Overhead and Performance of Low Latency Live Streaming using MPEG-DASH

Nassima Bouzakaria, Cyril Concolato, Jean Le Feuvre
Institut Mines-Télécom; Telecom ParisTech; CNRS LTCI
46, rue Barrault, 75013 Paris, France
{nassima.bouzakaria, cyril.concolato, jean.lefeuvre}@telecom-paristech.fr

***Abstract*— HTTP Streaming is a recent topic in multimedia communications with on-going standardization activities, especially with the MPEG DASH standard which covers on demand and live services. One of the main issues in live services deployment is the reduction of the overall latency. Low or very low latency streaming is still a challenge. In this paper, we push the use of DASH to its limits with regards to latency, down to fragments being only one frame, and evaluate the overhead introduced by that approach and the combination of: low latency video coding techniques, in particular Gradual Decoding Refresh; low latency HTTP streaming, in particular using chunked-transfer encoding; and associated ISOBMF packaging. We experiment DASH streaming using these techniques in local networks to measure the actual end-to-end latency, as low as 240 milliseconds, for an encoding and packaging overhead in the order of 13% for HD sequences and thus validate the feasibility of very low latency DASH live streaming in local networks.**

Keywords—HTTP Streaming; Live Streaming; Low Latency; MPEG-DASH; Video Encoding; Overhead.

I. INTRODUCTION

HTTP Streaming technologies have been introduced recently to deliver multimedia streams, taking into account the constraints of today's networks. They try to overcome the deployment issues of other protocols such as RTP/RTCP/RTSP in environments where firewalls, Network Address Translation (NAT) or UDP traffic filtering are used. HTTP Streaming offers similar features to RTP/RTCP/RTSP streaming as it can be used for on demand or live services, and can be adaptive to network bandwidth fluctuations. However, some key factors are differentiating: HTTP streaming leverages existing HTTP infrastructures (proxies, caches, content delivery networks) to make an efficient use of the network when targeting a large number of clients, in a way similar to multicast streaming but without the deployment issues; relies on a client-centric approach to perform network adaptation, similar to using multiple multicast RTP streams and letting the client decide; and eases content repurposing from live to on-demand and vice-versa. An important standard in the field of HTTP streaming is the MPEG Dynamic Adaptive Streaming over HTTP (DASH) standard [5].

While packet-based streaming solutions, e.g. using RTP, can achieve latency, in the order of frames, HTTP streaming solutions such as DASH are not used today for such very low latency streaming. The major reason for that is that HTTP streaming relies on a segmentation process, whereby encoded media frames are aggregated into segments (in the DASH terminology) used as: a download unit in HTTP requests; a buffering unit to smooth network bandwidth variations; an indication of the boundaries to perform seamless switching between streams encoded with different bitrates. For these purposes, segments typically start with a Random Access Point (e.g. an IDR frame in the AVC coding format), and last for a few seconds. Such encoding and segmentation therefore introduce a delay which is not acceptable for low latency streaming, and in particular for live.

Traditionally, very low latency steaming is required for interactive or bidirectional applications such as video conferencing or live streaming with voting. Despite the benefits of HTTP streaming explained earlier, DASH is not initially adapted for such low latency. In this paper, we would like to investigate how to achieve very low latency, i.e. latency similar the one achievable with RTP, but using DASH, to benefit from its advantages in particular, in scenarios such as interactive streaming or hybrid delivery. In the hybrid delivery scenario, where DASH streaming over broadband network is combined with a broadcast service, the latency of the DASH system should be lower than the broadcast, and if no additional buffer is introduced in the broadcast (at the client side or at the encoder side), this means that the DASH system should achieve very low latency. Even in local area networks, where the network jitter is small but where bandwidth can vary (for example when shared between users), adaptive HTTP streaming can still be useful and could benefit from very low latency, e.g. in a local broadcast of a live event.

In this paper, we push the use of DASH to its limits to evaluate different aspects related to latency. We consider the use of DASH over HTTP 1.1 where "chunked-transfer encoding" is used and rely on specific parts of segments, called fragments, being downloaded before entire segments are ready. Additionally, to have meaningful low latency in DASH, we use low latency video coding tools, in particular the Gradual Decoding Refresh feature of the AVC standard. In this paper, we will evaluate the usefulness of this approach both in terms of latency and overhead.

Section II of this paper will present the state of the art in HTTP live streaming. Section III will describe our approach and propose an evaluation of the introduced overhead. Section IV will describe some experiments made to validate the approach and Section V will conclude the paper and propose future work.

II. RELATED WORKS

Several research papers have been published regarding live or low latency streaming over HTTP. Lohmar et al. [4] proposed an analysis of the different delays in an HTTP streaming chain and compared it with an RTP streaming chain. They found that an HTTP streaming chain with typical segment duration of 1s would introduce a delay in the order of 3s compared to RTP. These additional seconds come from: the segmentation process, a segment is advertised only when it is fully produced; the uncertainty related to when the segment is fetched; and the download time of the segment. The paper also proposes a measurement of the overhead showing that the overhead of HTTP and ISOBMFF-based media files decreases with the segment size and can be lower than RTP for segments longer than 2s.

Swaminathan et al. [3] proposed a low latency HTTP streaming approach using HTTP chunked-transfer encoding, and an analytic model to evaluate different client/server communication strategies. The paper showed that with chunked-transfer encoding and with a proper download strategy, the latency does not depend on the segment duration, as shown in [4], but depends on the duration of the HTTP chunks, while also preserving a small initial delay. However, the proposed approach still uses long chunks of 1s, and does not describe what happens for shorter chunks.

Introducing a low latency streaming technique is only meaningful if the associated media coding is also low latency. It would be useless to provide a means to deliver a video sequence frame by frame if the encoder used several future reference frames. Additionally, if the client cannot process the initial frames it receives because they are not Random Access Points (RAP), the streaming system would be inappropriate. Inserting RAPs too often, at worse at every frame, would increase the bitrate tremendously. Fetching previous RAPs and decoding faster than real-time is also an option but requires higher processing on the client. The Gradual Decoding Refresh (GDR) concept of the Advanced Video Coding (AVC) standard is interesting in this respect. Hannuksela et al. presented this tool and studied the associated overhead in [1] and found that "the average bitrate loss of GDR compared to periodic IDR was between 11 and 17%". The paper however does not consider its use in HTTP streaming and does not evaluate the overhead for high-definition sequences.

Kofler et al. [6] studied the impact of the use of the ISO Base Media File Format (ISOBMFF), in HTTP streaming systems, including DASH, but when delivering videos encoded using the Scalable Video Coding (SVC) standard. The authors measured the overhead introduced by the ISOBMFF and the HTTP requests and report that the approach is inefficient for bitrates lower than 1Mbps.

As a summary, to the best of our knowledge, there is no existing research work studying the overhead and appropriateness of HTTP streaming using DASH with the combined usage of HTTP "chunked-transfer" encoding, GDR encoding and the ISO Base Media File Format. This is the goal of this paper.

III. LIVE DASH STREAMING LATENCY

A. Basic DASH Latency

The DASH standard relies on a client-driven streaming approach described in [5]. The client first fetches a description of the streaming session: the Media Presentation Description (MPD). It parses it and chooses the best representation suiting its needs. A representation is one of the encoded media streams with unique characteristics, such as bitrate, resolution or language. Then the client starts requesting segments from the server. A media segment is a part of a stream with a unique HTTP address, packaged for delivery and starting with a RAP. To respect real-time playback, the client continuously compares the download duration of each segment, the segment playback duration and its buffer occupancy, and in some cases, it switches to another representation which best matches its needs. Finally, if indicated by the server, the client updates the MPD from time to time to retrieve new segment information. As in RTP-based streaming systems, a DASH client uses a buffer for two purposes: to adjust to network jitter; and to cope with encoding constraints when bidirectional predicted frames or variable bit rate are used. The DASH client is informed of this latter part through the *minBufferTime* attribute in the MPD.

To enable the client to determine precisely when a new segment is ready and make the necessary request, DASH relies on the *availabilityStartTime* attribute in the MPD. It indicates the UTC time at which the first segment is entirely made available. Hence, as opposed to other HTTP streaming approaches, DASH requires that both servers and clients are synchronized on a common clock, the UTC clock. This approach enables clients to make only the necessary requests for segments, at the right time.

B. Low Latency DASH

Following the above description and in accordance to Lohmar et al. [4], the latency in DASH is affected by: the segmentation delay; the asynchronous fetch of media segments; the time to download the segments; and the buffering at the client side. This analysis can also be applied when replacing segments with smaller chunks of data. As indicated in [3], if segments are further divided in smaller parts and these parts delivered using HTTP chunks, the segmentation delay can be reduced to the duration of a chunk. Additionally, in cases where segments can reliably be produced at the precise times indicated by the MPD, the delay due to the asynchronous fetch can also be reduced. And in local networks, where the jitter is smaller, the buffering at the client side can also be drastically reduced. So under some circumstances, it can be possible to use DASH for very low latency systems.

However, in the first version of the DASH standard, the MPD only indicates the *availabilityStartTime* (AST) value. The client having no knowledge of how segments are produced, e.g. if they are available progressively, it typically sends a request only when an entire segment has been generated. This introduces a latency of one segment duration at least (d_s). To reduce this latency, we participated in the definition of the amendment 1 of the DASH standard proposing the introduction of the new *availabilityStartTimeOffset* (ASTO) attribute in the MPD.

The *availabilityStartTimeOffset* attribute indicates the difference between the *availabilityStartTime* of the segment and the UTC time at which the server can start delivering data for this segment, e.g. using HTTP chunks. Typically, this latter time corresponds to the time at which one or more fragments are available. This is a fundamental change: with the presence of this attribute, a client is now aware that a fragment of the segment is available earlier than the segment. The client is also capable of making the necessary request for the current segment at the right time that will not have him wait or that will not return an HTTP 404 response, although the segment is not fully produced. This however requires that the server is able to send out the fragment earlier, possibly as soon as it has been completely generated, e.g. using HTTP chunks, or that the server can keep the client waiting until the segment is produced, as in the "server wait" approach described in [3]. If the *availabilityStartTimeOffset* is chosen to match the time at which the first fragment is fully produced, the latency can be reduced to the duration of a fragment (dc). The relationship between the *availabilityStartTime*, *availabilityStartTimeOffset*, segment duration (d_s) and fragment duration (d_c) is shown in Fig. 1. λ represents a margin introduced in the computation of ASTO to cope with UTC mismatch between the client and the server.

$$ASTO = AST - (AST - d_s + d_c + \lambda) = d_s - d_c - \lambda$$

IV. EXPERIMENTATIONS AND RESULTS

A. Implementation

To experiment with the proposed approach for low latency live DASH streaming, we have implemented a complete DASH streaming system, as depicted in Fig. 2 and detailed below. We rely on a specific behavior of the DASH encoder/server. The DASH encoder produces DASH compliant segments as depicted in Fig. 3, composed of multiple fragments, produces MPD according the amendment 1 of DASH and sets the *availabilityStartTimeOffset* as being the generation end time of the first fragment of the segment minus the margin. If an HTTP 1.0 client, i.e. not capable of using HTTP chunks, requests the segment before it is fully produced, the DASH server will simply make the client wait until the segment is fully produced. This is similar to the "server wait" approach described in [3]. However, if the client is HTTP 1.1 capable, the server will start sending chunked segments immediately. As indicated in [3], such approach should lead to a streaming system with latency in the order of the duration of a chunk.

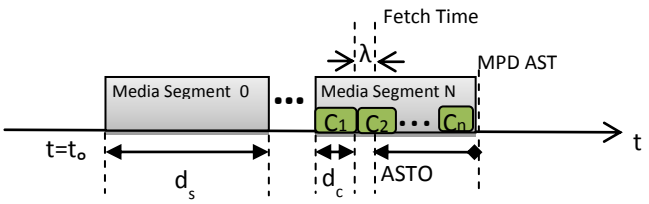

Fig. 1. Determination of the availability time of a media fragment in DASH.

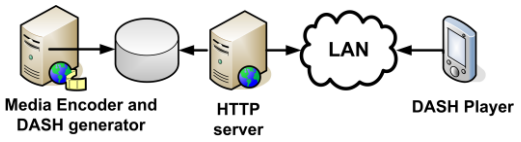

Fig. 2. Architecture of the experimented streaming system.

1) Content generation: The content generation part of the system is in charge of three tasks: encoding the video in real-time, in particular using the GDR scheme; segmenting and formatting the video segments according to the ISOBMFF; and generating the DASH Media Presentation Description (MPD).

In these experiments, we use the DashCast tool from the GPAC[a] project to encode the input video into multiple DASH representations, all using GDR encoding, with different resolutions and bitrates, as suggested in [7]. Additionally, we have configured DashCast to produce ISOBMFF segments of 2s duration. Segments are composed of several fragments. Each fragment consists of at least two boxes represented by the codes "moof" and "mdat", as depicted in Fig. 3. In our experiments, we have used different number of fragments per segments, ranging from 1 fragment carrying one video frame to 1 fragment carrying the whole segment.

2) Content distribution: The segmented media and associated MPD are then deployed on the Web Server and are then fetched by the client through a Local Area Network (LAN). In this work, we have implemented an intelligent web server based on the NodeJS[b] framework. This server serves media segments in a specific manner. When it receives a request from a client for a specific and (fully or partially) available segment, it indicates that the data will be sent using HTTP 1.1 "chunked-transfer" encoding, then starts the parsing of the segment to detect ISOBMFF fragments. When a new fragment is published, the server sends out the fragment as a chunk. With this approach, the download of the segment can start before the segment is completely ready and published. In our system, the server detects the end of a segment by the presence of a new "eods" box (End of Dash Segment) specifically introduced for the coordination between DashCast and our Web server. This approach is similar to the one used in [3] (and the so-called "post metadata") with the difference that our server is located on the same physical machine as the encoder and therefore communication between the encoder and server is done through disc input-output monitoring.

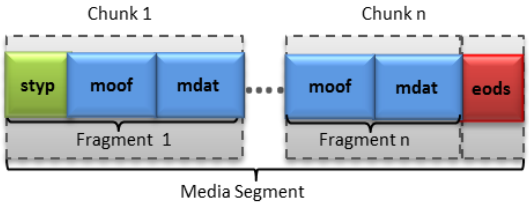

Fig. 3. Structure of a DASH media segment.

[a] http://gpac.wp.mines-telecom.fr

[b] http://nodejs.org/

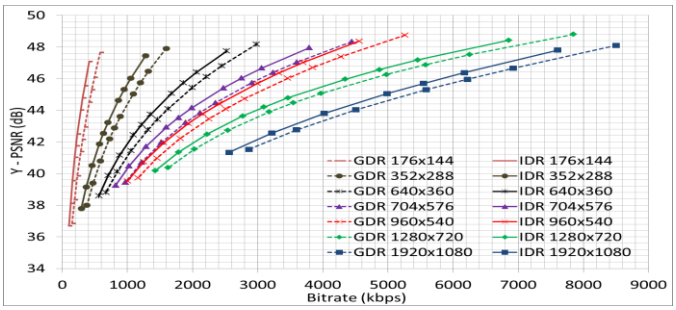

Fig. 4. Comparison of the GDR/IDR encoding of the Big Buck Bunny sequence.

In future work, we plan to extend this approach to separate the encoder from the web server.

3) DASH client: In our approach, the client is a compliant DASH client with HTTP 1.1 capability, i.e. it is able to receive data transferred using chunked-transfer encoding; and is capable of processing incomplete segments, i.e. media frames are parsed in ISO fragments and dispatched before the complete segment is received.

B. Results

In order to validate our approach, we have conducted two types of experiments: experiments to measure the overhead introduced by the selected coding and packaging tools, and experiments to measure the latency of system. This section details these two parts.

1) Overhead measurements: The total overhead of our approach can be decomposed into: the overhead introduced by the GDR video coding scheme; the overhead introduced by the packaging and segmenting of GDR-encoded videos into ISOBMFF fragments; and the overhead introduced by the download of ISOBMFF fragments as HTTP 1.1 chunks.

a) Encoding overhead: For our experiments, we used two video sequences, initially compressed with the AVC format, with the characteristics reported in TABLE II. We encoded these sequences, using the open source x264[c] encoder, at different resolutions (ranging from QCIF to full HD) and different bitrates (ranging from 150kbps to 9Mbps).

The video sequences were encoded in two modes: with and without GDR. When using IDR, we set the GoP size to be 1 second, corresponding to a typical DASH segment starting

TABLE I. BITRATE FOR DASH REPRESENTATIONS

Resolution	Bitrates (kbps)
1920x1080	8000
1280x720	4500
704x576	2000
960x540	2250
640x360	1600
352x288	1200
176x144	100

[c] https://www.videolan.org/developers/x264.html

TABLE II. INPUT VIDEO SEQUENCE CHARACTERISTICS

Sequence	Bit rate (kbps)	Frame rate (fps)	Resolution	GoP size	Duration
RedBull[d]	6 000	24	1920x1080	15 s	1 min
Big Buck Bunny[e]	9 000	29.97	1920x1080	3 s	10 min

with a RAP. With GDR, we set the roll distance to 8. In both cases, no B frames were used and only 1 reference image was used for prediction. Additionally, we used the "crf" option of x264, which tries to achieve a given quality and used the same quality for both GDR and IDR versions. We kept only the sequences which resulted in a bit rate lower than the initial one. The exact command line for GDR encoding is provided below:

```
x264 --sar 1:1 -o output_video.h264 input_video.h264
--ref 1 --crf <x> --bframes 0 --keyint 8 --vf
resize:<w>,<h> --b-pyramid none --fps <f> --preset
veryslow --tune psnr --psnr --intra-refresh
```

Fig. 4 shows the PSNR comparison for the Big Buck Bunny sequence. The comparison for the Red Bull sequence exhibits the same pattern. The results, summarized in TABLE III. , confirm the finding of [1] but extend these results for HD content: the GDR coding incurs an overhead, computed following Bjontegaard metric [8], between 7% and 25% at roughly constant quality or a PSNR difference between 0.44 dB and 2.55 dB at equivalent bitrate. It can be seen that the GDR encoding is less penalizing at higher resolutions, probably because the space for inter/intra prediction is larger, given that the same number of GDR frame divisions has been used for all resolutions.

b) ISOBMFF overhead: The packaging in ISO Base Media files used in our DASH streaming system also introduces an overhead which can be decomposed in: an initial overhead due to the packaging of raw media data (e.g. AVC NAL Units) into the structured ISO format; a specific overhead for the storage of GDR encoded videos; and the additional overhead due to the segmentation and fragmentation required in DASH.

The initial overhead does not depend on the bitrate but does depend on the number of ISO samples, i.e. the number of frames per seconds. Our experiments on the Big Buck Bunny and Red Bull sequences show that the additional overhead introduced by the simple storage of IDR or GDR encoded sequences in MP4 files compared to the raw AVC sequence is negligible. The overhead is of course the biggest for the sequence with lowest bitrate (e.g. 150 kbps). Typically, across video sequences, resolutions and bitrates, we have between 0.0019% and 0.26% for the storage of IDR sequences in ISOBM files compared to raw AVC files. The specific overhead introduced in GDR compared to AVC is between 0.0042% and 0.37%. Hence, storing GDR sequences costs a lot more than storing IDR sequences, the overhead is

[d] http://download.tsi.telecom-paristech.fr/gpac/dataset/dash/mmsys13/video/redbull_10sec/

[e] http://www.bigbuckbunny.org/index.php/download/

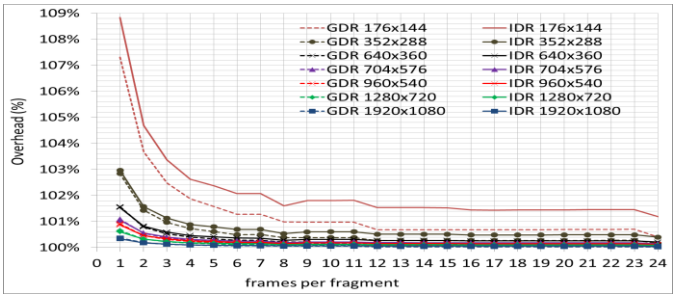

Fig. 5. Comparison of the overhead introduced by the ISOBMF storage and fragmentation of GDR/IDR streams.

sometimes more than twice, but compared to the video bitrate, this overhead is still negligible.

When it comes to fragmented MP4, considering a constant duration segment of 1s, the overhead depends on the number of fragments per segment, i.e. on the number of frames per fragment, and on the difference between the GDR and IDR signaling.

Fig. 5 shows the result for the Big Buck Bunny sequence, with the overhead computed with respect to the raw AVC sequence. We can see that the fragmentation introduces an overhead, which decreases as the number of frames per fragment increases. For all resolutions and bitrates, it roughly is fewer than 2%, except for fragments smaller than 4 frames, where it can reach up to 9%. For this reason, in section 2, we will use fragments made of 5 frames. Then, we can see that, for a given number of frames per fragment, the fragmentation overhead decreases with the video resolution and bitrate. Interestingly, since the GDR coding overhead is high for the sequence with the lowest resolution (25-28%), the additional fragmentation overhead is relatively smaller (7%) than the overhead introduced by the fragmentation on the IDR sequence (9%).

c) HTTP overhead: Finally, the last overhead introduced by the system is in the delivery of content over HTTP. In typical DASH scenario, an HTTP request is made for every media segment. The size of the request is highly dependent on the information present in the header (descriptions of the user agent, of the server, list of accept headers, use of byte ranges…). In [4], the authors report a typical size of 140 bytes. In [6], the authors report a size of 280 bytes. We can assume an average size of 200 bytes per request. However, in our approach what matters more is the overhead introduced by the chunk-transfer encoding.

TABLE III. BJONTEGAARD METRIC FOR GDR/IDR COMPARISON

	Big Buck Bunny		Red Bull	
	delta PSNR (dB)	***Delta bit rate (%)***	***delta PSNR (dB)***	***Delta bit rate (%)***
1920x1080	0.4401	7.0532	0.4554	7.6407
1280x720	0.4437	7.9916	0.4379	8.6911
704x576	0.6596	10.8178	0.6201	11.3784
960x540	0.5827	9.3572	0.5164	9.4989
640x360	0.7747	11.7519	0.6967	12.1939
352x288	1.2582	17.1335	1.0357	16.5746
176x144	2.5585	28.2752	1.8412	24.6922

According to the HTTP 1.1 specification[f], each chunk is composed of a string giving the size of the chunk, followed by extensions, followed by the carriage return and line feed characters (CRLF), the chunk data and another CRLF. The last chunk is a chunk of size 0 followed by an optional trailer and CRLF. Assuming a sequence encoded at 8 Mbps using GDR encoding, the fragmentation for this high bitrate will add an overhead of 2% for 1 frame per fragment, and so the average fragment size will be expressed on 4 hexadecimal characters. In total, for chunks with no extension or trailer and with each chunk corresponding to one fragment, in turn corresponding to either 1 frame or the full segment, each chunk will be at most 8 bytes bigger than the fragment. So at most, at 25 fps, this gives an additional bitrate of 1.6 kbps, representing 0.02% of the raw video bitrate, which can be considered negligible.

d) Total overhead: As a summary, the total overhead for the delivery of a GDR encoded video sequence, stored in fragmented ISOBMF, delivered with HTTP 1.1, using 1 chunk per fragment, is mainly the result of the encoding and of the fragmentation. This overhead will vary, as reported in TABLE IV. between 12.5% and 56.30% of the same sequence encoded without GDR using 1 GoP per fragment and 1 fragment per segment, typical in DASH. We note that the lower overhead is for HD sequences and can be acceptable in some scenarios.

2) Latency: To validate our approach with respect to latency, we have instrumented DashCast (respectively MP4Client) to log the different UTC times at which a frame was encoded (resp. decoded), at which a fragment was fully produced (resp. a chunk was fully received), at which a segment was fully produced (resp. received). We then run streaming sessions with DashCast, our modified Web Server and MP4Client in a local network. MP4Client was configured to remove all network buffering.

We set the segment duration d_s to 2 seconds and the fragment duration d_c (equal to the chunk duration) to 200 milliseconds to obtain 10 chunks per segment. The following DashCast command was run, grabbing the screen of the computer at a resolution of 800x600 pixels, at 25 frames per second, encoding it according to the configuration given in the `dashcast.conf` file with GDR encoding, using '`eods`' marker and with ASTO equal to $d_s - d_c$.

```
DashCast -vf x11grab -v :0 -seg-dur 2000 -frag-dur
200 -gdr -live -conf dashcast.conf -ast-offset -1800
-vres 800x600 -vfr 25 – seg-marker eods
```

TABLE IV. VIDEO SEQUENCE BITRATE OVERHEAD (GDR WITH 1 FRAME PER FRAGMENT VS IDR WITH 1 GOP PER FRAGMENT)

	Big Buck Bunny	**Red Bull**
1920x1080	12.50%	13.39%
1280x720	21.64%	17.60%
704x576	19.56%	19.20%
960x540	17.05%	22.05%
640x360	15.22%	24.00%
352x288	30.45%	32.92%
176x144	56.30%	55.00%

[f] https://tools.ietf.org/html/rfc2616

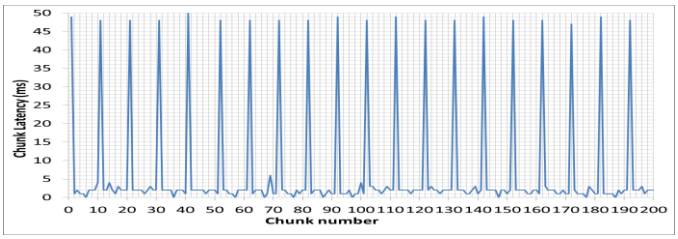

Fig. 6. Chunk latency (fragmentation output/client reception).

DashCast and MP4Client were run on different machines, whose system times where configured to use Network Time Protocol (NTP). However, we noticed important mismatch between the times used by both machines and therefore decided not to rely on NTP and used a dedicated NTP-inspired mechanism to synchronize the machines, as follows. Upon sending the MPD to the client, the web server adds an extra HTTP header indicating its UTC system time. When the client receives the MPD from the server, it fetches its own UTC system time, substracts the time read from the HTTP headers and obtains an estimated UTC time difference between the two machines. This difference is assigned to λ. This mechanism is very simple, and omits the delivery time of the MPD. We plan to improve that in future work. Additionally, it is calculated just once and applied every time the availability start time of a segment is compared to the client system time. In the future, we plan on using the reception times of the chunks to adjust this UTC difference.

Fig. 6 shows the chunk latency, i.e. the difference between the adjusted UTC times at which a fragment was fully produced by DashCast and at which it was received by MP4Client client as an HTTP chunk. As can be seen on the figure, the chunk latency is of the order of 2-3 milliseconds for every chunk, except for the first chunk of each segment, where it is in the order of 50 ms. This is due to the fact that the UTC time adjustment is only approximate. Manual checking shows that the real UTC time difference is 50 ms greater. For this reason, the client actually requests the first chunk 50 ms too late. This latency is however reduced for the next chunk as it is pushed by the server and no request is made by the client.

Fig. 7 shows the frame latency, i.e. the difference between the time at which a frame was encoded and at which it was decoded. With 25 frames per seconds, and 200 ms per chunk, a chunk contains 5 frames. The figure shows 5 peaks, one per frame, corresponding to the latency of each frame in the chunk, the first frame being delayed most (160 ms), and the last suffering almost no latency.

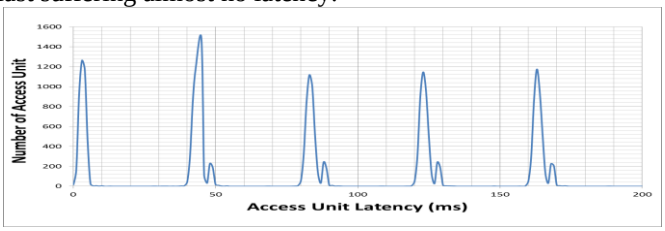

Fig. 7. Access unit latency (encoder output/decoder input).

These experiments validate that, in local networks; very low latency down to 160 ms (5 frames), between the encoder output and the decoder input, can be achieved using DASH amendment 1. Noting that the encoding and decoding times of a frame are lower than its duration (40 ms), the end-to-end latency is lower than 240 ms.

V. CONCLUSION

HTTP Streaming is the new approach for streaming video over the Internet, for live and on demand cases. However, current approaches, in particular using DASH, are not deployed for low latency live services. In this paper, we proposed to use the amendment 1 of DASH in combination with Gradual Decoding Refresh encoding and to deliver media frames up to the frame. We measured the overhead introduced by the GDR encoding and the associated fragmentation. We showed that especially for high definition content, the overhead in the order of 13% can be acceptable. We also described an implementation of a streaming system comprising a DASH live encoder generator, a DASH-aware web server and a DASH client. With this system, we validated the approach for very low latency streaming in local networks, with latency as low as 240 ms. In future work, we plan to examine how such low latency system will behave in real content delivery networks, and to further exploit the combined use of GDR and chunk encoding to enable fetching segments not from their start, reducing the initial delay and enabling faster switching.

REFERENCES

[1] Hannuksela, M.M.; Ye-Kui Wang; Gabbouj, M., "Random access using isolated regions," Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on , vol.3, no., pp.III,841-4 vol.2, 14-17 Sept. 2003, doi: 10.1109/ICIP.2003.1247376

[2] Y.-K. Wang and M.M. Hannukela "Gradual decoder refresh using isolated regions," Joint Video Team document JVT-C074, May 2002. Available online ftp://np.imtc-files.org/jvt-experts/2002-05-Fairfax/JVT-C074.doc, access July 1st, 2013.

[3] Swaminathan, V.; Sheng Wei, "Low latency live video streaming using HTTP chunked encoding," Multimedia Signal Processing (MMSP), 2011 IEEE 13th International Workshop on , vol., no., pp.1,6, 17-19 Oct. 2011, doi: 10.1109/MMSP.2011.6093825

[4] Lohmar, T.; Einarsson, T.; Frojdh, P.; Gabin, F.; Kampmann, M., "Dynamic adaptive HTTP streaming of live content," World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a , vol., no., pp.1,8, 20-24 June 2011, doi: 10.1109/WoWMoM.2011.5986186

[5] Sodagar, I., "The MPEG-DASH standard for multimedia streaming over the internet," MultiMedia, IEEE , vol.18, no.4, pp.62,67, April 2011, doi: 10.1109/MMUL.2011.71

[6] Kofler, I.; Kuschnig, R.; Hellwagner, H., "Implications of the ISO base media file format on adaptive HTTP streaming of H.264/SVC," Consumer Communications and Networking Conference (CCNC), 2012 IEEE , vol., no., pp.549,553, 14-17 Jan. 2012, doi: 10.1109/CCNC.2012.6180986

[7] M. Grafl, C. Timmerer. H. Hellwagner, W. Cherif, A. Ksentini, "Hybrid scalable video coding for HTTP-based adaptive media streaming with high-definition content", in Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), Madrid, Spain, June 2013.

[8] Bjøntegaard G., "Calculation of average PSNR differences between RD-curves", ITU-T Q.6/SG 16 Video Coding Experts Group (VCEG), Doc. VCEG-M33, Austin, Texas, USA, Apr. 2001. Available online http://wftp3.itu.int/av-arch/video-site/0104_Aus/VCEG-M33.doc, accessed July 1st, 2013.