework

BABAK TARAGHI, (Member, IEEE), HERMANN HELLWAGNER, (Senior Member, IEEE), AND CHRISTIAN TIMMERER, (Senior Member, IEEE)
Christian Doppler Laboratory ATHENA, Universität Klagenfurt, 9020 Klagenfurt am Wörthersee, Austria

Corresponding author: Babak Taraghi (babak.taraghi@aau.at)

This work was supported in part by the Austrian Federal Ministry for Digital and Economic Affairs; in part by the National Foundation for Research, Technology, and Development; and in part by the Christian Doppler Research Association.

ABSTRACT Live media streaming is a challenging task by itself, and when it comes to use cases that define low-latency as a must, the complexity will rise multiple times. In a typical media streaming session, the main goal can be declared as providing the highest possible Quality of Experience (QoE), which has proved to be measurable using quality models and various metrics. In a low-latency media streaming session, the requirements are to provide the lowest possible delay between the moment a frame of video is captured and the moment that the captured frame is rendered on the client screen, also known as end-to-end (E2E) latency and maintain the QoE. This paper proposes a sophisticated cloud-based and open-source testbed that facilitates evaluating a low-latency live streaming session as the primary contribution. **L**ive **L**ow-**L**atency **C**loud-based **Ad**aptive **Vi**deo **S**treaming **E**valuation (LLL-CAdViSE) framework is enabled to asses the live streaming systems running on two major HTTP Adaptive Streaming (HAS) formats, Dynamic Adaptive Streaming over HTTP (MPEG-DASH) and HTTP Live Streaming (HLS). We use Chunked Transfer Encoding (CTE) to deliver Common Media Application Format (CMAF) chunks to the media players. Our testbed generates the test content (audiovisual streams). Therefore, no test sequence is required, and the encoding parameters (*e.g.*, encoder, bitrate, resolution, latency) are defined separately for each experiment. We have integrated the ITU-T P.1203 quality model inside our testbed. To demonstrate the flexibility and power of LLL-CAdViSE, we have presented a secondary contribution in this paper; we have conducted a set of experiments with different network traces, media players, ABR algorithms, and with various requirements (*e.g.*, E2E latency (*typical/reduced/low/ultra-low*), diverse bitrate ladders, and catch-up logic) and presented the essential findings and the experimental results.

INDEX TERMS Live Streaming, low-latency, HTTP adaptive streaming, quality of experience, objective evaluation, open-source testbed.

I. INTRODUCTION

Recent statistical studies show a large percentage of the Internet and mobile network traffic consists of audiovisual streaming. Video constituted around 70% of all global mobile network traffic in 2022. This is expected to increase by around 30% annually until the end of 2028, when it is forecast to account for 80% of global mobile data traffic [1]. HTTP Adaptive Streaming (HAS) encodes a multimedia file at multiple bitrates, video resolutions, audio sample rates, and other factors, *i.e.*, *representations*. Each representation will then be split into temporal segments and stored on an HTTP server. On the client side, the media player runs an Adaptive Bitrate (ABR) algorithm to select, for each segment, the most suitable representation to be downloaded. The ABR decision is made by taking into account many factors, and it affects the QoE significantly. QoE is commonly indicated by the Mean Opinion Score (MOS). MOS can be divided into two

The associate editor coordinating the review of this manuscript and approving it for publication was Alessandro Floris.

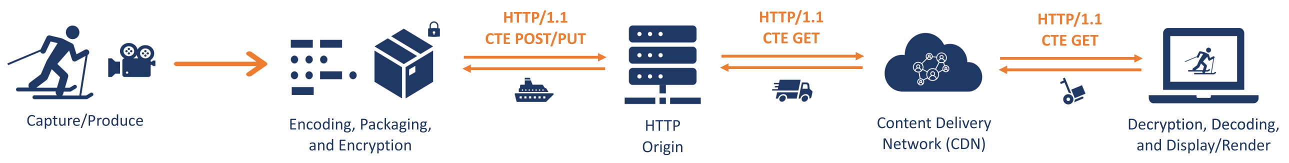

Capture/Produce
Encoding, Packaging, and Encryption
HTTP/1.1 CTE POST/PUT
HTTP Origin
HTTP/1.1 CTE GET
Content Delivery Network (CDN)
HTTP/1.1 CTE GET
Decryption, Decoding, and Display/Render

FIGURE 1. Low-latency live streaming by HTTP Chunked Transfer Encoding (Illustration inspired by a keynote at ACM MMSys'22 by Ali C. Begen - A master's toolbox and algorithms for low-latency live Streaming).

types, predicted and perceived. While perceived MOS can be obtained only by subjective evaluation, various models for predicting the MOS exist.

The term *latency* in live media streaming is the time difference between the moment the actual event is captured on camera or the live feed comes out of a playout server and when the end user sees the content on their device's screen, *i.e.*, E2E latency. Historically an online live streaming service has a latency somewhere between 30 seconds to over 60 seconds (high latency) depending on the viewer device capability and the used video workflow. Nowadays, the challenge for such a service is to lower the live streaming latency to a range closer to linear broadcast signal latency, 3 to 5 seconds (low-latency) or sometimes lower (ultra low-latency and near real-time) [2]. With more devices capable of playing audiovisual content with low-latency, research and technological development in this area gained more attention.

Media players implement a function referred to as catch-up logic to overcome a challenge that low-latency live media streaming puts on the table. The procedure defines the strategy of how the player should catch up with the live edge. Two approaches to this problem are commonly used. The first is to increase or decrease the playback speed to reach a point where the content is available, and the achieved latency is closest to the defined target latency. It can be controlled by adjusting the maximum and minimum playback speed rates. The catch-up logic will decide how fast or slow the player should play the content within the defined playback speed range to get closer to the target latency. The second approach is to jump over a few frames of content or introduce a stall event to reach a point where the content is available, and the achieved latency is closest to the defined target latency. With this strategy, the catch-up logic should decide how many frames should be dropped from the player buffer or how long the playback should be stalled to get closer to the target latency.

Chunked Transfer Encoding (CTE) is a data transfer mechanism in version 1.1 of the Hypertext Transfer Protocol (HTTP) [3]. The end-to-end perspective of such an approach for live media streaming is shown in Figure 1.

Introduced in 2016, the Common Media Application Format (CMAF) [4] is a standard transport container for streaming VoD and linear media using MPEG-DASH [5] or HLS [6]. Thanks to CMAF in low-latency mode and with CTE, the player requests for incomplete media segments, which will be served by smaller units called *chunks*. Each chunk can be 500 milliseconds or lower, depending on the encoder configuration. As shown in Figure 2, a chunk is the smallest referenceable media unit, containing a *moof* and *mdat* atom. The *mdat* box holds a single Instantaneous Decoder Refresh (IDR) frame, which is required to begin every segment. A segment is a collection of one or more *fragments*, and a fragment is a collection of one or more chunks. The *moof* box is required by the player for decoding and rendering individual chunks [2].

Using CTE, the server can deliver chunks of data with undetermined size, which enables transferring the generated CMAF chunks as they come out from the encoder and without the need to be written on the disk (if caching the data was not desired). The chunked transfer coding wraps content in order to transfer it as a series of chunks, each with its size indicator, followed by an optional trailer section containing trailer fields. CTE enables content streams of unknown size to be transferred as a sequence of length-delimited buffers, which enables the sender to retain connection persistence and the recipient to know when it has received the entire message [3].

In this paper, we present our sophisticated open-source testbed: **L**ive **L**ow-**L**atency **C**loud-based **Ad**aptive **Vi**deo **S**treaming **E**valuation (`LLL-CAdViSE`). LLL-CAdViSE is designed to be highly scalable in terms of a number of real media players placed in experiments with different network profiles. Our testbed is capable of evaluating real live streams with both HLS and MPEG-DASH. The encoder and packager components are fully customizable, and the parameters can be adjusted based on the scenarios defined by the evaluator. This framework will record various significant metrics from the streaming session, enabling it to measure the latency precisely. At the end of each experiment, the testbed is capable of automatically generating a single *.mp4* file that is a chain of played segments stitched together, combined with the real events and defects that happened in the experimental session. Our testbed does not require to be supplied by video-on-demand (VoD) test content. It generates highly dynamic (randomly moving objects transforming into different shapes and with constantly changing color and size) video content and a constant beep as the audio track. Figure 3 is a sample frame of the generated video in an experiment. The QoE calculation based on the ITU-T P.1203 model [7] is integrated into the framework. Since this model does not consider the meaningfulness of the streamed content, it can facilitate the process of providing predicted MOS automatically.

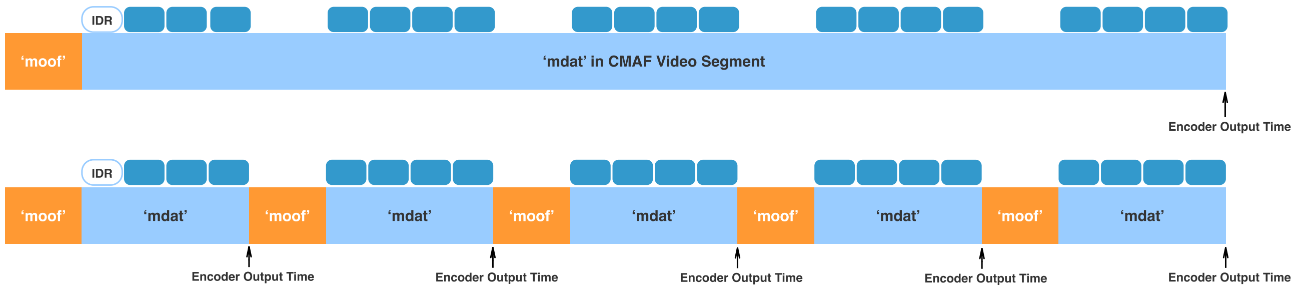


FIGURE 2. Top: A CMAF segment containing a sequence of 20 samples, Bottom: A same size CMAF chunked segment with multiple 'moof' boxes, which decreases the possible encoder output time.

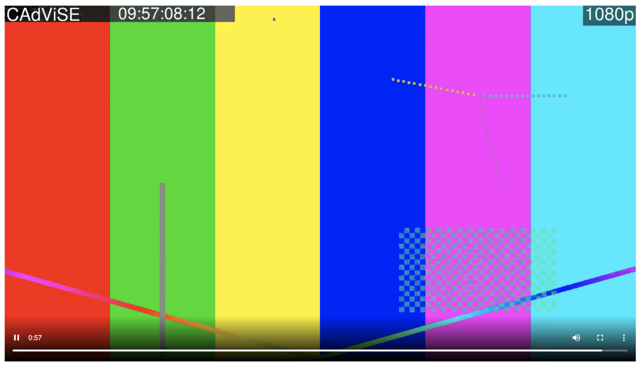


FIGURE 3. A single frame of the highly dynamic (randomly moving objects transforming into different shapes and with constantly changing color and size) video generated by the LLL-CAdViSE server.

The remainder of the paper is organized as follows. Section II reviews related work, followed by the introduction of our testbed architecture in Section III. Next, the experimental setup is presented in Section IV. In Section V, the experimental results and key findings are described in detail. Finally, Section VI concludes the paper with an overview of the main contributions.

II. RELATED WORK

In 2017, AdViSE [8] was introduced as a framework for testing media players. This testbed was developed to be deployed on physical machines and evaluate the performance of the media players by manipulating the network link between the server and its clients. The presented framework is used for the comparison and testing media players in the context of adaptive video streaming over HTTP in Web/HTML5 environments. AdViSE uses a customized Mininet-based software-defined network (SDN) for network emulation and bandwidth shaping. Through a Web management interface, it conducts experiments and runs the adaptive media players.

In 2020, CAdViSE [9] was introduced, a cloud-based version of the same concept as in AdViSE, with multiple improvements. This testbed is capable of deployment onto a cloud infrastructure such as Amazon Web Services (AWS). It records comprehensive logs from the test streaming session and provides visualized statistics over the results on the Bitmovin Analytics[1] dashboard or similar applications. In another work [10], the authors further developed the functionalities of CAdViSE to suit their project needs, leading to the implementation of a new network node (an *edge*) in the testbed for repackaging the content more efficiently. CAD-LAD [11] is another extension of CAdViSE, which presented the implementation of some parts of the Common Media Client Data (CMCD) standard on this testbed.

Video BenchLab [12] presents an open and flexible benchmarking platform to measure the performance of streaming media workloads. Video BenchLab can be used with any existing media server and provides a set of tools for researchers to experiment with their platforms and protocols.

Ramos-Chavez et al. [13] presented a testbed for the MPEG Network Based Media Processing (NBMP) standard [14], implementing all the standard's components. The testbed includes many configurable functions for load generation, monitoring, data collection, and visualization. The testbed is used to test dynamic adaptive HTTP streaming functions under different workloads in a standardized and reproducible manner.

Stohr et al. [15] presented a framework to evaluate open-source HAS media players. They compared the performance and extracted the strengths and weaknesses of the media players. A significant observation in their study is stated as the importance of the target buffer size and the player implementation compared to the choice of the ABR algorithm.

Yadav et al. [16] proposed a combination of techniques collectively called QLive in which they have tried to resolve the fundamental problem of bandwidth estimation for low-latency live streaming by using existing chunk parser and filtering of downloaded chunk data. They have used an Apache Web server to host the dash.js media player. The FFmpeg encoder with CMAF packager and origin ran on the server provided by Streamline.[2] The server and client ran on two Linux-based machines connected by a router and used the tc NetEm network emulator to control the network bandwidth.

[1]https://bitmovin.com/docs/analytics, accessed Jan. 5, 2023
[2]https://github.com/streamlinevideo/low-latency-preview, Feb. 21, 2023

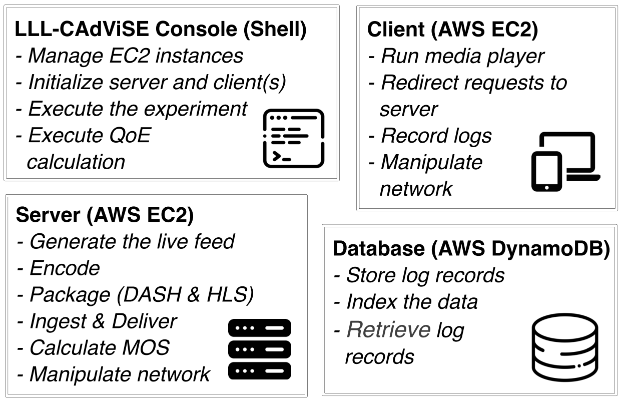


FIGURE 4. Live low-latency CAdViSE system components.

Taha et al. [17] presented a testbed for evaluation of QoE in HAS. The testbed uses physical machines to execute experiments. It can measure metrics such as startup delay, frequency of quality switches, number and length of stall events, CPU usage, and energy consumption. A CDN testbed is proposed by Taha [18] that consists of two mechanisms based on the performance of server load and network congestion to deliver fast the segmentation of adaptive video streaming and redirect clients' requests to appropriate surrogate servers.

Our testbed is a highly scalable and sophisticated evaluation framework for HAS media players and ABR algorithms. In contrast to mentioned related works, LLL-CAdViSE uses modern cloud infrastructure. It is also the first comprehensive low-latency enabled live streaming emulator over the cloud infrastructure that precisely measures various significant metrics. LLL-CAdViSE allows for E2E latency measurement alongside a variety of significant metrics such as stall events duration, startup delay, seek duration, quality switches, played bitrate for each segment, and playback speed rate. Our testbed is capable of evaluating any Web-based HAS media player with ease, and updating the media players or evaluating other media players is as simple as replacing an HTML file with the new media player. The calculation of predicted MOS using the ITU-T P.1203 quality model is integrated with this framework, and our testbed is available as open-source software.

III. LLL-CAdViSE ARCHITECTURE

LLL-CAdViSE[3] is designed and developed on top of a testbed introduced in 2020, called CAdViSE[4] [9]. CAdViSE provides a framework for the automated evaluation of media players and their ABR algorithms. The presented system has a modular architecture and provides a set of tools for analyzing and visualizing obtained results. CAdViSE utilizes a cloud infrastructure to instantiate, control, and continuously monitor the two network node types in a HAS session, the server, and the clients. The established network connection between the nodes is managed and manipulated by predefined network profiles using simple JSON files.

Similar to its predecessor, LLL-CAdViSE follows the same goal, to provide a testbed that makes the execution of experimental streaming sessions maintainable and scalable. We have completely redesigned and redeveloped the testbed so that actual live streaming is enabled and low-latency factors are considered. Firstly, our framework generates the audiovisual test content; therefore, unlike CAdViSE, we do not need to provide encoded and packaged VoD content for each experiment. A simple file called *streams.json* will steer the content generator to define a number of streams with different characteristics, *i.e.*, bitrates, and resolutions. Secondly, by introducing various adjustable parameters on the server such as (*-tune zerolatency, -ldash and -lhls*), the produced content will be delivered to the clients with low latency enabled. LLL-CAdViSE, as a plus, also provides experimental low-latency HLS streams for media players capable of playing such content. The components of LLL-CAdViSE are shown in Figure 4 and described in detail in the following.

The LLL-CAdViSE console is in charge of (*i*) spinning-up and management of Amazon Elastic Compute Cloud[5] (EC2) instances, (*ii*) initialization of server and clients with required libraries, applications, and packages, (*iii*) execution of defined experiments, and optionally (*iv*) calculation of predicted MOS.

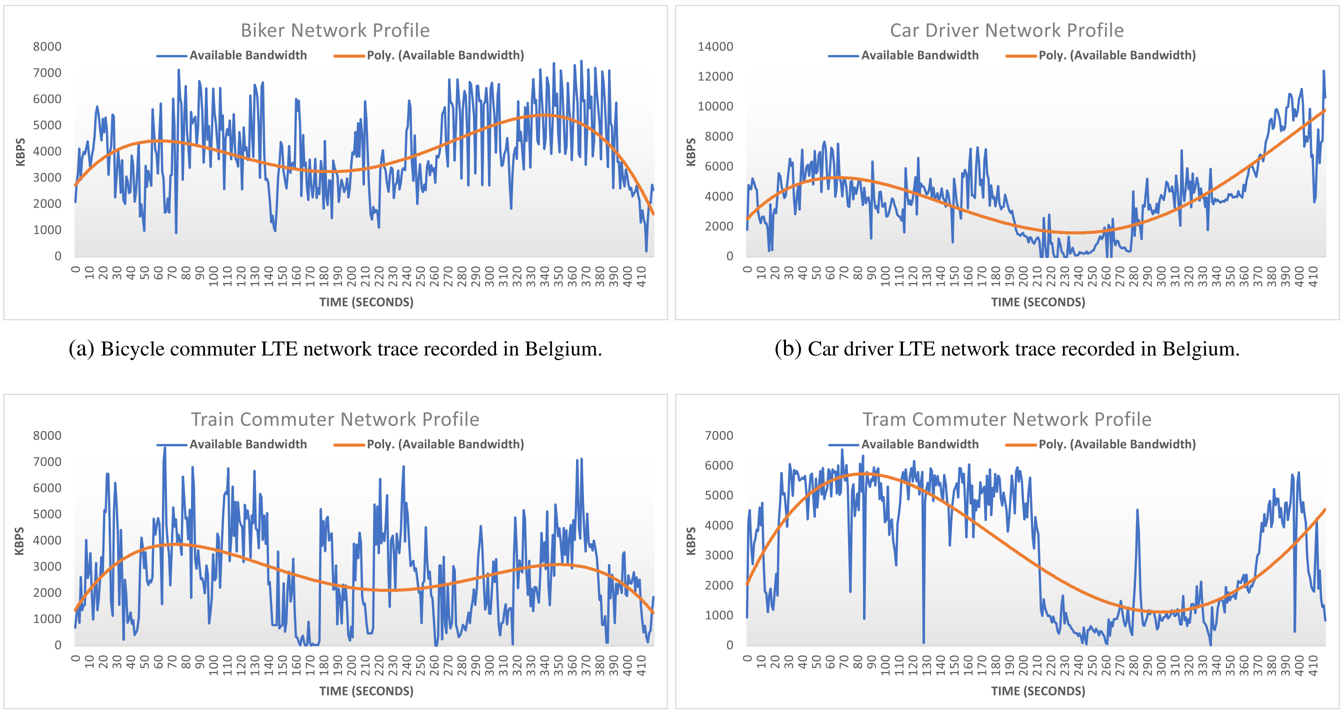
EC2 instances are requested from the cloud provider to be instantiated and available in a specific cloud region and under a single cluster to minimize the network latency introduced between the nodes. Once the EC2 instances are up and running, two types of initializers, starters and configuration scripts will be injected into each machine, specifically created for the server and clients. The initializer scripts will be invoked to install the required libraries and packages.

Next, the starter scripts will be executed synchronously on all nodes (server and clients), which run the stream producer, encoder, and packager (called the engine) and bring up the two HTTP servers on the server EC2 instance. The first HTTP server, the ingest server, receives the data from the stream engine via the HTTP PUT method, caches the ingress data in the memory (cache object), and stores it on the disk while it is being streamed. The second HTTP server, called the delivery server, will initially be on standby to receive requests from the clients. The received requests will be directly served from the stored data in the cache object if it is still available; otherwise, it will be from the disk. If the received request is seeking for *.m4s* files the value of *Transfer-Encoding* on the HTTP response object will be set to *Chunked*.

On the client EC2 instance(s), the starter script will bring up an HTTP server to facilitate the communication between the media player and the delivery server while recording the logs of each request and the media player events into the database. The starter script will also kick off the streaming session by sending the play command to the media players.

[3]https://github.com/cd-athena/LLL-CAdViSE, accessed Jan. 5, 2023
[4]https://github.com/cd-athena/CAdViSE, accessed Jan. 5, 2023

[5]https://docs.aws.amazon.com/ec2/, accessed Jan. 5, 2023



(a) Bicycle commuter LTE network trace recorded in Belgium.

(b) Car driver LTE network trace recorded in Belgium.

(c) Train commuter LTE network trace recorded in Belgium.

(d) Tram commuter LTE network trace recorded in Belgium.

FIGURE 5. Various network profiles trimmed to 420 seconds were used for the experiments. The polynomial (Poly) trendline of the available bandwidth with order 4 depicts the fluctuations from a broader perspective.

Clients can be HLS or DASH media players, and since the engine is enabled to generate CMAF chunks, the delivery server is capable of serving both types accordingly. The HTTP server on each client node also receives the events raised by the media players and stores them in the database for later retrieval. Another task defined in the starter scripts is manipulating the network shape based on the selected network profile on both server and client machines. We use Wondershaper[6] scripts to limit the bandwidth of network adapters for each machine. We also use Linux *tc* commands to introduce the network delays. After the experiment duration, the server retrieves the recorded logs from the database. It calculates the following metrics: *playbackDuration, stallsDuration, startUpDelay, seekedDuration, qualitySwitches, minBitrate, maxBitrate, averageBitrate, minLatency, maxLatency, averageLatency, minPlaybackRate, maxPlaybackRate, averagePlaybackRate*.

If declared by the evaluator via the *–withQoE* option on the LLL-CAdViSE console, the predicted MOS will also be computed using the ITU-T P.1203 QoE model (mode 1), and the results will be available immediately after the experiment is finished. The authors of [19] made a comparison between some of the available QoE models and presented the correlation of results with actual perceived MOS in their subjective evaluations. ITU-T P.1203 proved to have the most accurate results in multiple scenarios. This model integrates predictions based on a large set of training and validation data. Four modes are available within this model: 0, 1, 2, and 3, in which mode 0 takes the least metadata as the input, whereas mode three needs full access to the bitstream. In mode 1, which we used in our framework, audio/video codec, video resolution, framerate, audio/video bitrate, frame type, and frame size are considered while calculating the MOS.

As mentioned before, mode 1 of this quality model considers audio/video codec, video resolution, framerate, audio/video bitrate, frame type, and frame size.

The final results for all clients will be stored in an Amazon Simple Storage Service[7] (S3) bucket and in CSV file format. If the evaluator desires, the testbed can also use the recorded logs and stitch back the streamed media segment for each client to generate a single *.mp4* file. This generated *.mp4* file can later be used for further investigation of the experimental session's video quality and the occurred events with possible defects. A use case of this functionality can be found in [20].

IV. EXPERIMENTAL SETUP

We performed experiments with the following configurations to demonstrate our testbed's capabilities and functionalities.

LLL-CAdViSE provides a platform for evaluating many characteristics of a media streaming system and producing a wide variety of results (plots). An extensive set of parameters and factors can be configured; each might affect the outcome differently. The stated configuration and setup are chosen as a sample, providing insight into the available options.

[6]https://github.com/magnific0/wondershaper, accessed Jan. 5, 2023

[7]https://docs.aws.amazon.com/s3/, accessed Jan. 5, 2023

TABLE 1. Provided streams (representations) by the origin server in the experimental setup.

Index	Resolution	Video Bitrate	Audio Bitrate
1	256x144	100kbps	128kbps
2	384x216	375kbps	128kbps
3	640x360	750kbps	128kbps
4	1024x576	1500kbps	128kbps
5	1280x720	3000kbps	128kbps
6	1920x1080	5000kbps	128kbps

We compared the performance of different ABR algorithms and two widely used media players for the HLS and MPEG-DASH. Inspired by the work in [21], we have selected six representations, shown in Table 1. The selection covers a range of very low to high-definition quality streams the server provides for our evaluation. The first representation (index 1 in the table) has an audio bitrate of 128kbps, which is higher than the video bitrate; this is not usual. This representation is not recommended for a production environment, but here we use it for the sake of testing corner cases. The last representation (index 6 in the table) provides a video resolution of 1920 × 1080 which is higher than the restricted view-port as mentioned in IV-B and also, in the P.1203 model, defined the view-port as such. If the media player requests content with a higher resolution than what it is restricted to, it can negatively affect the QoE by risking the introduction of unwanted stall events. And at the same time, the P.1203 model does not give any credit when a higher-than-viewport video content resolution is played. When such requests are observed in the stored logs of the testbed, it can be considered room for improvement in the media player.

We have set four target latencies (TL) in our experiments according to the claim in [22] that 1s, 3s, 5s, and 10s latencies are the most expected TLs in live streaming scenarios.

The duration of individual experiments is equal to the length of provided network profiles, 420 seconds. We did three runs for each scenario, *i.e.*, ABR algorithm, a network profile, and the target latency. Each media player was tested in an isolated network, and there was no competition over the available bandwidth between the media players as a single separated server was serving each. The configuration for FFmpeg[8] as our encoder and packager was identical among all the experiments, as shown in Table 3. In the following, we will describe the setup with details for the experiments.

A. CLOUD ENVIRONMENT

According to the number of concurrent tasks we have defined for the server, we have selected a *m5ad.12xlarge* size EC2 instance. This instance provides a 10 Gbps network bandwidth suitable for our experiments. Amazon *m5ad* instances feature AMD EPYC 7000 series processors with an all-core turbo clock speed of 2.5 GHz. The *12xlarge* instance has 48 CPU cores and 192 GiB of available memory. With Amazon *m5ad* EC2 instances, local NVMe-based SSDs are physically connected to the host server and provide block-level storage coupled to the instance's lifetime. The selected instance size has two SSD volumes of 900 GB.

For the clients in our test environment, we have selected *m5ad.2xlarge*, which benefits from the same capabilities as the server node but with fewer resources. This instance provides up to 10 Gpbs network bandwidth. There are 8 CPU cores and 32 GiB of available memory for client instances with one SSD volume of 300 GB.

B. MEDIA PLAYERS

We have integrated two well-known media players to support the evaluation of MPEG-DASH and HLS streaming. For MPEG-DASH, we have used *dash.js* media player version 4.4.1, and for HLS, we have used *hls.js* media player version 1.2.0. Both players have been placed in a simple HTML file with a *Video* element, which restricts the view-port resolution to 1280 × 720. Via a native API integration, we then listen for all the events fired by the HTML video element and, alongside the occurrence timestamps, store them in the database through the HTTP server instantiated on the client EC2 instances. We have enabled the low-latency mode on both players. For *dash.js*, we signal the TL and the playback rates without further modification on the media player defaults through the manifest file. For *hls.js*, we set the following configuration: *startLevel=−1, enableWorker=true, liveSyncDuration=0, liveMaxLatencyDuration=[TL], testBandwidth=true, low LatencyMode=true*, and *abrController.targetLatency=[TL]*.

We have tried to minimize the changes in media players' configurations and leave the default values as they are. There might be possible adjustments and tuning on both media players, which would lead to better results than what is presented in Section V.

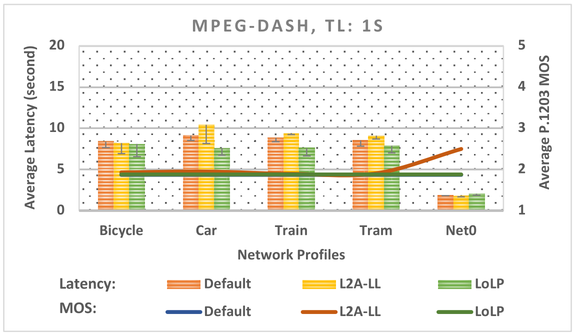
C. ABR ALGORITHMS

Three ABR algorithms have been prepared for our experimental evaluations in combination with the two media players. The default algorithms for both *dash.js* and *hls.js* have been used, plus the *L2A-LL* and *LoLP* algorithms, which are explained in the following subsections.

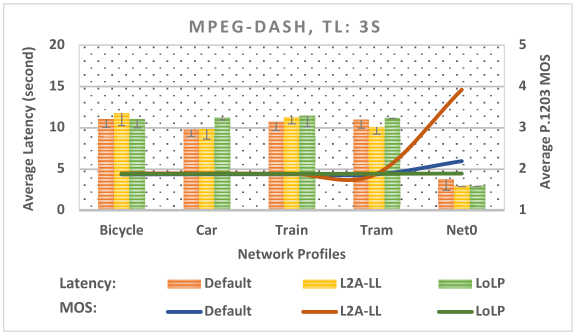
1) Learn2Adapt-LowLatency

According to the published work in [23], Learn2Adapt-LowLatency (L2A-LL) is a novel bitrate adaptation algorithm based on online learning and the Online Convex Optimization (OCO) theory that is a general framework for decision making which leverages convex optimization to allow for efficient algorithms, tailored for low-latency streaming. It performs well over a broad spectrum of network profiles in real experiments due to its design principle: its ability to learn. It does so without requiring parameter tuning, modifications according to application type, statistical assumptions for the channel, or throughput estimation. The robustness property of L2A-LL allows it to be classified in the small set of bitrate adaptation algorithms that mitigate the main

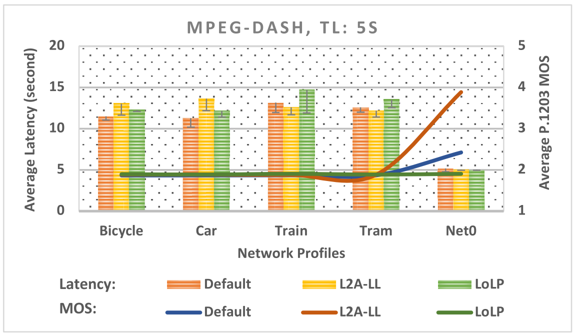
[8]https://ffmpeg.org, accessed Jan. 5, 2023



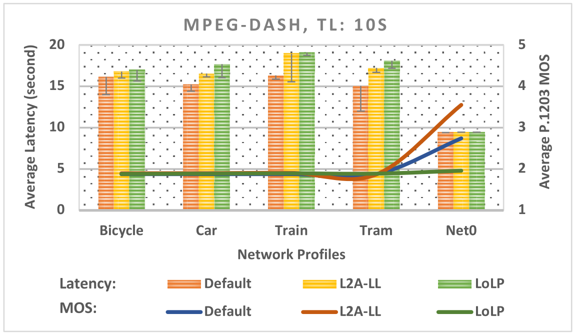
(a) Comparison with a target latency of 1 second.



(b) Comparison with a target latency of 3 seconds.

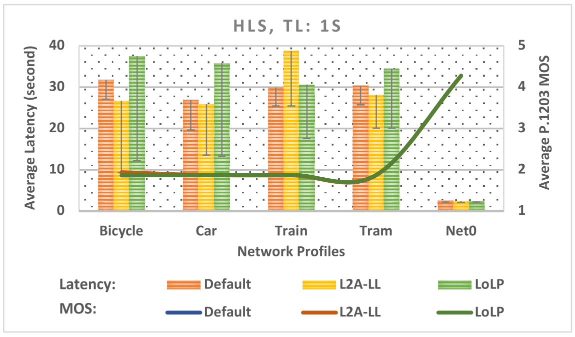


(c) Comparison with a target latency of 5 seconds.

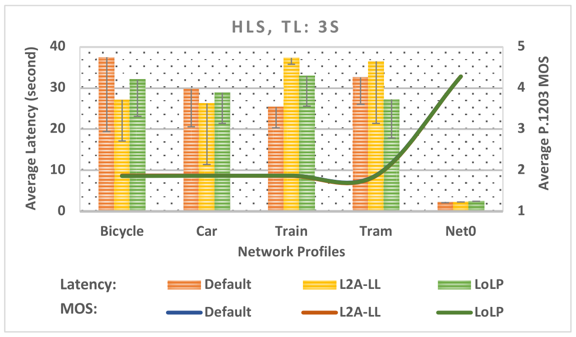


(d) Comparison with a target latency of 10 seconds.

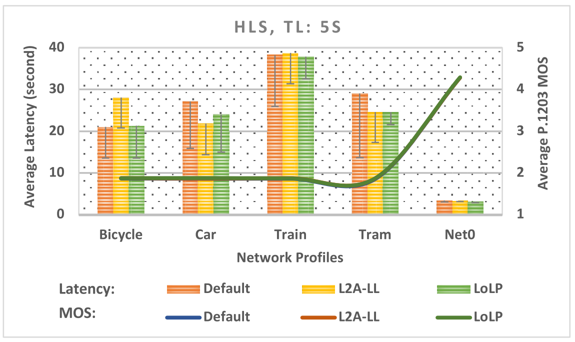
FIGURE 6. Average latency and predicted MOS comparison of three ABR algorithms implemented on dash.js media player with four given target latencies and five network profiles (Note that average latency range is from 0 to 20 seconds).



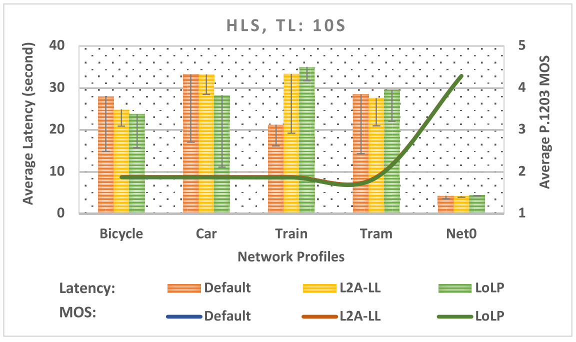
(a) Comparison with a target latency of 1 second.



(b) Comparison with a target latency of 3 seconds.



(c) Comparison with a target latency of 5 seconds.



(d) Comparison with a target latency of 10 seconds.

FIGURE 7. Average latency and predicted MOS comparison of three ABR algorithms implemented on hls.js media player with four given target latencies and five network profiles (Note that average latency range is from 0 to 40 seconds).

TABLE 2. LLL-CAdViSE raw results of low-latency live streaming with MPEG-DASH. Each row represents average values for three experiments.

Experiment[a]	Stall[b]	StartUp[c]	Seek[d]	Switch[e]	Bitrate[f]	Latency[g]	Playback Rate[h]	MOS[i]
ldash-def-bike-1s	143.30	8.69	18.90	0	100k-100k-100k	1.18-31.26-8.40	1-1.05-1.04	1.87
ldash-def-car-1s	147.42	11.03	12.63	0	100k-100k-100k	1.76-36.78-9.11	1-1.05-1.04	1.87
ldash-def-train-1s	163.95	10	14.68	0	100k-100k-100k	1.64-33.90-8.84	1-1.05-1.04	1.87
ldash-def-tram-1s	174.15	8.49	16.48	0	100k-100k-100k	1.66-24.89-8.53	1-1.05-1.04	1.87
ldash-def-net0-1s	16.41	4.61	0.04	0	100k-100k-100k	0.94-3.07-1.86	1-1.05-1.04	1.87
ldash-l2a-bike-1s	186.19	8.99	15.41	22.33	100k-750k-350k	1.32-26.84-8.21	1-1.05-1.03	1.93
ldash-l2a-car-1s	206.51	12.71	27.97	23	100k-1000k-402k	1.09-36.54-10.42	1-1.05-1.02	1.95
ldash-l2a-train-1s	213.87	13.29	26.5	22.66	100k-750k-298k	1.61-29.19-9.41	1-1.05-1.03	1.9
ldash-l2a-tram-1s	200.88	11.10	36.22	19.66	100k-750k-336k	1.81-25.26-9.09	1-1.05-1.0	1.91
ldash-l2a-net0-1s	15.69	4.71	0.06	59.33	316k-5800k-1339k	0.94-3.05-**1.8**	1-1.05-1.03	**2.5**
ldash-lolp-bike-1s	146.96	12.23	17.01	1	100k-750k-104k	0.92-28.28-8.10	1-1.05-1.036	1.88
ldash-lolp-car-1s	125.55	9.31	10.22	1	100k-750k-104k	1.49-24.12-7.60	1-1.05-1.04	1.88
ldash-lolp-train-1s	170.87	10.36	28.44	1	100k-750k-105k	1.66-21.53-7.61	1-1.05-1.04	1.88
ldash-lolp-tram-1s	133.03	9.52	19.35	1	100-750-104k	1.35-24.91-7.90	1-1.05-1.03	1.88
ldash-lolp-net0-1s	18.48	4.64	0.06	1	100k-750k-103k	0.98-4.2-2.08	1-1.05-1.04	1.88
ldash-def-bike-3s	131.35	8.88	12.22	20.66	100k-500k-172k	3.41-36.56-11.02	1-1.05-1.04	1.87
ldash-def-car-3s	120.17	7.03	11.09	21	100k-500k-146k	3.36-30.19-9.8	1-1.05-1.04	1.87
ldash-def-train-3s	139.37	12.88	13.5	17	100k-625k-145k	3.18-31.10-10.73	1-1.05-1.04	1.87
ldash-def-tram-3s	173.86	7.48	19.35	13.66	100k-500k-137k	3.14-30.40-10.94	1-1.05-1.03	1.87
ldash-def-net0-3s	5.25	4.74	0.03	10.66	100k-5800k-722k	2.93-5.86-3.77	0.99-1.03-1.01	2.19
ldash-l2a-bike-3s	176.47	9.77	17.12	28.33	100k-1000k-289k	3.45-37.38-11.76	1-1.05-1.03	1.89
ldash-l2a-car-3s	124.18	10.71	14.50	29.33	100k-1000k-279k	3.25-30.58-9.82	1-1.05-1.03	1.89
ldash-l2a-train-3s	158.35	10.75	17.96	30.33	100k-1000k-243k	3.48-30.49-11.29	1-1.05-1.04	1.88
ldash-l2a-tram-3s	148.83	15.35	22.22	22.33	100k-1000k-226k	3.28-29.68-10.11	1-1.05-1.036	1.88
ldash-l2a-net0-3s	0	4.75	0.07	206.33	750k-5800k-4384k	2.84-3.24-**2.94**	0.98-1.02-1	**3.92**
ldash-lolp-bike-3s	148.85	9.49	23.24	50	100k-1250k-238k	3.33-32.10-11.04	1-1.05-1.03	1.88
ldash-lolp-car-3s	165.32	9.32	26.38	57	100k-1000k-202k	3.40-37.34-11.25	1-1.05-1.03	1.88
ldash-lolp-train-3s	179.65	12.14	23.96	41	100k-1000k-209k	4.08-29.80-11.44	1-1.05-1.03	1.88
ldash-lolp-tram-3s	167.60	7.57	25.49	50.33	100k-1000k-212k	3.37-31.36-11.2	1-1.05-1.03	1.88
ldash-lolp-net0-3s	0	4.76	0.06	8	100k-5800k-256k	2.86-3.24-2.95	0.98-1.02-1	1.89
ldash-def-bike-5s	112.07	9.67	7.84	26	100k-625k-157k	5.36-33.83-11.48	1-1.05-1.04	1.87
ldash-def-car-5s	112.02	7.86	15.28	18	100k-625k-151k	5.15-30.37-11.28	0.99-1.05-1.03	1.87
ldash-def-train-5s	142.17	29.53	15.56	16.66	100k-500k-143k	5.48-32.84-13.12	1-1.05-1.03	1.87
ldash-def-tram-5s	143.76	7.36	16.73	15.33	100k-500k-132k	5.34-34.2-12.59	1-1.05-1.03	1.87
ldash-def-net0-5s	1.82	4.62	0.01	36.33	100k-5800k-1077k	4.95-6.44-5.17	0.99-1.03-1.00	2.42
ldash-l2a-bike-5s	162.68	16.05	31.16	26.33	100k-1000k-315k	5.46-40.57-13.06	1-1.05-1.03	1.89
ldash-l2a-car-5s	179.90	7.18	35.99	23.33	100k-1250k-261k	5.74-40.20-13.64	1-1.05-1.03	1.88
ldash-l2a-train-5s	150.61	14.05	18.31	24	100k-750k-206k	5.34-32.6-12.60	1-1.05-1.03	1.88
ldash-l2a-tram-5s	137.84	14.07	24.72	31.33	100k-1000k-237k	5.59-29.82-12.16	1-1.05-1.03	1.88
ldash-l2a-net0-5s	0	4.76	0.02	205.33	750k-5800k-4340k	4.94-5.92-5.05	0.98-1.02-1	**3.88**
ldash-lolp-bike-5s	137.45	20.21	17.72	58	100k-1250k-243k	5.67-32.14-12.28	1-1.05-1.02	1.89
ldash-lolp-car-5s	138.79	18.81	14.58	44	100k-750k-193k	5.43-31.78-12.14	0.99-1.05-1.03	1.88
ldash-lolp-train-5s	179.42	16.09	27.82	50.33	100k-1250k-234k	5.41-44.3-14.75	1-1.05-1.02	1.9
ldash-lolp-tram-5s	151.11	10.65	30.72	45	100k-1250k-246k	5.30-33.36-13.54	1-1.05-1.03	1.88
ldash-lolp-net0-5s	0	4.71	0.03	6	100k-5800k-321k	4.87-5.21-**4.99**	0.98-1.02-1	1.9
ldash-def-bike-10s	99.85	10.27	10.68	21.66	100k-625k-169k	10.04-30.98-16.15	0.98-1.05-1.04	1.87
ldash-def-car-10s	108.3	6.74	13.45	16	100k-625k-173k	8.10-31.97-15.25	0.95-1.05-1.03	1.87
ldash-def-train-10s	122.62	12.11	14.61	15.33	100k-625k-169k	9.19-34.36-16.32	0.98-1.05-1.04	1.87
ldash-def-tram-10s	93.6	8.99	8.53	14.33	100k-500k-134k	9.29-31.41-15.04	0.98-1.05-1.03	1.87
ldash-def-net0-10s	0	4.69	0	26	100k-4366k-1258k	5.41-10.09-**9.48**	0.95-1.01-0.99	2.74
ldash-l2a-bike-10s	133.24	10.6	10.33	32.33	100k-1000k-273k	9.39-35.38-16.79	0.98-1.05-1.03	1.89
ldash-l2a-car-10s	124.87	7.26	10.77	38.33	100k-750k-243k	8.72-30.99-16.54	0.96-1.05-1.02	1.89
ldash-l2a-train-10s	164.59	15.73	24.32	28.33	100k-750k-252k	10.33-47.78-18.97	1-1.05-1.03	1.90
ldash-l2a-tram-10s	142.22	14.67	28.97	26	100k-750k-232k	8.71-34.52-17.22	0.96-1.05-1.03	1.86
ldash-l2a-net0-10s	0	4.68	0	195.33	625k-5800k-3915k	5.41-10.09-**9.48**	0.95-1.01-0.99	**3.55**
ldash-lolp-bike-10s	107.37	12.31	12.55	72	100k-1000k-253k	9.64-33.10-16.99	0.98-1.05-1.03	1.88
ldash-lolp-car-10s	145.45	14.50	15.04	56.66	100k-2000k-234k	10.97-33.66-17.67	1-1.05-1.03	1.88
ldash-lolp-train-10s	167.35	13.09	25.21	42.33	100k-1500k-227k	10.92-47.04-19.1	1-1.05-1.03	1.89
ldash-lolp-tram-10s	138.82	11.11	17.8	40.66	100k-1000k-220k	10.32-47.05-18.08	0.99-1.05-1.02	1.88
ldash:lolp:net0:10s	0	4.71	0	18.66	100k-5800k-455k	5.42-10.09-9.49	0.95-1.01-0.99	1.95

[*] All time values are in seconds.
[a] Experiment title, format: "[streaming protocol]-[ABR algorithm]-[network profile]-[target latency]" (def: Default, l2a: L2A-LL).
[b] Average of the sum of stall events duration.
[c] Average start-up delay.
[d] Average of the sum of seek events duration.
[e] Average quantity of quality switches.
[f] Playback bitrate (min-max-avg) in kbps.
[g] Latency (min-max-avg).
[h] Playback rate (min-max-avg).
[i] Average MOS predicted by the ITU-T P.1203 quality model.

TABLE 3. Origin server encoder and packager configuration parameters in the experimental setup.

Parameter	Value	Parameter	Value
Video Encoder	libx264	GOP Size	48
Audio Encoder	aac	Video Preset	Faster
Tune	zerolatency	Video Profile	Main
Segment Duration	2 seconds	fflags	genpts
Fragment Duration	1 second	mov Flag	cmaf
Update Period±	30 seconds	Pixel Format	yuv420p
Video Buffer	$Bitrate \times 2 \div 3$	Write PRFT	True
Segment Type	mp4	FPS	24
HTTP Method	PUT	Min. Rate-	0.95
HTTP Option	Chunked Post	Max. Rate+	1.05

Click for further adjustable parameters.
± Manifest update period time.
- Minimum playback rate.
+ Maximum playback rate.

limitation of existing HAS approaches: the dependence on statistical models for the unknowns. L2A-LL is lightweight while concurrently facilitating easy adjustments to consider other streaming applications and use cases.

2) LOW-ON-LATENCY PLUS

The Low-on-Latency or, in short, LoL ABR algorithm [24] was a significant step forward in multi-bitrate low-latency live streaming. Low-on-Latency Plus (LoLP) [25] is the enhanced version of the LoL ABR algorithm, which tries to implement improvements in three main areas: LoL used hard-coded parameters computed from an offline training process in the rate adaptation algorithm, and this was seen as a significant barrier in LoL's wide deployment. Second, LoL's objective was to maximize a collective QoE function. Yet, specific use cases require singular QoE, which had to be accommodated. Third, the adaptive playback speed control failed to produce satisfying results in some scenarios. LoLP is designed explicitly for LLL streaming and delivers good QoE for any TL.

D. NETWORK PROFILES

We have selected four real LTE network traces [26], plus a static network profile without limitations on the available bandwidth, for our experimental setup. The selection covers various network trace scenarios, *i.e.*, (*i*) a commuter by train, (*ii*) a biker, (*iii*) a commuter by tram, (*iv*) a car driver, and (*v*) the static network profile called network 0 (net0). All the network profiles apply a network latency of 80 milliseconds, which is an average and typical latency in modern networks [20]. We have trimmed the network profiles to 420 seconds. The shape of the network profiles are depicted in Figures 5a, 5b, 5c, and 5d. We have omitted network 0 from the figures since it adds no limitation over the available bandwidth between the network nodes (*i.e.*, up to 10Gpbs as described in IV-A).

V. RESULTS AND FINDINGS

To demonstrate the power and potential of evaluating a low-latency live streaming session with our testbed, LLL-CAdViSE, we have executed different experiments described in Section IV. Here we will present the essential findings and the results from the experimental low-latency live streaming evaluations. The obtained results with minor processing and in raw format can be found in Table 2 and Table 4, respectively.

The evaluated media players have taken different approaches to catch up with the live edge. As presented in Tables 2 and 4, the playback rate for the hls.js player always remains one but with a higher overall seek time whereas the dash.js player has played the content sometimes up to the max allowed playback rate and in few cases slowed down the playback speed. None of the approaches has been dictated to the players as a catch-up logic.

As seen in Figures 6 and 7, MPEG-DASH performs better than HLS in providing low-latency in live streaming. For instance, with a target latency of one second shown in Figures 6a, and 7a, MPEG-DASH streaming has the highest average latency of 10.42 with the L2A-LL ABR algorithm and in the Car network profile. At the same time, HLS can reach an average latency of 38.81 with the L2A-LL ABR algorithm and in the Train network profile. This pattern can also be seen with other defined target latencies. In general, MPEG-DASH outperforms HLS in low-latency live streaming.

A. MPEG-DASH

The comparison between different low-latency ABR algorithms in terms of achieved latency and the predicted MOS with the MPEG-DASH standard are shown in Figures 6a, 6b, 6c, and 6d. We executed experiments, stored the logs of significant metrics in the database, and plotted these diagrams to better understand the ABR algorithms' performance. The blue, red, and green lines represent the average predicted MOS, and the orange, yellow, and green bars represent the average latency. The error bars represent the standard deviation values. As standard deviation is always a positive integer we have illustrated them in one direction to save space on the diagrams. We have marked (bold) the best performance (closest average latency to the TL and highest QoE score) of ABR algorithms for each TL in Table 2. It has been discovered that when a target latency is set for media players, they tend to sacrifice the QoE in favor of achieving the closest value to the target latency, which might not be the intention in all cases and could be better configured to serve the exact need.

According to the experiments and the shown data, when the network profiles are challenging with lots of fluctuations, *e.g.*, a car driver network profile (Figure 5b), all algorithms, *i.e.*, the player default algorithm, L2A-LL, and LoLP are set to prefer providing low quality but deliver better latency. It can be seen that when the network profile allows, *e.g.*, in network 0, the MOS values increase while the latency remains in the target latency range.

It is also observed that in all experimental scenarios, the LoLP algorithm, unlike the other two algorithms, provides very low QoE (green line in Figures 6a, 6b, 6c, and 6d), and

TABLE 4. LLL-CAdViSE raw results of low-latency live streaming with HLS. Each row represents average values for three experiments.

Experiment[a]	Stall[b]	StartUp[c]	Seek[d]	Switch[e]	Bitrate[f]	Latency[g]	Playback Rate[h]	MOS[i]
lhls-def-bike-1s	85.32	1.20	38.11	0.66	100k-191k-102k	2.40-54.45-31.81	1-1-1	1.86
lhls-def-car-1s	91.73	0.41	23.78	2.66	100k-500k-104k	2.25-46.26-26.91	1-1-1	1.87
lhls-def-train-1s	90.42	0.67	25.85	2.66	100k-533k-110k	2.97-46.34-29.94	1-1-1	1.87
lhls-def-tram-1s	84.44	1.35	20.83	2	100k-375k-115k	2.75-52.96-30.43	1-1-1	1.87
lhls-def-net0-1s	58.99	0.05	65.45	3	100k-5800k-5723k	1.85-3.55-2.48	1-1-1	4.28
lhls-l2a-bike-1s	65.75	0.39	12.2	2	100k-500k-206k	3.33-39.62-26.72	1-1-1	1.94
lhls-l2a-car-1s	106.61	0.6	41.64	3	100k-500k-106k	2.4-46.39-25.92	1-1-1	1.87
lhls-l2a-train-1s	84.84	0.45	18.40	0	100k-100k-100k	2.63-59.32-38.81	1-1-1	1.86
lhls-l2a-tram-1s	108.22	0.48	34.98	1.33	100k-283k-107k	2.31-46.53-28.12	1-1-1	1.87
lhls-l2a-net0-1s	56.95	0.05	66.06	3	100k-5800k-5723k	1.42-3.52-2.44	1-1-1	4.28
lhls-lolp-bike-1s	128.19	0.43	47.04	2.66	100k-500k-116k	2.31-66.67-37.57	1-1-1	1.86
lhls-lolp-car-1s	84.86	0.39	14.23	1.66	100k-408k-106k	2.65-52.47-35.82	1-1-1	1.86
lhls-lolp-train-1s	102.18	0.24	23.60	2.66	100k-500k-105k	3.13-59.59-30.66	1-1-1	1.87
lhls-lolp-tram-1s	79.9	0.31	19.79	2.33	100k-533k-106k	2.09-51.66-34.52	1-1-1	1.87
lhls-lolp-net0-1s	52.08	0.05	53.09	3	100k-5800k-5724k	1.75-3.58-**2.4**	1-1-1	**4.28**
lhls-def-bike-3s	96.31	1.06	23.56	0.66	100k-191k-101k	2.39-57.42-37.47	1-1-1	1.87
lhls-def-car-3s	82.89	0.36	25.18	1.33	100k-283k-102k	2.62-43.95-29.82	1-1-1	1.87
lhls-def-train-3s	81.53	0.83	15.62	1.33	100k-283k-106k	2.45-39.75-25.42	1-1-1	1.87
lhls-def-tram-3s	122.95	0.71	29.47	1.66	100k-408k-104k	2.46-64.59-32.6	1-1-1	1.88
lhls-def-net0-3s	50.69	0.05	7.64	3	100k-5800k-5724k	1.9-4.32-2.30	1-1-1	4.28
lhls-l2a-bike-3s	58.45	0.43	9.86	2.66	100k-500k-108k	2.37-37.16-27.12	1-1-1	1.87
lhls-l2a-car-3s	92.31	0.57	34.96	2.66	100k-500k-113k	2.32-39.95-26.26	1-1-1	1.87
lhls-l2a-train-3s	69.78	0.75	5.41	0.66	100k-191k-100k	2.66-54.47-37.36	1-1-1	1.86
lhls-l2a-tram-3s	97.91	0.76	18.54	2	100k-375k-107k	4.85-56.30-36.5	1-1-1	1.86
lhls-l2a-net0-3s	70.48	0.05	61.28	3	100k-5800k-5720k	1.49-4.56-2.4	1-1-1	4.28
lhls-lolp-bike-3s	99.2	0.37	25.21	2	100k-408k-113k	3.02-52.25-32.08	1-1-1	1.86
lhls-lolp-car-3s	85.25	1.22	15.93	2	100k-375k-106k	2.38-48.31-28.9	1-1-1	1.86
lhls-lolp-train-3s	95.35	1.13	12.59	1.33	100k-283k-111k	2.71-52.26-32.97	1-1-1	1.87
lhls-lolp-tram-3s	97.98	0.47	32.94	2.33	100k-408k-115k	2.44-46.95-27.25	1-1-1	1.87
lhls-lolp-net0-3s	70.61	0.05	57.71	3	100k-5800k-5720k	1.47-4.69-**2.54**	1-1-1	**4.28**
lhls-def-bike-5s	107.67	2.59	13.46	0.66	100k-191k-116k	1.91-40.31-21.02	1-1-1	1.87
lhls-def-car-5s	106.91	0.23	13.85	1.33	100k-283k-101k	2.19-52.86-27.09	1-1-1	1.86
lhls-def-train-5s	81.69	0.73	10.10	0.66	100k-191k-104k	2.89-55.35-38.22	1-1-1	1.86
lhls-def-tram-5s	60.59	0.87	2.89	0	100k-100k-100k	2.47-42.50-29.01	1-1-1	1.86
lhls-def-net0-5s	25.21	0.053	6.01	3	100k-5800k-5729k	1.65-5.18-**3.41**	1-1-1	4.28
lhls-l2a-bike-5s	78.38	0.61	26.59	3	100k-625k-108k	2.67-48.24-28.03	1-1-1	1.87
lhls-l2a-car-5s	92.44	0.29	17.40	1.33	100k-283k-103k	2.31-42.53-21.88	1-1-1	1.86
lhls-l2a-train-5s	61.49	0.63	7.61	2	100k-375k-110k	2.83-49.45-38.7	1-1-1	1.87
lhls-l2a-tram-5s	89.36	0.37	12.57	2.33	100k-408k-118k	2.31-44.04-24.56	1-1-1	1.87
lhls-l2a-net0-5s	15.89	0.05	3.58	3	100k-5800k-5731k	1.87-5.39-3.31	1-1-1	4.28
lhls-lolp-bike-5s	81.27	0.22	15.52	3	100k-500k-114k	2.56-37.15-21.29	1-1-1	1.87
lhls-lolp-car-5s	85.79	0.11	22.24	2.66	100k-533k-105k	2.56-43.26-24.01	1-1-1	1.87
lhls-lolp-train-5s	80.25	0.81	12.31	0.66	100k-191k-105k	2.49-57.37-37.81	1-1-1	1.87
lhls-lolp-tram-5s	92.68	0.65	26.81	2	100k-408k-104k	2.86-40.87-24.58	1-1-1	1.87
lhls-lolp-net0-5s	13.85	0.05	1.61	3	100k-5800k-5731k	1.67-5.09-3.12	1-1-1	**4.29**
lhls-def-bike-10s	62.99	0.86	5.95	1.33	100k-283k-111k	3.53-45.3-27.98	1-1-1	1.87
lhls-def-car-10s	79.20	1.22	6.08	2	100k-375k-108k	2.62-53.56-33.29	1-1-1	1.87
lhls-def-train-10s	92.54	0.96	26.03	0.66	100k-191k-101k	3.33-36.24-21.25	1-1-1	1.87
lhls-def-tram-10s	102.34	0.39	12.46	2	100k-375k-111k	2.96-57.47-28.47	1-1-1	1.87
lhls-def-net0-10s	7.82	0.05	0.02	3	100k-5800k-5732k	1.61-5.64-4.24	1-1-1	4.29
lhls-l2a-bike-10s	62.30	0.58	4.99	1.33	100k-316k-106k	2.65-42.11-24.87	1-1-1	1.87
lhls-l2a-car-10s	64.98	0.74	0.92	3	100k-500k-112k	3.65-49.72-33.23	1-1-1	1.87
lhls-l2a-train-10s	81.09	0.21	7.27	2.66	100k-500k-112k	2.64-55.31-33.37	1-1-1	1.87
lhls-l2a-tram-10s	98.72	0.19	9.04	2.66	100k-533k-107k	2.54-50.61-27.61	1-1-1	1.87
lhls-l2a-net0-10s	7.09	0.05	0.02	3	100k-5800k-5732k	1.53-5.58-4.21	1-1-1	4.29
lhls-lolp-bike-10s	105.74	0.17	15.56	2	100k-408k-106k	2.60-42.66-23.78	1-1-1	1.87
lhls-lolp-car-10s	88.45	0.39	7.24	3	100k-500k-113k	2.26-48.26-28.26	1-1-1	1.87
lhls-lolp-train-10s	88.57	0.88	4.74	0.66	100k-191k-101k	3.53-57.62-35.04	1-1-1	1.86
lhls-lolp-tram-10s	82.23	0.45	6.86	2.33	100k-408k-112k	2.78-53.86-29.58	1-1-1	1.86
lhls-lolp-net0-10s	7.14	0.05	0.02	3	100k-5800k-5732k	1.55-5.73-**4.41**	1-1-1	**4.29**

[*] All time values are in seconds.
[a] Experiment title, format: "[streaming protocol]-[ABR algorithm]-[network profile]-[target latency]" (def: Default, l2a: L2A-LL).
[b] Average of the sum of stall events duration.
[c] Average start-up delay.
[d] Average of the sum of seek events duration.
[e] Average quantity of quality switches.
[f] Playback bitrate (min-max-avg) in kbps.
[g] Latency (min-max-avg).
[h] Playback rate (min-max-avg).
[i] Average MOS predicted by the ITU-T P.1203 quality model.

even when the network profile does not limit the available bandwidth, *i.e.*, in network 0.

The L2A-LL algorithm assessment results, presented with bold font in Table 2 for the network 0 profile and the TL of three seconds, show better performance in delivering expected latency and providing the highest QoE score (2.94 seconds and 3.92 MOS), and this is also the case for almost all other challenging network profiles for this ABR algorithm in MPEG-DASH low-latency live streaming.

Unlike hls.js, the dash.js media player and the three examined ABR algorithms for live streaming keep the average latency very close to the defined target latency in almost all scenarios, as shown in Figure 6.

Further investigation is advised, *e.g.*, by running more experiments with each algorithm and with different configurations (which was not the intention of our designed experiments, but rather the comparison between the ABR algorithms) and comparing retrieved metrics for a specific algorithm in order to find out what is the root cause of any poor decision. Additionally, other metrics, *e.g.*, stall events' duration or the number of quality switches, shown in Table 2, can give better insights into where the ABR algorithms could be improved.

B. HLS

The results and comparisons of achieved latency and the predicted MOS between the ABR algorithms with HLS streaming are shown in Figures 7a, 7b, 7c, and 7d. The blue, red, and green lines represent the average predicted MOS, and the orange, yellow, and green bars represent the average latency. The error bars represent the standard deviation values. As standard deviation is always a positive integer we have illustrated them in one direction to save space on the diagrams. We have marked (bold) in Table 4 the best performance (closest average latency to the TL and highest QoE score) of ABR algorithms for each TL.

An immediate observation is that the defined target latency (set by *abrController.targetLatency*) is not honored by hls.js and the plugged algorithms. In some cases, such as shown in Figure 7c, the algorithms have more difficulties with the Train commuter network profile (Figure 5c).

The average latencies for HLS live streaming proved higher when the TL is set to be one second and with the network 0 profile. On the other hand, when the TL is set to ten seconds, the achieved latency with network 0 is less than ten. This indicates that hls.js with our simple configuration (refer to Section IV) has other priority parameters and factors rather than being dictated by a defined TL.

As seen in Table 4, the MOS values for all ABR algorithms and network profiles except for the network 0 profile in HLS streaming remain very low. This leaves room for major improvement, specifically when the network profiles are more challenging.

The LoLP algorithm assessment results in HLS low-latency live streaming, presented with bold font in Table 4 for the network 0 profile and the TL of three seconds, show better performance in delivering the expected latency and providing the highest QoE score (2.54 seconds and 4.28 MOS) in comparison with other ABR algorithms and with other network profiles and defined TLs.

VI. CONCLUSION

As the main contribution, this paper introduces our sophisticated cloud-based and open-source testbed, LLL-CAdViSE, a framework for evaluating HAS from different perspectives. LLL-CAdViSE provides multiple functionalities, *i.e.*, evaluations of live media streaming significant metrics such as stall events and quality switches, precise measurement of media streaming E2E latency, assessment of objective QoE, and helps with preparation of a single media file to further investigate the possible defects in the experimental live streaming session. As a second contribution, we extensively tested well-known media players and ABR algorithms using LLL-CAdViSE. The results show that the L2A-LL ABR algorithm plugged into the dash.js media player and using MPEG-DASH low-latency live streaming outperforms other ABR algorithms in providing the closest latency to a target latency and maintaining a high QoE score. Our testbed is publicly available on GitHub with the following link and can be used for the evaluation of different live media streaming scenarios, media players, and ABR algorithms:

https://github.com/cd-athena/LLL-CAdViSE

REFERENCES

[1] Ericsson. (2022). *Ericsson Mobility Report*. [Online]. Available: https://www.ericsson.com/4ae28d/assets/local/reports-papers/mobility-report/documents/2022/ericsson-mobility-report-november-2022.pdf

[2] C. Mueller. (2018). *Low Latency Streaming: What is It and How Can It be Solved?* [Online]. Available: https://bitmovin.com/cmaf-low-latency-streaming/

[3] (IETF). (2022). *HTTP/1.1*. [Online]. Available: https://httpwg.org/specs/rfc9112.html

[4] *Information Technology—Dynamic Adaptive Streaming Over HTTP (DASH)*, Standard 23000-19:2020, 2020. [Online]. Available: https://www.iso.org/standard/79106.html

[5] *Information Technology—Dynamic Adaptive Streaming Over HTTP (DASH)*, Standard 23009:2022, 2022. [Online]. Available: https://www.iso.org/standard/65274.html

[6] R. Pantos and W. May. (2017). *HTTP Live Streaming*. [Online]. Available: https://www.rfc-editor.org/info/rfc8216

[7] *Parametric Bitstream-Based Quality Assessment of Progressive Download and Adaptive Audiovisual Streaming Services Over Reliable Transport—Video Quality Estimation Module*, Standard 1203. [Online]. Available: http://handle.itu.int/11.1002/ps/P1203-01

[8] A. Zabrovskiy, E. Kuzmin, E. Petrov, C. Timmerer, and C. Mueller, "AdViSE: Adaptive video streaming evaluation framework for the automated testing of media players," in *Proc. 8th ACM Multimedia Syst. Conf.* New York, NY, USA: Association for Computing Machinery, Jun. 2017, pp. 217–220, doi: 10.1145/3083187.3083221.

[9] B. Taraghi, A. Zabrovskiy, C. Timmerer, and H. Hellwagner, "CAdViSE: Cloud-based adaptive video streaming evaluation framework for the automated testing of media players," in *Proc. 11th ACM Multimedia Syst. Conf.*, May 2020, pp. 349–352, doi: 10.1145/3339825.3393581.

[10] J. Aguilar-Armijo, B. Taraghi, C. Timmerer, and H. Hellwagner, "Dynamic segment repackaging at the edge for HTTP adaptive streaming," in *Proc. IEEE Int. Symp. Multimedia (ISM)*, Dec. 2020, pp. 17–24, doi: 10.1109/ISM.2020.00009.

[11] M. Nguyen, B. Taraghi, A. Bentaleb, R. Zimmermann, and C. Timmerer, "CADLAD: Device-aware bitrate ladder construction for HTTP adaptive streaming," in *Proc. 18th Int. Conf. Netw. Service Manag. (CNSM)*, Oct. 2022, pp. 198–204, doi: 10.23919/CNSM55787.2022.9964669.

[12] P. Pegus, E. Cecchet, and P. Shenoy, "Video BenchLab: An open platform for realistic benchmarking of streaming media workloads," in *Proc. 6th ACM Multimedia Syst. Conf.* New York, NY, USA: Association for Computing Machinery, Mar. 2015, pp. 165–176, doi: 10.1145/2713168.2723145.

[13] R. Ramos-Chavez, R. Mekuria, T. Karagkioules, D. Griffioen, A. Wagenaar, and M. Ogle, "MPEG NBMP testbed for evaluation of real-time distributed media processing workflows at scale," in *Proc. 12th ACM Multimedia Syst. Conf.* New York, NY, USA: Association for Computing Machinery, Jul. 2021, pp. 173–185, doi: 10.1145/3458305.3463380.

[14] *Information Technology—-Coded Representation of Immersive Media—-Part 8: Network Based Media Processing*, Standard 23090-8:2020, 2020. [Online]. Available: https://www.iso.org/standard/77839.html

[15] D. Stohr, A. Frömmgen, A. Rizk, M. Zink, R. Steinmetz, and W. Effelsberg, "Where are the sweet spots? A systematic approach to reproducible DASH player comparisons," in *Proc. 25th ACM Int. Conf. Multimedia*. New York, NY, USA: Association for Computing Machinery, Oct. 2017, pp. 1113–1121, doi: 10.1145/3123266.3123426.

[16] P. K. Yadav, A. Bentaleb, M. Lim, J. Huang, W. T. Ooi, and R. Zimmermann, "Playing chunk-transferred DASH segments at low latency with QLive," in *Proc. 12th ACM Multimedia Syst. Conf.* New York, NY, USA: Association for Computing Machinery, Jul. 2021, pp. 51–64, doi: 10.1145/3458305.3463376.

[17] M. Taha, J. Lloret, A. Ali, and L. Garcia, "Adaptive video streaming testbed design for performance study and assessment of QoE," *Int. J. Commun. Syst.*, vol. 31, no. 9, p. e3551, Jun. 2018, doi: 10.1002/dac.3551.

[18] M. Abdullah, "A novel CDN testbed for fast deploying HTTP adaptive video streaming," in *Proc. 9th EAI Int. Conf. Mobile Multimedia Commun.*, 2016, pp. 65–71, doi: 10.4108/eai.18-6-2016.2264163.

[19] B. Taraghi, M. Nguyen, H. Amirpour, and C. Timmerer, "Intense: In-depth studies on stall events and quality switches and their impact on the quality of experience in HTTP adaptive streaming," *IEEE Access*, vol. 9, pp. 118087–118098, 2021, doi: 10.1109/ACCESS.2021.3107619.

[20] B. Taraghi, A. Bentaleb, C. Timmerer, R. Zimmermann, and H. Hellwagner, "Understanding quality of experience of heuristic-based HTTP adaptive bitrate algorithms," in *Proc. 31st ACM Workshop Netw. Operating Syst. Support Digit. Audio Video*, Jul. 2021, pp. 82–89, doi: 10.1145/3458306.3458875.

[21] B. Taraghi, H. Amirpour, and C. Timmerer, "Multi-codec ultra high definition 8K MPEG-DASH dataset," in *Proc. 13th ACM Multimedia Syst. Conf.* New York, NY, USA: Association for Computing Machinery, Jun. 2022, pp. 216–220, doi: 10.1145/3524273.3532889.

[22] Bitmovin. (2022). *The 6th Annual Bitmovin Video Developer Report*. [Online]. Available: https://bitmovin.com/wp-content/uploads/2022/12/bitmovin-6th-video-developer-report-2022-2023.pdf

[23] T. Karagkioules, R. Mekuria, D. Griffioen, and A. Wagenaar, "Online learning for low-latency adaptive streaming," in *Proc. 11th ACM Multimedia Syst. Conf.* New York, NY, USA: Association for Computing Machinery, May 2020, pp. 315–320, doi: 10.1145/3339825.3397042.

[24] M. Lim, M. N. Akcay, A. Bentaleb, A. C. Begen, and R. Zimmermann, "When they go high, we go low: Low-latency live streaming in dash.js with LoL," in *Proc. 11th ACM Multimedia Syst. Conf.* New York, NY, USA: Association for Computing Machinery, May 2020, pp. 321–326, doi: 10.1145/3339825.3397043.

[25] A. Bentaleb, M. N. Akcay, M. Lim, A. C. Begen, and R. Zimmermann, "Catching the moment with LoL$^+$ in twitch-like low-latency live streaming platforms," *IEEE Trans. Multimedia*, vol. 24, pp. 2300–2314, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9429986

[26] J. Van Der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck, "HTTP/2-based adaptive streaming of HEVC video over 4G/LTE networks," *IEEE Commun. Lett.*, vol. 20, no. 11, pp. 2177–2180, Nov. 2016, doi: 10.1109/LCOMM.2016.2601087.

BABAK TARAGHI (Member, IEEE) received the bachelor's degree in information technology, in 2015, and the M.S. degree in software engineering from the University Technology Malaysia, in 2017. He is currently pursuing the Ph.D. degree with the Institute of Information Technology (ITEC), Alpen-Adria-Universität Klagenfurt (AAU), with the ATHENA Project with a focus on adaptive video streaming. He possesses a strong background in software development engineering with many years of professional experience in software solution design and construction. His research interests include multimedia communication, streaming, adaptation, and the quality of experience. Further information is available at https://tiny.one/tbabak

HERMANN HELLWAGNER (Senior Member, IEEE) is currently a Full Professor of computer science with Alpen-Adria-Universität Klagenfurt (AAU), where he leads the Research Group Multimedia Communication (MMC), Institute of Information Technology (ITEC). Earlier, he held a position as an Associate Professor with the Technical University of Munich (TUM) and a Senior Researcher with Siemens Corporate Research, Munich. He has published widely on parallel computer architecture, parallel programming, and multimedia communication and adaptation. His current research interests include distributed multimedia systems, multimedia communication and adaptation, QoS/QoE, and communication in multi-UAV networks. He was a member of the Scientific Board of the Austrian Science Fund (FWF) from 2005 to 2016 and the FWF Vice President from 2013 to 2016. He is currently a Senior Member of the ACM. He is also a member of the CD Senate of Christian Doppler Forschungsgesellschaft (CDG). Further information is available at https://www.itec.aau.at/hellwagn/

CHRISTIAN TIMMERER (Senior Member, IEEE) is currently a Full Professor of computer science with the Institute of Information Technology (ITEC), Alpen-Adria-Universität Klagenfurt (AAU), and also the Director of the Christian Doppler (CD) Laboratory ATHENA. His research interests include multimedia systems, immersive multimedia communication, streaming, adaptation, and the quality of experience, where he has coauthored seven patents and more than 300 articles. He was the General Chair of WIAMIS 2008, QoMEX 2013, MMSys 2016, and PV 2018, and has participated in several EC-funded projects, notably DANAE, ENTHRONE, P2P-Next, ALICANTE, SocialSensor, COST IC1003 QUALINET, ICoSOLE, and SPIRIT. He also participated in ISO/MPEG work for several years, notably in the area of MPEG-21, MPEG-M, MPEG-V, and MPEG-DASH, where he also served as a Standard Editor. In 2012, he has cofounded Bitmovin to provide professional services around MPEG-DASH, where he holds the position of the Chief Innovation Officer (CIO)—the Head of Research and Standardization. Further information is available at http://timmerer.com