Applicative-Layer Multipath for Low-Latency Adaptive Live Streaming

Patrice Houzé
Orange Labs – B-Com
patrice.houze@orange.com

Emmanuel Mory
Orange Labs
emmanuel.mory@orange.com

Géraldine Texier Gwendal Simon
IRISA/Telecom Bretagne
firstname.lastname@telecom-bretagne.eu

***Abstract*—End-users in online video services are sensitive to the overall quality of the video at screen, but also, and more importantly, to other factors, including the latency between the video generation and the playback for live videos. If the improvement of the quality of the perceived video has been well investigated by the multimedia community, the impact of delay, latency, and re-buffering has not received a significant attention. Adaptive streaming technologies, which have been widely adopted in the recent years, contribute to this lack of consideration since vendors usually recommend the introduction of significant extra-latency. In this paper, we address the problem of low-latency live video streaming. We propose an implementation of multi-path video delivery at the applicative level, which exploits the information provided in the latest version of the video delivery standards. We present an implementation of a video player, which takes advantage of multi-path networking to enable video playback using TCP with a latency below 100 ms. By initiating the multi-path delivery from the client side, our mechanism is compatible with current network equipments and does not require any change neither at the server nor in the middleboxes.**

***Index Terms*—Live video streaming; Multi-path networking; HTTP rate-adaptive streaming**

1. Introduction

Recent studies [5] have demonstrated that the engagement of end-users in online video services is directly linked to the quality of experience (QoE) of the video player. End-users are sensitive to the overall quality of the video at screen, but also, and more importantly, to the *delay* for the video to start, to the *latency* between the video generation and the playback (for live videos) and to the number of interruptions due to video *re-buffering*. The multimedia community has developed multiple techniques to improve rate-distorsion trade-offs, but the other aspects related to delay, latency, and re-buffering have not received a significant attention. This is especially true for the HTTP rate-adaptive streaming (HAS) technologies, which have been widely adopted in the recent years. For example, the vendors of HAS systems for live streaming recommend the introduction of an extra-latency of 10 seconds before starting.

In this paper, we address the problem of low-latency live video streaming. Our goal is to minimize the latency between the time a new video frame is generated at the source, and the time it is played in the end-users' screen. To achieve this objective, we leverage multi-path networking.

Multi-path networking is considered as a promising technique to improve content delivery. The motivation behind multi-path networking comes from the observation that most of the recent devices have several network interface cards (NICs): typically Wi-fi, cellular, and Wi-fi direct. Instead of fetching data from one path only, the idea is to exploit the multiple paths simultaneously. The advantages include a better resilience to changes in network conditions, and consequently a more stable video streaming.

Some multi-path protocols have been developed at the transport level, notably Multi-Path Transmission Control Protocol (MPTCP). Unfortunately, several reasons explain why these protocols have not been successfully deployed in the network yet. First, the performances of MPTCP are below the expectations, especially when the underlying paths have heterogeneous characteristics [2], [13]. Second, MPTCP requires implementations in both end-points but most of the clients' operating systems and most of the providers' servers do not support it yet. Third, many current middleboxes do not support MPTCP. It is typically not possible to use MPTCP in the french cellular networks. Overall, the video providers are reluctant to implement MPTCP in their servers, so the deployment of MPTCP is still restricted to a tiny fraction of the Internet.

The contributions of this paper are as follows:

- We explore an implementation of multi-path video delivery *at the applicative level*. We present a technique, which exploits the information provided in the latest version of the video delivery standards from both ISO base media file format (ISOBMFF) and MPEG dynamic adaptive streaming over HTTP (DASH). We show that it is possible to initiate multi-path delivery from the client side. The main advantage of our technique is that it can be implemented without any change neither at the server nor in the middleboxes. It is thus compatible with current network equipments.
- We apply our applicative-level multi-path delivery to the problem of low-latency live video stream. We present an implementation of a video player, which enables video playback with a latency below 100 ms although state-of-the-art techniques target latency in the order of seconds [1], [8]. We develop a novel

approach of HAS where we adapt the video bit-rate to the transmission delay so that we prevent re-buffering events. More generally, we show that it is possible to implement low-latency adaptive live streaming based on Transmission Control Protocol (TCP).

The proposals that are described in this paper open several perspectives, not only in terms of implementation in practical systems, but also regarding research perspectives. We discuss a set of open problems related to the applicative-level multi-path and to the low-latency video delivery to conclude the paper.

2. Related Works

To the best of our knowledge, no previous study has explored the application of multi-path networking to low-latency video delivery. Thus, we highlight in the following the studies that deal with low-latency video delivery, and then those that deal with multi-path networking.

The implementation of live streaming on the Internet is traditionally based on the Real-Time Protocol (RTP), which was especially designed for end-to-end real time audiovisual data transfer on User Datagram Protocol (UDP). In the recent years, the main content providers have switched to DASH technologies based on TCP. In DASH, the video is cut into *segments* for a better delivery management. For live stream, a segment is available when the process of coding the whole segment terminates. The analysis made in [16] demonstrated that the additional delay due to this segmentation process and to the download time in TCP (including retransmission time) imposes a delay of at least 3 s for a video (provided that the video is cut into 1 s-long segments). The integration of *HTTP chunked transfer encoding* mechanism into DASH delivery is introduced in [9] to reduce the live latency. This integration is developed further in [1], [8], [17]. The main idea is that the server transmits partial segments (a *chunk*) before the entire segment is published, which leads to a reduction of the latency to one to two chunk duration. Moreover the mechanism is independent of the segment duration. In [15], experiments conducted on a local network show that an end-to-end latency of 240 ms can be achieved (with an encoding and packaging overhead of 13%) when the conditions are ideal.

In this paper, we also leverage HTTP chunked encoding but we combine it to multi-path delivery, which enables additional latency reduction. We also develop a novel approach for adaptive streaming where the choice of the video representation depends on frame downloading delay.

The surge of attention on multi-path networking has resulted in many proposals, including MPTCP at the Internet Engineering Task Force (IETF). Unfortunately, various studies have shown that the performances of MPTCP are below the expectations. The MPTCP packet scheduler is identified as the main weakness of the protocol. The packets that have to be transmitted are buffered into the MPTCP *output queue*. Today's schedulers pick the packets in the queue following a first-in first-out (fifo) process, and send them into the TCP *sending buffer* of one of the available links, according to the scheduler policy. However, packet losses occurring on one path can generate *head-of-line* blocking at the receiver side. The literature includes papers on bandwidth and buffer management [6], windows congestion revamping [7], cross-layer scheduling [2] and retransmission processes [11], but none of these works address low-latency for streams. Regarding multi-path delivery for video content, the Multi-Path Real-time Transport Protocol (MPRTP) proposal [14] targets latency among others objectives, but it relies on RTP and UDP, which are not used by today's content providers. Other works (notably [4]) proposes joint optimization of path selection and rate allocation, but the latency is not addressed and these proposals do not deal with implementation issues.

In this paper, we target low-latency. As far as we know, no previous work has proposed to leverage multi-path on TCP for delay reduction. Our objective is thus first to demonstrate that application-level implementation of multi-path delivery is possible, and second to show that multi-path is a way to reduce latency.

3. Background and Model

We introduce now the key concepts behind the problem of latency in video streaming, and especially in DASH. Then, we recall the main principles of video encoding.

3.1. Latency in Live Video Streaming

We define the *latency* as the time difference between the time at which a given frame of the video is generated at the server side and the time at which this said frame is displayed at screen. In the case of interactive multimedia services such as cloud gaming, and conversational services, the goal is to get a latency below 100 ms so that the distance is imperceptible. Subjective studies have shown that latency greater than 200 ms significantly degrades the QoE [3]. The other not-interactive live video services are less sensitive to latency by nature, but low-latency has become a requirement nonetheless, typically for inter-destination synchronisation [12].

To keep it simple, we consider a simple client-server architecture. In today's wide-scale video delivery, the chain of actions involve several actors including video contributors, cloud providers, and content delivery network (CDN). For the sake of clarity, we gather all these actors into an abstracted service provider, which we call *server*. We represent in Figure 1 the chain of the main operations done by the three identified actors: server, network, and client.

Each of the events in Figure 1 takes some time. We go over these times in the following (except for capturing the video flow at the server and playing the decoded stream at the client, which do not take a significant time).

The encoding of a raw video can now be done by software encoders in commoditized computer as long as the quality expectations are not high. For large-resolution videos, for example 4*k*, and high-frame rate videos, for example 60 frames per second (fps), one still has to use modern graphics processing unit (GPU) cards and specific machines. The main challenge is to ensure that one frame

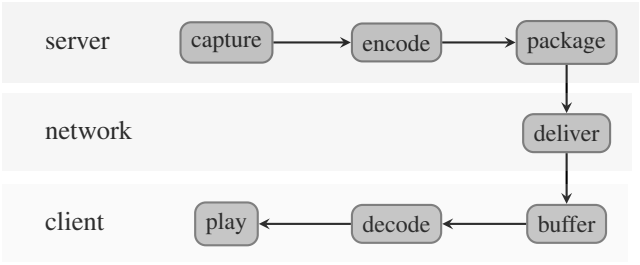

Figure 1. Chain of operations for the delivery of live video

is generated every x ms, where x depends of the frame rate (for example $x = 40$ ms for the traditional 25 fps).

The packaging of video data is a set of operations that is done on the encoded data so that the stream can be interpreted and read by standard video players. The preparation of the content is mainly its integration into a *multimedia container*. In general, the recent video containers follow the ISOBMFF, which specifies a structure and metadata for multimedia content. The format contains a series of *boxes*, which provide information on the timing and the structure of the actual video data (the *mdat* box). The packager is in charge of (i) cutting the video data into *segments* (typically 2 s of video), (ii) setting all the metadata boxes of the container, and (iii) generating the *manifest* file, which provides the necessary information to download consecutive segments and to switch from one video quality to another.

The delivery of data in recent HAS systems is *pull-based*; the client initiates the delivery. We represent in Figure 2 the different delays. Such a representation is trivial but we recall it because, as we will see later, the distinction between the downstream latency and the total download time is key for our multi-path delivery technique.

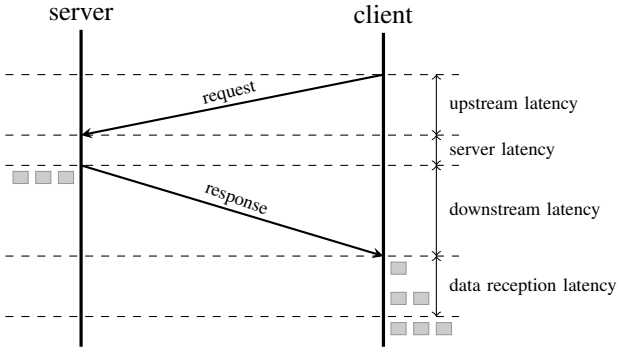

Figure 2. Dissecting network latency for a TCP stream of three packets

At the client side, buffering aims to accommodate burst of traffic, packet losses, and also the variation of bit-rate in the video stream due to the differences between frames. The larger is the buffer, the surer is the video playing but of course the larger is the end-to-end latency. In most of today's client implementations, the size of the buffer is set at the beginning of the session: the client downloads three segments in advance, and then starts playing.

Finally, the decoding is the process of dealing with consecutive frames to re-build a series of images. As explained in Section 3.3, some frames require information from future frames. So, to decode the video, the decoder has to wait for the future frames to arrive. The packager inserts an information, called *minbuffertime*, which says the number of frames that must be buffered for a safe decoding. In general, the *minbuffertime* is in the order of four frames, but it depends on the Group of Picture (GOP) structure.

3.2. Latency in HAS

The HAS technologies, for example DASH, have not been designed for low-latency live streaming. The concept of video segmentation introduces in particular extra-latency since a live video segment should be entirely encoded and packaged before being available for download from the clients. Some proposals have been recently adopted in DASH to reduce the impact of segmentation on the latency.

First, to avoid the announcement of segment availability, the manifest file can now be set as *dynamic* for live streams. The idea is that a template provides the information for all future segments.

Second, to avoid waiting for the whole segment to be completed, the standard has introduced the concept of *sub-segments*, which is a fraction of the segment that can be transmitted as soon as it is encoded. One of the purposes of the introduction of sub-segment is to use HTTP chunked transfer on the sub-segment scale, as proposed in [1], [8]. Two attributes (*availabilityTimeComplete* and *availability-TimeOffset*) have been added so that requests based on HTTP *byte-range* are possible. The fields, called *ssix*, are stored in a specific box of the media container, called *sidx*. The manifest file of a DASH system contains a pointer toward the *ssix* of every available representation.

3.3. Frame Structure in Video Stream

A video encoder converts the original sequence of images (arrays of pixel values) into a video bit-stream. The decoder does the opposite. The idea that is now adopted in video compression is the principle of hierarchical structure of video stream data. The bit-stream is cut into independent GOP, each GOP being cut into frames, which have temporal dependencies with regards to their types: Intra (I), Predicted (P) or Bidirectional (B) pictures.

Since they encode different information, the frames have not the same size. In general, I frames are heavier than P frames, which are heavier than B frames. We show an example in Figure 3 with one GOP randomly chosen from a video. Depending on the encoder, the size differences across frame types can differ. The size difference also depends on the scene that should be encoded and on the synchronization between GOPs and brutal scene changes.

The consequences of the frame size difference include that, with respect to a given bandwidth at the client side and a inter-frame delay x ms, some frames require more than x ms to be fully transferred although some others can be

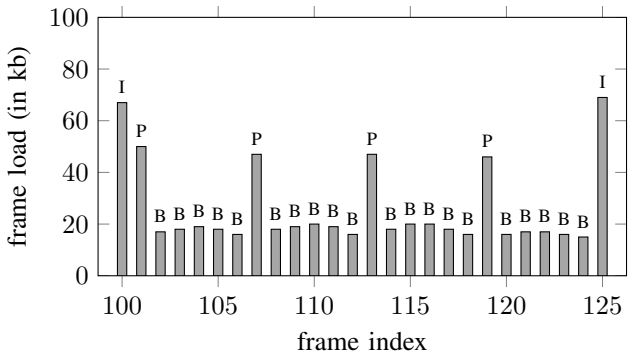

Figure 3. Size of the different frames for one GoP (movie *sintel* encoded at 9,000 kbps and 25 fps, GoP randomly chosen).

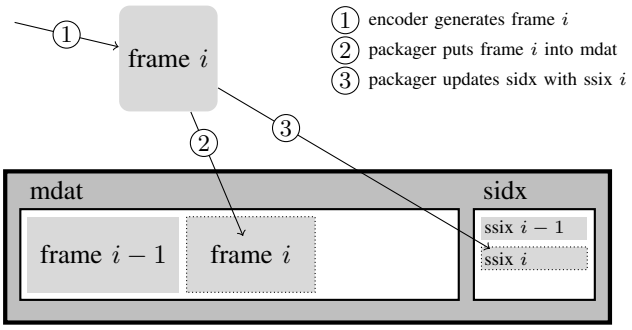

Figure 4. Process at the server side. Here only one segment of one representation is depicted.

transferred in less than x ms. Buffering is the adopted solution to compensate this difference, at the price of additional latency.

4. Our Proposal

We present in this Section our proposal for multi-path delivery of live video. We first give a short description of this technique, and then we go into the implementation details.

4.1. Proposal in a Nutshell

The technique that we propose in this paper requires the following implementation on both server and client sides. Please note that this implementation can be easily done since it is a combination of various technological triggers that are already available in the standards.

4.1.1. Server side. The encoder and the packager should coordinate so that the *ssix* field in the multimedia container always contains the byte-range of all the frames that have been generated. Since the manifest file contains a pointer toward the *sidx* box of each video representation, the client can refresh the information from the *ssix* and be informed about the last frame as soon as it is encoded and available.

For each video representation, the packager gets a new frame every x ms. Once it gets it, its role is to continue building the segment that it prepares according to the ISOBMFF: (i) it adds the new frame to the *mdat* box of the segment, and (ii) it adds in the *ssix* field the offsets that define the byte-range of the new frame in the *mdat* box. These actions should be done as soon as the frame is ready, so every x ms. The whole process is depicted in Figure 4.

At the end of the segment, the *mdat* box is completed and the *sidx* box is conform to the norm. Thus, the segment can be safely stored for further use (time-shifted streaming for example) and re-used by any DASH server.

4.1.2. Client side. The client should implement the recurring process of getting from the *ssix* the information about the new available frame every x ms. Then, the client has to fetch the whole frame before the deadline, which is the time at which this frame should be played. Since we target a minimum latency, the deadline is close. Moreover, the client has to make sure that fetching the current frame does not require more than x ms because a new frame will then be available and will have to be fetched too. Given that some frames are heavier than other, we need a technique to fasten the fetching of some frames (mostly I-frames and some P-frames). Here, we propose to leverage multi-path. The client dispatch the fetching on the available links. For example when two paths are available, a part of the frame is fetched from one path while the remaining of the frame from the other part. Such a dispatching is depicted in Figure 5 in an example where the path 0 has a low downstream latency but the bandwidth is not large and the path 1 has a longer downstream latency but a larger bandwidth. Here, the two paths take exactly the same time to fetch two packets out of the four packets that have to be downloaded. By combining both paths, fetching the four packets is faster than when only one path is used, as shown with the path 0 in Figure 5.

4.2. Implementation

We now refine the description of our proposal.

4.2.1. Byte-Range Dispatching Among Paths. We consider that the client can use a set N of transport paths. When the paths in N are homogeneous, the frame can be divided into byte-ranges of the same size and allocated to the paths. Since the long-term performances of the paths are similar, the short-term performances are close (depending on packet loss), and the byte-range inter-arrival time is small. This case is typical of *DSL bonding*, when several network connections from the same operator (for example in a building) are shared by the clients. However, when the paths have heterogeneous performances, the byte-range fetched on a slow path may arrive later than the byte-range on a fast path, which can result in a head-of-line blocking event. Indeed, the transfer delay depends on the properties of the network and on the amount of concurrent traffic. Hence, we need to allocate larger byte-ranges to the faster paths so that the variation of the transfer delay is absorbed.

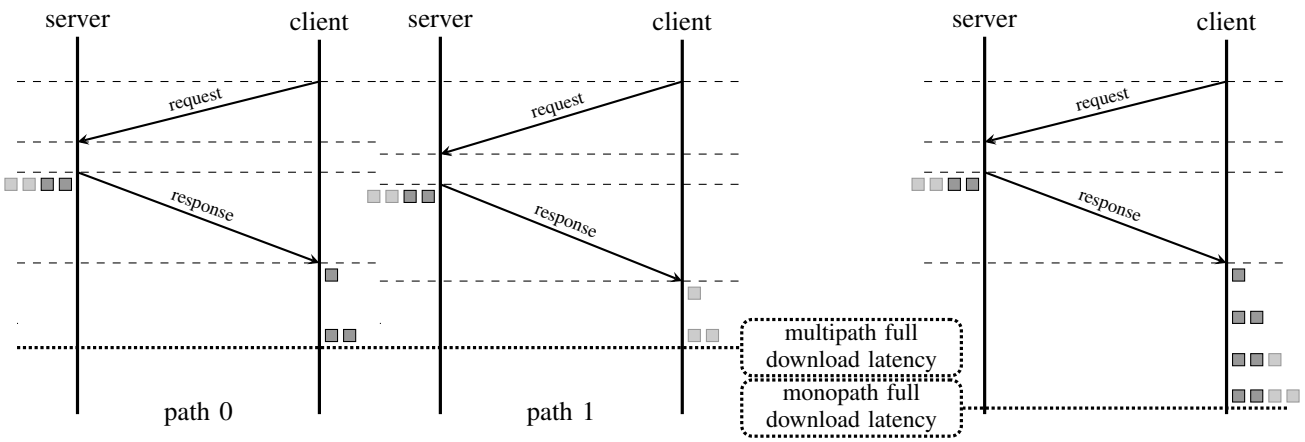

Figure 5. Multi-path delivery. On the left, two paths are used concurrently; the path 0 fetches the dark gray packets while the path 1 fetches the light gray packets. On the right, only one path is used, here the path 0. We show two bye-ranges (dark and light gray), each containing two TCP packets.

Finding the best size of byte-ranges in order to receive them with a small interarrival time when paths have heterogeneous performance can be formulated in an integer linear program (ILP). We aim at minimizing the time needed by the slowest path to fetch the byte-range from the server. We introduce a function f_i, which takes in input a byte-range of size s, which is expressed in a number of Maximum Transmission Unit (MTU), and returns the time t_i needed to get it from the server on the path $i \in N$. We define a binary variable α_{is}, which equals 1 if the byte-range allocated to the path $i \in N$ is of size s, 0 otherwise (for all the other sizes). Constraint (2) ensures that the frame of size S is entirely sent to the client. Constraint (3) ensures that only one byte-range is allocated to the path $i \in N$.

$$\textbf{minimize} \quad \max_{i \in N}(f_i(\alpha_{is} \cdot s)) \tag{1}$$

$$\textbf{s.t.} \quad \sum_{i \in N} s \cdot \alpha_{is} = S, \qquad \forall s \tag{2}$$

$$\sum_{i \in N} \alpha_{is} = 1, \qquad \forall i \in N, \forall s \tag{3}$$

$$\alpha_{is} \in \{0, 1\}, \qquad \forall i \in N, \forall s \tag{4}$$

This ILP is a variant of the well-known knapsack problem, for which efficient solutions have been proposed based on dynamic programming. It is possible to implement an optimal solution of the dispatcher in practice since it runs in polynomial time of the number of paths, which is not expected to be large.

4.2.2. Delay Settings and Adaptive Streaming. Our goal is to minimize the latency so that the playback delay can be also minimized. The playback delay is decided by the video player. In traditional HAS systems, the playback delay is conservatively set in the order of seconds. With our proposal, playback delay in the order of hundreds of milliseconds becomes possible. However, the playback delay is a decision that is taken at the beginning of the streaming session while the client has only fetched one (or a few) frame and while the size of the future frames is unknown. If the playback delay is set too conservative, the system will be under-optimal while a too aggressive playback delay will put the streaming system in a situation where there will be not enough time to fetch the biggest frames, especially in case of packet losses. To address this issue, we leverage the adaptive feature of HAS systems.

We illustrate our idea with an example. Let us pick a well-known representative video (*tears*) with two representations, which have been encoded with the High Efficiency Video Coding (HEVC) encoder Test Model (HM)[1] at 25 fps into two resolutions: one at 720p and a bit-rate target of 6,000 kbps, and another with a resolution $4k$ and a bit-rate target at 20,000 kbps. Let us consider for the sake of example that the playback delay has been set so that, given network conditions and performances, it is *impossible* to get frames that are greater than 5,000 kb and that it is *hard* to get those that are greater than 3,500 kb. We show in Figure 6 the cumulative density function (CDF) of the frame sizes for both video representations. For the $4k$ resolution, more than a third of the I-frames (and some P-frames) are greater than 3,500 kb, therefore, given the constraints, the chosen playback delay is too agressive *for this resolution*. But HAS systems allows the client to switch. As can be seen, all the frames of the 720p resolutions are smaller than 3,500 kb.

Based on this observation, we propose a novel adaptive mechanism, where the size of the biggest frame of the segment is taken into account to decide the representation to download. Fortunately, the biggest frame is commonly the very first frame of a segment since a segment usually starts with an I-frame. Hence, when a new segment starts, the client looks at the size of the first frame of all representations in order to choose the representation with the highest bit-rate *and* from which it can get the frame on time with high probability. Consider for example the case of the 20 first seconds of the *tears* video with four available representations and segment of 2 s. We show in Figure 7 the size of

1. https://hevc.hhi.fraunhofer.de/

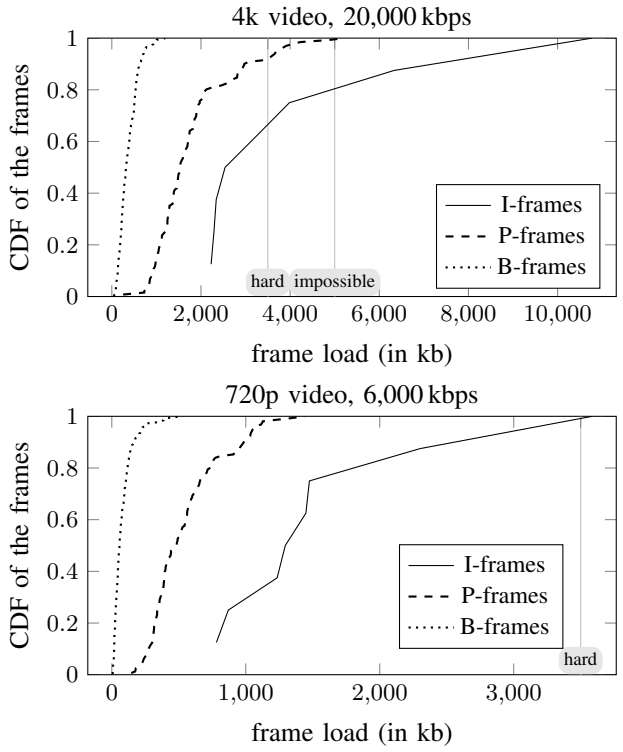

Figure 6. CDF of the frame sizes for two video representations

the biggest frame within each segment for each of the four representations. As can be seen, the only representation for which *all* segments can be fetched given the aforementioned constraints (frame lesser than 3,500 kb) is the representation at 720p with bit-rate target 2,000 kbps. However, the majority of the segments of the $4k$ representations can be fetched on time.

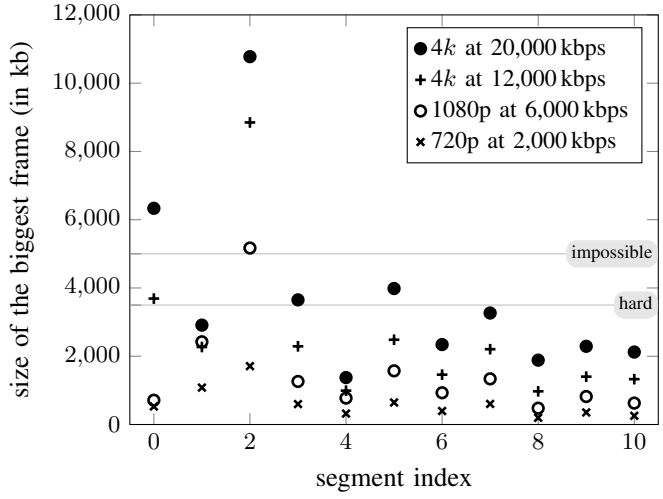

Figure 7. Size of the biggest frame of each segment (video *tears* encoded with HM HEVC encoder)

5. Evaluations

We have developed a full platform for testing and validating our proposal in realistic settings, especially regarding the network conditions and the implementation of our DASH-compatible mechanism. We present in the following the results of a campaign of subjective QoE tests, which we conducted in Spring 2015. Due to lack of room, we do not provide other results.

The campaign was done in a certified ITU-R BT 1788[2] laboratory. The method we used is called SAMVIQ, which is the recommendation to determine the video quality from accurate quality scores. The notation for short duration sequences (18 s) matches a continuous quality scale (0-100) using items (from bad to excellent). Observers are non-expert viewers and abnormal results are rejected using the recommended correlation threshold (0.85). The sequences are chosen with a wide range of characteristics regarding movements, textures, and details.

We consider a delivery latency of 60 ms. With such latency, it is possible to stay live and to accommodate the biggest frames without preventing the next frame to be fetched. In total, given the time to encode and decode, the delay is 100 ms. Our videos are encoded into 7 representations, from 480p at 500 kbps to 1080p at 8,000 kbps. For this experiment, the client can access five homogeneous xDSL links with 4 Mbps downlink bandwidth and a 30 ms Round-Trip Time (RTT). It corresponds to the case of *DSL bonding* with a pool of shared network connections.

We compare the video that we get from our proposal, considering the multipath delivery and the adaptive mechanism to the two extrema videos. The *upper bound* corresponds to the best quality video (1080p at 8,000 kbps). With respect to the network conditions, the only way to get such a video is to introduce a very large delay at the beginning of the stream session. The *lower bound* corresponds to the best video representation that can be streamed without interruptions given the latency delay without multipath delivery. Results are given in Figure 8.

The results given in these QoE measurements are promising. Indeed, the videos that have been watched by the end-users correspond to a stream that has only a 100 ms delay from the frame generation at the source. Despite this constraint, the quality of the video is in general close to the quality of the best representation, which can only be streamed with a significant delay. Furthermore, our stream significantly improves the QoE of the video representation that can be obtained with respect to the 60 ms latency constraint. These results demonstrate that our multipath delivery enables frame fetching in less than 60 ms and the adaptive mechanism provides quality variations.

6. Conclusive Discussion

We have presented in this paper a proposal for multipath delivery of live streams. Our goal is to minimize

2. https://www.itu.int/rec/R-REC-BT.1788-0-200701-I/en

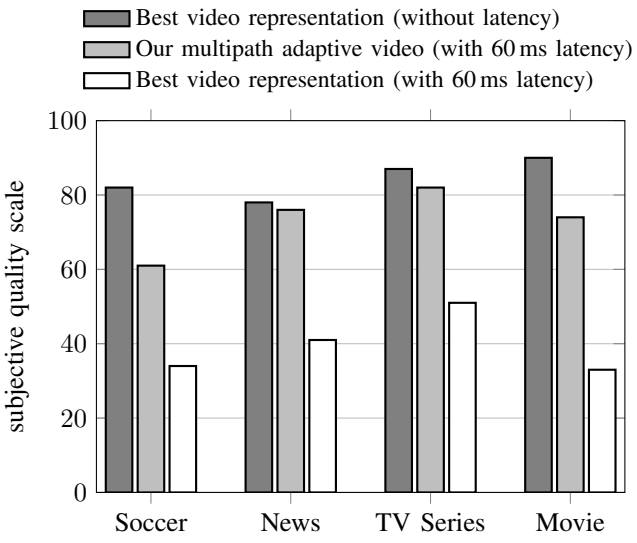

Figure 8. Subjective QoE measurements for four representative videos

the latency, which is a key requirement of end-users. We propose an implementation of multi-path *at the applicative level*, which is a novel approach for live streaming. This approach leverages several recent features that have been adopted by the most popular video standards in MPEG and ISO. A first contribution of the paper is the overall mechanism, which enables applicative-level multi-path. To the best of our knowledge, it is the first applicative-level multi-path delivery for live video streaming. A second contribution of the paper is the demonstration that our proposal, coupled with a novel rate-adaptive mechanism, significantly improves the standard solution.

This paper opens several research perspectives:

- Our proposal contributes to the recent global efforts toward reuniting the transport and the applicative layers [2], [6]. What we showed here is that the features that are introduced at the applicative level now allow the implementation of smart solutions on top of the traditional TCP. Our solution can typically be implemented by a video provider that implements its own software for both the client and the server, which is the case of most of the big video players like YouTube, Youku Tudou, and Dailymotion. But of course, our proposal would be more efficient if finer-grained information about the transport layer were available. In particular, the size of the congestion windows, the RTT, and the last packet loss are information that can be exploited in our proposal to anticipate the performances of the paths, and to achieve better performances. The implementation of such mechanisms is among our future works.
- We manipulate here the live video streams at the *frame scale*. In particular, the multi-path scheduler does not consider any structure within the frame when it divides the byte-ranges among the available paths. However, the HEVC norm has introduced the concept of *tiles* [10], which are independent elements within a frame. One of our future works is to allocate the byte-ranges among paths so that the most reliable paths get the most significant tiles in a frame, *i.e.*, the tiles that is the most important for the overall QoE of the scene.
- The subjective QoE measurements that we have presented in this paper show that the end-users accept frequent representation switches, typically every two seconds. This result are counter-intuitive for experts in multimedia QoE since it is generally assumed that too many switches have a negative impact on the QoE. Our future works will also include some additional testing about this finding.

References

[1] N. Bouzakaria, C. Concolato, and J. Le Feuvre. Overhead and performance of low latency live streaming using MPEG-DASH. In *Proc. of IISA*, pages 92–97, 2014.

[2] X. Corbillon, R. Aparicio-Pardo, N. Kuhn, G. Texier, and G. Simon. Cross-layer scheduler for video streaming over mptcp. In *Proc. of ACM MMSys*, 2016.

[3] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld. Gaming in the clouds: Qoe and the users' perspective. *Mathematical and Computer Modelling*, 57(11-12):2883–2894, 2013.

[4] D. Jurca and P. Frossard. Video packet selection and scheduling for multipath streaming. *IEEE Trans. Multimedia*, 9(3):629–641, 2007.

[5] S. S. Krishnan and R. K. Sitaraman. Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs. *IEEE/ACM Trans. Network*, 21(6):2001–2014, 2013.

[6] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli. DAPS: intelligent delay-aware packet scheduling for multipath transport. In *Proc. of IEEE ICC*, 2014.

[7] T. Kurosaka and M. Bandai. Multipath TCP with multiple acks for heterogeneous communication links. In *Proc. of IEEE CCNC*, 2015.

[8] J. Le Feuvre, C. Concolato, N. Bouzakaria, and V. Nguyen. MPEG-DASH for low latency and hybrid streaming services. In *Proc. of ACM MM*, 2015.

[9] T. Lohmar, T. Einarsson, P. Frojdh, F. Gabin, and M. Kampmann. Dynamic adaptive HTTP streaming of live content. In *IEEE Int. Symp. WOWMOM*, 2011.

[10] K. M. Misra, C. A. Segall, M. Horowitz, S. Xu, A. Fuldseth, and M. Zhou. An overview of tiles in HEVC. *J. Sel. Topics Signal Processing*, 7(6):969–977, 2013.

[11] C. Raiciu, C. Paasch, S. Barré, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley. How hard can it be? designing and implementing a deployable multipath TCP. In *Proc. of USENIX NSDI*, 2012.

[12] B. Rainer, S. Petscharnig, and C. Timmerer. Merge and forward: self-organized inter-destination multimedia synchronization. In *Proc. of ACM MMSys*, 2015.

[13] A. Singh, C. Goerg, A. Timm-Giel, M. Scharf, and T. Banniza. Performance comparison of scheduling algorithms for multipath transfer. In *Proc. of IEEE GLOBECOM*, 2012.

[14] V. Singh, S. Ahsan, and J. Ott. MPRTP: multipath considerations for real-time media. In *Proc. of ACM MMSys*, 2013.

[15] I. Sodagar. The MPEG-DASH standard for multimedia streaming over the internet. *IEEE MultiMedia*, 18(4):62–67, 2011.

[16] V. Swaminathan and S. Wei. Low latency live video streaming using HTTP chunked encoding. In *Proc. of IEEE Int. Workshop on Multimedia Signal Processing (MMSP)*, 2011.

[17] S. Wei and V. Swaminathan. Low latency live video streaming over HTTP 2.0. In *Proc. of ACM NOSSDAV*, 2014.