HTTP Video Streaming: the quality - reliability - latency triangle

Matteo Saloni, *Member, IEEE*
Master degree in Computer Science, University of Trento, Trento, Italy

***Abstract*—Nowadays, IP video traffic is responsible for a vast amount of the global IP traffic over Internet, an increasing trend fueled not only by the usage of personal computers, but also by the rising in prominence of mobile devices and smart-tvs as primary entertainment consumption devices among users.**

As of 2018, the leading paradigm of delivery is *HTTP-based adaptive streaming*, a technique which carries pre-encoded video segments over HTTP/TCP packets. While many efforts have been spent towards reaching a satisfying visual experience, one critical point remains mostly untackled by all the prominent HAS protocols: the *latency of delivery*. With the term *latency* we indicate the time passed from the instant a video is captured to the moment it gets displayed on the user's device, a key aspect in the viewing of live events such as sports or happenings.

In this white-paper we will evaluate some innovative approaches and technological advancements which can re-balance the triangle between *visual quality*, *network reliability and occupation* and *delivery latency*, by first examining the reasons which lead to an increase in latency for HAS protocols, and then investigating some of the most promising proposals in the field.

I. Introduction

In a world where consumers are increasingly considering Internet access as a basic *utility*, data consumption rates are on an explosive path. According to studies [1], IP video traffic is expected to reach 82 percent of all IP traffic by 2021, up from 73 percent in 2016. As such, network providers and content distributors must face a continuous struggle for bandwidth and efficiency, in order to deliver the expected video quality to both PC and mobile users.

Both consumers and content producers are fully accustomed to the usage of video streaming for a variety of service types such as on-demand, live and time-shift viewing, but also for chat and social media updates among many others. The established expectation is being able to receive high quality video, in terms of visual and audio details, which leads to the usage of high resolution content, compressed with advanced and extremely optimized codecs and delivered over high speed networks.

While many modern techniques have been developed over the years to improve the video streaming experience over the *best-effort* Internet network, one single approach has won the favor among content providers and device producers, **HTTP Adaptive Streaming (HAS)**. Every HAS protocol leverages the same idea: temporally divide a single video into same-length segments and encode each of these into many different quality levels. Eventually, clients will reproduce the video content by downloading a list of segments over HTTP.

As we will see in the following section, this idea can offer many advantages to content producers, while at the same time ensuring the availability of visually satisfying video renditions for every use case, at the expense of the delay which elapses between the acquisition of the video and the visualization on the user's screen (fig. 1).

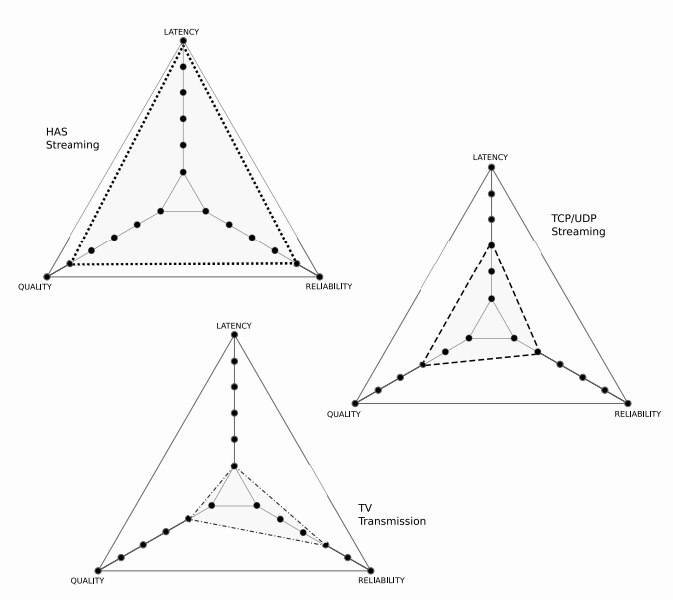

Fig. 1. Quality - reliability - latency balancing for *HAS*, *TCP/UDP streaming* and traditional *broadcast TV*.

While many on-demand use cases can live with such delay, others like live content delivery, interactive systems or synchronized viewing can't withstand a variable delay in the order of tens of seconds without a serious disruption in the user's experience. In the next sections we will present some of the most promising approaches targeted at reducing and fixing the latency, in an effort to improve the HAS adoption among all the application fields.

II. Adaptive Streaming: quality and network challenges

Traditionally, TV-based broadcast systems have been widely employed to consume video content, a task which can be considered *passive*, in opposition to *interactive tasks* performed with personal computers. Technological improvements, both to screens and displays, along with the growth in connectivity availability and the widespread adoption of personal,

portable devices like smartphones, have eventually shifted the paradigm: *personal computers* (in the broadest sense of the word) have become the primary video consumption devices for a large slice of population, especially among the younger generations. Even TVs are becoming *smart*, by transforming into an embedded system composed of a computer, a display and a network access card.

This technological shift poses a remarkable amount of problems, previously unknown to the broadcast TV ecosystem, which worked under a precise set of assumptions:

- TV transmissions are broadcasted in a single quality level, independently of viewing device: every user receives the very same stream;
- Broadcasts are pushed from the producer through the distributing network until they reach the receiving devices, there is no on-demand and a single network path carries all the available channels.

With Internet-based streaming instead, mostly due to the widespread adoption of the so called *post-PC* devices like smartphones and smart-TVs, the following problems need to be solved by content distributors:

- different display sizes and resolutions, bounded with a variety of viewing distances, rise the need for *many different video flavors* with different resolutions, codec optimizations and bandwidth usage;
- network layouts, flexible routing paths and a widespread lack of visibility of end user devices over the Internet, due to firewall and *Network-Address-Translation (NAT)* usage, drive the adoption of *HTTP* as the transport protocol for video content.

Moreover, the ever-growing list of use cases for video streaming diverges from the traditional TV architecture, where a content provider distributes the content to all the viewers at once, in a *push-based* approach. Nowadays, the leading paradigm is a *pull-based* approach, where users request, and subsequently receive, a content from a provider.

Various approaches and patents have been developed to ensure that viewers can obtain a visual quality suitable for the device characteristics, mainly in terms of perceived quality and network requirements, which eventually led to the development of *HTTP Adaptive Streaming* protocols, which embed video content inside HTTP packets, by segmenting a continuous stream into many fixed-duration consecutive segments, which can be read and eventually reproduced as a continuous whole video by players (fig. 2).

The usage of HTTP means that HAS providers can leverage pre-existing delivery networks, along with caching and proxies, to distribute their load in a cheaper and more robust way than what is possible with *Real-time Transport Protocol(RTP)* or *User Datagram Protocol (UDP)* based protocols. Perhaps more importantly, common network firewalls, and also protocol and content filters employed by Internet service providers (ISPs), corporate networks and telecom carriers, usually avoid putting restrictions over HTTP traffic, because the protocol is used for web surfing. As such, network traversal is generally ensured for HTTP streaming protocols.

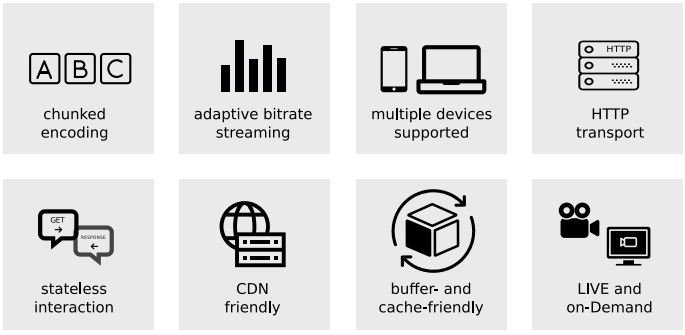

Fig. 2. HTTP Adaptive Streaming main advantages.

Additionally, HAS dictates that clients initiate the streaming, by requesting a list of segments to the provider, a simple trick which removes the need for streaming servers to keep long-lasting socket connections with clients. Instead, when needed clients will make a distinct connection for each of the video segments, as dictated by their own viewing progression.

This particular concept also enables HAS to deliver the *adaptive* part, by preemptively encoding the same video into many different *quality levels* and by letting the client pick the most appropriate rendition during the playback via rate adaptation heuristic. By downloading a *master playlist*, which contains the references to every rendition available on the server, the video player can instantaneously transition between quality levels, simply by switching to a different *media playlist* inside the same group. Source video sequences are at first temporally divided into fixed-duration blocks, and then each fragment is individually encoded into multiple renditions. As such, all the files derived from the same segment contain exactly the same fraction of the source video, and they are *temporally aligned* within the global time-line. Consequently, they are perfectly interchangeable during reproduction, without experiencing any misalignment, frame drop or hiccup. Furthermore, the entire adaptation and bandwidth estimation process is executed by clients, an approach which frees servers from the need to maintain session state information on every active client, and from the need to estimate and then deliver the most suitable rendition. Ideally, different players could employ different techniques for quality selection, which could be tuned to favor visual quality, bandwidth usage, or even computational complexity and power usage, all in an autonomous way.

The ultimate goal is to maximize the **Quality of Experience (QoE)** [2]: a measurement which objectively evaluates all the aspects that impact the human perception of multimedia content like visual quality, number of playback interruptions, number of quality switches.

All these techniques employed by HAS led to a strict dependency on the video segmenting and encoding process, which must produce video fragments and playlists *in advance*, in order to let clients know how to move forward in the reproduction at any given time. There is simply no way for servers to contact the clients to *push* additional content towards them, except embedding the references into the playlist. While this is perfectly acceptable for on-demand content, which can be prepared and assembled beforehand,

live or near-live scenarios (eg. Digital-Video-Recorder) can't work under these assumptions. The solution adopted by the prominent HAS protocols like *Dynamic Adaptive Streaming over HTTP (DASH)* and *HTTP Live Streaming (HLS)* mandates the adoption of an **open-ended playlist**, where servers append segment references as soon as they are ready, and players continuously download the updated playlist at regular intervals during reproduction, to update their knowledge of the video stream.

III. The latency problem

HAS protocols can ensure an exceptional visual experience, but at the expense of an aspect often forgotten: the *latency of delivery*, the camera-to-display delay which in current deployments can reach the order of tens of seconds. Many different steps in the adaptation and distribution process contribute to this built up in the delay, but ultimately it is the very nature of HAS approaches which requires a valuable amount of time, mainly due to the segmentation and distribution phases.

We can identify two principal areas of interest, which added together negatively impact the latency of delivery:

- start-up time, which involves the time required to start a video reproduction;
- end-to-end latency, which involves the delay between the video capture and the effective display.

We will briefly mention the player startup to highlight how it affects the end-to-end latency, and then analyze it for live events, which can be ultimately be referred to as **lag time**, meaning the time the stream itself lags behind the actual event or live broadcast.

A. Start-up time

Reproduction startup can be seen as the amount of time between the moment the user starts the player and the moment the video starts playing. These days, industry averages range in the 900 - 1,200 millisecond (ms) range from request to playback-ready state, but some customized applications can reach averages around the 500 - 650 ms range, by leveraging hardware optimizations and decoding acceleration paths available on specific platforms.

When using segment-based protocols, as in the case of HAS, players need to download at least a whole segment, and then buffer a minimal amount of frames in order to start reproduction. As such, low startup time can be obtained only if segments are small and already available on the server, otherwise clients will need to download a large amount of data before being able to decode and display the first video frame.

In *lossy* video codecs, visual quality strictly relates to available bandwidth. One of the key aspects in obtaining bandwidth efficiency derives from the ability to buffer and temporally optimize entire sequences (frames) of related images during the encoding. When a video is segmented into chunks of limited duration, the encoding process is executed independently for each chunk and every resolution or quality level. The natural consequence is the inability to properly optimize the encoding for small segments. Low *startup times* with HAS require small segments, because players need to download them fully before the decoding phase, but small segments can't be used to deliver an high QoE. Large segments, on the other hand, ensure a proper visual quality, at the expense of startup times.

Given a properly optimized player, in any case *startup time* adds to the latency introduced by servers in preparing and distributing the content, which dominates the *end-to-end latency* for HAS protocols.

B. End-to-end latency

As previously described, HAS protocols works under the assumption that a video stream can be processed and prepared in advance, before clients initiate the reproduction: players need to download a playlist containing at least a single suitable segment to start the reproduction (cfr. figure 3).

This process can be briefly summarized by the following steps :

- First, an encoder takes the actual video signal and converts it into a digital format, usually a lossless codec suitable for acquisition and processing but unusable for HAS delivery;
- the digital video is converted in different formats, usually selecting lossy codecs, and temporally segmented into fixed-length chunks;
- if required, chunks are also transformed into different resolutions or quality levels, to support the *adaptive* reproduction;
- a *playlist* is prepared, containing the references (usually URLs) to all the video chunks;
- the actual files are distributed over a *Content Delivery Network (CDN)*, a globally distributed set of servers which will receive the client requests and eventually deliver the files over HTTP.

Each step requires a finite amount of time to be executed, and the adding up of all the processing and distribution times contributes to an industry-standard delay of 30 - 60 seconds, as usually experienced with HAS. While **on-demand video** leverages pre-existing content, **live streaming** has no other choice but to execute the whole process in real-time, starting from a live video source up to the client playback.

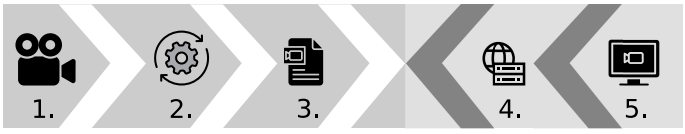

Fig. 3. HTTP Adaptive Streaming video pipeline:
1. Camera acquisition and digitalization; 2. Segmenting and encoding; 3. Video files and playlist assembly; 4. CDN distribution; 5. Client reproduction. The first three steps are executed in a *push-oriented* video-production process, while the last two are *pull-based*, initiated by clients.

The **acquisition** of a video signal can be completed in a limited amount of time, thanks to the usage of hardware accelerated encoders which can deliver digital video to the following processing block with an average delay in the order of *a couple (or tens) of frames*, i.e. well under a second. Consequently, the actual delay depends on the **segmenting**, **transcoding** and **distribution** phases.

The **video segmenting** minimal delay strictly depends on the duration selected for video segments: at the very minimum the *segmenter block* has to buffer a whole segment interval of the incoming video, plus whatever frames are needed to satisfy the video codecs requirements, and the additional amount needed to ensure a steady processing. For example, with a 5 second duration the first chunk will be available at best 5 seconds after the video acquisition.

Each video chunk has to be transformed, in a multi-step process which

- **scales** the resolution multiple times to produce different renditions;
- **encodes** each version in an appropriate codec;
- **packages** each video with the audio track into a *container file* suitable for the distribution;
- **updates** the playlist with a reference linking every video chunk into the correct position.

Given an adequate computational capability server-side, and an optimally engineered trans-coding software, all these steps can be executed together, leveraging the resources common to every rendition to reduce the overhead of producing multiple adaptations of the same source. Nevertheless, a finite, discrete amount of time will be required, an amount which nowadays can be estimated at best around the *single-second* mark for our *5 seconds long* sample.

After transforming the video into many different files, *origin servers* need to **distribute** the content towards the *edge nodes*, usually a CDN which ensures the load distribution and the end-user reachability. Both *push-* and *pull-based* approaches are equally valid, but usually web distribution networks employ a pull approach, which requires a user HTTP request on an edge node to trigger the download of the source file from the origin server. Afterwards, each subsequent request for the same file will be directly delivered by the CDN node, saving the upstream round-trip and reducing the delivery delay. But the first client request involves a FETCH from the origin server, an operation which introduces a sizable delay, which is directly transferred into the end-to-end latency measure.

Lastly, the client player will need to **request the playlist**, individuate the most appropriate rendition, fetch the segment playlist and the **download** the video segments in order to be able to reproduce the actual video content.

As we can see, it is the protocol nature, and the video processing it mandates, which introduces all the various delays. In the following section, we will evaluate some of the proposed and suitable solutions for reducing the lag.

C. Client synchronization: the soccer problem

As a direct consequence of the increased and unpredictable delay introduced by HAS protocol, **client playback synchronization** becomes a challenging argument during the streaming of live events. As previously described, each client before starting the reproduction of a live stream has to perform a number of steps, all of which are initiated right on the device itself, and not on the server side. Given that state-of-the-art HAS protocols do not explicitly envision a client synchronization procedure, each player has to autonomously decide how to handle the *start-up* procedure, particularly in terms of:

- where in the video playlist start downloading segments;
- how many video files download and cache before beginning reproducing them;
- how much buffer, in terms of data or frames, keep in player to ensure a consistent reproduction.

The first point is luckily addressed by the most recent revisions of both *DASH* and *HLS*, the two prominent HAS protocols in use today, which both offer an annotation scheme for content producers to instruct players on the way to handle **live playlists**: the adopted solution utilizes either the adoption of a specific amount of segments, counting from the end of the playlist, to instruct clients about the starting point, or the time-stamping of segments to provide a precise chronological and temporal advice to players. Either way, these protocols can at least ensure that no client will start processing the video playlist outside a small window, which comprises only a couple of segments.

Even after this reduction of randomness in the playlist parsing, HAS clients can actually start the reproduction during a window of time large at minimum 2x/3x the duration of a single video segment, depending on the instant they download the playlist and the internal amount of caching and buffering done. This can amount to tens of seconds, even in deployments optimized for live events and with clients accessing the web with an adequate and stable network connection. More commonly, when considering real usage with mobile clients, unreliable connections and un-optimized video segmenting, the amount of de-synchronization between any two clients accessing the same stream could be measured in several tens of seconds or even minutes.

As a result, it is almost unavoidable to experience a perceptible disconnection between the contents displayed on different screens, even if the reproduction is started at the same time by users. The implication on client experience is the inability to reproduce the same video concurrently on more then one screen, in a synchronized fashion, a common concept withing an home with multiple TVs. In fact, traditional broadcast over air or cable, thanks to the *push-based* approach, ensures that every single client will receive the very same content *at the very same time*, minus the transmission latency. Start the reproduction of a TV channel, and the content will be the same on every additional device connected to the same channel over the same network path, independently of the playback start (fig. 4).

Fig. 4. TV broadcasting pipeline:
1. Camera acquisition and digitalization; 2. Media adaptation and encoding; 3. Air/Sat/Cable distribution; 4. Client reproduction.
The whole the process happens in *push*, from content producers towards end users.

This eradicated expectation is regularly disappointed by

HAS streaming.

This problem manifests itself also during highly popular events, like sports matches, which are followed by a vast amount of people in a given area. For example, *soccer matches* in Italy are followed by many individuals, both at home or within public places like pubs, clubs, even theaters. The rapid pace of action, a characteristic of many games, implies that during a couple of seconds the state of the match could change, favoring first one and then the opposing party. When clients are *un-synchronized*, these topical moments will be received and displayed on screen at different times, according to the real-world delay accumulated by the single device. As such, the people reaction, often shared via social networks, messages, live blogs or even by voice, can effectively disclose the result of a single game action to all those in reach, which have yet to see it take place on their displays, due to the delay. By taking inspiration from Huysegems et al. [3], we can call this *spoiling* of live sports events the **soccer problem**: the result of an action is spoiled by social media or neighbors when the goal has not yet been shown on the screen.

While this problem can be dismissed as a minor inconvenience, it is nonetheless a deviation from an established habit, which can impact negatively on the reception and thus adoption of HAS streaming within the live sports scenario.

IV. Current and proposed solutions

Given the importance of *video streaming* as a first-class citizen of the web, techniques and protocols are a hot topic not only for scholars and researchers, but also for the big technological companies, which invest money and resources in order to improve the technological landscape and the user experience. This section will briefly highlight some of the viable and promising approaches, which could help reducing the *latency problem* without compromising the efficiencies and excellences achieved by the current state-of-the-art HAS protocols.

A. The naive: super-short segments

If we analyze the design principles and the video pipeline behind HAS protocols, we can intuitively understand that employing large segments, while optimal for video encoders, has an immediate drawback on the time required to *acquire, segment, encode and deliver* those fragments. By reducing the segment size, we can thus immediately gain back the time lost during the video pipeline, and reduce at the same time the **startup delay** and the **end-to-end latency**. This *naive* solution, while easily implementable, can be adopted only when working with segments no less than a couple of second long, otherwise the overhead generated by the *encoding codec*, along with the induced increase in the amount of *HTTP requests* will negatively impact on the protocol efficiency.

Putting aside those concerns, we can speculate on the usage of *super-short* segments, with a sub-second duration. In this case, the duration of a single segment will barely impact on the end-to-end latency, due to the minimal amount of time required for the acquisition and encoding processes. Given an average sub-100ms latency obtainable when using a Content Delivery Network, we can easily compare the time required to transmit a single packet from client to server, and back, to the duration of a single super-short segment. Due to the round-trip dictated by HTTP streaming, where clients needs to issue an HTTP GET to receive a video file from the server, we can conclude that the simple usage of shorter segments alone can not solve the latency problem. Furthermore, the mere adoption of smaller segments can not help in *synchronizing* different players viewing the same content, because the mere reduction of the *end-to-end* latency, even when under the 10 seconds mark, can still negatively impact the live sports experience, where a sub-second synchronization is mandatory.

B. Partial-segments

Improving on the idea of *short-segments*, recent works (cfr. [4], [5] and [6]) have explored the feasibility of providing clients with *partial-segments* early on during the encoding process. Thanks to the integration of *HTTP chunked transfer encoding*, clients can start downloading a partial file as soon as the server starts writing it, with the caveat that further byte blocks of the same file can be accessed via different HTTP conditional requests. This approach can avoid the encoding and caching problems associated with the usage of short segments, and at the same time enable clients into experiencing shorter *start-up delays* and *end-to-end latency*.

One problematic area, shared with the previous approach, remains: the explosion of HTTP requests and the related network overhead. Similarly to the *short-segment* approach, clients need to issue a request over HTTP for smaller blocks, leading to an increase in the overhead and ultimately in a reduction of available bandwidth for video content, at the expense of visual quality.

C. HTTP/2: Push-based approaches

As previously introduced, HAS protocols leverage HTTP as the transport protocol for video segments. Both the two major streaming protocols, namely *HTTP Live Streaming* [7] and *MPEG-DASH* [8], were defined and published before the *HTTP/2* standardization, which happened in 2015 [9]. The new major version of the HTTP protocol provides new features targeted at reducing the load time during web browsing and improving the bandwidth utilization. One of those features could help in improving video streaming protocols, *server-side push*. This technique enables the origin server to send data directly to clients, after an initial request, without awaiting the reception of subsequent GET request. Wei et al. [6] explore this feature, by proposing and evaluating a server push based low-latency mechanism in a MPEG-DASH prototype. By employing a 1-second segment duration, along with a server push of k-segments after each client request, they consistently manage to reduce the *end-to-end* latency for live streaming down to 10 seconds. Moving from the same hypothesis, van der Hooft et al. [10] propose a novel approach which can successfully utilize *super-short* segments, with a sub-second duration, to achieve a *end-to-end* latency of around 6 seconds, with a *start-up time* around the 1 second mark.

These promising results demonstrate how the combination of HTTP/2 and super-short segments can greatly alleviate the *latency problem*, but they require further research to optimize and improve the video codecs and the HTTP infrastructure, in order to guarantee an adequate visual quality under every network condition.

The server-based push approach could also be used to implement a client synchronization protocol, to solve the *soccer problem*. Furthermore, some frameworks recently published [11] explore the usage of server profiling and client storage to reduce the *startup time*, even for the worst-case situation of mobile networks with large *Round Trip Time (RTT)*.

D. 5G networks and edge computing

The advent of 5G networks is stimulating investments and research funding towards *edge computing*, an approach which leverages network edges to provide micro and virtual *data-centers* closer to users than in traditional centralized approaches. Moving computational and storage capabilities towards users immediately reduces the response time, improving on the idea behind *Content Delivery Networks*. In order to effectively exploit this novel *proximity* between users and servers, the network should dynamically and pro-actively schedule and destroy service instances on the most suitable nodes, taking into account contextual knowledge and network information [12].

Scoca et al. [13] explore the consequences of moving latency-sensitive applications to *edge computing*, with a major focus on live streaming. They demonstrate significant improvements in bandwidth utilization, and more interestingly in end-to-end latency. Their studies show that strict proximity of end users to the nodes delivering the video stream reduces the delay not only because of the reduced network path or physical distance, but also that moving the trans-coding process as well further contributes to the delay reduction, thanks to the cut of another network trip between CDN nodes and origin servers.

Closely related to edge computing, research in the 5G networks area also can improve video streaming over the Internet, and help reduce the *latency problem* (cfr. [14], [15], [16] and [17]). The various approaches explore the requirements for video streaming and the techniques which can improve the delivery of content over 5G networks. In particular, researches focusing on caching and network path optimization, along with virtual CDN usage, have shown interesting results in optimizing the bandwidth efficiency and reducing the lag experienced when consuming video streams with current-gen networks.

E. Application level multi-path

Another different approach, targeted at reducing the latency between video acquisition and content playback, has shown interesting results: *multi-path video delivery*. While the idea of multi-path in data networks has been widely researched during the years, Houz et al. [18] propose a novel approach which addresses the *latency problem* by implementing multi-path at the application level. By leveraging network level multi-path and the TCP protocol, their video player can reach a latency below 100ms. This impressive result is obtained by leveraging features exposed by video standards like *MPEG* and *ISO*, but also by heavily modifications of the DASH protocol and further tuning of video encoding: by optimizing the delivery of single frames, and leveraging the multi-path network to transport each type of frame over the most suitable link, they demonstrate the ability to reach a 60ms delay target.

While interesting and promising, this solution can be only partially adopted by general purpose HAS protocols, since the encoding, packaging and delivery of the video stream requires an unified approach over the *transport* and *application* layers, and a custom software stack for both server and clients. As such, it could be adopted by large scale video providers, which control and implement the software for both the client and the server. In order to reach mass adoption, this approach requires extensive modification to streaming protocols, players, CDN networks which would probably require years to be completed.

F. Hybrid Cloud/P2P distribution

Video providers utilizing HAS protocols heavily rely on fast connections and distribution networks to guarantee an adequate reachability and performance level. Streaming over HTTP ensures that video segments can be treated as regular files, delivered via a multitude of HTTP caching proxies, thus avoiding the cost associated with a dedicated streaming infrastructure. Nevertheless, regular CDNs can be a pricey commodity for high volume providers. As such, many have explored alternatives solution, which leverage inexpensive *peer-to-peer* (P2P) networks to reduce the number of servers required to support a given amount of users (cfr. [19], [20] and more recently [21]). With the *hybrid P2P/Cloud* approach, the users viewing the video content could become members of the distribution network, relying each segment to those in the proximity with their own upload link, or thanks to *cloud computing* edge network devices could act as the relaying party. The cost reduction for the providers becomes an immediate penalty for the *quality-of-service*, since often end users do not possess bandwidth or computational capability comparable to CDN servers. Moreover, the relay process introduces an additional step in the delivery chain, with the obvious result of an immediate increase in the *end-to-end latency*. Regarding the *startup delay*, the usage of P2P networks could in theory provide an advantage to users, in the very specific case that one of their peers (i.e. one on the same network path) locally possesses the required video segment. This advantage anyway is negated by the time required for nodes to receive a list of available peers, either from a server or by autonomously exploring the network.

A recent research by Provensi et al. [22] explores the implementation of a *cloud-assisted* P2P streaming system, where latency is kept at reasonable levels by combining P2P dissemination of video segments with a cloud based self-organizing system. Their work demonstrates that an hybrid approach can greatly reduce infrastructure costs for providers, and at the same time ensure an *end-to-end latency* under the 20 seconds mark, a promising achievement which could support the live streaming use case. Moreover, the utilization of such

a hybrid approach could provide the basis for the implementation of a *synchronization scheme*, targeted at solving the *soccer problem*

V. Conclusion

Live video streaming applications and platforms are increasingly popular among end users. Major web players like *YouTube*, *Facebook*, *Amazon*, along with traditional TV broadcasters, are aggressively targeting live event coverage and transmission, reaching millions of users all over the world. While current state-of-the-art HAS protocols can ensure an high visual quality and an optimal viewing experience over a wide range of devices, there is no accepted solution to guarantee a minimal end-to-end latency for live content. Many interesting approaches, as briefly depicted in the previous section, could help in minimizing the various delays built in the video delivery pipeline, but none of them is sufficient by itself. We can thus speculate that hybrid approaches, combining the usage of *short-segments*, *edge computing*, *HTTP/2 push* and potentially *P2P delivery*, could provide an adequate solution in keeping latency under control, and maybe even in implementing a synchronization protocol which could support the live sports or interactive broadcast use cases. The complexity associated with all the different approaches will probably ensure that no globally usable solution will be adopted in a reasonable time window, but specific approaches will surely be adopted for the most relevant cases by the big players, which can leverage content acquisition and processing, network distribution and player capabilities within their own organizations. Nevertheless, we hope that standard, widely accepted streaming protocols like *MPEG-Dash* and *HTTP Live Streaming* will eventually evolve and thus avoid a fragmentation of the streaming landscape towards proprietary or vertically-integrated solutions, with the risk of hindering the competition, hampering the open research and ultimately worsening the user experience.

VI. Acknowledgment

This project is supported by a student grant of the *IEEE Smart Cities Initiative*.

References

[1] C. V. N. Index, "The zettabyte era–trends and analysis," *Cisco white paper*, 2017.

[2] I. T. Union, "ITU-T Recommendation P.10/G.100," *Vocabulary for performance and quality of service*, 2006.

[3] R. Huysegems, J. van der Hooft, T. Bostoen, P. Rondao Alface, S. Petrangeli, T. Wauters, and F. De Turck, "HTTP/2-based methods to improve the live experience of adaptive streaming," in *Proceedings of the 23rd ACM international conference on Multimedia*. ACM, 2015, pp. 541–550.

[4] N. Bouzakaria, C. Concolato, and J. Le Feuvre, "Overhead and performance of low latency live streaming using MPEG-DASH," in *Information, Intelligence, Systems and Applications, IISA 2014, The 5th International Conference on*. IEEE, 2014, pp. 92–97.

[5] J. Le Feuvre, C. Concolato, N. Bouzakaria, and V.-T.-T. Nguyen, "MPEG-DASH for low latency and hybrid streaming services," in *Proceedings of the 23rd ACM international conference on Multimedia*. ACM, 2015, pp. 751–752.

[6] S. Wei and V. Swaminathan, "Low latency live video streaming over HTTP 2.0," in *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*. ACM, 2014, p. 37.

[7] R. Pantos and W. May, "HTTP Live Streaming: draft-pantos-http-live-streaming-06," *Published by the Internet Engineering Task Force (IETF)*, vol. 24, 2011.

[8] I. Sodagar, "The mpeg-dash standard for multimedia streaming over the internet," *IEEE MultiMedia*, no. 4, pp. 62–67, 2011.

[9] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," Internet Requests for Comments, RFC Editor, RFC 7540, May 2015.

[10] J. Van Der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck, "An HTTP/2 push-based approach for low-latency live streaming with super-short segments," *Journal of Network and Systems Management*, vol. 26, no. 1, pp. 51–78, 2018.

[11] J. van der Hooft, C. De Boom, S. Petrangeli, T. Wauters, and F. De Turck, "An HTTP/2 push-based framework for low-latency adaptive streaming through user profiling," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–5.

[12] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[13] V. Scoca, A. Aral, I. Brandic, R. De Nicola, and R. B. Uriarte, "Scheduling latency-sensitive applications in edge computing." in *CLOSER*, 2018, pp. 158–168.

[14] J. Qiao, Y. He, and X. S. Shen, "Proactive caching for mobile video streaming in millimeter wave 5g networks." *IEEE Trans. Wireless Communications*, vol. 15, no. 10, pp. 7187–7198, 2016.

[15] G. Carrozzo, F. Moscatelli, G. Solsona, O. P. Gordo, M. Keltsch, and M. Schmalohr, "Virtual cdns over 5g networks: Scenarios and requirements for ultra-high definition media distribution," in *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*. IEEE, 2018, pp. 1–5.

[16] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. Leung, "Cache in the air: exploiting content caching and delivery techniques for 5G systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 131–139, 2014.

[17] C.-F. Lai, R.-H. Hwang, H.-C. Chao, M. M. Hassan, and A. Alamri, "A buffer-aware HTTP live streaming approach for SDN-enabled 5G wireless networks," *IEEE network*, vol. 29, no. 1, pp. 49–55, 2015.

[18] P. Houzé, E. Mory, G. Texier, and G. Simon, "Applicative-layer multipath for low-latency adaptive live streaming," in *Communications (ICC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–7.

[19] I. Trajkovska, J. Salvachua Rodriguez, and A. Mozo Velasco, "A novel p2p and cloud computing hybrid architecture for multimedia streaming with qos cost functions," in *Proceedings of the 18th ACM international conference on Multimedia*. ACM, 2010, pp. 1227–1230.

[20] X. Jin and Y.-K. Kwok, "Cloud assisted p2p media streaming for bandwidth constrained mobile subscribers," in *Parallel and Distributed Systems (ICPADS), 2010 IEEE 16th International Conference on*. IEEE, 2010, pp. 800–805.

[21] F. Wang, J. Liu, M. Chen, and H. Wang, "Migration towards cloud-assisted live media streaming," *IEEE/ACM Transactions on networking*, vol. 24, no. 1, pp. 272–282, 2016.

[22] L. Provensi, F. Eliassen, and R. Vitenberg, "A cloud-assisted tree-based p2p system for low latency streaming," in *2017 International Conference on Cloud and Autonomic Computing (ICCAC)*. IEEE, 2017, pp. 172–183.