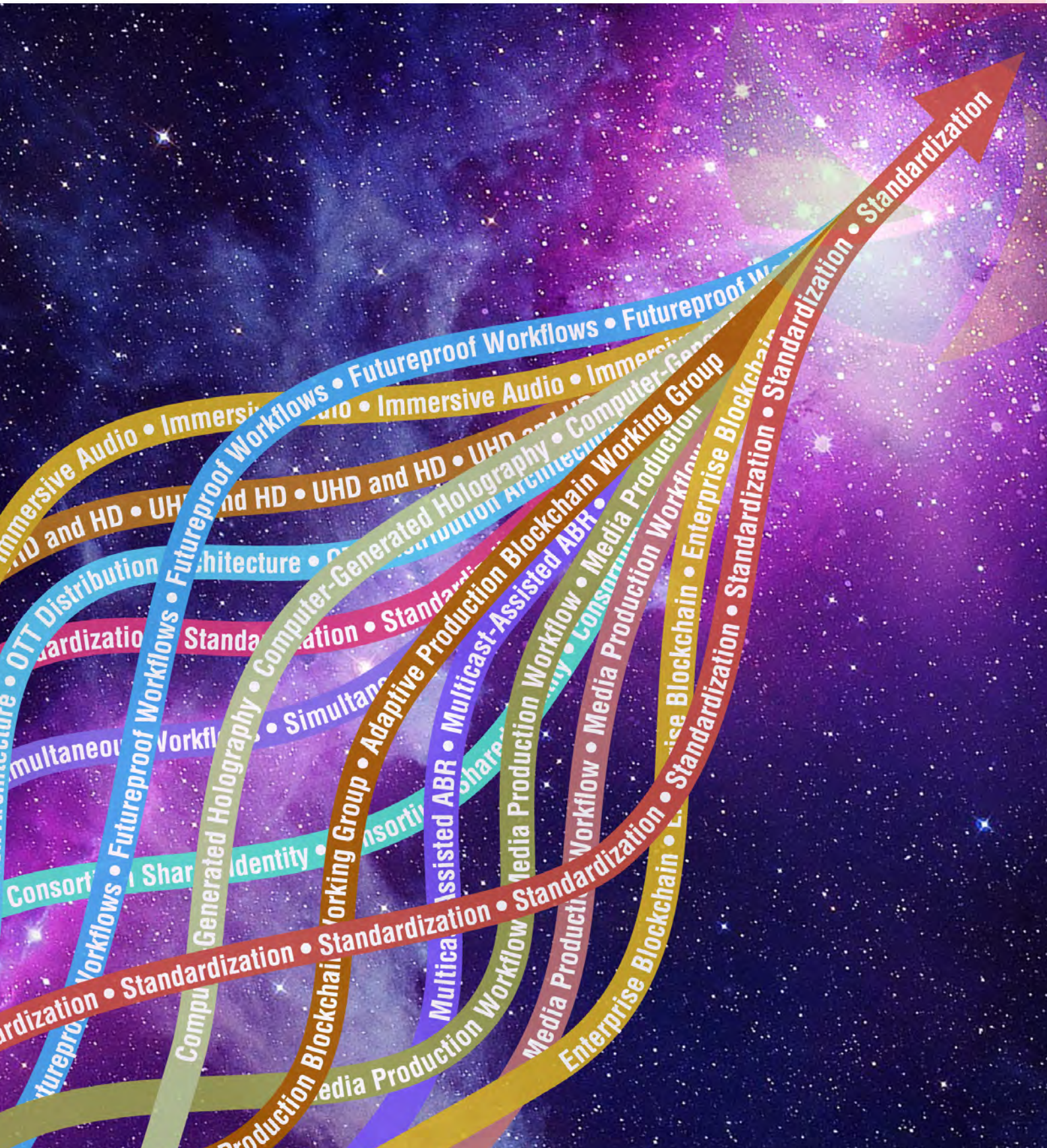
SMPTE

MOTION IMAGING JOURNAL

Covering Emerging Technologies for the Global Media Community



Server-Side Segment Selection for Low-Latency Streaming—S4S

By Guillaume Bichot and Nicolas Le Scouarnec

Introdução:

No universo OTT, a baixa latência desafia a estimativa de largura de banda feita pelos dispositivos ou players, que se baseiam nos sistemas ABR atuais. Porém se a baixa latência quebrar essa confiança da estimativa fornecida por esses dispositivos, toda a qualidade de experiência do usuário (QOE) vai por água a baixo. No presente artigo vamos ver como um conjunto de recursos combinados pode garantir tanto a estimativa fornecida pelos dispositivos quanto a QOE sejam preservadas.

Senhoras e Senhores apresento-vos o *Server-Side Segment Selection for streaming-S4S*, ou se o traduzirmos livremente como **Seleção de Segmento do lado do Servidor de Streaming-S4S**, que permite que os dispositivos interajam com um servidor habilitado para S4S garantindo, assim, uma melhora da experiência geral. Artigo obrigatório para todos que desejam entender como melhorar a QOE diante dos desafios da Baixa Latência de Rede. Boa Leitura!

Tom Jones Moreira

Abstract

Over-the-top (OTT) streaming growth is driven by the increasing number of wireless connected devices such as tablets and phones. An enabler for using these devices is adaptive bitrate (ABR) streaming that allows devices to select at each moment the stream quality that best fits the available bandwidth. The overall experience is highly dependent on the bandwidth estimation. The current approach to bandwidth estimation in HTTP-based ABR players is challenged by the evolution toward low-latency protocols (DASH CTE—Dynamic Adaptive Streaming over HTTP—Chunked Transfer Encoding) or (HLS LL—HTTP Live Streaming—Low-Latency), which result in micro-burst of traffic. Indeed, estimation at the client side, at the HTTP level, relies on the assumption that a relatively large segment of data is available and can be sent at the link speed, which is not the case anymore with low-latency protocols. To address this issue, we propose a novel scheme Server-Side Segment Selection for streaming (S4S), which allows players to interact with an S4S-enabled server to improve the overall experience. First, bandwidth estimation is done at the server side, using transport's congestion control sender-side stats, leading to more precise estimates even in the presence of small burst, especially when using modern congestion control algorithms such as Bottleneck Bandwidth and Round-trip propagation time (BBR) or Performance-orientated Congestion Control (PCC). Second, we define a protocol for allowing the network to control the bandwidth versus quality tradeoff.

Keywords

Low-latency adaptive bitrate (ABR) streaming for live content

Introduction

Adaptive bitrate (ABR) streaming is a streaming technology that is prevalent today when streaming video-on-demand (VOD) and live content as popularized with protocols such as HTTP Live Streaming (HLS) from Apple and Dynamic Adaptive Streaming over HTTP (DASH), a standard specification from the Motion Picture Experts Group (MPEG) consortium. These protocols are converging in terms of the content format [e.g., fragmented MPEG MP4/Common Media Application Format (CMAF)]. Their differences lie mostly in the way they write and manage the manifest/playlist file. Beyond these two major protocols, there are numerous proprietary protocols more or less derived from the previous ones.

S4S is a technology compliant with ABR streaming that forces the bitrate selection based on its own bandwidth estimation. Because it performs the bandwidth estimation on the server, S4S is not constrained by the player environment (sandbox, API) and can inherently provide a more accurate estimation.

Fundamentally, adopting the client/server paradigm, and HTTP-based ABR protocol gives the control to the terminal (more specifically to the player). It has some obvious advantages: the server can be a dumb file server; the player selects the video-bitrate based partly on the function of its status (buffer level) and the capacity of the host (codec, screen resolution).

However, it has also a serious drawback: the player neither knows the network environment nor cares about it. Its bitrate selection strategy is only driven by a heuristic for: (1) avoiding buffer starving, (2) reducing quality switches, and (3) maximizing the video bitrate. Consequently, a player tends to be greedy; it downloads in advance, requests video segments for which the bitrate exceeds the maximum bottleneck bandwidth, and possibly operates multiple concurrent Transmission Control Protocol (TCP) connections. It can definitively impact neighboring players even if the

bitrate selection strategy is identical. This is particularly true when the network and possibly the content delivery network (CDN) are overloaded in the case of, for example, very popular live or VOD events.

Furthermore, to properly proceed with its heuristic, the player needs to rely on a good network-bandwidth estimation. It is, however, known that this evaluation is not always accurate nor consistent across platforms or software versions and it is very challenging when talking about low latency.

ABR streaming has been created for mobile handsets that are typically attached to the network through a shared medium (Wi-Fi, Cellular). The bandwidth estimation drawbacks mentioned earlier are exacerbated when the players are in competition sharing the same bottleneck. As discussed in Bichot et al.,[1] the bitrate selection may become unstable and/or unfair because the concurrent players ignore each other and, as ABR streaming is bursty by nature, the bandwidth estimation can be wrong.

It is typical in VOD or live streaming for the player to manage a buffer of 30 sec while, in low-latency streaming (about 5 sec glass to glass), the player cannot rely heavily on its buffer anymore. Consequently, the bandwidth estimation and prediction become even more prevalent.

Recently, new methods have been published for allowing low-latency ABR streaming based on the support of the MPEG-CMAF[2] format wherein a typical video segment/file (e.g., 2 sec) is fragmented into a set of chunks having a duration of tens to hundreds of milliseconds generated at the rhythm of the encoding and transported along the delivery chain without buffering up to the terminal. However, it has been shown[1,3] that bandwidth estimation in that context is very challenging.

Low latency is required for video live streaming. In live streaming, CDNs are exposed to overloading especially when a content is popular. The number of simultaneous connections may increase significantly putting the CDN at risk, the CDN being unable to deliver such amounts of data. There is no practical and efficient way to mitigate such overload. A simplistic approach would be to overprovision the CDN/network, while a more reasonable approach is to force the player somehow to consider a maximum bitrate. This is possible today through modifying the manifest/playlist file but there is no guarantee that the manifest file will be downloaded by the players and if so in the [almost] same time.

It turns out that the service operator has no control over the player behavior and in particular regarding the video bitrate selection strategy. Doing so, however, would permit addressing various use cases where controlling finely, dynamically, and in a consistent way the video streaming bitrate provides a more efficient usage of the bandwidth for a better quality of experience.

Our contribution is Server-Side Segment Selection for streaming (S4S), a technology enabling various features that solve or mitigate the aforementioned drawbacks. It enables the service operator to finely control the bandwidth versus quality tradeoff in a consistent way. While our previous work[1] presented the general benefits of Server-Side Segment selection, this paper addresses more specifically the case of live streaming and presents the set of features that make low-latency streaming possible in a tough environment.

Others have experimented server-side approaches in controlling ABR streaming[4–9] although none of them have combined a server-side bandwidth estimation with server-side bitrate selection for addressing low-latency HTTP-based adaptive streaming.

S4S: Controlling the Streaming Experience From the Server Side

S4S is a technology compliant with ABR streaming that forces the bitrate selection based on its own bandwidth estimation. Because it performs the bandwidth estimation on the server, S4S is not constrained by the player environment (sandbox, API) and can inherently provide a more accurate estimation.

Transparent Mode

In this mode of operation, the S4S-capable server selects the bitrate and delivers self-initializing segments on [player] request. The strength of this mode is that it works with many existing players without any changes, making it deployable on servers as of today without changes on the client sides. The player is not aware of the S4S presence; it behaves similarly to conventional ABR, attempting to control the quality/bitrate selection. A self-initializing segment (compatible with MPEG-DASH[4]) is a media segment with a prepended initializing segment. The player supporting the self-initializing segments must be capable of parsing the CMAF header information present at the beginning of the returned [self-initialized] segment.

In addition to the self-initializing segments, the S4S server relies on specific uniform resource identifiers (URIs) declared on the manifest, preserving the stateless nature of the S4S server. Again, this is fully compatible with the MPEG-DASH[4] standard and therefore transparent to the terminal.

The server always decides whether to perform the quality/bitrate selection on the requested segment or simply fulfills the player's selection indicated in the request.

Last but not least, the server works with HTTP/2. While in a previous work,[1] we presented the S4S solution as workable over HTTP 1.1, despite a loss of accuracy, we do recommend HTTP/2 and future versions as it avoids dealing with several TCP connections per player, providing a more stable and accurate framework, which is particularly important when targeting low-latency streaming in wireless/mobile environments.

Non-Transparent Mode

Because it lacks client-side information (buffer level, current resolution), in transparent mode, the server-side decision algorithm may be suboptimal in some contexts and can be improved by adding cooperation between the server and the player.

When the player is S4S aware, there is an exchange of information with the server for increasing the quality of experience. The information is exchanged relying on custom HTTP headers for example. This section gives an overview of such a protocol.

The server is capable of working in two sub-modes. The server-driven mode is about the server being authoritative and controlling tightly the streaming session quality/bitrate switching. The server-driven mode requires the support of self-initializing segments as with the transparent mode.

The client must indicate, in all the requests, the list of supported representations (bandwidth and absolute URL of the segment to download). In addition, the player must join the audio and video buffer levels as well as their status (pause, play, stall).

The server must indicate in all responses, the representation (i.e., the selected bitrate) currently served. In addition, it must join in all media response its bandwidth estimation and maximum bitrate. The maximum bitrate is understood by the client as the maximum allowed bitrate (or available bandwidth) imposed by the network for various reasons (fair bandwidth sharing, decrease the load on the CDN). This is particularly useful in the second mode: the server-assisted mode.

In the server-assisted mode, the client is basically on its own regarding quality/bitrate selection while it can/must use server information as explained further in the paper. The server-assisted mode allows the player to select the bitrate relying on the server information, increasing the accuracy of the player heuristic. However, there is no guarantee that the adaptation is uniform and it precludes a tight and instantaneous limitation of the bandwidth usage. The operator and thus the server always decides to which mode the client should comply. The mode can be selected for the entire session (e.g., on a client/player basis) or can possibly change at any time during a session.

Case of Low Latency

Typical latency for live streaming may be about 30 sec or more. The ABR algorithm relies on the buffer level as well as available bandwidth estimation. Low-latency streaming targets must approach live latency as provided by an IPTV service that is about 5 sec glass to glass. With such a target, the player's buffer is very limited (a few seconds), precluding bulky burst transmissions. In other words, the server can no longer transmit a segment of several seconds as a bulky burst of bits but must smooth the transmission, sending more regular, smaller bursts. A segment is split into chunks, ranging from a video frame duration (e.g., 40 msec) to hundreds of milliseconds.

These chunks are typically generated by the packager. Ideally, they contain a structure of video frames without forward references (i.e., the pictures can be decoded immediately).

The streaming must be as regular as possible. DVB has published a low-latency extension to the DVB-DASH specification[10] that is also mostly adopted by the Dash-IF consortium and will be published as part of their next implementation guideline release.[11] The specification relies on the MPEG-CMAF[2] format combined with HTTP CTE (Chunk Transfer Encoding). With such a transmission mode, the time to download a segment is close to the segment's duration, which makes a player's bandwidth evaluation based on HTTP transactions roughly equivalent to the segment encoding bitrate. This is problematic for most of the players. Because they cannot estimate the maximum available bandwidth properly, they stick to one quality/bitrate only.

How It Is Addressed Today

A few attempts have been made to solve that problem. Bentaleb et al.[3] proposed Adaptive Streaming with Chunked Transfer Encoding (ACTE), a throughput-based ABR strategy relying on bandwidth prediction based on recursive linear regression (RLS), a method depending on bandwidth estimation. The latter does not work properly, especially when chunks are small and thus interchunk times tend to be significant. This is due to the difficulty in identifying chunk boundaries. It is worth noting that the best scheme (active probing) is the one that provides the best results but suffers from the inherent-induced load over the medium.

As part of the MultiMedia Systems Conference (MMSys) grand challenge,[12] Lim et al.[13] went a step further, relying on an unsupervised learning (self-organizing maps)-based algorithm for bitrate selection. The features are a QOE modeling (a combination of the typical metrics that characterize an ABR streaming session: bitrate selected, number of bitrate switches, rebuffering time, live latency, and playback speed), a throughput estimate, the buffer occupancy, and the latency. They increased the efficiency of the bandwidth estimation proposed in Ref. 3 through a better detection of the chunk boundaries but with adding a dependency on the video format.

Karagkioules et al.[14] proposed an online learning method based on the online convex optimization (OCO) theory wherein the throughput estimation serves in computing the adversary function.

Karagkioules et al.[14] and Lim et al.[13] were the winner and runner-up, respectively, of the grand challenge.[12] Both exhibited similar results although they[13] had a better stall duration result (divided by 2) especially for the "cascade" network profile (the available bandwidth varies every 15–20 sec) despite a number of quality switches that is two to four times greater.

The challenge conditions (segment duration of 0.5 sec, chunk duration of 33 msec, RTT =0) make the experiment results inapplicable to typical live "broadcast" programs.

And S4S?

In low latency, it is important to first tackle the player's rebuffering (stalling) before anything else. Because the buffer is small (a few seconds) and corresponds to a known latency target (e.g., 3 sec), the bitrate selection should follow some form of adaptive increase, multiplicative decrease (AIMD) schema. However, it is important to note that a player/client involved in low-latency streaming is inherently weaker regarding concurrent players. This is because in low latency, there are much more "holes" because interchunk delivery time slots provide more opportunities for greedy players (involved, e.g., in VOD streaming where the buffer can store several tens of seconds) to eat the bandwidth.

S4S supports any form of low-complexity bitrate selection strategy/adaptation and provides the advantage of being consistent across platforms and players. Being at the server side, it enables first a better estimate of the network/transport behavior.

S4S provides an accurate throughput measurement provided by the congestion control module attached to the transport layer. Our first implementation was based on bottleneck bandwidth and RTT (BBR) that is supported by both TCP and quick UDP internet connections (QUIC). BBR has a dedicated maximum bottleneck bandwidth evaluation mechanism that is reactive on the rising edge (i.e., when the throughput goes up significantly) but may be relatively slow on the falling edge (i.e., when the throughout goes down significantly). This is because the BBR bandwidth evaluation is based on the maximum delivery rate over a sliding time window that is itself based on the RTT.

The plot shown in **Fig. 1** illustrates the case. It is a low-latency streaming session over HTTP2/TCP with BBR activated. The bottleneck bandwidth variation setup was the following: min bw = 1200 kb/s, max bw = 3500 kb/s, number of steps = 3, and step duration = 20 sec. The RTT was set to 180 msec that has an impact on the BBR bandwidth evaluation window as explained earlier. The green curve is the raw BBR bandwidth metric. One can see that over the falling edge, the BBR_bw remains high and falls down a few seconds late. When the video quality/bitrate is selected based on this metric only, the player may be forced to switch too late and may experience a video freeze.

The mitigation consists in using the delivery rate information (provided by the BBR module through a system API) for detecting a falling edge and for computing the bandwidth estimation allowing a quicker reaction (blue curve in the figure). The orange dash curve corresponds to the video quality/bitrate selected by the S4S server.

Note that the blue curve shows a significantly smaller measured bandwidth. This is because our bandwidth estimation takes into account (is reduced by) the bandwidth consumed by the audio (roughly 130 kb/sec).

Figure 2 illustrates a simple test (rtt = 180 msec) with two players DASH-IF 3.1.0. Three substreams/bitrates are available (800 kb/s, 1200 kb/s, and 2500 kb/s). On the left side, the player is running in low latency (target 3 sec) and the bitrate selection is done by the player (the dashed orange curve). On the right side, the player controls the bitrate selection and the low latency is not activated. However, the content is served through HTTP CTE. No stall is observed. However, the player on its own (right side) estimates the bandwidth according to the video bitrate and is therefore unable to switch to higher bitrate.

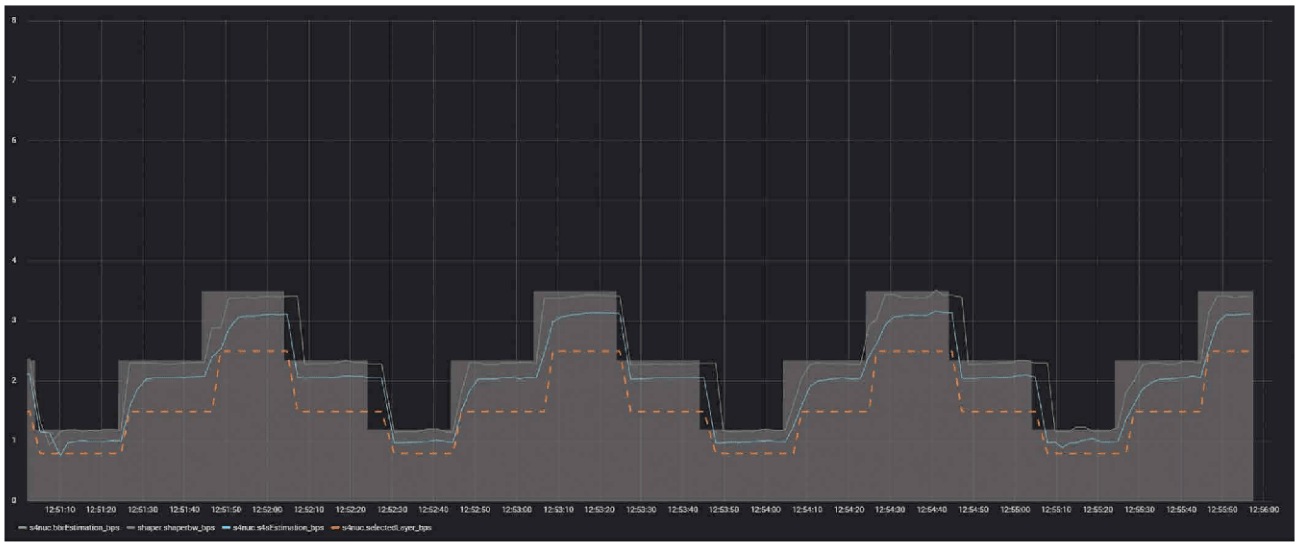


FIGURE 1. Compensating BBR slow falling edge detection.

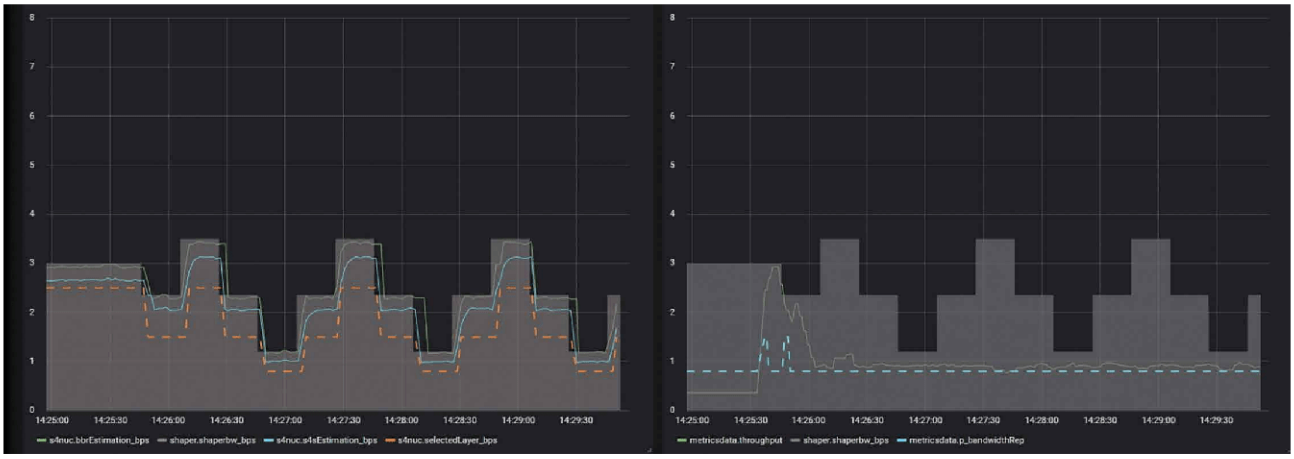


FIGURE 2. S4S streaming in low latency versus nonlive streaming over HTTP CTE.

The next plot (**Fig. 3**) illustrates the test with a smaller rtt (30 msec) with, on the right side, the player, controlling the bitrate selection, having the low-latency feature enabled. The player is capable of switching, but it frequently selects the wrong bitrate, resulting in stalls. Note that the throughput estimation computed by the Dash-IF player appears to be a little strange.

The last plot (**Fig. 4**) illustrates the case where two sessions share the same network and therefore bandwidth. The maximum bottleneck bandwidth has been doubled for theoretically allowing a fair share of it. On the right side, the player is streaming from a conventional DASH server in the VOD mode (without low latency and without the content being delivered through HTTP CTE). On the left side, the other player is running in low latency and controlled by the S4S server. The rtt is fixed to 30 msec. One can see that the streaming is more challenging for S4S than the previous cases as the right-side player tends to be greedy relying on a rather large buffer (10–20 sec of buffer), whereas on the left side, the player keeps its buffer level around 3 sec. Yet S4S does a good job maintaining its buffer level without stalling and still delivering the highest bitrate possible most of the time.

What About the Chunk Duration?

There is an important network feature to be considered that is the round-trip time (RTT). It has been obfuscated in the grand challenge[12] as the tests run locally (i.e., within the same machine) without any emulation of the RTT). The RTT can vary in real life (physical transmission time, bottlenecks). Assuming that the CMAF chunk is a bulk transfer unit, its size must be large enough to accurately estimate the maximum available bandwidth. Without proper care, the throughput estimation could be limited by the RTT and therefore preclude bitrate switching at the right time.

One solution to this problem is to have the delivery chain support dynamic chunk size, which computes the chunk size as a function of a maximum RTT between the cache server and the terminal. With S4S, the cache server evaluates the RTT and feeds a process that computes the minimum bulk transfer size enabling the estimation of the available bandwidth as large as the "biggest" representation (i.e., the one that is associated with the biggest bitrate).

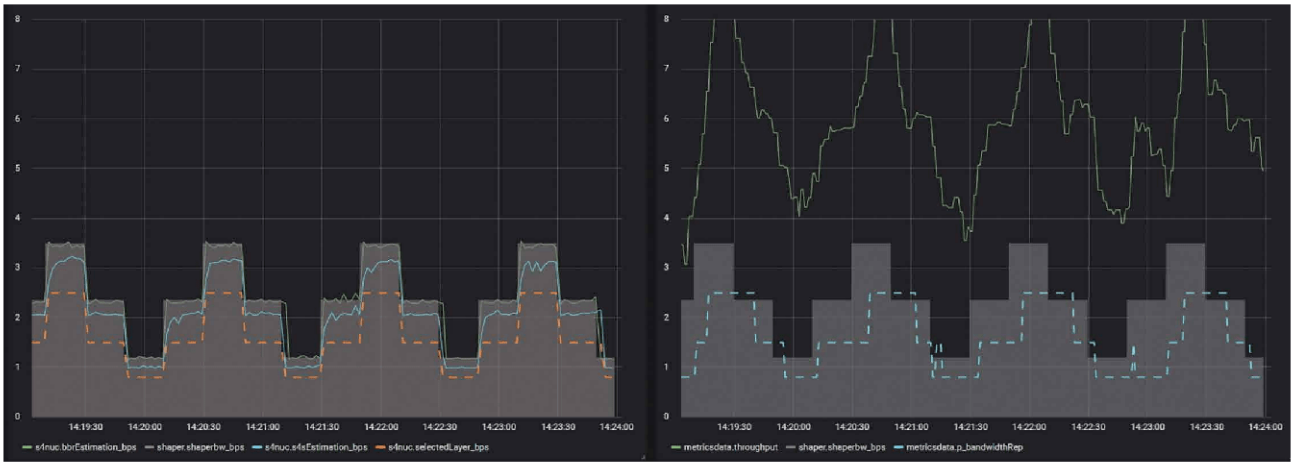


FIGURE 3. S4S streaming in low latency versus convention live streaming (with low latency enabled).

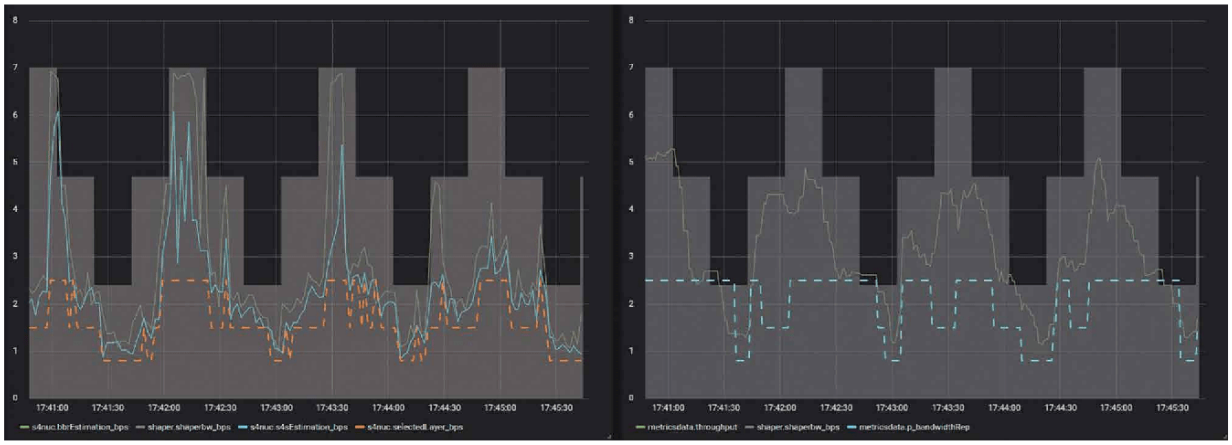


FIGURE 4. Live low-latency streaming competing with greedy VOD streaming.

Assuming N sub-streams $(1,\ldots, N)$, each associated with an average bitrate B, the minimum transfer bulk duration (mTBD) for the sub-stream i is defined according to the following:

$$\mathrm{mTBD}_i = \frac{\mathrm{C}\cdot\max\left(\mathrm{B}_0\cdot.\mathrm{B}_\mathrm{N}\right)\cdot\mathrm{mRTT}}{\mathrm{B}_i} \qquad (1)$$

with

i: sub-stream i

mRTT: maximum RTT between the cache server and the terminal

Bi: Bitrate (in bits per second) associated with sub-stream i

N: Maximum number of representations (quality sub-streams)

C: constant allowing to add a margin that compensates rough mRTT estimation as well as B as an average bitrate indication. Note that this is precisely the transmission time between the server and the terminal that matters here and not exactly the round-trip time. Therefore, mRTT can be possibly halved considering that the RTT is equally distributed among the uplink and downlink.

The process then builds the bulk transfer unit on the fly by concatenating **m** chunk units together that verify the following:

$$\sum_{j=1}^{j=m} Cdj \le mTBDi < \sum_{j=1}^{j=m+1} Cdj \qquad (2)$$

with:

j: chunk j

Cd: Chunk duration in seconds

Conclusion

Low latency challenges bandwidth estimation done at the player side in current ABR systems, preventing good user experience. A server-side approach relying on bandwidth estimation provided by modern transport congestion control (such as BBR) combined with a set of discussed features successfully prevented stalling while allowing high bitrate streaming.

We have shown the importance of keeping the RTT as low as possible. This is beneficial for streaming, in general, and for low-latency streaming and therefore S4S, in particular. This is why we are developing a cache infrastructure/framework that places our cache servers in locations with low and consistent RTTs.

Because such technology is better evaluated statistically, as the next step, we plan to experiment with the S4S solution under diverse conditions met on the field via an A/B testing framework.

References

1. G. Bichot, P. J. Guery, and N. Le Scouarnec, "How to Optimize ABR Video Delivery with Server-Side Quality Control," *Proc. 2020 NAB Broadcast Engineering and Information Technology Conf.*, May 13, 2020.
2. International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) 23000-19, "Information Technology—Multimedia Application Format (MPEG-A)—Part 19: Common Media Application Format (CMAF) for Segmented Media."
3. A. Bentaleb, C. Timmerer, A. C. Begen, and R. Zimmermann, "Bandwidth Prediction in Low-Latency Chunked Streaming," *NOSSDAV'19*, Amherst, MA, Jun. 21, 2019.
4. K. Khan and W. Goodbridge, "Server-Based and Network-Assisted Solutions for Adaptive Video Streaming," *Int. J. Adv. Networking Appl.*, 09(03): 3432–3442, 2017.
5. H. Mao, R. Netravali, and M. Alizadeh, "Neural Adaptive Video Streaming with Pensieve," *SIGCOMM '17*, Los Angeles, CA, USA, 2017.
6. S. Petrangeli, J. Famaey, M. Claeys, S. Latré, and F. De Turk, "QoE-Driven Rate Adaptation Heuristic for Fair Adaptive Video Streaming," *ACM Trans. Multimedia Comput. Commun.* Article 28, Vol. 12 (2), Oct. 2015.
7. A. El Essaili, D. Schroeder, D. Staehle, M. Shehada, W. Kellerer, E. Steinbach, "Quality-of-Experience Driven Adaptive HTTP Media Delivery," *IEEE Int. Conf. Commun. (ICC 2013)*, Jun. 2013.
8. R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang, "QDASH: A QoE-Aware DASH System," *MMSys'12*, Chapel Hill, NC, Feb. 22–24, 2012.
9. L. De Cicco, S. Mascolo, and V. Palmisano, "Feedback Control for Adaptive Live Video Streaming," *Proc. Second Annu. ACM Conf. Multimedia Syst. (MMSys'11)*, ACM, pp. 145–156, Feb. 2011.
10. European Telecommunications Standards Institute (ETSI) TS 103 285 V1.3.1 (2020-02), Digital Video Broadcasting (DVB);

MPEG-DASH Profile for Transport of ISO BMFF Based DVB Services over IP Based Networks.
11. "Guideline for Implementation: DASH-IF Interoperability Points. Change Request for Low Latency Modes for Dash." [Online]. Available:https://dashif.org/guidelines/#agreed-crs-for-next-version
12. "Grand Challenge on Adaptation Algorithms for Near-Second Latency," *ACM MMSys*, 2020.
13. M. Lim, M. N. Akcay, A. Bentaleb, A. C. Begen, and R. Zimmermann, "When They Go High, We Go Low: Low-Latency Live Streaming in dash.js with LoL," *MMSys '20: Proc. 11th ACM Multimedia Systems Conf.*, pp. 321–326, May 2020.
14. T. Karagkioules, R. Mekuria, D. Griffioen, and A. Wagenaar, "Online Learning for Low-Latency Adaptive Streaming," *MMSys '20: Proc. 11th ACM Multimedia Systems Conf.*, pp. 315–320, May 2020.
15. International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) 23009, "Information Technology—Dynamic Adaptive Streaming over HTTP (DASH)."

About the Authors

Guillaume Bichot joined Broadpeak in August 2018 as the head of Exploration Team with a focus on addressing and developing innovation. Prior to working at Broadpeak, he was a principal scientist at Technicolor and Thomson, where he led international teams/activities related to image and networking. His areas of expertise include video streaming, IoT, wireless, broadband access, cloud networking-NFV, and virtualization. He has contributed to many collaborative projects, standardization bodies, and authored numerous papers at industry conferences.

Nicolas Le Scouarnec is a research engineer on the Exploration Team at Broadpeak. His current research focuses on modern internet protocols for video streaming and reliable, high-performance CDN systems. Prior to joining Broadpeak, he was a senior scientist in distributed systems, security, and high-performance machine learning at Technicolor. He holds a Ph.D. in computer science from INSA de Rennes, Rennes, France.