QOE ENHANCEMENT THROUGH COST-EFFECTIVE ADAPTATION DECISION PROCESS FOR MULTIPLE-SERVER STREAMING OVER HTTP

Joachim Bruneau-Queyreix[1] [2], *Mathias Lacaud*[2], *Daniel Negru*[1], *Jordi Mongay Batalla*[3], *Eugen Borcoci*[4]

[1] Univ. Bordeaux, LaBRI, UMR 5800 F-33400 Talence, France, {jbruneau,negru}@labri.fr

[2] Viotech Communications, Versailles, France, {jbruneau,mlacaud}@viotech.net

[3] Warsaw University of Technology, Poland, jordim@interfree.it

[4] University Politehnica of Bucarest, Romania, eugen.borcoci@elcom.pub.ro

ABSTRACT

Single-source HTTP Adaptive Streaming protocols (HAS), such as MPEG-DASH, have become the de-facto solutions to deliver video over the Internet. By avoiding buffer stalling events that are mainly caused by the lack of throughput at client or at server side, HAS protocols increase end-user's Quality of Experience (QoE). We propose to extend HAS capabilities to a pragmatic DASH-compliant Multiple-Source Streaming solution (MS-Stream) that simultaneously utilizes several servers. MS-Stream offers the opportunity to obtain higher QoE by exploiting expanded bandwidth and link diversity in heterogeneous distributed streaming infrastructures, such as distributed home-gateways or geographically distributed set-top-boxes belonging to Over-The-Top video service providers. This paper exposes a cost-effective two-phase adaptation process with dual (i.e., bitrate and number of sources) adaptation decisioning prior segment request and in-segment download adaptation. Our approach was empirically evaluated for on-demand video streaming over the Internet. An online demonstration is also made available [1].

1. INTRODUCTION

According to Cisco's white paper [2], video traffic will experience a tremendous growth of 91.7%, and is expected to reach 81.8% of the total Internet traffic by 2020. With the advent of technology characterized by the increasing offers of video services, end-users' Quality of Experience (QoE) has become a crucial factor for their success or failure. The use of HTTP Adaptive Streaming (HAS) has been a way for technical enablers to tend toward satisfying end-users' QoE. Single-source HAS protocols (such as the widely adopted Dynamic Adaptive Streaming over HTTP standard -*DASH*-) adapt the delivered content bitrate according to the observed throughput at client side or to the client buffer occupancy. Consequently, HAS protocols tend to avoid the video the freezing events that are mainly caused by the lack of throughput at the client or at the server side. Technical enablers have proposed, developed and commercialized solutions to overcome the server-side bottleneck problem by implementing Content Delivery Networks infrastructures (CDNs). CDNs provide a single intermediate point geographically close to end-users along with high throughput and storage capacity. In the event of bottleneck occurring at the server side (i.e.: CDNs reaching their upload throughput capacity limit), horizontally and vertically scaling CDNs' infrastructures can enable higher throughput to end-users, and avoid QoE degradation. Nevertheless, in view of the drastic video traffic growth forecasts, this solution will increase deployment and maintenance costs, eventually making CDN services prohibitively pricey for Content Providers looking for high QoE content delivery.

This work is part of the European CHIST-ERA DISEDAN project, supported by the European Future & Emerging Technologies scheme (FET)

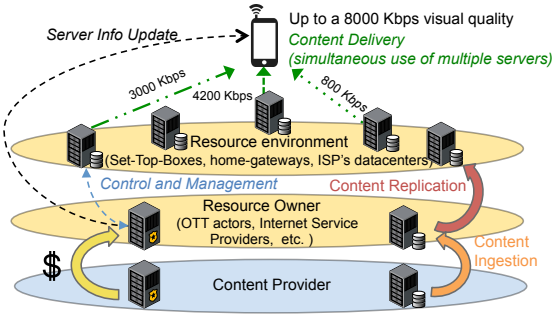

Fig. 1. MS-Stream solution overview

In order to alleviate both the scalability cost issue and the insufficient throughput due to server-side bottleneck, we propose an evolving and pragmatic solution where a client can simultaneously use several source servers with heterogeneous capacities (set-top-boxes, commodity devices, cloud servers, etc.). Such a solution would provide the opportunity to obtain higher QoE by exploiting expanded bandwidth, link diversity and reliability in distributed and highly heterogeneous streaming infrastructures that scale horizontally at significantly lower costs compared to CDN solutions.

Based on the work in [3], we propose a dynamic adaptive Multiple-Source Streaming over HTTP protocol (MS-Stream) as an evolution of HAS solutions (and more specifically DASH) that simultaneously uses several servers for the download of each video segment. MS-Stream clients request several servers to create and deliver independent sub-segments (also referred as sub-streams). When retrieved, the requested sub-segments are merged in order to reconstruct the original requested content quality. MS-Stream incorporates a two-phase adaptation algorithm with dual adaptation decisioning prior segment request (i.e., bitrate and number of sources, based on a bottleneck estimation method) and in-segment download adaptation. In the event of sub-segment loss or out-dated delivery, content playback interruption is avoided, and content quality may only be affected for a short time-lapse. While MS-Stream introduces a bandwidth consumption overhead required to achieve streams independence, we suggest a method to tend to minimize and limit it up to a given percentage (*X%*) of the content bitrate. Moreover, MS-Stream clients can estimate the presence and type of bottleneck and act consequently by adding or removing servers to the streaming session, so as to avoid QoE degradation and to reach a given video bitrate (*Y Mbps*). Thanks to its codec agnosticism and DASH-compliance, MS-Stream represents an evolving solution that can be applied to many scenarios. For instance, MS-Stream presents the opportunity for new technical enablers to participate in the video delivery value chain, competing with CDN providers. As depicted in Fig.1, a new actor (owning heterogeneous resources) could benefit from the MS-Stream solution to propose to Content Providers video delivery services targeting a video bitrate (*Y Mbps*) to be delivered

to a specific number of users.

In this paper, we empirically evaluate and compare the QoE metrics of MS-Stream and uni-source DASH in a video-on-demand scenario (section 5) similar to an Over-The-Top (OTT) video provider benefiting from its set-top-boxes geographically distributed over the Internet at end-users' premises. The overhead of our solution is also evaluated as well as its bottleneck estimation method. Before detailing the MS-Stream solution in section 3 and its two-phase adaptation algorithm in section 4, we outline the related work (section 2).

2. RELATED WORK

HAS protocols have seen important interest from the industry and the research community, mainly due to their capabilities to render smooth video playback to the consumers, hence a better QoE. The authors in [4] and [5] introduced the DASH framework of Netflix, the largest DASH streaming provider in the world, and outlined that a user is always bound to one server, regardless of the available throughput between the user and the server. [5] also indicates that QoE could greatly benefit from the venue of a practical HAS that can actually utilize multiple servers simultaneously. Originally not suited for a multiple-server purpose, the proposed bitrate adaptation decisions in [6] are asynchronously taken, imposing segments to be retrieved one after another without considering completion time. Therefore, applying [6] to multiple-source streaming will result in requests being most of the time delivered out-of-date due to path heterogeneity, leading to buffer depletion and video freezing. In contrast with this approach, our proposed MS-Stream solution considers requesting several servers simultaneously and abandoning segment requests, based on synchronization rules. The contribution in [7] presents Presto, a streaming protocol designed to use several servers simultaneously in order to improve QoE by providing better fairness, efficiency and stability at server side. However, Presto performs very poorly should any of the considered network paths suffer degradation. In [8], an evolution of DASH is put forward using multiple servers assisted with the Scalable Video Coding technique where content is generated over a "single base layer" providing a minimum visual quality and several "enhanced layers" designed to ameliorate the visual quality on top of the base layer. Clients simultaneously request segment layers from several servers. Dependency between layers requires segment scheduling; failing in retrieving the base layer will prevent the end-user from watching the video. In contrast with [7] and [8], the segments retrieved in our approach can be independent from each other and are aggregatable, thus providing uninterrupted streaming even when segments are lost. The work in [9][10] relies on the same principles presented in our work (i.e., independent and aggregatable sub-segments). However, due to the fact that sub-segments in [9][10] are created prior the content consumption phase, flexibility and adaptability to network paths heterogeneity are drastically limited to the number of available sub-segments, leading to many quality fluctuations throughout streaming sessions. Moreover, sub-segments placement within a distributed system is an obvious issue of [9][10], requiring expanded storage capacity in each server to provide scalability. In our work, sub-segments are generated for each video segment, during consumption phase and upon client specification, which brings the needed flexibility and adaptability for distributed streaming infrastructures. Finally, while the papers [9][10] do not address the issue of limiting the additional bandwidth consumption costs resulting from their approach, our paper proposes a method to limit and minimize such costs.

Another approach in multi-path streaming over the Internet is the use of the multi-path TCP protocol (MPTCP). Many studies have shown that MPTCP performances are below the expectations, especially under heterogeneous network characteristics [11] [12]. In MPTCP, a scheduler assigns each packet from the MPTCP output queue to one available TCP sending buffer. Packet losses occurring on one path can generate head-of-line blocking at the client side, potentially leading to streaming buffer dryness. Many papers addressed this issue by focusing on windows congestion management [13], cross-layer scheduling [11], bandwidth and buffer management [14] and retransmission processes [15] at the transport layer. Our approach differs from MPTCP streaming by performing both multi-source/multi-path delivery and adaptation at the application layer, using TCP as its transport mode. Finally, by benefiting from the non-dependency between sub-segments, MS-Stream provides an elegant way to handle stream synchronization from multiple sources.

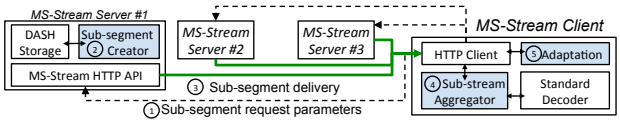

Fig. 2. MS-Stream client/server architecture

3. ARCHITECTURE AND SUB-SEGMENT CREATION

We propose MS-Stream (Multiple-Source Streaming over HTTP) as a practical solution simultaneously exploiting multiple heterogeneous servers for consumers' QoE enhancement, inducing only limited additional costs in terms of consumed network resources. The MS-Stream overall client/server architecture is depicted in Fig. 2 (additional modules -compared to DASH- are highlighted in blue). The MS-Stream content delivery steps consist of: (1) the client instructs MS-Stream servers to generate and deliver sub-segments through the MS-Stream HTTP API; (2) the Sub-stream creator generates the requested sub-segments from the existing set of content segments (each segment's duration being of a few seconds) made available at different bitrates in the DASH Storage; (3) sub-segments transit on the network; (4) the Sub-stream Aggregator module merges the received sub-segments so as to reconstruct the original content quality; Finally, (5) server and content bitrate adaptation mechanisms are ran in order to tend towards the targeted content bitrate (Y Mbps) and the maximum bandwidth consumption overhead ($X\%$). Y and X are given parameters prior the session.

The sub-segment generation scheme utilized in our approach relies on the principles brought by Multiple Description Coding (MDC) [16]. A video segment is generated over multiple sub-segments. Video quality increases with the number of sub-segments merged; the original quality is obtained when all sub-segments are used. The number of sub-segments for a given segment is an evolving parameter during the streaming session and is determined by the client according to the heterogeneous characteristics of the available paths and servers, and to the targeted bitrate. We present a list of specificities included in our sub-segment generation scheme, which would ease its adoption by streaming providers: video-codec standard compatibility, tunable redundancy, low additional complexity and the possibility to create as many different sub-segments as needed. Video standard compatibility allows receivers to use a standard decoder module to decode sub-segments. A very low-complexity pre-decoding step is required to merge sub-segments, thus preserving standard compatibility. Sub-segments can be totally independent from each other; this is made possible by copying some common information (i.e., redundancy) into them. Such non-dependency between flows provides high reliability in heterogeneous or unreliable networks, as any independent sub-segment can be lost without interrupting the streaming session. The more the redundancy is, the greater the network bandwidth consumption overhead. MS-Stream has embedded mechanisms to control the degree

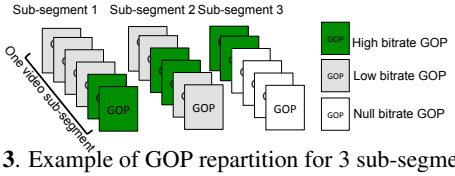

Fig. 3. Example of GOP repartition for 3 sub-segments

of redundancy based on servers throughput and network conditions (further discussed in section 4). Finally, although most of the MDC approaches proposed in the literature [16] rely on two sub-segments only, our solution can generate any number, providing better bitrate scalability and resilience to outages and delay variations[16].

We focus on a hybrid solution based on temporal and compressed data domains[16] by interleaving the Group Of Pictures (GoPs) available at two different bitrate representations of the same segment. Examples of GoP-based sub-segments are depicted in Fig.3. Reconstructing the original content quality is achieved by selecting the GoPs of higher size in the pool of available sub-segments. Should some sub-segments be missing for content reconstruction at client side, the content is still playable in sub-optimal visual quality according to the received quality at the GoP level.

4. TWO-PHASE ADAPTATION ALGORITHM

MS-Stream is an evolutionary protocol where a pragmatic usage of servers over multiple paths is suggested in order to reach a targeted bitrate, to avoid video freezing and to limit the extra bandwidth costs. Fig. 4 presents the protocol's two-phase adaptation algorithm. The first phase consists of prior-download adaptation decisions for the upcoming segment, composed of three steps: sub-segment requests generation based on GoP distribution to adapt sub-segments' bitrate to network resources heterogeneity; overhead selection and bitrate adaptation to increase QoE and limit bandwidth overhead; bottleneck estimation and server adaptation to provide a flexible usage of network resources and reach the required throughput mapping the targeted bitrate. The second phase consists in performing in-segment download adaptation so as to ensure smooth video playback.

4.1. Prior-download: Sub-segment requests design based on GoP distribution for path heterogeneity adaptability

Video streaming in network environments with high resource heterogeneity in terms of server's throughput capacities can be addressed via adaptability in the requested throughput on each path.

This throughput is a function of the number of GoPs at the selected bitrate B and at the redundant L bitrate (L can be 0 Kbps) that a client requests to servers so as to generate sub-segments composed of G GoPs in total. Prior the streaming session, a MS-Stream client retrieves a manifest file containing information about the available MS-Stream servers as well as the number of GoPs per video segment. The MS-Stream client uses this information to specify the GoP sub-segment composition through parameters in HTTP requests. Let us consider the use of M servers simultaneously. Let us define $GoP_{i,B,m}$ so that $GoP_{i,B,m} = 1$ when server m is assigned the delivery of GoP i at bitrate B and $GoP_{i,B,m} = 0$ otherwise. The MS-Stream client requests each server to deliver a sub-segment so that the following equations are satisfied under the constraint that the delivery of GoP i at the selected B bitrate is assigned to one server only: $\sum_{i=1}^{G} GoP_{i,B,m} \leq G$ and $\sum_{m=1}^{M} \sum_{i=1}^{G} GoP_{i,B,m} = G$. The MS-Stream's sub-segment design module is in charge of designing sub-segments' GoP compositions so that the sub-streams' bitrates map the estimated throughput on each path (estimations are made after the previous sub-segments downloads). The flexibility of this approach depends on the number of GoPs per segments. Indeed,

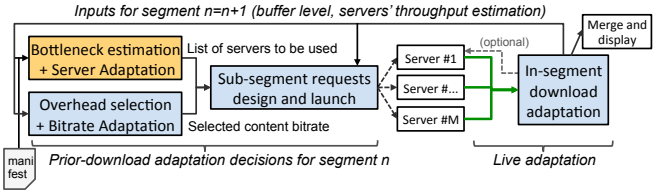

Fig. 4. MS-Stream adaptation and consumption algorithm. The *Bottleneck Estimation* and *Server Adaptation* module is only used every two segment downloads for the purpose of bottleneck estimation.

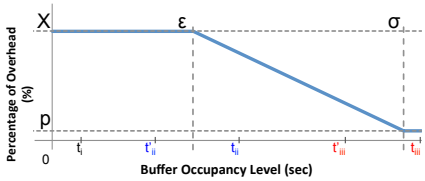

Fig. 5. Overhead selection

a MS-Stream client can use at most M_{max} servers, where M_{max} denotes the number of GoPs per segments.

4.2. Prior download: Overhead selection and bitrate adaptation step for QoE enhancement and bandwidth costs limitation

As in DASH, the MS-Stream protocol adapts video bitrate to the available network throughput in order to avoid video stalling and QoE degradation. Upon sub-segments delivery, a MS-Stream client estimates the status of each used path (by measuring the observed throughput), and computes the available cumulated throughput on all paths. A candidate bitrate *B* is selected based on the estimated global throughput and on the available buffered content playout. The client then assigns to each server the delivery of sub-segments according to GoP distribution decisions As in DASH, many adaptation policies can be implemented (defining more aggressive or conservative consumption behaviors).

MS-Stream also addresses the issue of limiting and minimizing bandwidth overhead coming from redundant data copied into sub-segments. Bandwidth overhead is defined as the percentage of data transiting on the network, which does not take part into the displayed content. We have $TransmittedData = PlayedData + NonPlayedData$, therefore we can define $Overhead = 1 - PlayedData/TransmittedData$. The capacity of a streaming system to accommodate as many clients as possible with a *Y Mbps* content bitrate is a function of its global aggregated upload throughput from all servers. This upload throughput capacity is linearly decreased by the bandwidth consumption overhead, hence impacting the infrastructure scalability, which in turn impacts on consumer's QoE. Minimizing this overhead is therefore an essential issue. To tackle this problem, we propose to limit this overhead by lowering the amount of redundancy copied into sub-segments according to the streaming session's steadiness and stability based on the buffered content playout. Concretely, we propose to ensure a minimum degree of network-outages and server-failures resiliency by requiring a maximum $X\%$ of redundancy when the buffer level is below a pre-defined lower bound ε. Fig. 5 depicts the relation between the buffered content and the redundant data percentage selection (overhead) for the next segment download. Fig 5 also shows a buffer level upper bound σ over which the percentage of redundant data cannot decrease lower than p (p can be 0%). The request design module is in charge of mapping this percentage to the closest number of redundant GoPs to be copied into sub-segments so as to create the highest number of independently playable sub-segments. Although a great value of X increases the maximum overhead, it also provides bet-

Table 1. Variables

Symbol	Description
M_n	Number of servers used for segment *n*: $M_n \in [1..M_{max}]$
S_{M_n}	Set of M_n server(s) used for segment *n*,$S_{M_n} = \{s_{m,n}\}_{m\in[1..M_n]}$
thr_n	Observed throughput for segment *n*
X_{M_n}	The set of observed throughputs for each server $\in S_{M_n}$ for segment *n*: $X_{M_n} = \{x_{m,n}\}_{m\in[1..M_n]} = \{throughput(s_{m,n})\}$
δ	Maximum allowed bandwidth overhead: $\delta = Y * X\%$
τ	Required throughput for the targeted video quality ($\tau \geq Y + \delta$)
ΔT_n	Throughput difference between 2 segments: $\Delta T_n = thr_n - thr_{n-1}$
ΔM_n	Evolution of the number of simultaneously used server between two segment downloads:$\Delta M_n = M_n - M_{n-1}$
μ	Percentage used in the computation of bnd_n
bnd_n	Throughput bound determining whether the global throughput is stable: $bnd_n = \mu * thr_n$
γ	Percentage determining whether a throughput varies highly in time
λ_n	The number of servers which delivered throughput varies highly $\lambda_n = card(\{s_{m,n} \in S_{M_n} : \lvert x_{m,n} - x_{m,n-1}\rvert > \gamma * x_{m,n-1}\})$

Table 2. Bottleneck estimation

Estimation	ΔM_{n-1}	ΔT_{n-1}	λ_n
Server	> 0	$\in [-bnd_{n-1}; +\infty]$	λ_{Low}
Server	0	$*$	λ_{Low}
Client	$*$	$\in [-\infty, bnd_{n-1}]$	λ_{High}

ter resiliency to network outages as it allows MS-Stream clients to request a greater number of independently playable sub-segments. Note that the optimal trade-off between X, p, ε and σ is not discussed here and is planned for future works.

4.3. Prior-download: Bottleneck estimation and server adaptation step for a flexible usage of network resources

Providing flexibility in the amount of heterogeneous network resources used to reach a targeted video bitrate (Y Mbps) is another objective of MS-Stream and is achieved in this step. Two main causes can explain the lack of throughput received at the client side: First, the presence of a bottleneck within the client's environment (over-used communication medium, limited throughput capacity); second, the presence of a bottleneck at the server side (i.e.: servers reached their upload throughput capacity). We propose a method to detect and estimate both the presence and the type of bottleneck that prevents the delivery of the video targeted bitrate. Then, based on this estimation, MS-Stream clients are able to take decisions regarding the addition or removal of servers to the streaming session. The overall server adaptation decisioning is outlined in algorithm 1. The bottleneck estimation is based on a two-segment-download-duration observation window. This window is dimensioned as such in order to monitor (1) the impact of server add/keep/remove decisions (ΔM_{n-1}) on the global throughput capacity (ΔT_{n-1}) during the first segment download, and (2) the system's stability in terms of the number of servers (λ_n) whose delivered throughput varies highly throughout the window's duration. The value of λ_n is used to determine the competing TCP connections' stability in time. Table 1 lists the variables used for bottleneck estimation, and table 2 exposes the different cases where bottlenecks can be detected and estimated.

The adaptation of the simultaneously used server number follows the standard Additive-Increase Multiplicative-Decrease (AIMD) algorithm to avoid network congestion when a client-side bottleneck is detected. In the presence of a client-side bottleneck, the number of servers is halved in order to avoid client link congestion that could lead to erroneous GoP distribution decisions due to the unstable nature of multiple TCP connections competing for a limited bandwidth. When a server-side bottleneck is detected, the number of servers is increased (by selecting a server in the server list made available in the manifest file) aiming at reaching a higher delivered throughput. From the client side, the type of bottleneck cannot always be determined. Therefore, should the bottleneck-type be unknown for a long sequence of segment downloads, the number of servers is incremented so as to trigger observable behaviors favorable for bottleneck-type detection, c.f. table 2. When the decision to add a server to the streaming session is taken, the MS-Stream has no previous record of the throughput provided by this new server. Consequently, this server's contribution to the streaming session will be as low as possible (one GoP at lowest quality only) in order to insure a minimum impact of sub-stream loss on end-user's QoE. It ought to be noted that if a server cannot deliver at least this in the given delay, it is removed from the used servers.

Algorithm 1: AIMD Server Adaptation

```
if thr_n < τ then
    bottleneck ← estimateBottleneck()
    if bottleneck == client then M ← M/2
    else if bottleneck == server then M ← M + 1
    else if M is the same for l segments then M ← M + 1
```

4.4. In-segment download adaptation: resiliency to heterogeneous network characteristics

The MS-Stream algorithm introduces adaptation decisions and actions during the download phase in order to ensure smooth video playback experience in the event of suddenly low performing paths. Some sub-segments (the ones fully composed of GoPs or the ones with GoPs complementing the already received GoPs) can be independently played and some are dependent from the not yet delivered sub-streams. $M_{n,i}$, $M_{n,d}$ respectively denote the number of servers currently delivering independent and dependent sub-segments. $M_{n,done}$ is the number of servers that are done delivering sub-streams for segment *n* and $bufLevel$ denotes the buffer level occupancy in seconds. In order to avoid video freezing due to out-of-date sub-segments delivery, three client-side in-segment download adaptation rules have been designed:

(i) for a given content segment, if at least one playable video segment can be reconstructed and $bufLevel < t_i$, then all current sub-stream downloads are abandoned. This reactive rule ensures uninterrupted video experience while providing a temporary sub-optimal visual quality to the end-users. By monitoring sub-segment download progress, estimations are made on sub-segments completion time;

(ii) in the event where $M_{n,i} \geq 1$ and $bufLevel > t_{ii}$, if all $M_{n,i}$ servers are estimated to finish delivering their sub-segments when $bufLevel < t'_{ii}$, the most performing available server (throughput-wise) $\widehat{s}$ is selected: $\widehat{s} = s_{m,n} \in \{\{S_{M_{n,i}} \cup S_{M_{n,done}}\} : x_{m,n} = max[X_{M_n}]\}$. According to the already delivered GoPs, to the missing ones (from the late independent $M_{n,i}$ sub-streams), and to $\widehat{s}$'s throughput, the MS-Stream client computes the optimal independent sub-stream that could be delivered from $\widehat{s}$ before $bufLevel < t'_{ii}$. If such a sub-stream exists, the late $M_{n,i}$ downloads are canceled and the new sub-stream is requested from $\widehat{s}$.

(iii) in the event where $M_{n,d} = M_n$ and $bufLevel > t_{iii}$, if $k_{\in[1..M_{n,d}]}$ of the $M_{n,d}$ servers are estimated to finish delivering their sub-segments when $bufLevelt'_{iii}$, the most performing available server $\widehat{s}$ is selected $\widehat{s} \in \{S_{M_{n,k}} \cup S_{M_{n,done}}\}$. The MS-Stream client computes the optimal dependent sub-stream that could be delivered from $\widehat{s}$ before the end of the late *k* sub-streams. If such a sub-segment exists, the *k* downloads are abandoned and the new sub-segment is requested from $\widehat{s}$. These two reactive rules provide the means to re-assign sub-segment downloads to the most available server in order to provide uninterrupted streaming. Moreover, by setting $t_i < t_{ii} < t_{iii}$ (fig.5) with t_{iii} high enough, rule (ii) and

Table 3. Dataset

Resolution	Bitrate	Resolution	Bitrate	Resolution	overheadBitrate
1920x1080	8Mbps	1280x720	4Mbps	1920x1080	250Kbps
1920x1080	7Mbps	1280x720	3Mbps	1280x720	200Kbps
1920x1080	6Mbps	720xx480	2Mbps	720x480	100Kbps
1280x720	5Mbps	720xx480	1Mbps		

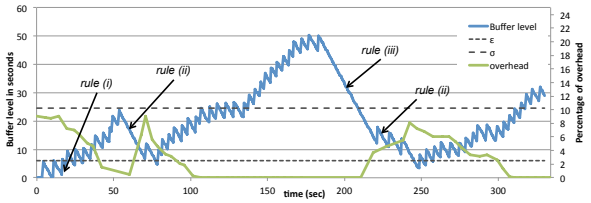

Fig. 6. Buffer Level and % of overhead

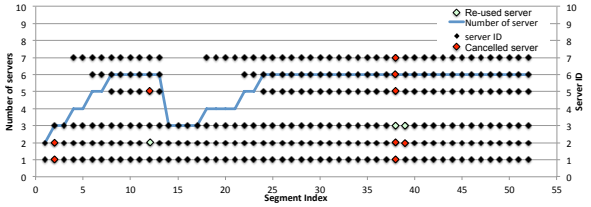

Fig. 7. Used servers

(iii) seek to maintain the buffer level in its area and to keep a lowest possible amount of overhead (between ε and σ for rule (ii); and as close as possible to the upper bound σ for rule (iii)).

5. EVALUATION AND PERFORMANCE RESULTS

The MS-Stream client was implemented on top of the open-source DASH-IF dash.js code. The MS-Stream server was implemented in Java. In order to perform its complete evaluation, we first functionally validated the correctness of execution of each phases of the proposed algorithm in a controlled environment. Second, we focused on the bottleneck estimation method, as well as the CPU overhead coming from sub-segments generation. Finally, we empirically evaluated the MS-Stream behavior in a real deployed environment over the Internet, similar to the way an OTT provider would use it. The 10-minute Big Buck Bunny video (with 6-second segments, each containing 12 GoPs) was used for our experiments (c.f. Table 3).

5.1. Two-phase adaptation algorithm functional validation

For the purpose of this study, we used a set of 10 MS-Stream servers provisioned with the video. The equivalent of a 25 Mbps aggregated upload throughput was fairly distributed over the 10 servers, limiting each with a 2500 Kbps upload capacity. Additionally, we applied a 100ms delay to each server. We set the variables $\varepsilon = t_{ii} = 6$secs, $\sigma = t_{iii} = 25$secs, $X = 10\%, p = 0\%$, $\mu = 10\%, \gamma = 30\%$ and $\tau = 14$Mbps. We monitored over time the buffer level in relation with the overhead variation (Fig.6), the number of used servers and their IDs (Fig.7), the observed throughput on each server stacked in Fig.8, and the displayed video bitrate with the overhead bitrate and the global observed throughput (Fig. 9). From segment 1 to 8, the client detected server-side bottleneck and increased the number of used servers (up to 6) so as to reach the targeted network throughput τ. In the mean time, as the buffer occupancy level increased and reached a value higher than σ, the amount of overhead decreased from 10% to 0%. At segment 10, we purposely applied a download rate limitation at the client side. We can observe that the client applied the in-segment download adaptation rule (ii) and abandoned the downloads of two sub-segments before re-launching a request for a lighter sub-segment composed of 3 Mbps GoPs. Two segment downloads later, the client detected a client-side bottleneck

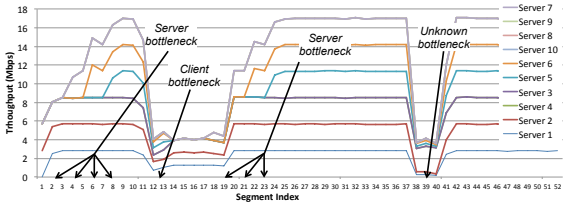

Fig. 8. servers' throughput

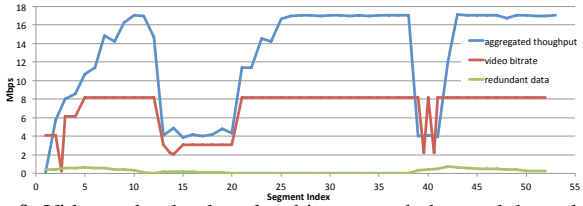

Fig. 9. Video and redundant data bitrates, and observed throughput

Table 4. Bottleneck estimation accuracy

Applied bottleneck	Estimated server bottleneck	Estimated client bottleneck	Estimated unknown bottleneck
Server	92.16%	3.72%	4.12%
Client	20.58%	65.46%	13.95%

and halved the number of used servers and avoided network congestion. We stopped the client download rate limitation at segment 19, enabling the detection of a server-side bottleneck, resulting in the consequent increase in the number of servers used. The client bottleneck was removed at segment 19 and the client could detect server-side bottleneck and increased the number of servers used. When the buffer level reached 50sec, we applied throughput upload limitations on 9 of the 10 servers for the duration of three segment downloads. At this point, the client determined an unknown bottleneck type (due to the 2-segment-download observation window) and did not increase the number of servers. The rules (iii) and (ii) of the in-segment download adaptation were triggered in this time lapse in order to ensure uninterrupted video playback and to provide the best possible segments quality to the end-user.

5.2. CPU overhead and bottleneck estimation performance

Regarding the impact of sub-segment generation on server, we compared a DASH and a MS-Stream server based on the processing time required to handle simultaneous (from 1 to 1000) client requests (normal segments for DASH; and sub-segment generation with random GoP distribution for MS-Stream). Segment bitrate were randomly selected from a pool of uniformly distributed content bitrate representation from Table 3. We could show that the required time for a MS-Stream server to process 99% of the requests is roughly 11% greater than for a DASH server.

Regarding the bottleneck estimation method performances, we re-used the same test-bed as in section 5.1. We respectively set λ_{Low} and λ_{High} to 50% and 75% of the number of currently used servers. We successively applied and removed 300 client-side download bandwidth variations and 300 server-side upload throughput maximum capacity variations (applied on one or many servers at the same time, but not all of them together). Then, we observed the bottleneck-type estimation made by the MS-Stream client. The results are exposed in Table 4. Our method performs well enough for the server-side bottleneck (in 92% of the cases). However, the client-side bottleneck estimation has room of improvement with 65% of good estimation, and 20% of the cases estimated as server ones. Future work will address the issue of variation of λ_{Low} and λ_{High} and their impact in the bottleneck estimation.

Table 5. Evaluation set

Eval ID	Global available throughput	Number of clients	Number of servers	X	ε $(= t_{ii})$	σ $(= t_{iii})$	τ (Mbps)
DASH_0	12.5 Mbps	1	1	-	-	-	-
DASH_1	25 Mbps	2	1	-	-	-	-
DASH_2	25 Mbps/10 servers=2.5 Mbps	2	1	-	-	-	-
MS-Stream_0	12.5 Mbps	1	10	10%	6	25	11.5
MS-Stream_1	25 Mbps	2	10	10%	6	25	15
MS-stream_2	25 Mbps	2	10	10%	12	35	15
MS-stream_3	25 Mbps	2	10	15%	12	35	15

Table 6. Evaluation results

Eval ID	Mean Bitrate (Mbps)	Quality Changes	Rebuff--ering	Start-up delay	Bandwidth overhead	Used servers
DASH_0	7.91	5.10	0	3.38 sec	0%	1
DASH_1	7.64	8.04	0.58	4.11s sec	0%	1
DASH_2	0.86	40.1	30.10	8.75 sec	0%	1
MS-Stream_0	7.89	5.21	1.16	3.07 sec	8.91%	9.52
MS-Stream_1	7.03	10.32	3.58	3.48 sec	4.37%	7.59
MS-Stream_2	7.49	8.11	2.07	3.13 sec	9.43%	7.68
MS-Stream_3	7.55	9.28	1.35	3.09 sec	11.12%	7.61

5.3. Empirical evaluations

In order to fully evaluate MS-Stream performance, we decided to set-up a real experiment over Internet as if MS-Stream was deployed by an OTT provider in users' set-top-boxes (cheap commodity and heterogeneous servers). We compared MS-Stream to a uni-source DASH solution (with expensive high-end and well-located server, using the open source dash.js from DASH-IF). To this end, we deployed 10 MS-Stream servers as set-top-boxes in users' homes in different locations in Europe: 3 in Bordeaux, 3 in Paris, 2 in Heraklion, 1 in Warsaw and 1 in Bucharest. The clients were located in Bordeaux. Table 5 summarizes the 7 evaluated streaming applications: *DASH_0* and MS-*Stream_0* with a single-client, *DASH_1/2*, MS-*Stream_1/2/3* with two concurrent clients. Each application targeted the 8Mbps bitrate (i.e., $Y = 8Mbps$). The global available throughput was fairly distributed among all servers. We empirically monitored the perceived quality at the consumer's side, i.e., QoE. To this end, we have derived a set of criteria, each considered essential for the QoE of video streaming services: quality distribution throughout the streaming session (Fig.10), average number of quality changes, mean displayed bitrate, average number of rebuffering, average start-up delay. Finally, we monitored the average number of simultaneously used servers along with the average bandwidth overhead (both essential for the efficiency of MS- Stream). The latter metrics are presented in Table 6. We repeated the video 50 times per application; representing a total playback time of 100 hours.

In the *DASH_0* and *MS-Stream_0* evaluations, the same amount of throughput is made available for one client. The obtained QoE metrics are similar in both cases, along with the targeted bitrate (8Mbps) reached most of the time. Regarding the *DASH_1* and *MS-Stream_1/2/3* evaluations, two clients are concurrently competing for the same network resources, hence introducing bandwidth diversity and variability in the network. The QoE results are equivalent, except for a few more rebuffering in the MS-Stream cases. Related to MS-*Stream_1/2/3*, the impact of $\varepsilon, \sigma, t_{ii}, t_{iii}$, and X on QoE and on the percentage of overhead exists. Indeed, higher values of $\varepsilon, \sigma, t_{ii}$, and t_{iii} enable a lower amount of rebuffering, a higher mean bitrate, and a better quality distribution. However, the actual percentage of overhead increases. Furthermore, a greater value of X permits the clients to avoid further rebuffering as more servers are allowed to deliver redundant data. As above-mentioned, the study on the optimal value of these parameters is left for future works. Finally, *DASH_2* shows that high QoE streaming with one commodity DASH server (such as set-top-boxes) for several clients is not feasible.

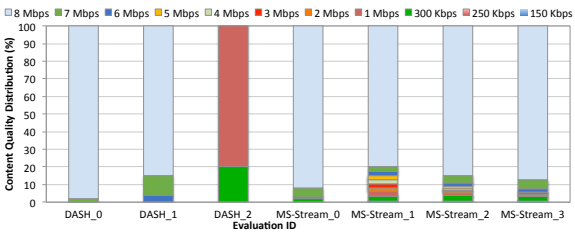

Fig. 10. Quality distribution throughout streaming sessions

6. CONCLUSION

Within this paper, we presented a solution to cost-effectively upgrade the legacy dynamic adaptive streaming protocols over HTTP to multiple-source streaming in heterogeneous environments. The proposed standard-compliant sub-segment generation and the two-phase adaptation algorithm were exposed so as to provide resiliency, flexibility and adaptability to network resource heterogeneity. The empirical evaluation exposed that MS-Stream in distributed heterogeneous environments can perform as good as high-end uni-source optimally-deployed DASH solutions. Thereby, our proposal shows that video streaming could be moved to distributed heterogeneous infrastructures in order to provide cheaper video delivery solutions compared to the existing CDNs and with respects to the drastic video traffic increase forecasts. Future works will address the optimal variables trade-offs in overhead selection and bottleneck estimations, and will combine MS-Stream and P2P streaming to further reduce video streaming deployment costs.

7. REFERENCES

[1] J. Bruneau-Queyreix, M. Lacaud, and D. Negru, "A multiple-source adaptive streaming solution enhancing consumer's perceived quality," in *IEEE Consumer Communications and Networking Conference (CCNC), demonstration track*, Las vegas, United States, 2017, `http://msstream.net`.

[2] "Cisco Visual Networking Index, 2016," .

[3] J. Bruneau-Queyreix, M. Lacaud, D. Negru, J. Mongay Batalla, and E. Borcoci, "Adding a New Dimension to HTTP Adaptive Streaming through Multiple-Source Capabilities," *IEEE MultiMedia*, 2017.

[4] Mark Watson, "HTTP Adaptive Streaming in practice," .

[5] V. K. Adhikari et al., "Unreeling netflix: Understanding and improving multi-CDN delivery," *IEEE INFOCOM*, 2012.

[6] G. Tian and Y. Liu, "Towards Agile and Smooth Video Adaptation in HTTP Adaptive Streaming," *IEEE/ACM Transactions on Networking*, vol. 24, no. 4, 2016.

[7] S. Zhang et al., "Presto: Towards fair and efficient HTTP adaptive streaming from multiple servers," *IEEE International Conference on Communications (ICC)*, 2015.

[8] W. Pu, Z. Zou, and C. W. Chen, "Dynamic Adaptive Streaming over HTTP from Multiple Content Distribution Servers," *IEEE Global Communications Conference (GLOBECOM)*, 2011.

[9] J. Bruneau-Queyreix et al., "Multiple Description-DASH: Pragmatic streaming maximizing users Quality of Experience," *International Conference on Communications (ICC)*, 2016.

[10] J. Bruneau-Queyreix et al., "MS-Stream: A multiple-source adaptive streaming solution enhancing consumer's perceived quality," *IEEE Consumer Communications and Networking Conference (CCNC)*, 2017.

[11] X. Corbillon et al., "Cross-layer scheduler for video streaming over MPTCP," *ACM International Conference on Multimedia Systems (MMSys)*, 2016.

[12] A. Singh et al., "Performance comparison of scheduling algorithms for multipath transfer," *IEEE Global Communications Conference (GLOBECOM)*, 2012.

[13] N. Kuhn et al., "DAPS: Intelligent delay-aware packet scheduling for multipath transport," *IEEE International Conference on Communications (ICC)*, 2014.

[14] T. Kurosaka and M. Bandai, "MPTCP with multiple ACKs for heterogeneous communication links," *IEEE Consumer Communications and Networking Conference (CCNC)*, 2015.

[15] C. Raiciu et al., "How Hard Can It Be? Designing and Implementing a Deployable MPTCP," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012.

[16] M. Kazemi et al., "A review of multiple description coding techniques for error-resilient video delivery," *Springer Multimedia Systems*, vol. 20, no. 3, pp. 283–309, 2013.