Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

A DASH server-side delay-based representation switching solution to improve the quality of experience for low-latency live video streaming

Román Belda [a], Pau Arce [a], Juan Carlos Guerri [a,*], Ismael de Fez [b]

[a] Institute of Telecommunications and Multimedia Applications, Universitat Politècnica de València, Spain
[b] Universidad Internacional de Valencia, Spain

ARTICLE INFO

Keywords:
DASH low latency
Live streaming
Testbed
Multimedia software open source
Performance evaluation

ABSTRACT

This work addresses the integration of real-time transmission systems, including IP cameras and production systems (like OBS or vMix), that use protocols such as RTSP (Real Time Streaming Protocol) or SRT (Secure Reliable Transport), with content distribution technology based on LL-DASH (Low Latency DASH -Dynamic Adaptive Streaming over HTTP-), taking advantage of the fact that DASH offers significant well-known advantages for content distribution over the Internet and via CDNs (Content Delivery Networks). Considering the limitations of the LL-DASH standard regarding the adaptation to network conditions, this paper proposes a new solution called Server-Side Representation Switching (SSRS). SSRS uses an approach based on the server measuring the delay in the requests made by clients, whose variation may be due to a decrease in bandwidth, as occurs in Wi-Fi networks with a high number of clients. To evaluate the effectiveness of the proposed solution, a testbed has been developed that allows the performance evaluation of both the LL-DASH system and the solution based on server-side decision-making. In addition, the developed solution has been compared with known algorithms (L2A and LoL+) integrated into the Dash.js player. The results show that the Server-Side Representation Switching solution offers a good trade-off between the transmitted quality and the final delay measured at the client, compared to the other algorithms evaluated. Moreover, it holds the advantage of being straightforward to implement and does not require any modifications to the players used.

• Networks • Network performance evaluation • Network performance analysis.

1. Introduction

It is well known that multimedia content distribution, and in particular video streaming, currently dominates global Internet traffic, and its importance will be even greater in the future. In this scenario, HTTP-based content distribution systems, such as the DASH (Dynamic Adaptive Streaming over HTTP) standard [1], revolutionized the way video content is streamed on VoD platforms. Its popularity can be attributed to its scalability, universality, and compatibility with network equipment [2]. DASH allows for the use of standard web servers for content distribution over HTTP (Hypertext Transfer Protocol), enabling the full benefits of CDNs (Content Delivery Network). Additionally, web technology can be played on browsers, and multimedia traffic is transparent to intermediate routers because of the widespread use of HTTP in Internet browsing.

Specifically, the main advantages of using DASH can be summarized in three main points: 1) Reduced management complexity; 2) Cost reduction; and 3) Improved scalability. Regarding the first advantage, by allowing HTTP video transmission using TCP ports 80 and 443 and eliminating the need to deploy and manage a separate caching infrastructure, the management complexity is highly reduced. In addition, in most corporate networks, some level of restriction at the protocol and port level are used to minimize the likelihood of attacks. However, ports 80 and 443 are almost always open for generic web traffic flow and therefore for HTTP video. Likewise, ports using other video streaming protocols such as RTMP, SRT, RTSP, etc. are not always open, hindering or blocking these protocols.

With regards to cost-reduction, non-HTTP transmission protocols increase the cost of the infrastructure as they require specific hardware and software in the server, forming a parallel infrastructure to the network of the rest of the services. In addition, inefficient content caching can increase the amount of bandwidth required to transmit popular videos over the network. However, HTTP technology leverages the existing HTTP server network, allowing organizations to save costs

* Corresponding author.
E-mail address: jcguerri@iteam.upv.es (J.C. Guerri).

https://doi.org/10.1016/j.comnet.2023.109961
Received 14 April 2023; Received in revised form 12 July 2023; Accepted 31 July 2023
Available online 9 August 2023

that would otherwise be spent on specialized hardware and software. And as access to video content increases, HTTP caching proxies dramatically reduce bandwidth costs over accessing uncached video. Finally, regarding improved scalability, the ubiquity of HTTP servers and the protocol's native support for perimeter caching make HTTP the ideal choice for streaming large-scale live events and on-demand content for very frequent access.

As more and more streaming platforms emerge, including those that offer live content (such as Youtube Live or Twitch, among others), low latency solutions are becoming increasingly necessary. Proposals for low-latency adaptive HTTP streaming, such as Low-Latency HLS (HTTP Live Streaming) [3] and Low-Latency DASH [4], are being developed to minimize the delay from video packetization to playback. However, other factors can contribute to added delays, such as encoding delay or acquisition delay through protocols like RTMP (Real Time Messaging Protocol), SRT (Secure Reliable Transport), or RTSP (Real-Time Streaming Protocol). These protocols are widely used in commercial IP (Internet Protocol) cameras and real-time content distribution systems like Teams, Zoom, IPTV, OBS, and vMix. While DASH relies on clients requesting video segments, LL-DASH pushes content while it is being generated, making it unsuitable for standard web servers [5]. Server and encoder implementations can be found in literature and online repositories [6], but there are no IP cameras that support DASH or LL-DASH at a commercial level, making the study and development in this field interesting.

Another key aspect to consider in the context of low-latency streaming is the Quality of Experience (QoE) of the viewers. In traditional VoD (Video on Demand) streaming, buffering and slow start times are common issues that can lead to a poor QoE. In live streaming, additional challenges arise due to the real-time nature of the content and the need for low latency. As such, it is important to develop strategies to minimize rebuffering and maximize QoE in low-latency streaming scenarios. This could include techniques such as predictive buffering, dynamic bit rate adaptation, and network-aware streaming algorithms.

Also, another process that has to be properly evaluated and configured in live streaming systems to achieve a trade-off between the delay introduced and the quality observed, is the transcoding process. This task will allow to properly adapt the input protocol (RTSP, SRT, …) and the video characteristics (codec, bitrate, resolution, …) to the DASH-based distribution systems.

In this regard, this paper presents a testbed for LL-DASH [7], aiming to explore the factors that contribute to end-to-end delay. The evaluation of the latency for different video origins and manifest parameters will help researchers and developers improve the performance of streaming platforms. This is particularly important in use cases that require interactivity, like e-learning, online conferences, and gaming. By minimizing delays, the user experience can be greatly enhanced, leading to increased engagement and improved satisfaction. As the world of streaming platforms continues to expand, the demand for live content and interactivity is increasing, which makes the development of low-latency solutions for HTTP streaming a primary concern. Furthermore, this work also proposes a server-based solution for Low Latency video streaming whose aim is to improve the Quality of Experience of the users.

The rest of the paper is organized as follows: Section 2 presents the state of the art related to LL-DASH streaming technologies; Section 3 presents the main factors and protocols related to latency, such as encoding, segmentation or network delivery; Section 4 explains the Low Latency solution proposed in this work; Section 5 introduces the proposed testbed, while Section 6 presents the evaluation carried out using the aforementioned testbed; Finally, Section 7 summarizes the main conclusions as well as the future work.

2. State of the art

Low-Latency DASH technology is based on a technique called ``chunked transfer encoding'', which is used in HTTP to send data as a series of chunks, rather than as a single response. Thus, the server splits the response into smaller chunks and sends them to the client, each preceded by a size indicator. This allows the client to begin processing the response before the server has finished sending it, resulting in faster and more efficient data transmission. The disadvantage of using this type of transfer, which allows segments to be generated and sent simultaneously, is that traditional adaptation algorithms do not correctly calculate the available bandwidth.

Therefore, there are different solutions in the literature dealing with this problem. For instance, in Bentaleb et al. [8] authors present the ACTE (Adaptive Streaming with Chunked Transfer Encoding) algorithm, which uses a sliding window and an online linear adaptive filter as an alternative to overcome this drawback. The same authors present in Bentaleb et al. [9] the Automated Model for Prediction (AMP) algorithm, which utilizes a set of bandwidth prediction models that are dynamically selected to optimize performance in low latency scenarios. Along the same lines, in Lyko et al. [10] it is presented a new algorithm called Llama (Low Latency Adaptive Media Algorithm). This algorithm uses two independent measures of throughput on different time scales: one to decide whether to decrease the video quality and one to decide about increasing the video quality. Other algorithms to highlight regarding low latency scenarios are L2A [11], LoL+ [12], and Stallion [13]. Precisely both the L2A and LoL+ algorithms are used in this work in order to carry a comparison regarding the proposed solution. For that reason, these algorithms will be further explained in Section 5.2.2.

Analyzing the most widely used DASH players, it is important to highlight Dash.js, the result of an initiative of the DASH Industry Forum. Aforementioned algorithms, such as L2A and LoL+, are integrated in this player. In [14] it is presented an implementation of an algorithm called SARA (Segment Aware Rate Adaptation), in which VBR (Variable Bitrate) coding is considered and therefore with a different size of generated segments. However, this algorithm is not oriented for low latency scenarios. On the contrary, Li et al. [15] presents a solution called Fleet, which has also been implemented in Dash.js and it is oriented for low-latency live video streaming. Fleet is based on a stochastic model predictive controller that considers networks conditions and client states in order to carry out bitrate adaptation. Specifically, Fleet uses a bandwidth measurement module, based on an HTTP chunk measurement, and a bitrate adaptation module which employs a practical evaluation model. In the same line, a very complete study regarding evaluation of Dash.js for low-latency scenarios can be found in Taraghi et al. [16], which presents a framework to assess live streaming. Although real scenarios are difficult to assess, the results presented in this paper are obtained using an actual development of an LL-DASH server and also using Dash.js at the client. However, network congestion, packet loss and bandwidth constraints are simulated in order to evaluate the performance of the proposed approach in specific network conditions. In this regard, Arunruangsirilert et al. [17] presents a performance evaluation of low-latency live streaming using MPEG-DASH over a commercial 5 G network in Thailand.

Regarding live streaming, one of the most used protocols nowadays is SRT, which appeared in order to overcome different limitations of broadcast communications. In [18] authors implement an adaptive bitrate algorithm for the SRT protocol. In this way, they manage to include one of the advantages of DASH technology, which is the adaptation of the bitrate depending on the state of congestion. The same authors focus the study in Viola et al. [19] considering the need to use the ``chunked transfer encoding'' transmission proposed in CMAF (Common Media Application Format) to reduce the delay in 5G network applications. A similar solution is proposed in Li et al. [20], where it is presented a design a fuzzy logic controller based ABR for low latency live video streaming with Chunked Transfer Encoding (CTE). In order to make bitrate decisions, the algorithm takes into consideration player buffer size, throughput mean and throughput standard deviation.

From the point of view of end-to-end delay, in Tashtarian et al. [21]

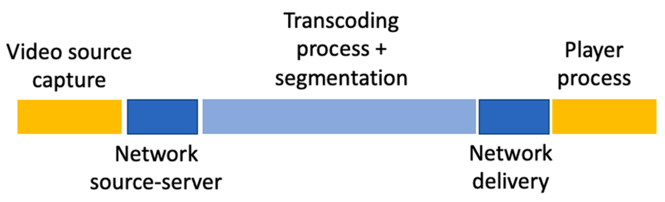


Fig. 1. Processes introducing delay in multimedia transmission.

the authors focus on providing an end-to-end solution through what they call HxL3 architecture (HTTP/x-based Low-Latency Live streaming architecture). This solution is agnostic to the codecs (H.264, H.265, …), application protocols (HTTP/1.1 or HTTP/2.0), streaming format (DASH, HLS), transport protocol (TCP -Transport Control Protocol- or UDP -User Datagram Protocol-) and CDNs. And in Bouzakaria et al. [22], the authors focus primarily on the study of the overhead introduced by streaming content using LL-DASH. The use of transmission using ``chunked transfer encoding'' introduces an increase in headers and therefore overhead in the network.

Regarding the different low latency solutions, in Durak et al. [23] an in-depth comparison is made between the low latency solution proposed by Apple (HLS LL) in 2019, the standard LL-DASH solution of 2019 and Low-latency HTTP Live Streaming (LHLS) that was first introduced by Twitter's Periscope in 2018 and then improved by Twitch in 2019, also describing the differences between the different solutions. Another work focused on low latency scenarios for HLS is [24], where the authors integrate existing LL-DASH ABR schemes in hls.js video player.

Unlike other papers that also present results based on their own testbeds [21,22,25], this work presents an open-source web service that supports chunked transmissions as well as FFmpeg scripts to quickly deploy a LL-DASH evaluation scenario. In addition, this work is an extension of previous work carried out by the authors regarding the DASH protocol, specifically on the evaluation of QoE (Quality of Experience) in DASH [26] and the impact of DASH streaming on Energy Efficient Ethernet [27].

3. Low latency: factors and protocols

According to a report by Bitmovin [28], on the main concerns of companies in the video streaming sector, the problem of latency comes first (41%), second is the controlling cost (e.g., bandwidth, storage,) (33%), and third is device compatibility (32%).

Low latency definition depends to a large extent on the application in which the distribution system is framed. In fact, depending on the application (video on demand, live streaming, videoconferencing, etc.) the latency requirements (in addition to bandwidth requirements, loss tolerance, etc.) are different. To give an example at each end of the multimedia applications, it can be seen how video conferencing or security applications need much lower latency than VoD applications (such as Netflix, HBO, Youtube…) since video conferencing is an interactive and real-time application with very demanding delay requirements (around 150–200 ms).

Generally, real-time, ultra-low latency and low latency solutions involve using protocols such as WebRTC, RTP/RTSP or SRT (protocols used by applications such as Teams, Zoom, Skype, or IPTV services) while, at the other end, HLS and DASH protocols can be found that offer latencies in the order of 20–30 s typically (used by applications such as Netflix, Youtube, HBO, etc.). However, it can be seen that within the 1–5 s range, any protocol can be utilized. At this point, it is important to note that while WebRTC or RTP/RTSP protocols are implicitly designed to offer these low latency performances, HLS or DASH protocols need new functionalities both on the encoding and on the packaging side.

In a simplified form, and considering the presented scenario, the main processes that introduce delay in the system are shown in Fig. 1. Of the different processes involved, this work will focus on the transcoding, segmentation, and communication processes between the player and the server. Because there is no control over the capture process, and the video may arrive with different protocols and encoding parameters (frame rate, bitrate, resolution, etc.), it is not possible to carry out segmentation, and a transcoding process is necessary beforehand.

The main aspects that contribute to the final latency observed in the player and depicted in Fig. 1 are discussed below.

3.1. Encoding

The first aspect to decide is the type of encoder that can be used to carry out the encoding while minimizing delay. Although previous studies have analyzed the efficiency of different encoders and the resulting quality, such as the R-D (rate-distortion) curve, a study is conducted where both the VMAF (Video MultiMethod Assessment

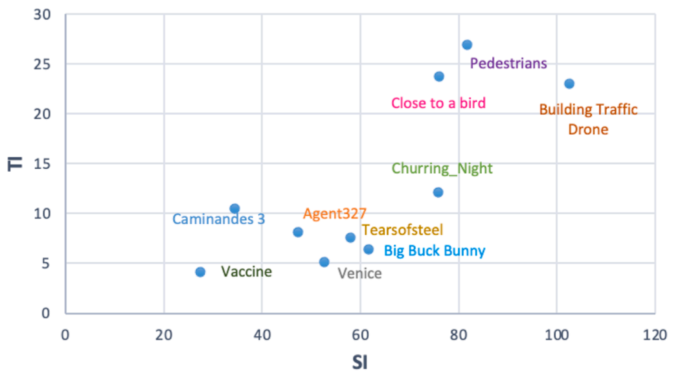


Fig. 3. Evaluation of spatial information (SI) and temporal information (TI).

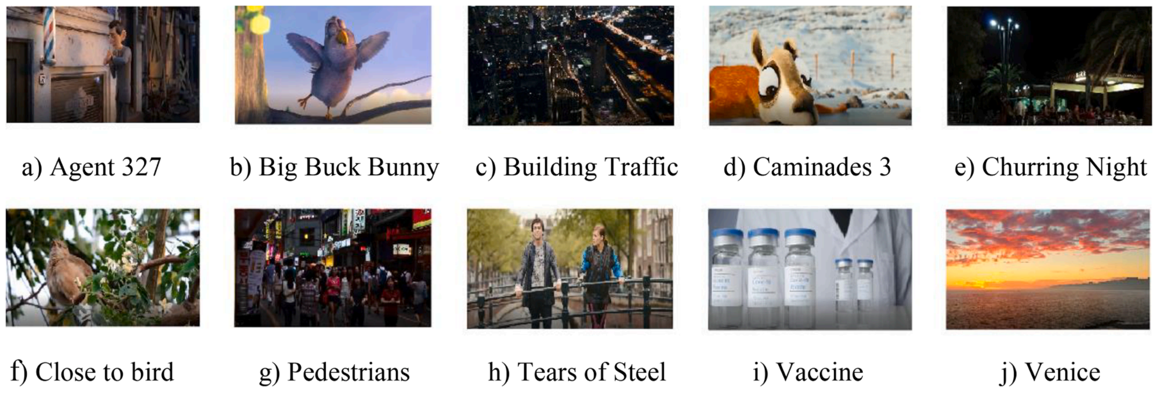
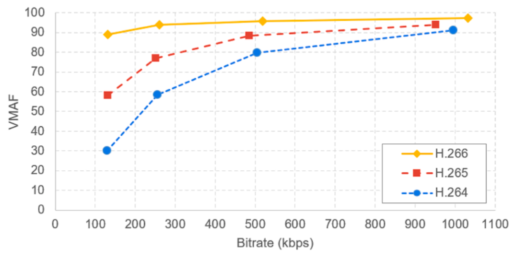
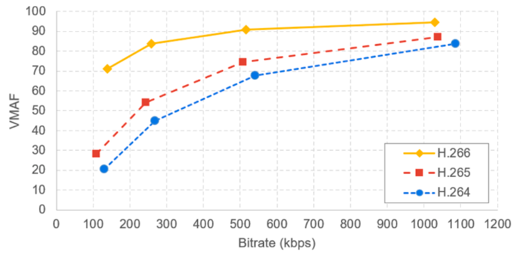


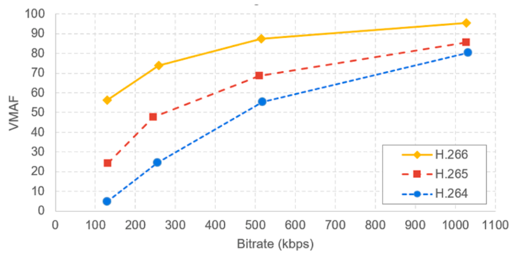
Fig. 2. Frames of the videos used to perform encoding analysis.



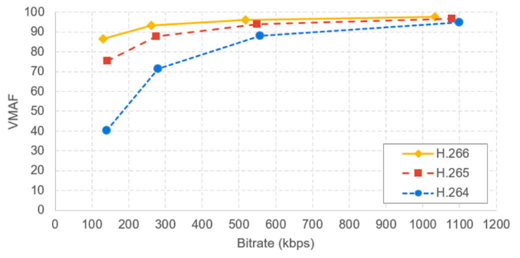
a) Agent 327



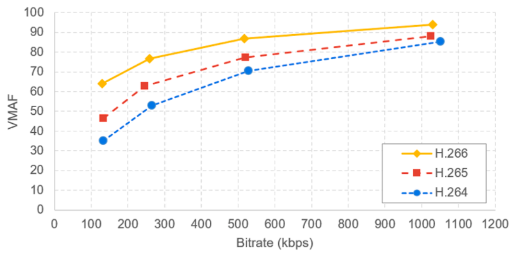
b) Big Buck Bunny



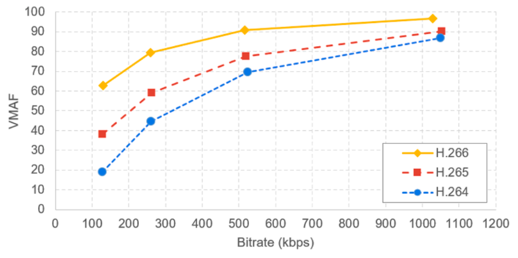
c) Buildings traffic 327



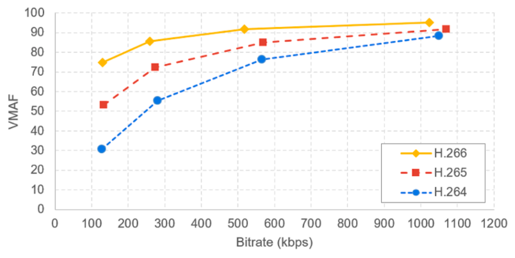
d) Caminandes3



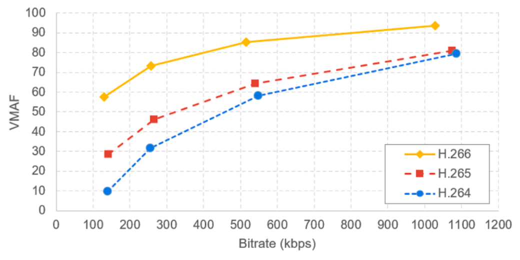
e) Churring night



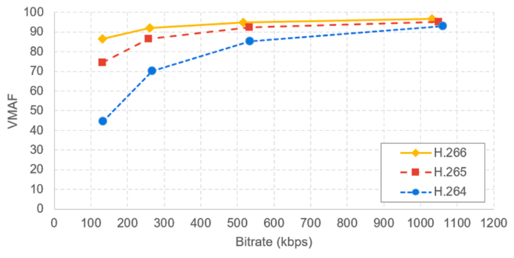
f) Close to a bird



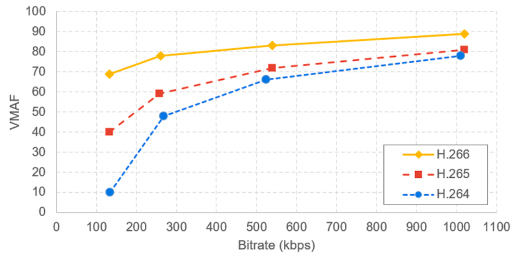
g) Pedestrians



h) Tears of Steel



i) Vaccine



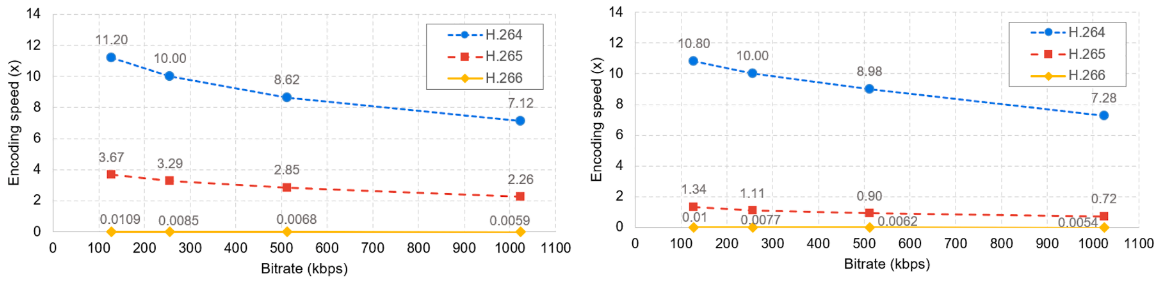
j) Venice

Fig. 4. Evaluation of VMAF for H.264, H.265 and H.266 codecs.

Fusion) and the encoding time will be considered as a comparative measure to assess the feasibility of encoders. Currently, there are different encoders widely used by devices such as cell phones, cameras, tablets, etc., such as H.264/AVC (Advanced Video Coding) and H.265/HEVC (High Efficiency Video Coding), and recently the specifications of other encoders, such as H.266/VVC (Versatile Video Coding), have been published.

To conduct the study, 10 videos have been selected, obtained from Blender (www.blender.org) and Videvo (https://www.videvo.net), all with the same characteristics: duration of 30 s, resolution of 1280 × 720, 24 fps, 4:2:0 sampling format (yuv420p), and 8-bit bit depth. Fig. 2 depicts a frame from each video.

Figure 3 shows the spatial information (SI) and temporal information (TI) values of the 10 videos used for the encoder comparison. Scenes



a) Agent 327 b) Pedestrians

Fig. 5. Evaluation of encoding speed for H.264, H.265 and H.266 codecs.

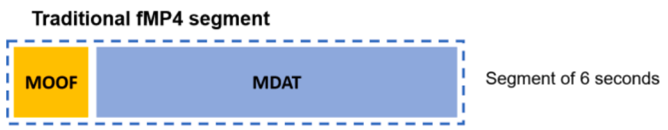


Fig. 6. Traditional fMP4 segment (e.g., 6 s).

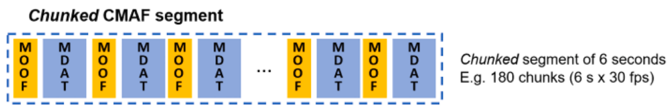

Chunked CMAF segment
MOOF
MDAT
…
Chunked segment of 6 seconds
E.g. 180 chunks (6 s x 30 fps)

Fig. 7. CMAF chunked segment (e.g., 180 chunks).

with a lot of detail have higher spatial complexity or information (SI) than those with less detail. Similarly, scenes with a lot of motion have higher temporal complexity or information (TI) than those with less motion. For example, videos such as *Vaccine* or *Agent 327* have scenes with less detail and less motion compared to sequences like *Pedestrians* or *Close to bird*, where the scenes are more complex due to the presence of more details and motion. Additionally, the behavior of the different encoders on each of these scenarios is analyzed and how they affect the visual quality of the encoded videos.

Among all the possible measures to evaluate video quality (PSNR, SSIM, VMAF, …), VMAF has been considered for the assessment because it takes into account perceptual quality factors that are more closely aligned with how humans perceive video quality. VMAF has been shown to perform well across a wide range of video content, including high-motion, low-bitrate, and high-resolution content. This makes it a versatile and reliable metric for video quality measurement in a variety of contexts. Hence, Fig. 4 shows the result of measuring VMAF with respect to the bitrate for all sequences (128 kbps, 256 kbps, 512 kbps and 1024 kbps), with a resolution of 1280 × 720 pixels and encoded at 24 fps. It can be seen, for example, that for *Agent 327* sequence (Fig. 4(a)), for a VMAF = 90 requires about 129 kbps with H.266, while H.264 and H.265 require almost 1 Mbps.

Regarding the VMAF values obtained, it is fulfilled in all sequences that the H.266 encoder offers better quality than H.264 and H.265. Specifically, the VMAF improvement of H.266 with respect to H.265 in the *Agent 327* sequence is [30.93; 16.87; 7.53; 3.27] points corresponding to the bitrates of [128k, 256k, 512k, 1024k], and [58.97; 35.45; 16.06; 6.17] points corresponding to the bitrates of [128k, 256k, 512k, 1024k] with respect to H.264. It is observed in all sequences that the quality improvement is greater when the bitrate is lower. In this case, it can be seen that this improvement is much greater in the case of a 128k bitrate. An improvement of 30.93 VMAF points indicates that the subjective quality perceived by the user is much better in H.266 than in H.265 for a bitrate of 128 kbps. For the 1024 kbps bitrate, the VMAF improvement is only 3.27 VMAF points, indicating that the perceived video quality change when using H.266 is very low compared to using H.264 or H.265. Regarding the improvement that H.265 introduces over H.264, it is still significant for low bitrates.

However, for low latency systems, it is necessary to use other metrics, such as the encoding time used by each encoder. For this, it can be used the encoding speed relative to the frame rate. According to the results shown in Fig. 5, for example, for a low-quality (128 kbps) encoding of the *Agent 327* sequence, an encoding speed of 11.2× in H.264 means that it is capable of encoding 268 fps; in H.265, the speed of 3.67× implies being able to encode 88 fps; and in H.266, the speed of 0.0109× allows only encoding 0.26 fps. Comparing the encoding speed for the bitrate of 1024 kbps, 170 fps, 54 fps, and 0.14 fps could be reached in H.264, H.265 and H.266, respectively. The same comparison for the *Pedestrian* sequence shows that for medium quality (512 kbps), both the H.265 and H.266 offer inefficient results. In this way, there is a trade-off between encoding efficiency and speed coding. In the case of low latency scenarios, where the delay is the key factor, both H.265 and H.266 codecs can be discarded for the transcoding process.

On the other hand, the GoP (Group of Pictures) represents the pattern of appearance of the frames (I, P, B) in the encoded video. There are multiple possible GoP configurations: periodic/non-periodic; with/without B-frames; open/closed GoP; large size/small size; etc. Regarding latency, whether to use B-frames and the GoP size are of great importance in GoP configuration. And with regards to GoP size and latency, this feature has less influence on RTP-based protocols (WebRTC) than on HLS or DASH protocols. The reason is that protocols such as RTP transmit frame by frame, while HLS or DASH protocols transmit segments (groups of frames) that usually consist of a whole number of GoPs.

3.2. *Segmentation*

DASH and HLS are based on the transmission of segments using the HTTP protocol and adaptive algorithms. The fact of using segments introduces an intrinsic delay into the system, since it is necessary to wait for the generation of the segment in order to be able to transmit it. Typical values for the duration of such segments are 2 s, 4 s, 6 s or 10 s. Without considering the fact of having to use a buffer in the receiver, the segmentation process already introduces an unacceptable delay for many interactive or real-time applications. Figure 6 shows schematically the structure of a 6 s segment in the fMP4 (fragmented MP4) format. When using fMP4 file format, the encoded file is divided into segments that can be downloaded and played back. However, to start playback, it is necessary for the player to download the complete segment (*mdat*).

To reduce playback delay, a new container format called CMAF (Common Media Application Format) has been specified, which allows the segment to be divided into chunks and to start playback while new chunks are still being generated. Figure 7 shows how a segment has been split into one chunk per frame, ideal for real-time applications. However, there is a trade-off between the latency reduction achieved and other aspects such as the probability of interruptions due to congestion or the increase in headers for accessing the chunks.

Finally, to minimize the effect of congestion, all players use buffers to

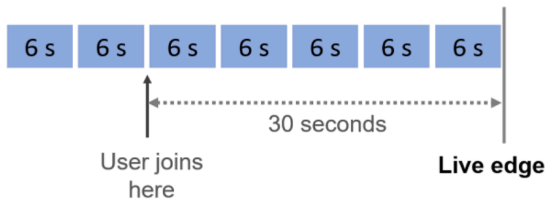


Fig. 8. Segments in player buffer (e.g., 30 s).

store a certain number of frames or segments before starting playback (Fig. 8). Again, there is a trade-off between buffer size and the probability of service interruptions. Typical values are in the range of 30 s buffer. However, depending on the applications (and specifically for real time) these values must be reduced to obtain a valid service.

Once the factors that affect latency have been analyzed, the protocols used for video transmission must be considered. In fact, each option will have different answers regarding latency, scalability, etc. that should be evaluated.

3.3. *Network delivery*

On the one hand, the exponential growth of traffic in wireless networks is posing a challenge due to the limited resource of the frequency spectrum. As users demand more bandwidth, it is important to seek out underutilized features of current wireless technologies to optimize spectrum efficiency. While there are mechanisms, such as MIMO (Multiple-Input Multiple-Output) technology, channel configuration, and limiting connections per access point, that aim to enhance Wi-Fi network performance, providing adaptive streaming services with satisfactory QoE remains a challenge in high-density wireless environments with 50–100 or even 500–600 users.

DASH technology was originally designed for Internet content broadcasting. However, when deployed in shared Wi-Fi environments, where there is a high concentration of devices such as in event halls, stadiums, auditoriums, buses, trains, ferries, etc., not only is the bandwidth per client reduced, but measuring the available bandwidth becomes challenging due to the high variability of wireless links. This variability creates problems for adaptive multimedia systems when determining the quality of video segments to be downloaded, as in the case of DASH client-driven architecture, where clients estimate bandwidth and request the appropriate quality from the server.

In this scenario, clients have limited information as they only measure transmission rates while downloading segments, after winning the media access contest. Unfortunately, during the next segment request, contention may increase due to other clients trying to access the channel again, interference from other networks or electromagnetic signals, or a loss of signal or coverage due to client mobility or the presence of obstacles between the client and the access point. In any of these cases, the bandwidth measured by the client does not provide an accurate indication of the quality required to maintain continuous playback and minimize the impact on QoE.

To tackle this issue, tutorials and novel proposals have emerged that rely on cross-layer solutions and coordination between the server, network, and clients [29–31]. In this sense, cross-layer mechanisms and, in particular, Server and Network Assisted DASH (SAND) technology [32], can serve as effective alternatives for carrying out additional processing to enhance the overall system performance.

Nevertheless, as mentioned earlier, low latency streaming systems entail greater complexity due to the need for adaptive algorithmic solutions that consider chunk-based transmission and highly variable bandwidth environments.

4. Server-side representation switching

One of the major advantages of using DASH for video streaming is that video servers do not need to maintain session state for clients. Each client manages its own playback position and requests the most relevant representation according to its circumstances (bandwidth, playback device used…). However, adapting video quality in low-latency streaming scenarios poses new challenges, in which server-side assistance can be useful.

When a video stream has multiple qualities sharing certain codec parameters, such as resolution and video codec, segments of different qualities can be interchanged and decoded using the same decoder instance without errors. For example, if a video is encoded with the same resolution and codec parameters, but at different bitrates, a single decoder instance can decode segments of any bitrate, generating decoded frames without errors. In this sense, Fig. 9 shows a single video decoder decoding a stream of segments from different representations. In the example, the video encoded with a bitrate B has a better quality (a higher bitrate) that the video encoded with a bitrate A.

Taking this into consideration, this work proposes a new solution for Low Latency Video streaming based on the information managed by the server. Thus, the solution proposed, called Server-Side Representation Switching (SSRS), leverages the aforementioned capability to select the quality of the representation for each segment that will be transmitted to video clients. To prevent clients from attempting to change the representation on their own, SSRS modifies the MPD to contain a single representation, thus making video players unaware of any change of representation.

In order for the server to determine which quality to send to each client for each request, a regular DASH server would need to maintain information about each client. However, SSRS can take advantage of the fact that it is broadcasting a Low-Latency DASH video stream to use the instants of time at which it receives requests for segments to calculate the representation to transmit. Figure 10 illustrates how the Low-Latency DASH server can easily calculate the delay of each client by simply comparing the time at which it receives the request with the moment it began receiving the required segment. The figure shows a single video decoder decoding a stream of segments from different representations.

Thus, SSRS does not need to maintain client state, since it only takes into account the delay of the request for the segment relative to the time at which the server began receiving that segment in order to calculate

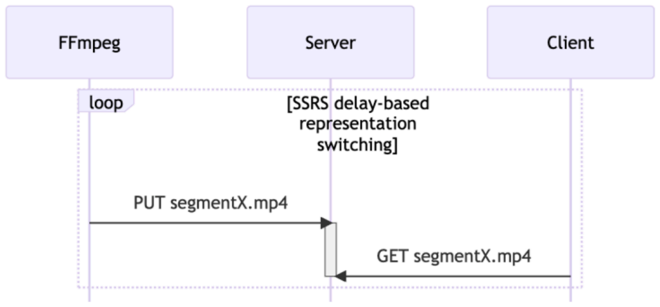


Fig. 10. Example of the performance of SSRS.

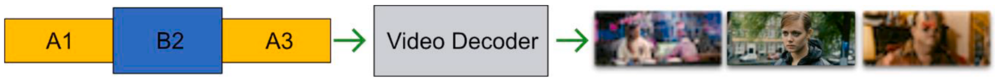


Fig. 9. Example of decoding a stream of segments.

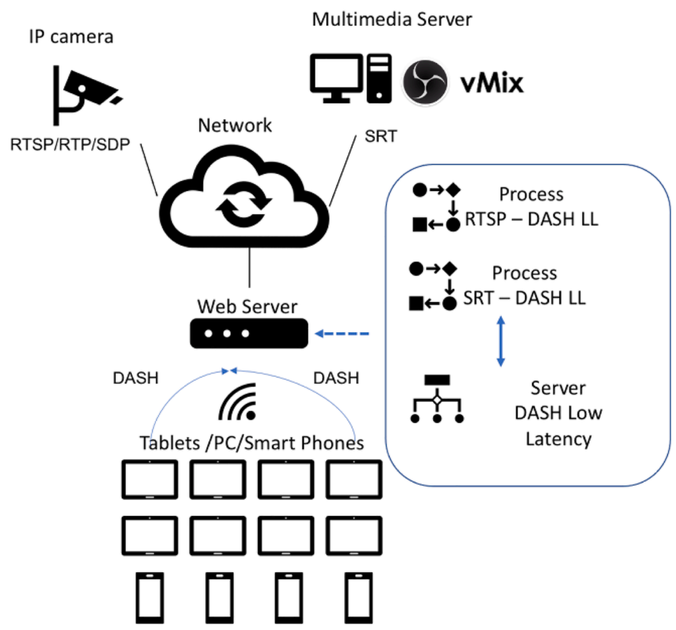


Fig. 11. Scenario - Real Time Production – LL-DASH Distribution.

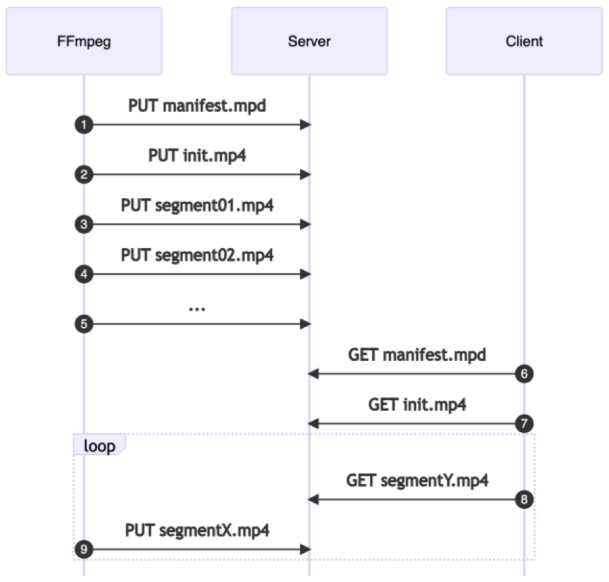


Fig. 12. DASH client-server sequence.

the representation to serve, as shown in the following equation:

$$rep_{segment\ X} = \max(0, round(rep\_\max - \lambda * \ delay)), \tag{1}$$

where $rep_{segment\ X}$ is the representation sent to the client for segment *X; rep_max* is the representation with the highest quality (that is, the representation encoded with the maximum bitrate); *delay* is the time (in seconds) between the instant of time the PUT request arrives at the server and the instant of time when the GET request arrives at the server for a specific segment; and λ is a weighting factor (the higher the λ, the more conservative the solution in terms of bitrate).

5. Methodology

5.1. Low latency DASH testbed

The situation arises where protocols intended for real-time transmission (from cameras, transmission equipment or production software) such as RTSP or SRT protocols, and the advantages of using LL-DASH technology for content distribution must coexist. Therefore, it is necessary to integrate both technologies to take advantage of the benefits of both. Figure 11 depicts schematically the proposed scenario for this purpose.

The figure shows different video content sources (cameras or content servers). The output of these sources uses RTSP or SRT. The server includes the developed processes that take care of the reception of these streams and their efficient transformation into video segments complying with the CMAF format for their use by the LL-DASH technology. On the other hand, a Web server has been implemented in Python that supports the distribution of content using the LL-DASH standard by sending chunks. Finally, the devices play the contents through DASH clients (such as Dash.js, Shaka-Player, etc.).

The use of standard HTTP servers for DASH is a key benefit of the technology but imposes a defined sequence of events where segment files must be available on the server prior to any request could be successfully handled. This sequence poses no inconvenience for VoD, where segments can be available on the servers in advance, or even on live streaming, where players can be playing some segments behind, but

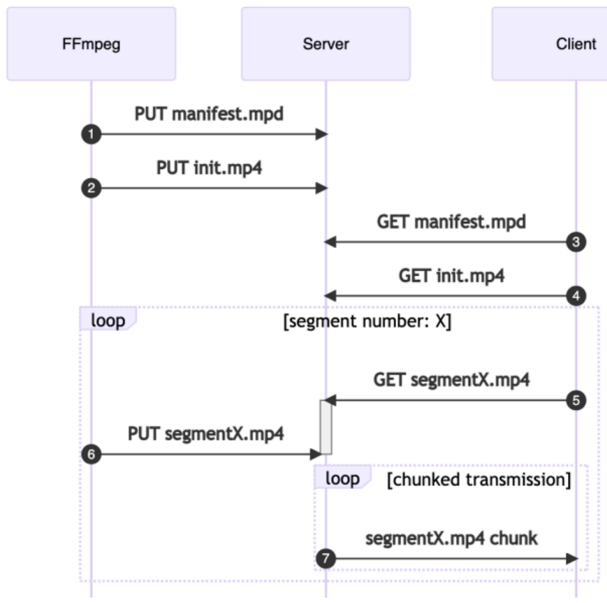


Fig. 14. Low Latency DASH client-server sequence.

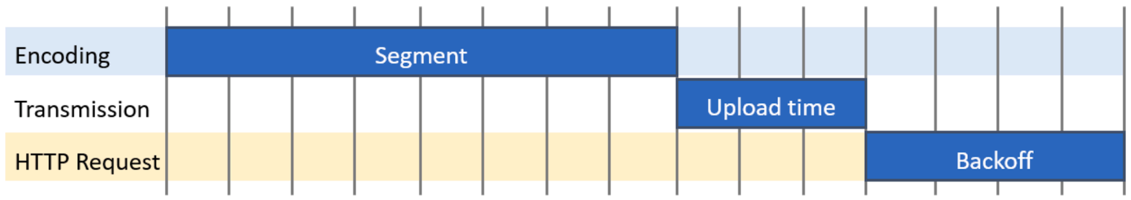


Fig. 13. DASH minimum client delay.

```
async def generate_partial_segment(segment: Segment):
    chunks = segment.chunks
    aux = 0
    while aux < len(chunks):
        try:
            await asyncio.wait_for(chunks[aux].event.wait(), 1)
            data = chunks[aux].data
            if data is not None:
                yield data
            aux = aux + 1
        except asyncio.TimeoutError:
            return
```

Fig. 15. Fast-ll *generate_partial_segment* function.

```
ffmpeg \
-fflags nobuffer \
-flags low_delay \

# SRT source
-i "srt://127.0.0.1:5000" \

# or RTSP source
-avioflags direct \
-f rtsp \
-i rtsp://root:pass@10.0.0.96:554/axis-media/media.amp \

-c:v libx264 \
-x264opts keyint=25:minkeyint=25:scenecut=-1 \
-tune zerolatency \
-profile:v baseline \
-preset veryfast -bf 0 -refs 3 \
-b:v 500k -bufsize 500k \
-utc_timing_url "https://time.akamai.com/?iso" \
-use_timeline 0 \
-format_options "movflags=cmaf" \
-frag_type duration \
-adaptation_sets "id=0, seg_duration=1, frag_duration=0.1, streams=v" \
-streaming 1 \
-ldash 1 \
-export_side_data prft \
-write_prft 1 \
-target_latency 0.5 \
-window_size 5  \
-extra_window_size 10 \
-remove_at_exit 1 \
-method PUT \
-f dash \
http://localhost:8000/test/manifest.mpd
```

Fig. 16. FFmpeg command line to transform RTSP into LL-DASH.

Table 1
Parameters used for the low latency DASH testbed.

Evaluation parameter	☐ Latency
Sources	☐ RTSP camera (*RTSP*)
	☐ FFmpeg test source (*Gen*)
	☐ SRT source (*SRT*)
Tools	☐ Dash.js version 4.7
	☐ Fast-ll server
Target latency	☐ 0.2, 0.5, 1 and 2 s

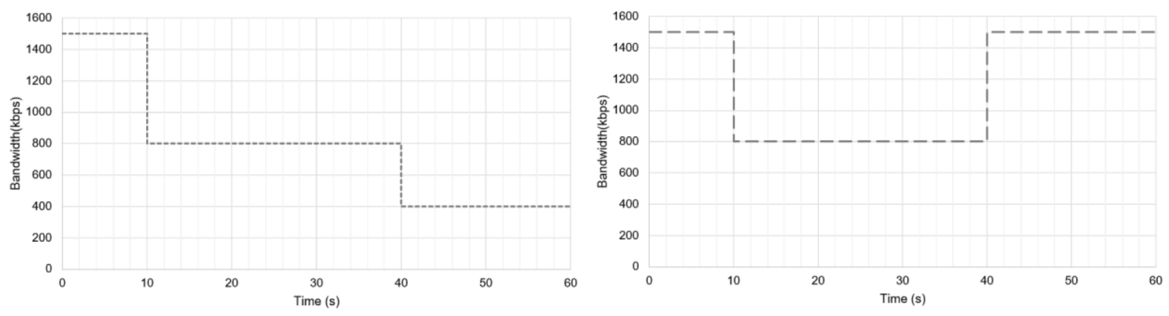
generates an unavoidable delay between the generation and the consumption of the content. Figure 12 shows how DASH clients must request, at least, a segment behind the sequence of generated segments ($Y < X$) in order to avoid HTTP errors.

This requirement generates a minimum delay equal to the sum of the time length of the segment, the time of the transmission to the server and the backoff time to avoid HTTP errors, as Fig. 13 depicts.

As previously stated, this minimum delay may suit some live streaming scenarios but not those which require low latency. For achieving low latency when using DASH, HTTP servers can no longer be static content servers but dynamically handle segment requests.

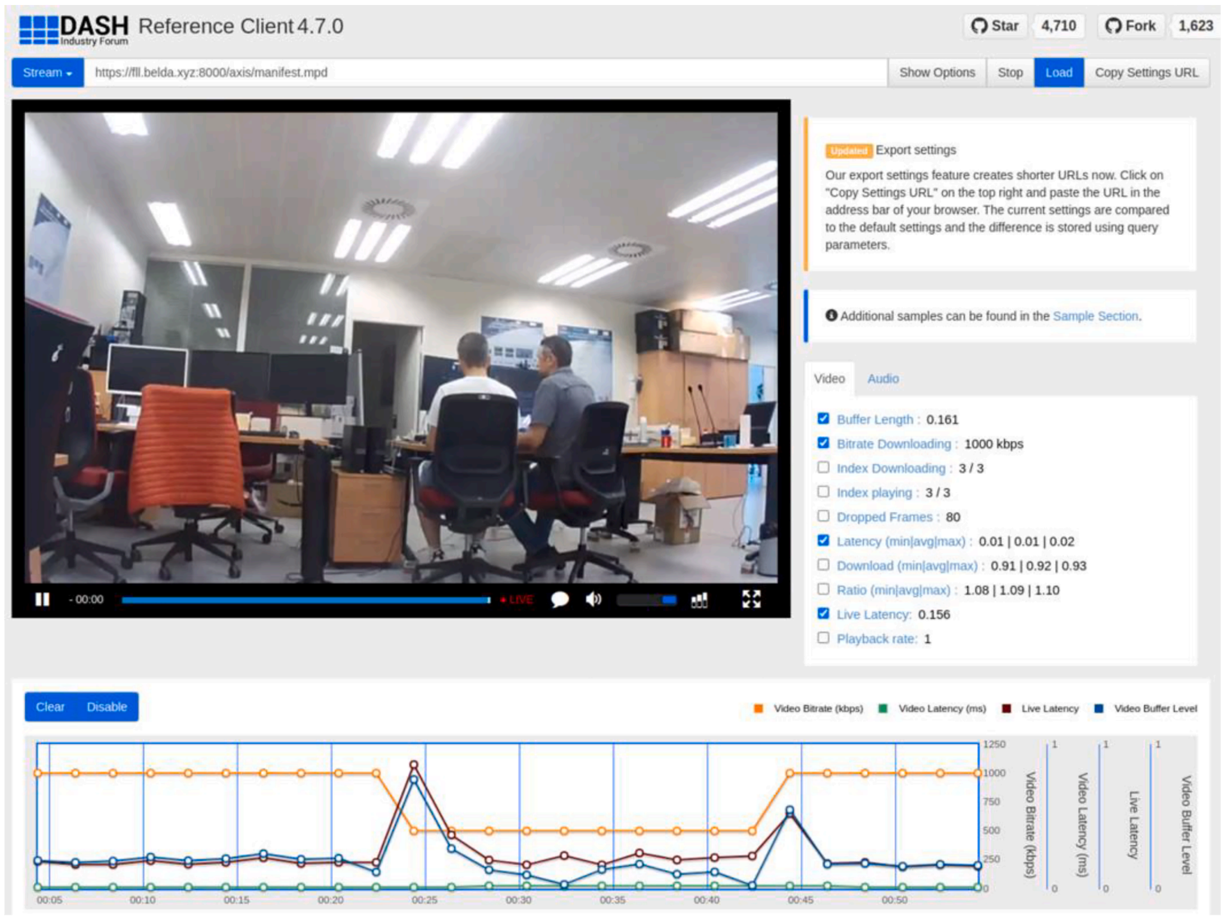
Figure 14 shows the required behavior of HTTP servers to handle LL-DASH clients. First, manifest and the initial segment must be uploaded to the server (events 1 and 2) before the content is accessible to the client (events 3 and 4). In live DASH, manifest is periodically generated and uploaded to the HTTP server, but it is omitted from the figure for clarity. Next, the client will begin to request segments based on the manifest, the target latency and the current time. In LL-DASH, those segment requests will reach the HTTP server even before the segment transmission from the source to the server has started. In this scenario, the HTTP server must retain the HTTP request long enough to wait the incoming segment or timeout otherwise, as represented between events 5 and 6 in Fig. 14.

The HTTP server, when the requested segment reception begins



a) Ladder b) Sinkhole

Fig. 17. Scenarios used to perform tests.



https://fll.belda.xyz:8000/axis/manifest.mpd

Updated Export settings

Our export settings feature creates shorter URLs now. Click on "Copy Settings URL" on the top right and paste the URL in the address bar of your browser. The current settings are compared to the default settings and the difference is stored using query parameters.

Additional samples can be found in the Sample Section.

- [✓] Buffer Length : 0.161
- [✓] Bitrate Downloading : 1000 kbps
- [ ] Index Downloading : 3 / 3
- [ ] Index playing : 3 / 3
- [ ] Dropped Frames : 80
- [✓] Latency (min|avg|max) : 0.01 | 0.01 | 0.02
- [ ] Download (min|avg|max) : 0.91 | 0.92 | 0.93
- [ ] Ratio (min|avg|max) : 1.08 | 1.09 | 1.10
- [✓] Live Latency: 0.156
- [ ] Playback rate: 1

Fig. 18. Screenshot of the testbed.

(event 6), starts sending the content of the segment to the pending requests, as it arrives, in the form of chunked transmission.

Aiming to test the described behavior, a Python HTTP server for LL-DASH has been developed based on FastAPI [33] framework, called Fast-ll. Figure 15 shows an excerpt of the server where a generator is created to add received chunks to the response of clients while they are being received. Fast-ll handles manifest and segments in-memory so no copy is stored on disk. The HTTP server must be fed using HTTP PUT requests to add content, and DELETE requests to remove content (when the segments exceed the windows defined in the manifest, thus no client will request it). The process of using HTTP methods to manage the content on the Fast-ll can be done by the FFmpeg tool [34] when the appropriate set of parameters is provided.

For reference, Fig. 16 includes a complete command-line that uses FFmpeg to access an SRT or RTSP stream, recodes the video stream, generates a low latency DASH stream and uploads the different objects (manifest and segments) to the locally running Fast-ll server.

Specifications for all parameters in Fig. 16 can be found in Kempf [35]. Among all parameters, there are three of particular relevance: 1) -ldash 1: specifies the LL-DASH mode; 2) -target_latency 0.5: latency that the client will try to achieve; and 3) -format_options "movflags=cmaf": the container format.

The source code of Fast-ll server and the scripts used in the work can be found in the Git repository https://github.com/robelor/fast-ll [36].

5.2. Evaluation setup

With the development introduced in this work, it is straightforward

```
{
    "name": "Hikvision Camera",
    "stream": "hik",
    "type": "RTSP",
    "input": "rtsp://XXX:XXX@192.168.0.89:554/h264/ch1/main/av_stream",
    "targetFps": "10",
    "segmentDuration": "1",
    "fragmentDuration": "0.1",
    "intraInterval": "10",
    "serverSideStreamSwitching": true,
    "qualities": {
      "video": [
        {
          "targetWidth": "640",
          "targetBitrate": "250"
        },
        {
          "targetWidth": "640",
          "targetBitrate": "500"
        },
        {
          "targetWidth": "640",
          "targetBitrate": "1000"
        }
      ]
    },
    "targetLatency": "0.2"
  }
```

Fig. 19. Example of Fast-ll streaming configuration.

Table 2
Parameters used for the server-side representation switching evaluation testbed.

Evaluation parameters	☐ Latency
	☐ Buffer length
	☐ Bitrate
Algorithms	☐ SSRS ($\lambda = 1$)
	☐ L2A
	☐ LoL+
Scenarios	☐ Wi-Fi AC
	☐ Ladder (1500 kbps, 800 kbps, 400 kbps)
	☐ Sinkhole (1500 kbps, 800 kbps, 1500 kbps)
Tools	☐ Dash.js version 4.7
	☐ Fast-ll server
Video encoding parameters	☐ Codec: H.264
	☐ Representations: 250 kbps, 500 kbps, 1000 kbps
	☐ Resolution: 480p

to setup a test environment to evaluate LL-DASH tools and clients.

In this work, two different types of evaluations have been carried out in order to evaluate both the Low Latency DASH testbed proposed as well as the Server-Side Representation Switching solution, explained in the following sections.

5.2.1. Setup of the low latency DASH testbed

First of all, to demonstrate the suitability of the Low Latency DASH testbed, an evaluation of the parameter target latency when generating LL-DASH content using FFmpeg has been performed. The evaluation is

Table 3
Latency (in seconds) measured by Dash.js.

	0.2 s	0.5 s	1 s	2 s
Gen	0.21078	0.41942	0.95998	1.87212
RTSP	0.35892	0.48984	0.97928	1.98286
SRT	0.20634	0.49404	1.01040	2.02898

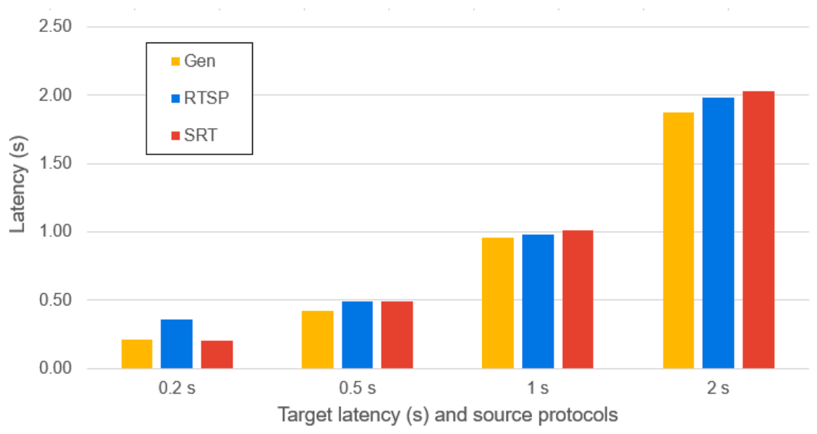
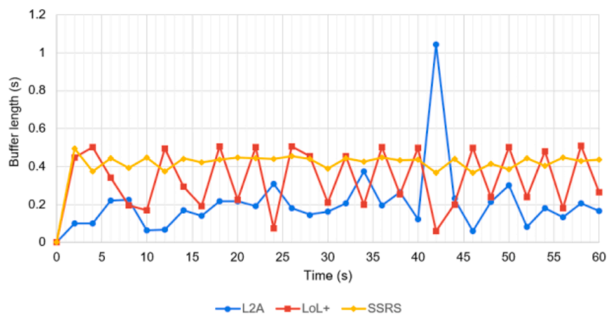
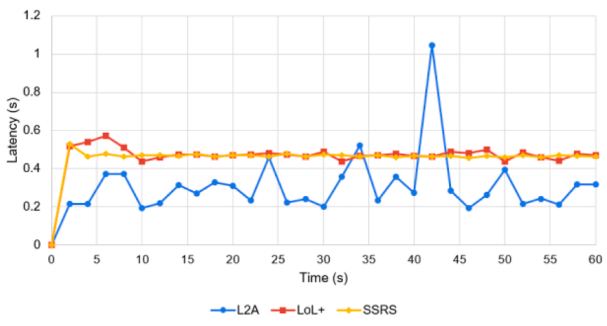


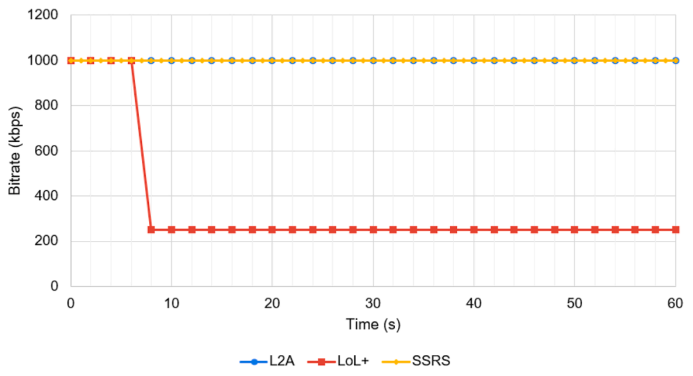
Fig. 20. Latency measured by the Dash.js client for different target latency values and source protocols.



a) Buffer length evaluation



b) Latency evaluation



c) Displayed bitrate evaluation

Fig. 21. Evaluation of the Wi-Fi scenario.

Table 4
Evaluation of the Wi-Fi scenario.

	Buffer length (s)			Latency (s)			Displayed bitrate (kbps)		
	L2A	LoL+	SSRS	L2A	LoL+	SSRS	L2A	LoL+	SSRS
Average	0.21	0.34	0.43	0.31	0.48	0.47	1000	346.77	1000
Min	0.06	0.06	0.37	0.19	0.44	0.46	1000	250.00	1000
Max	1.04	0.51	0.45	1.05	0.57	0.48	1000	1000.00	1000
Std dev.	0.17	0.15	0.03	0.16	0.03	0.01	0	255.58	0
MSE	–	–	–	0.04	0.08	0.07	–	–	–

carried out using three different sources: an RTSP camera (*RTSP*), an FFmpeg test source (*Gen*) and SRT source (*SRT*) generated also using FFmpeg. Regarding the target latency parameter, the evaluated values are 0.2, 0.5, 1 and 2 s.

FFmpeg uses the value, in seconds, of this parameter to generate the Media Presentation Description (MPD) accordingly. For example, when 0.5 is specified as target latency parameter, the resulting MPD will incorporate "<Latency target="500"/>" inside the ``Service-Description'' tag as the MPD defines the value to be in milliseconds.

In order to carry out the measurements, two main tools has been used: Dash.js and Fast-ll server. This tools are explained in the next section.

Table 1 summarizes the parameters used for the testbed.

5.2.2. *Setup of the server-side representation switching evaluation*

In order to compare the performance of the Server-Side Representation Switching solution proposed in this work, two well-known algorithms have been used: L2A and LoL+. It is worth highlighting that both algorithms are integrated into the DASH-IF reference video player (Dash.js). L2A is based on online learning and the Online Convex Optimization (OCO) theory and does not require any parameter tuning nor throughput estimation, performing well over a wide spectrum of network profiles, as shown in Karagkioules et al. [11]. Likewise, LoL+ [12] offers a precise bandwidth prediction and rate adaptation algorithm for low latency scenarios, and it was designed in order to provide a good QoE for any given target latency.

These two algorithms, as well as the SSRS solution hereby presented, have been tested in three different scenarios: the first consists of a Wi-Fi scenario (Wi-Fi 802.11 ac), specifically a laboratory sited at Universitat Politècnica de València; the second, called “ladder” and shown in Fig. 17 (a), presents an initial bandwidth of 1500 kbps, 10 s later the bandwidth switches to almost the half (800 kbps), and 30 s later the bandwidth drops until 400 kbps; the third scenario, called “sinkhole” it is equal to the second scenario except for the last 20 s, where the bandwidth instead of decreasing, increases until 1500 kbps, as Fig. 17(b) depicts. Therefore, the duration of each test is 60 s.

In order to carry out the measurements, as in the previous study, Dash.js has been used [37], specifically version 4.7.0, as shown in Fig. 18. This tool provides accurate information about the selected representation (represented by its target average bitrate), the network latency (time from the request to the response arrival, in seconds) and live latency (difference between live time and playback position in seconds) among other parameters.

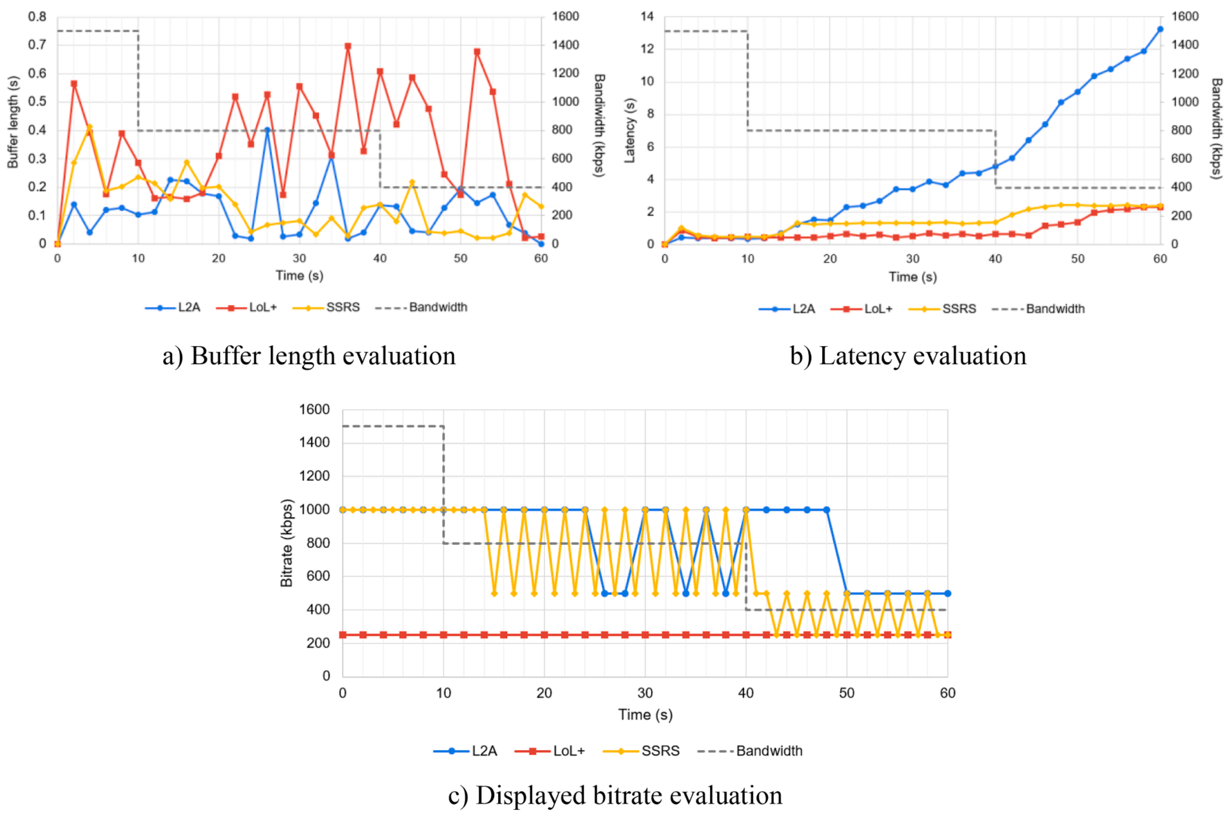


Fig. 22. Evaluation of the ladder scenario.

Table 5
Evaluation of the ladder scenario.

	Buffer length (s)			Latency (s)			Displayed bitrate (kbps)		
	L2A	LoL+	SSRS	L2A	LoL+	SSRS	L2A	LoL+	SSRS
Average	0.12	0.36	0.13	4.58	0.88	1.47	838.71	250.00	688.52
Min	0.00	0.02	0.02	0.35	0.41	0.47	500.00	250.00	250.00
Max	0.40	0.70	0.41	13.26	2.32	2.42	1000.00	250.00	1000.00
Std dev.	0.09	0.19	0.10	4.02	0.64	0.68	237.60	0.00	301.58
MSE	–	–	–	34.85	0.85	2.06	–	–	–

For the development of the tests, the Fast-ll server has been configured to re-encode the video stream from a Hikvision camera accessed via RTSP at a resolution of 1080p. This camera, depending on factors such as ambient light intensity, is able to change the frame rate at which it sends the video stream.

It is important to note that although the video obtained from the camera is already encoded using H.264, the video server must re-encode the video streaming to adapt it to the resolutions, frame rates and bitrates specified in the configuration.

Fast-ll has been configured to access the stream from this camera and generate three representations. All of them with a resolution of 480p. Figure 19 depicts an example of the stream configuration, showing the common parameters to all representations such as *targetFps, segmentDuration* or *fragmentDuration*, and the configuration of both the resolution and the bitrate of the three representations presented. The *serverSideStreamSwitching* parameter, if the configuration of the different representation allows it, indicates Fast-ll to use SSRS.

Regarding the SSRS solution, the tests have been carried out fixing $\lambda = 1$, as indicated in Eq. (1).

Finally, Table 2 summarizes the parameters used for the Server-Side Representation Switching evaluation testbed.

6. Results and discussion

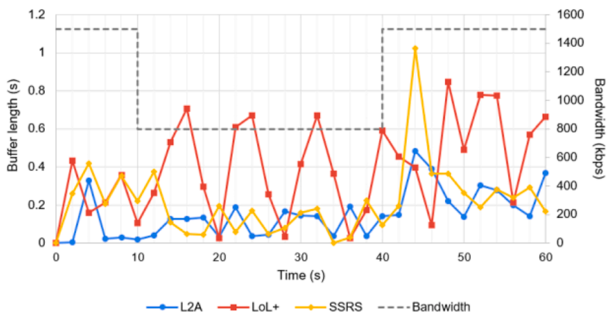
6.1. Evaluation of low latency DASH testbed

Figure 20 shows the average delay identified by Dash.js for each protocol and target delay. Numeric values can be seen in Table 1. This measured delay refers to the time between the segmentation process and the display time of video frames. Naturally, the overall delay will include delays introduced by the video sources, transport, and segmentation (Table 3).

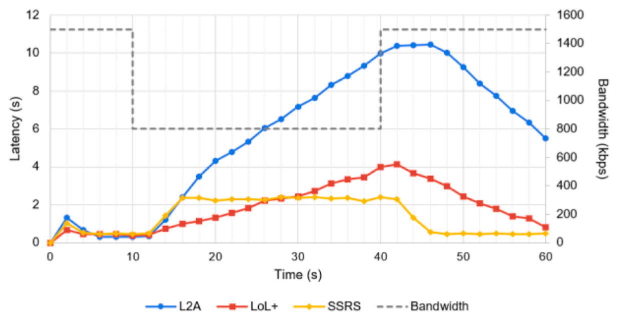
All video sources offer similar results, complying with the target latency in each case, except RTSP for the most restrictive case (0.2 s).

6.2. Evaluation of server-side representation switching

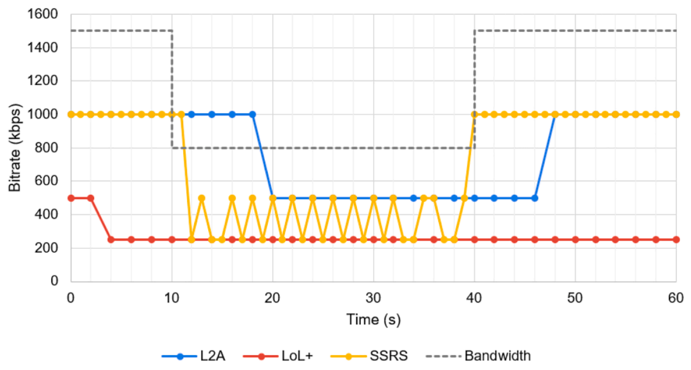
As aforementioned, in order to assess the new proposed algorithm, three different scenarios have been considered: a Wi-Fi scenario, a ladder bandwidth channel and a sinkhole bandwidth channel. The most representative parameters are *targetLatency* of 0.2 s, *segmentDuration* of 1 s, *fragmentDuration* of 0.1 s, and three *encodingTargetBitrates* of 250 kbps, 500 kbps and 1000 kbps, as shown in Fig. 19 for all the three scenarios. Starting with the first scenario, Fig. 21 shows the behavior of the three algorithms analyzed regarding the buffer length, the latency



a) Buffer length evaluation



b) Latency evaluation



c) Displayed bitrate evaluation

Fig. 23. Evaluation of the sinkhole scenario.

Table 6
Evaluation of the sinkhole scenario.

	Buffer length (s)			Latency (s)			Displayed bitrate (kbps)		
	L2A	LoL+	SSRS	L2A	LoL+	SSRS	L2A	LoL+	SSRS
Average	0.16	0.41	0.22	5.80	1.93	1.42	774.19	266.13	709.02
Min	0.00	0.03	0.00	0.31	0.40	0.46	500.00	250.00	250.00
Max	0.48	0.85	1.03	10.45	4.14	2.41	1000.00	500.00	1000.00
Std dev.	0.12	0.25	0.19	3.54	1.19	0.89	252.94	62.43	329.71
MSE	–	–	–	43.50	4.36	2.25	–	–	–

and the displayed bitrate at the client side. A summary of the results is shown in Table 4, which shows the average, minimum, maximum, and standard deviation values of the evaluated parameters. Also, regarding the latency, the table shows the Mean Squared Error (MSE) taking as a reference the target latency of 0.2 s.

Regarding buffer length, Fig. 21(a) and Table 4 reveal that the most stable solution is SSRS, with a buffer length around 0.45 s throughout all the transmission. L2A is the algorithm that provides the minimum average buffer length, with a remarkable peak of more than 1 s when $t = 42$ s. Both L2A and LoL+ offer a more fluctuating behavior regarding the buffer length. In the case of latency, both LoL+ and SSRS behave rather similar. The three algorithms provide good values regarding the MSE, thus fulfilling the target latency. Although L2A is the algorithm that provides the best average latency, it also has a remarkable peak with a latency higher than 1 s. Finally, with regards to displayed bitrate, Fig. 21 reflects that with L2A and SSRS, the displayed bitrate is the maximum available (1000 kbps), which is the expected result considering that there is no bandwidth throttling. On the contrary, LoL+ has a more conservative behavior, using the minimum quality (250 kbps) for the most of the transmission.

The results are not so stable when analyzing the different algorithms in more challenging channels. Regarding the ladder scenario, Fig. 22 depicts that there is a notable fluctuation in terms of buffer length. In this case, L2A and SSRS behave rather similar, providing alike values of average, minimum, maximum, and standard deviation according to Table 5. However, this similarity disappears when considering the latency, since the latency using L2A increases considerably when the channel bandwidth decreases (with values of more than 10 s at the end of the transmission). In the case of the proposed solution SSRS, the latency increases to 2.42 s. LoL+ provides the best results regarding latency (and it is the only algorithm that fulfills the target latency) at the expense of providing the worst displayed bitrate (250 kbps). Again, SSRS offers similar results to L2A, although providing slightly lower displayed bitrate.

The last scenario assessed is the sinkhole channel and results are shown in Fig. 23 and Table 6. The conclusions remain unchanged from the previous scenario: SSRS is the solution that offers the best trade-off among buffer length, latency and displayed bitrate. In this scenario, the SSRS provides the minimum MSE of the latency. Likewise, in this particular case, it is worth mentioning that the maximum buffer length of SSRS is higher than 1 s in the particular instant of time when the channel bandwidth increases, and that SSRS is the solution that offers minimum latency.

Finally, it is worth highlighting that a small buffer length entails better performance, as a longer buffer length results in higher latency during playback. However, if the buffer length reaches zero at any point meaning the buffer has emptied, this may indicate that the video could experience interruptions and degrade the QoE.

7. Conclusion

Currently, integration of real-time sources (using protocols such as SRT or RTSP) for distribution over the Internet or CDNs, using LL-DASH technology, is an interesting topic from a performance analysis point of view.

As the main contribution, this work presents a content distribution system for live streaming using Low Latency DASH along with a proposal for the selection of the transmitted quality, based on a server-side decision-making process called SSRS (Server-Side Representation Switching), to improve the offered QoE in certain scenarios. The evaluated parameters in the implemented testbed have been buffer length, latency, and displayed bitrate. Regarding the Low Latency DASH testbed, both RTSP camera, SRT source and FFmpeg test source, in general, comply with the target latency in each case (0.2, 0.5, 1 and 2 s).

Additionally, tests have been carried out in a real Wi-Fi environment without restrictions, as well as in two scenarios (ladder and sinkhole) where variations in the available bandwidth occur. The proposed solution has been compared to the L2A and LoL+ algorithms included in the Dash.js player. Analysing the results, it can be seen that the SSRS solution is the option that offers the best trade-off among the evaluated parameters. Specifically, in the ladder scenario, SSRS provides low values of latency (1.47 s) with a good displayed rate (688.52 kbps), in contrast to a latency of 4.58 provided by L2A and a displayed bitrate of 250 kbps of the LoL+ algorithm. In the sinkhole scenario, SSRS provides an average displayed bitrate of 709.02 kbps (close to 774.19 kbps provided by L2A) with an average and maximum latency (1.42 s and 2.41 s, respectively) much lower than the average and maximum latency of L2A (5.80 s and 10.45 s, respectively).

In conclusion, the work presents a promising approach for improving the user experience quality in real-time transmission systems and suggests possible future research lines in this area. The source code needed to deploy and run the presented LL-DASH testbed and the integration with RTSP and SRT source protocols is available on GitHub [36].

An interesting future work includes the development of a content source agnostic measurement system to be able to measure the delay in an automated way. Moreover, proposing new algorithms based on the SSRS solution from the server-side point of view is a worthwhile approach that can be explored further.

CRediT authorship contribution statement

Román Belda: Conceptualization, Methodology, Investigation, Software, Validation, Writing – original draft, Writing – review & editing. **Pau Arce:** Conceptualization, Methodology, Investigation, Software, Validation, Writing – original draft, Writing – review & editing. **Juan Carlos Guerri:** Conceptualization, Investigation, Software, Validation, Supervision, Writing – review & editing. **Ismael de Fez:** Conceptualization, Methodology, Investigation, Software, Validation, Writing – original draft, Writing – review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

I have shared the code in the git repository indicated in the paper

Acknowledgments

This work is supported by the Centro para el Desarrollo Tecnológico Industrial (CDTI) from the Government of Spain under the project “Nueva plataforma a bordo basada en redes 5G y Wi-Fi 6 para medios de transporte terrestre” (CDTI IDI-20210624).

References

[1] ISO/IEC 23009-1:2014. 2014. Dynamic adaptive streaming over HTTP (DASH) - Part 1: media presentation description and segment formats.
[2] A. Bentaleb, B. Taani, A.C. Begen, C. Timmerer, R. Zimmermann, A survey on bitrate adaptation schemes for streaming media over HTTP, IEEE Commun. Surv. Tutor. 21 (1) (2019) 562–585, https://doi.org/10.1109/COMST.2018.2862938.
[3] HTTP Live Streaming, second ed. (Internet-Draft), Internet Engineering Task Force (IETF), 2022.
[4] T. Stockhammer, C. Poole, T. Swindells, W. Law, I. Sodagar, A. Begen, T. Lohmar, and K. Hughes. 2017. DASH-IF/DVB Report on Low-Latency Live Service with DASH.
[5] DASH Industry Forum, Guidelines For Implementation: DASH-IF Interoperability Points, DASH Industry Forum, 2018.
[6] W. Law. 2020. Meeting live broadcast requirements – the latest on DASH Low Latency. DASH Industry Forum.
[7] DASH Industry Forum. 2020. Low-latency Modes for DASH. Retrieved from https://dashif.org/docs/CR-Low-Latency-Live-r8.pdf.
[8] A. Bentaleb, C. Timmerer, A.C. Begen, R. Zimmermann, Bandwidth prediction in low-latency chunked streaming, in: 29th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'19), Amherst, MA, USA, ACM, New York, NY, USA, 2019, p. 7, https://doi.org/10.1145/3304112.3325611.
[9] A. Bentaleb, A.C. Begen, S. Harous, R. Zimmermann, Data-driven bandwidth prediction models and automated model selection for low latency, IEEE Trans. Multimed. 23 (2021) 2588–2601, https://doi.org/10.1109/TMM.2020.3013387.
[10] T. Lyko, M. Broadbent, N. Race, M. Nilsson, P. Farrow, S. Appleby, Llama - low latency adaptive media algorithm, in: Proceedings IEEE International Symposium on Multimedia, ISM 2020, 2020, pp. 113–121, https://doi.org/10.1109/ISM.2020.00027.
[11] T. Karagkioules, R. Mekuria, D. Griffioen, A. Wagenaar, Online learning for low-latency adaptive streaming, in: Proceedings of the 11th ACM Multimedia Systems Conference (MMSys '20), New York, NY, USA, Association for Computing Machinery, 2020, pp. 315–320, https://doi.org/10.1145/3339825.3397042.
[12] A. Bentaleb, M.N. Akcay, M. Lim, A.C. Begen, R. Zimmermann, Catching the moment with LoL+ in twitch-like low-latency live streaming platforms, IEEE Trans. Multimed. 24 (2022) 2300–2314, https://doi.org/10.1109/TMM.2021.3079288.
[13] C. Gutterman, B. Fridman, T. Gilliland, Y. Hu, G. Zussman, Stallion: video adaptation algorithm for low-latency video streaming, in: Proceedings of the 11th ACM Multimedia Systems Conference (MMSys '20), New York, NY, USA, Association for Computing Machinery, 2020, pp. 327–332, https://doi.org/10.1145/3339825.3397044.
[14] A.C. Begen, M.N. Akcay, A. Bentaleb, A. Giladi, Adaptive streaming of content-aware-encoded videos in Dash.js, SMPTE Motion Imaging J. 131 (4) (2022) 30–38, https://doi.org/10.5594/JMI.2022.3160560.
[15] Y. Li, X. Zhang, C. Cui, S. Wang, S. Ma, Fleet: improving quality of experience for low-latency live video streaming, in: IEEE Transactions on Circuits and Systems for Video Technology, 2023, https://doi.org/10.1109/TCSVT.2023.3243901.
[16] B. Taraghi, H. Hellwagner, C. Timmerer, LLL-CAdViSE: live low-latency cloud-based adaptive video streaming evaluation framework, IEEE Access 11 (2023) 25723–25734, https://doi.org/10.1109/ACCESS.2023.3257099, 2023.
[17] K. Arunruangsirilert, B. Wei, H. Song, J. Katto, Performance evaluation of low-latency live streaming of MPEG-DASH UHD video over commercial 5G NSA/SA network, in: Proc. of the 2022 International Conference on Computer Communications and Networks (ICCCN), Honolulu, HI, USA, 2022, pp. 1–6, https://doi.org/10.1109/ICCCN54977.2022.9868877.
[18] R. Viola, Á. Martín, J.F. Mogollón, Á. Gabilondo, J. Morgade, M. Zorrilla, J. Montalbán, P. Angueira, Adaptive rate control for live streaming using SRT protocol, in: IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, BMSB, 2020, https://doi.org/10.1109/BMSB49480.2020.9379708.
[19] R. Viola, Á. Gabilondo, Á. Martín, J.F. Mogollón, M. Zorrilla, J. Moltalbán, QoE-based enhancements of Chunked CMAF over low latency video streams, in: IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, BMSB, 2019, https://doi.org/10.1109/BMSB47279.2019.8971894.
[20] Y. Li, X. Zhang, S. Wang, S. Ma, A fuzzy-based adaptation controller for low latency live video streaming, in: Proc. of the 2021 IEEE International Conference on Image Processing (ICIP), Anchorage, AK, USA, 2021, pp. 2169–2173, https://doi.org/10.1109/ICIP42928.2021.9506065.
[21] F. Tashtarian, A. Bentaleb, A. Erfanian, H. Hellwagner, C. Timmerer, R. Zimmermann, HxL3: optimized delivery architecture for HTTP low-latency live streaming, IEEE Trans. Multimed. (2022), https://doi.org/10.1109/TMM.2022.3148587.
[22] N. Bouzakaria, C. Concolato, J. Le Feuvre, Overhead and performance of low latency live streaming using MPEG-DASH, in: Proc. IISA 2014 - 5th International Conference on Information, Intelligence, Systems and Applications, 2014, pp. 92–97, https://doi.org/10.1109/IISA.2014.6878732.

[23] K. Durak, M.N. Akcay, Y.K. Erinc, B. Pekel, A.C. Begen, Evaluating the performance of apple's low-latency HLS, in: IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP), 2020, https://doi.org/10.1109/MMSP48831.2020.9287117.

[24] A. Bentaleb, Z. Zhan, F. Tashtarian, M. Lim, S. Harous, C. Timmerer, H. Hellwagner, R. Zimmermann, Low latency live streaming implementation in DASH and HLS, in: Proc. of the 30th ACM International Conference on Multimedia (MM '22), New York, NY, USA, Association for Computing Machinery, 2022, pp. 7343–7346, https://doi.org/10.1145/3503161.3548544.

[25] A. El Essaili, T. Lohmar, M. Ibrahim, Realization and evaluation of an end-to-end low latency live DASH system, in: Proc. IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, BMSB, 2018, https://doi.org/10.1109/BMSB.2018.8436922.

[26] P. Guzmán, P. Arce, J.C. Guerri, Automatic QoE evaluation of DASH streaming using ITU-T standard P.1203 and google puppeteer, in: Proc. of the 16th ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks (PE-WASUN '19), New York, NY, USA, Association for Computing Machinery, 2019, pp. 79–86, https://doi.org/10.1145/3345860.3361519.

[27] T.R. Vargas, J.C. Guerri, P. Arce, Study on the impact of DASH streaming services using energy efficient ethernet, in: Proc. of the 18th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks (PE-WASUN '21), New York, NY, USA, Association for Computing Machinery, 2021, pp. 89–94, https://doi.org/10.1145/3479240.3488527.

[28] Bitmovin Video Developer Report. 2021. Retrieved from https://bitmovin.com.

[29] E. Thomas, M.O. van Deventer, T. Stockhammer, A.C. Begen, J. Famaey, Enhancing MPEG DASH performance via server and network assistance, SMPTE Motion Imaging J. 126 (1) (2017) 22–27, https://doi.org/10.5594/JMI.2016.2632338.

[30] J.W. Kleinrouweler, B. Meixner, P. César, Improving video quality in crowded networks using a DANE, in: Proc. of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video, 2017, pp. 73–78, https://doi.org/10.1145/3083165.3083167.

[31] A.H. Zahran, J.J. Quinlan, K.K. Ramakrishanan, C.J. Sreenan, SAP: stall-aware pacing for improved DASH video experience in cellular networks, in: Proc. ACM MMSys, 2017, pp. 13–26, https://doi.org/10.1145/3083187.3083199.

[32] International Organization for Standardization. 2017. Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 5: server and network assisted DASH (SAND). ISO/IEC 203009-5:2017. Available online at: https://www.iso.org/standard/69079.html.

[33] FastAPI webpage. 2023. Retrieved from https://fastapi.tiangolo.com.

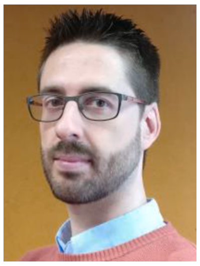
[34] FFmpeg webpage. 2023. Retrieved from https://ffmpeg.org.

[35] J.-B. Kempf, Implementing DASH Low Latency in FFmpeg, DASH Industry Forum, 2020.

[36] Git repository – robelor. 2023. Fast-II. Retrieved from https://github.com/robelor/fast-ll.

[37] DASH Industry Forum. 2023. Reference Client 4.7.0. Retrieved from: https://reference.dashif.org/Dash.js/nightly/samples/dash-if-reference-player/index.html.

Román Belda received the Computer Science degree in 2004 and the M.S. in Telematics in 2013 from the Universitat Politècnica de València (UPV), Valencia, Spain. In 2021 he obtained his Ph.D. in Telecommunications from the UPV. He currently works as a researcher at the Institute of Telecommunications and Multimedia Applications (iTEAM) and as a Lecturer at UPV and Universidad Internacional de Valencia. His areas of interest are mobile applications and multimedia transmission protocols.



Pau Arce received his Telecommunications Engineering degree and the M.S. in Telematics from the Universitat Politècnica de València (UPV), Spain, in 2005 and 2007 respectively. In 2014 he obtained his Ph.D. in Telecommunications from the UPV. Currently he is assistant professor and works as a researcher at the Institute of Telecommunications and Multimedia Applications (iTEAM). His research interests include multimedia QoS, routing on wireless ad hoc networks and performance evaluation of computer systems.

Juan Carlos Guerri received the M.S. and Ph.D. (Dr.Ing.) degrees, both in Telecommunication Engineering, from the Universitat Politècnica de València (UPV), Valencia, Spain, in 1993 and 1997, respectively. Since 2017 he has held the position of University Professor in E.T.S. Telecommunications Engineering at the UPV, where he leads the Multimedia Communications research group (COMM) of the iTEAM Institute. Currently the group's research lines are focused on the development of multimedia content distribution systems using adaptive systems (DASH, DASH LL) as well as on the evaluation of the quality of experience (QoE) through objective and subjective measures. His areas of interest also include the performance evaluation of new video codecs (such as VCC, LCEVC) as well as the design of codecs based on Artificial Intelligence.

Ismael de Fez received the Telecommunications Engineering degree and the M.S. degree in Telematics from the Universitat Politècnica de València (UPV), Valencia, Spain, in 2007 and 2010, respectively. In 2014, he obtained his Ph.D. in Telecommunications from the UPV. Currently, he is a researcher at the Multimedia Communications research group (COMM) of the Institute of Telecommunications and Multimedia Applications (iTEAM), UPV. His areas of interest are multimedia transmission over wireless networks and file transmission over unidirectional environments.