Received: 16 July 2018 | Accepted: 26 June 2019

DOI: 10.1002/nem.2081

SPECIAL ISSUE PAPER

WILEY

Muslin: A QoE-aware CDN resources provisioning and advertising system for cost-efficient multisource live streaming

Simon Da Silva[1] | **Joachim Bruneau-Queyreix**[1] | **Mathias Lacaud**[1,2] | **Daniel Negru**[1] | **Laurent Réveillère**[1]

[1] Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, Talence, France

[2] Joada SAS, Bordeaux, France

Correspondence
Simon Da Silva, Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, Talence F-33400, France.
Email: contact@simondasilva.fr

Funding information
CHIST-ERA, Grant/Award Number: DIONASYS ANR-14-CHR2-0004; H2020 LEIT Information and Communication Technologies, Grant/Award Number: Internet of Radio-Light 761992

Summary

Delivering video content with a high and fairly shared quality of experience is a challenging task in view of the drastic video traffic increase forecasts, as live video traffic will grow 15-fold by 2022. Currently, content delivery networks provide numerous servers hosting replicas of the video content, and consuming clients are redirected to the closest server. Then, the video content is streamed using adaptive streaming solutions. However, servers and network links often become overloaded during major events, and users may experience a poor or unfairly distributed quality of experience, unless more servers are provisioned. In this paper, we propose Muslin, a streaming solution supporting a high, fairly shared end users' quality of experience for live streaming, while minimizing the required content delivery platform scale. Muslin leverages on MS-Stream, a content delivery solution, which aggregates video content from multiple servers to offer a high quality of experience for its users. Muslin dynamically provisions servers and replicates content into servers and advertises servers to clients based on real-time delivery conditions. We have used Muslin to replay a 1-day video-games event, with hundreds of clients and several test beds. Our results show that our approach outperforms traditional content delivery schemes by increasing the fairness and quality of experience at the user side with a smaller infrastructure scale.

1 | INTRODUCTION

End users' Quality of Experience (QoE) is a crucial factor for the success of the increasing number of video streaming services and especially live streaming. According to Cisco,[1] video traffic is experiencing a tremendous growth and is expected to exceed 82% of the total Internet traffic by 2022, and live video will grow 15-fold to reach 17% of all video traffic by 2022. Most of the time, such traffic-increase forecasts are not followed by the necessary upgrade of core networks capacity because of the important costs it incurs, and major issues arise with respect to the QoE of such services. QoE and fairness are thus a rising issue as servers and network links become overloaded.

Content Delivery Networks (CDNs) are extensively used for the delivery of video content over the Internet. In such architectures, geographically distributed replica servers located as close as possible to the consuming clients are provisioned in advance with sufficient capacities using estimates of the expected workload. When accessing a content,

consuming clients are automatically redirected to the closest server so as to temper network congestion and achieve higher throughput. Although CDN solutions can handle a large volume of requests, they laboriously adapt to the highly dynamic and volatile nature of live streaming service audiences. As a consequence, the streaming infrastructure can rapidly be either overscaled incurring extra expenditures or under-sized and thus delivering degradated QoE to end users.

In addition to CDN-based solutions, streaming services usually rely on HTTP Adaptive Streaming (HAS) solutions, often relying on the widely adopted *Dynamic Adaptive Streaming over HTTP*(DASH) standard. Such solutions enable consuming clients to dynamically adjust the requested content bitrate according to the observed network conditions or to the client buffer occupancy. However, if a large amount of end users located under the same geographic area is simultaneously consuming the same streamed content, the nearest server may become rapidly overloaded. Some users may consequently suffer throughput degradation or content unavailability and may experience a poor or unfairly shared QoE as they compete for limited network and server resources.

We introduce `Muslin`, a streaming solution supporting a high, fairly shared end users' QoE for live streaming services over the Internet. To do so, `Muslin` relies on periodic feedbacks from `Muslin` clients during streaming sessions and a ranking score for servers provisioning and advertising. As shown on Figure 1, the `Muslin` server provisioning module periodically estimates the required throughput to dynamically adjust the infrastructure scale according to real-world needs. The `Muslin` server selection module then advertises relevant content servers to clients depending on multiple criteria such as distance, bandwidth, and server load. For content delivery, `Muslin` leverages on MS-Stream, a multiple-source streaming solution based on the DASH standard, in which a client can simultaneously use several servers to aggregate throughput from multiple channels and offer a higher QoE for its users (see Section 2.2).

This paper extends previous work[2,3] by providing additional experiment details and results, updated background on QoE including latest DASH and ITU-T works, several new relevant references, and many clarifications on `Muslin` with new figures and clarifications. The rest of this paper is organized as follows. Section 2 provides technical background about HTTP adaptive streaming, and Section 3 presents related research work. Section 4 describes the `Muslin` solution and introduces the provisioning and selection modules. We detail our experimental setup in Section 5 and present our evaluation results in Section 6. Finally, Section 7 concludes and presents future work.

2 | TECHNICAL BACKGROUND

Video content delivery solutions over the Internet have evolved a lot during the last two decades. Lately, HAS solutions have seen important interest in the industry and research, mainly due to their capabilities to render smooth video playback to the consumers, hence a better QoE. Many HAS solutions have emerged, such as Adobe HTTP Dynamic Streaming,[4] Apple HTTP Live Streaming,[5] Microsoft Smooth Streaming,[6] and the Dynamic Adaptive Streaming over HTTP (DASH) standard.[7]

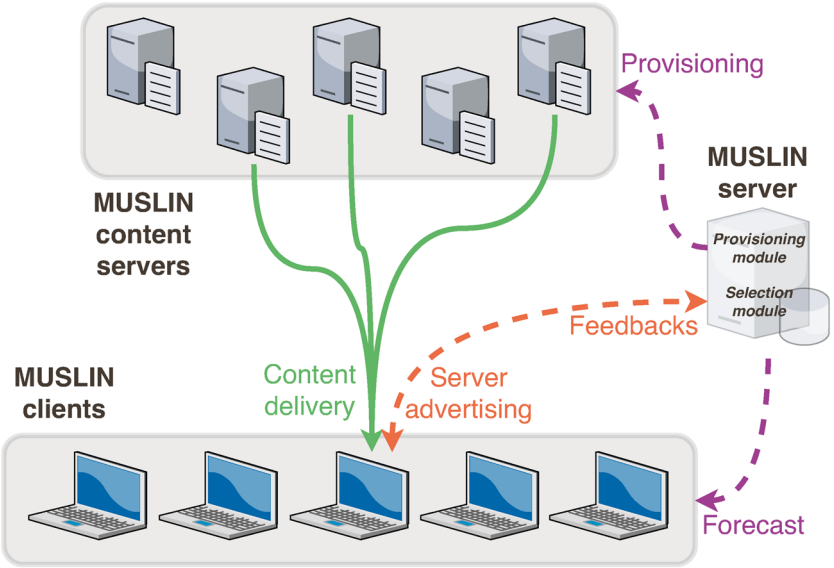

FIGURE 1 `Muslin` system overview

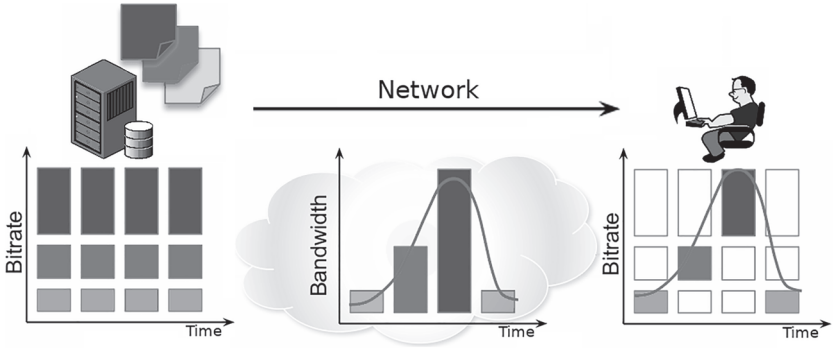

FIGURE 2 Dynamic Adaptive Streaming over HTTP (DASH) standard content delivery overview

2.1 | HTTP adaptive streaming and dynamic adaptive streaming over HTTP

The MPEG-DASH standard, widely adopted in the industry, aims at delivering uninterrupted multimedia content through the network via conventional HTTP traffic.[7] As shown in Figure 2, in a DASH server, different representations of the content split over segments of a few seconds are made available to the consuming client at alternative bitrates. Segments are composed of video frames' sequences gathered into independent units called *Groups of Pictures* (GoPs). A manifest file (the *Multimedia Presentation Description*, MPD) details the representations that are available for every segment and also provides a list of servers, where these segments can be accessed at. The MPD is initially handed out to the client, which then proceeds to retrieve the segments at the desired quality. During the streaming session, the client can dynamically switch the desired representation to another one so as to adjust to the network conditions or to its buffer status.

However, the work of Adhikari et al[8] advocates that QoE would greatly benefit from the venue of a practical HAS that can actually utilize multiple servers simultaneously. Even though there are some propositions for multiple servers streaming,[9,10] to the best of our knowledge, none of the existing approaches provide a high QoE through both redundancy between independent subsegments (to avoid rebufferings) and bandwidth aggregation (to reach a higher visual quality).

2.2 | Multiple-Source Adaptive Streaming (MS-Stream)

The MS-Stream over HTTP[11-14] solution is a proposition that extends the DASH standard, wherein a client can simultaneously utilize multiple servers in order to aggregate bandwidth over multiple links while being resilient to network and server impairments. When the bottleneck is located in the "last mile," that is to say at client side, MS-Stream performs similarly to traditional DASH-based streaming.

In MS-Stream, for a given video segment, each considered server delivers a video subsegment to the client. As shown in Figure 3, subsegments are generated by interleaving GoPs at different bitrates for the same segment: a high desired bitrate and a critically low bitrate (redundant bitrate). Some GoPs may also not be transmitted when enough redundancy is reached or under good delivery conditions. The redundant bitrate is set to critically low values (eg, 150 Kbps) in order to provide video playback at the lowest possible network transfer cost. Reconstructing the original content quality is achieved by selecting the GoPs of higher size in the pool of received subsegments at client side. Should some subsegments be missing, the content is still playable by relying on the redundant GoPs, hence displaying a suboptimal visual quality but providing reliability and less rebufferings in fluctuating network conditions.

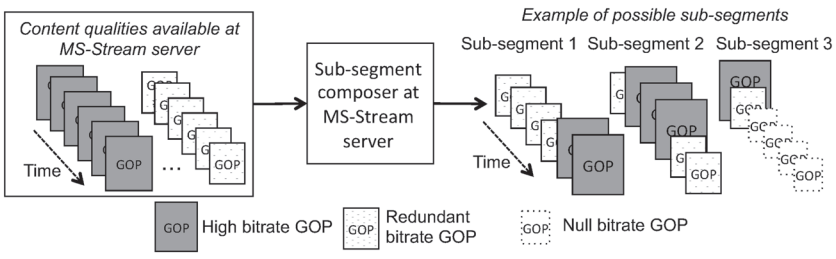

FIGURE 3 Subsegment generation and composition

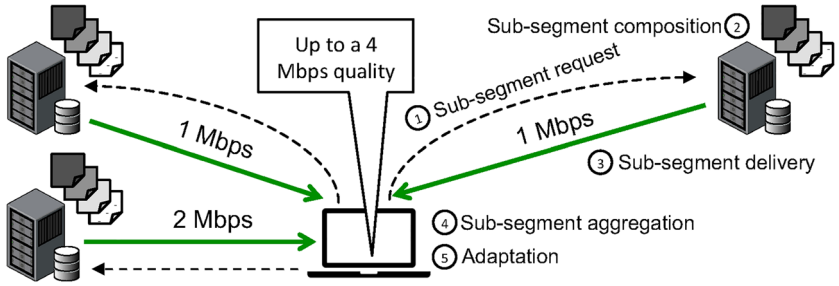

FIGURE 4 Multiple-Source Adaptive Streaming (MS-Stream) content delivery overview

An overview of the MS-Stream functioning is depicted in Figure 4. To enable the content delivery, an MPD file containing the available MS-Stream servers and video segments is delivered to the client before the content delivery and periodically refreshed during the streaming session. First, the client instructs MS-Stream servers to generate and deliver subsegments composed of GoPs from the representations available (listed in the MPD file). Then, the MS-Stream client merges the received subsegments to reconstruct a playable video segment with the highest possible visual quality given the available bandwidth. The client can adapt the number of simultaneously used servers according to the observed network conditions and to a targeted bitrate. The client attempts to minimize the bandwidth consumption overhead resulting from GoPs redundancy. It ought to be noted that the generation and aggregation of subsegments have very low processing footprints[14] as they only require to assemble already encoded GoPs available at different bitrates. A demonstration of MS-Stream is available online.[15]

3 | RELATED WORK

Video streaming is a trending topic both in research and in the industry, as consumers' demand is continuously growing. Servers provisioning, video content replication, and servers advertising are key problematics for CDN operators. Most CDN operators thus keep their policies secret,[16] as they often have a strong impact on cost and end-user QoE.

3.1 | Content replication policies

Optimizing content replication is a difficult task. Replicating content so as to minimize the network distance for requesting clients is NP-complete.[17] Therefore, advanced content caching algorithms are mostly heuristics. The most widespread content caching and replication techniques are based on greedy heuristic algorithms. It is usually done by maximizing a utility function[18] or minimizing a cost function.[19,20] Other policies consider social relationships between users and forecast the trending videos.[21] Our work is also based on a greedy iterative algorithm; however, it differs from these propositions. First, live content is only stored for a short time and required fast computation and decision, as opposed to on-demand streaming, where caching policies can converge over time.

Besides, some works use network awareness[22] and QoS metrics to route requests or to select servers. Zheng et al[23] base their approach on path latency optimization through multiple servers but not bandwidth or system scale. Similarly, Puntheeranurak et al[24] only take into account latency, delay, and jitter inside the network. As opposed to these approaches, `Muslin` aims at reaching a high end-user QoE and takes into account not only network measurements but also live clients' feedbacks to provision servers.

3.2 | Servers selection for a high and fairly shared QoE

Although CDN operators keep their strategies secret, the usual paradigm is to estimate the audience for an event and to provision enough servers near end users to withstand the demand.[16] Then, when clients request video content, the CDN strategy is to route their requests to the nearest server thanks to DNS[25] or IP anycast[26] and use HAS protocols for delivery. This behavior minimizes network-induced latency and lowers the probability to encounter congestion. For instance, Adhikari et al[8] introduced the DASH framework of Netflix, the largest DASH provider worldwide, and outlined that a user is always bound to one server, regardless of network issues. Consequently, one major drawback is that servers can get overloaded, and thus, some clients may receive a poor QoE or might even not have access to the content at all.

Muslin takes into account not only the distance but also the server bandwidth and requests failure (timeout) rate, enabling to provide a better QoE to the users.

Besides, there have been some attempts to reach a better QoE fairness between HAS clients. Georgopoulos et al[27] use Software Defined Networks to allocate bandwidth to each link, and Petrangeli et al[28] adapt the video bitrate requested by clients. However, to the best of our knowledge, all approaches towards higher QoE fairness are single-source oriented and do not consider dynamically advertising servers to the clients.

4 | MUSLIN: MULTISOURCE LIVE STREAMING

Muslin's goal is to provide a high and fairly shared QoE for live video content delivery. As QoE is subjective, it is a difficult challenge to evaluate the quality of experience of end users. QoE depends on many criteria, such as stalls, video resolution, encoding quality factor, bitrate fluctuation over time, and glitches. The ITU-T recently provided automated methods to algorithmically assess streaming QoE according to multiple factors in the P.1203 recommendation.[29] As it is complex and costly to take all parameters into account, Muslin tackles the main reasons why end users are not satisfied with their streaming experience, which are the number of rebuffering events, the average video bitrate displayed, and the number of quality changes during the session. Indeed, rebuffering events are considered the main negative impact on perceived QoE,[30] and both the average video bitrate and the quality changes have a significantly higher influence on QoE in adaptive streaming than other criteria. [31]

Muslin intends to solve the root causes for such QoE degradation, the two main reasons being (a) the server load and (b) the low bandwidth between the server and the client. Indeed, if a server is overloaded or if the network channel bandwidth to this server is low, clients requests to this server will timeout and cause rebufferings or visual quality degradation. Therefore, Muslin is able to monitor current delivery conditions to adapt its delivery schemes.

The Muslin system is composed of a Muslin server, MS-Stream clients, and MS-Stream content delivery servers with an additional Muslin layer to handle feedbacks and provisioning. Muslin clients send periodic feedbacks to the Muslin server, including the observed bandwidth from each server, the video subsegment requests failure (timeout) rate, their average displayed video bitrate, the number of rebufferings they experience, and the number of quality changes. Then, based on these feedbacks, the Muslin server accordingly scales the underlying delivery platform to provide a higher QoE to end users.

Fairness among users is mostly achieved thanks to the periodic feedbacks sent from the clients. They aim at monitoring the QoS and QoE each user is provided with and improve server provisioning and selection accordingly. Server and Network Assisted DASH (SAND),[32] introduced in MPEG-DASH Part 5, defines a standard for control messages exchanged between the servers and clients to report metrics. Muslinfeedback messages are currently not compliant with SAND, as Muslin was developed prior to this standard but will be in a future version for better interoperability. Besides, MS-Stream allows to maximize server throughput and reduce competition between clients when CDN servers' resources are saturated, as each client depends on multiple servers and is not bound to a specific one.

Muslin is specifically effective for live streaming, where churn rate can be very high, and important audience fluctuations can happen within seconds. In a VoD use case, clients' buffers can be larger, and there is less pressure to react in real time.

As illustrated in Figure 5, (**1**) the Muslin server dynamically provisions content servers and replicates content to available MS-Stream content delivery servers, which then register themselves to the selection module; (**2**) when a client

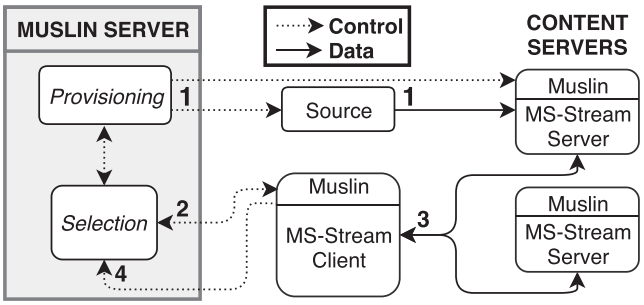

FIGURE 5 Muslin system architecture overview

requests an MPD file, the selection module replies with a list of available servers; (**3**) the client can access live content and begin the streaming session with the MS-Stream protocol; (**4**) `Muslin` clients send periodic feedbacks. In this section, we present in details the `Muslin` system and the `Muslin` server two main components, the provisioning module and the selection module.

4.1 | Provisioning module

The provisioning module goal is to decide on the number of servers to provision not only to answer end users throughput demand in video contents but also to maximize their QoE and minimize the required infrastructure scale. To do so, it periodically estimates the required throughput to fulfill the demand based on actual feedbacks and provisions a subset of servers to host the content. The provisioning module period T is equal to the length of two segments (typically 10 s).

4.1.1 | Audience forecast

In order to estimate the demand, `Muslin` computes the future number of clients during each period T. The current audience is defined as v_t. The estimated audience at the next iteration ($t+T$) is labeled $\widehat{v_{t+T}}$. Finally, Δv represents the change in number of viewers, that is to say $\Delta v = v_t - v_{t-T}$. `Muslin` estimates the audience with the following formula:

$$\widehat{v_{t+T}} = v_t + \Delta v. \tag{1}$$

As the actual replication is mostly based on clients feedbacks, a more accurate estimation is not required.

4.1.2 | Throughput estimation

`Muslin` throughput estimation algorithm uses the demand forecast $\widehat{v_{t+T}}$ to estimate how much throughput D the overall system must provide to the users. Each client tries to reach a target quality (highest available video bitrate) Q. Because of MS-Stream specification, the subsegments redundancy adds a network bandwidth overhead percentage O (up to a user-defined parameter). Besides, we introduce C, a dynamic corrective coefficient to address the network and server issues. It takes into account the mean average video bitrate B ($B \leq Q$) displayed by all clients watching the stream and the failure rate FR, which is the proportion of clients who failed to obtain in time the response of their last request from the server.

$$C = \frac{Q}{B} * (1 + FR) \tag{2}$$

The dynamic coefficient C allows the system to scale according to current clients QoE. It is then possible to compute the required system throughput that will be requested by the clients, using the following formula:

$$D = C * \widehat{v_{t+T}} * (Q + O). \tag{3}$$

4.1.3 | Provisioning decision

The provisioning module decides which servers to provision. To do so, the provisioning module periodically computes a server Ranking Score RS_s for each server s (including offline servers), based on clients and servers proximity and on feedbacks gathered periodically from all clients:

$$RS_s = (N_s * (1 - FR_s) * OBW_s)^{\frac{1}{3}}. \tag{4}$$

As shown in Equation 4, the RS_s takes into account the number of nearby clients N_s, the failure rate FR_s, and the average observed bandwidth OBW_s for each server s by computing a geometric mean. The higher the score, the more likely the server to be provisioned. For each server, the number of clients for which this would be the closest content server is computed as N_s. `Muslin` clients report when servers fail to deliver a subsegment in time. This measurement

is aggregated into the failure rate FR_s. It represents the ratio of delivery failures detected over the total number of clients that requested a subsegment from this server during the last T seconds. Besides, all clients can estimate the bandwidth from a specific server by observing delivered throughput in past requests. Muslin can compute the average observed bandwidth estimate OBW_s for each server s.

First, the RS_s of content servers is computed, and they are sorted by decreasing order. If the target throughput D is greater than the current system maximum available throughput, more servers are iteratively provisioned (by descending RS_s order) until D is reached. Else, if the system is over-provisioned, the servers are deprovisioned according to their RS_s in ascending order.

4.2 | Selection module

The Muslin selection module goal is to advertise a subset of available content servers to each client. To this end, we define a client-specific Ranking Score RS_{sc}, in order to reach a high and fairly shared QoE:

$$RS_{sc} = ((D_{max} - GD_{sc}) * (1 - FR_s) * OBW_s)^{\frac{1}{3}}. \quad (5)$$

To order the list of available content servers, the selection module computes the RS_{sc} for all server s and client c, based on feedbacks periodically sent by Muslin clients during streaming sessions. Similarly to the provisioning score, the RS_{sc} is based on the distance between each client and server and on clients feedbacks. As shown in Equation 5, the client-specific ranking score includes the maximum distance between any two places on Earth (D_{max} kilometers, roughly 20 000), the geographical distance GD_{sc} using geoIP data inferred from IP addresses, the video subsegment delivery failure rate FR_s of server s (ie, the percentage of requests the server was not able to handle on time), and the average observed bandwidth OBW_s between all clients and server s.

When clients request a video content, the selection module returns an MPD file containing servers sorted by descending RS_{sc} order. Then, Muslin clients decide how many servers they use, based on MS-Stream adaptation strategies. As illustrated in Figure 6, if nearby content servers are already overloaded, the Muslin server selects and advertises other content servers with a higher RS_{sc} to the client. It prevents content starvation from clients and allows fairness among users independently from their geographic position or nearby servers.

4.3 | Implementation and scalability discussion

The Muslin modules and Muslin content servers overlay are implemented in Java and run inside light-weight Docker containers. Muslin content servers are built on top of MS-Stream servers by adding the necessary glue code to manage the interaction with the Muslin provisioning and selection modules. All interactions with the Muslin

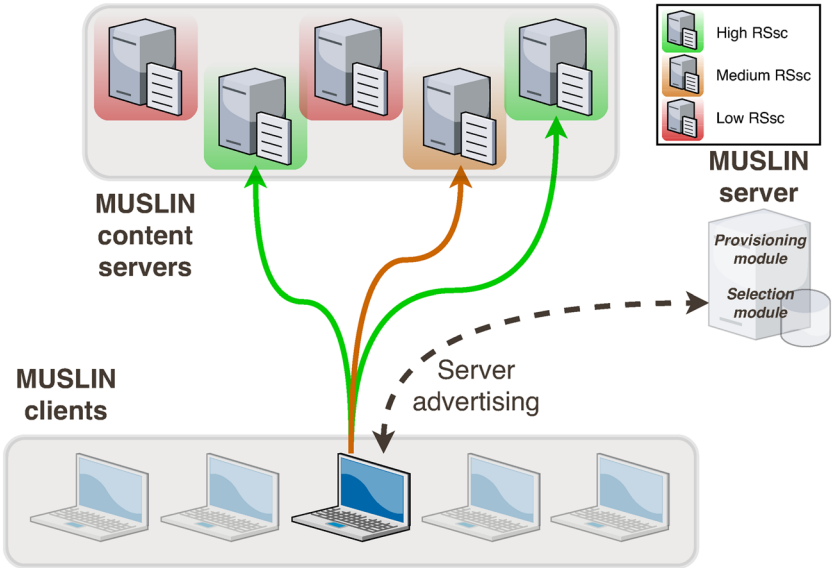

FIGURE 6 Muslin RS_{sc}-based servers selection example

modules fulfill the REST architecture style. Muslin clients are developed in pure JavaScript and run within any mobile or desktop Web browser. Clients extend MS-Stream clients by featuring periodic feedback reports to the Muslin server.

In terms of scalability issues, the Muslin system scales similarly to current HAS solutions as MS-Stream is compliant with the DASH standard. A scalability downside is due to the periodic clients' feedbacks as the Muslin server workload grows linearly with the number of clients. To solve this issue, we implement on the client a feedback request probability Pr to bound the number of feedbacks (see Equation 6).

$$\Pr = \min(1, N/v_t) \tag{6}$$

We thus ensure statistically that at most N clients will send a feedback for every period T, depending on the current audience v_t. With fewer feedbacks from the clients, the average estimated bandwidth and failure rates for servers are still correct but refreshed at a lower rate, resulting in temporary drops of QoE for some users.

Another scalability downside is due to the MPD refresh requests from Muslin clients every few segments or when they experience a poor QoE. Similarly to the clients' feedbacks, the Muslin server can become overloaded when too many clients request a new MPD file. To solve this issue, the Muslin selection module is distributed across several network nodes, each node only handling nearby clients requests (routed using classic DNS-based schemes).

5 | EXPERIMENTAL SETUP

In order to evaluate our approach, Muslin was deployed and compared with various strategies that are commonly used. In the remainder of this section, we describe in details each implemented strategy and then present the test beds and the audience trace we use for our experiments.

5.1 | Provisioning, forecast, advertising, and delivery policies

To evaluate Muslin, we implemented several common and alternative strategies as summarized in Table 1.

5.1.1 | Provisioning

Although CDN operators keep their strategies secret, the usual paradigm is to replicate content near end users and to balance the load across multiple servers. Therefore, we implement two provisioning policies: *geographical* and *random*. The *geographical* policy is aware of the clients' locations and replicates the content to servers near locations with the most clients. The *random* policy replicates content to randomly selected servers.

5.1.2 | Audience forecast

Usually, CDN operators try to estimate the audience for an event and then provision enough servers near end users in advance to withstand the demand. Therefore, we implement an *oracle* forecast, which is aware of the exact amount of viewers and their locations at any time. This policy is of course unreachable in real life, but it provides a best-case current paradigm comparison.

TABLE 1 Provisioning policies, audience forecast, selection policies, and delivery protocols

Provisioning	Forecast	Selection	Protocol
Muslin	Muslin	Muslin	MS-Stream
Geographical	Estimate	CDN	DASH
Random	Oracle	Random	
		Round Robin	

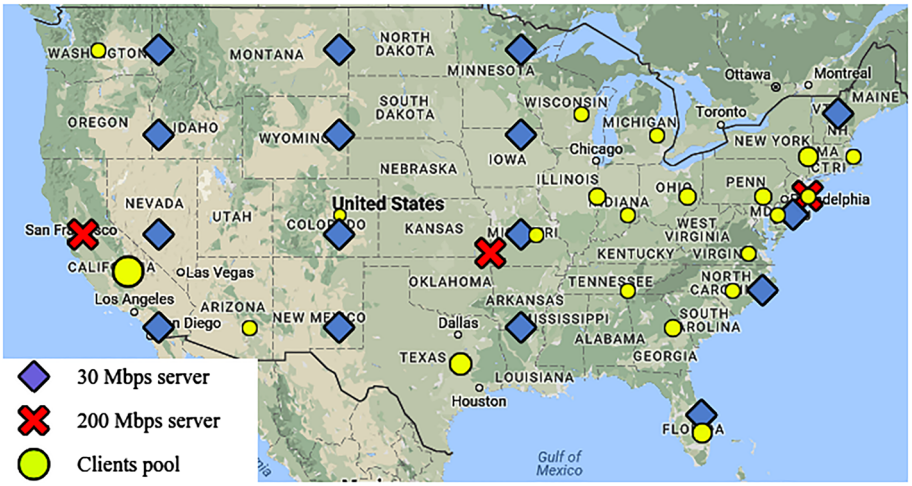

FIGURE 7 US map with points of presence and clients

On the contrary, in the *estimate* strategy, the audience is periodically estimated with the strategy depicted in Equation 1.

5.1.3 | Selection policy

We implement three selection policies called *CDN*, *Random*, and *Round Robin*. The *CDN* strategy is the most widespread one. It consists in routing clients to the nearest provisioned servers. In the *Random*policy, servers in the MPD file are randomly selected and sorted. The *Round Robin* policy balances the load among available servers, as servers within the MPD file are permuted for each new client request.

5.1.4 | Content delivery

To deliver video content, we used the multisource MS-Stream solution and the single-source DASH standard.

5.2 | Servers and clients setup

We evaluate our proposal in an actual environment. To do so, we set up 19 servers and 60 clients in our test beds according to the US map (see Figure 7). We also chose an actual audience trace to generate clients churn.

5.2.1 | Test beds

As shown in Table 2, we set up multiple Points of Presence (PoP) geographically distributed in the United States on a local network, by computing the latency and bandwidth between each client and server according to the geographical distance. Those PoP are set up according to two test beds. In the first test bed, 200-Mbps servers are available in three strategic locations (West, center, and East). In the second test bed, we use 30-Mbps servers located in *16* US states. We

TABLE 2 Available servers for each setup

ID	Location	Upload (Mbps)
3	California, USA	200
3	Kansas, USA	200
3	New York, USA	200
16	16 states	30

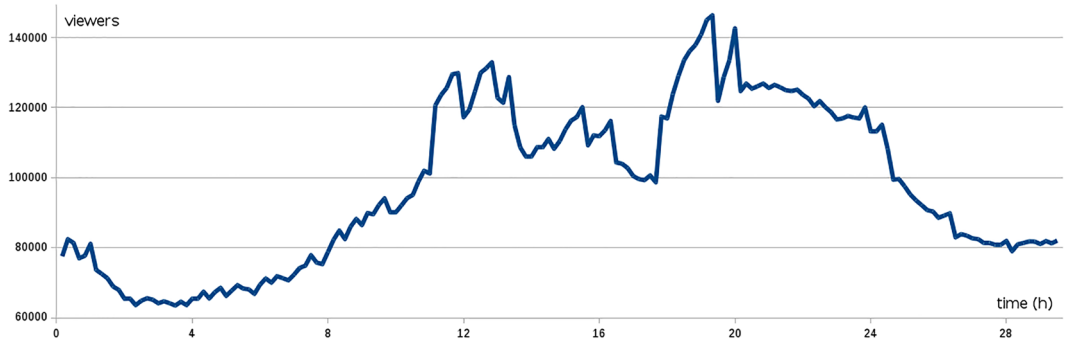

FIGURE 8 AGDQ audience trace

TABLE 3 Available video qualities

ID	Bitrate, bps
0	205.129
1	1.012.240
2	2.029.450
3	4.086.016
4	6.391.489

chose 16 locations as most CDN providers have between 10 and 30 PoP,[33] and Google provides 16 locations.[34] Besides, we selected 21 client pools locations in the contiguous US states. We randomly distributed the clients in the states using a weighted probability matching the state population (eg, California: 13% and Texas 10%) as shown in Figure 7 and saved the output to reuse the same toss in all the experiments.

5.2.2 | Audience trace

In order to be consistent for all experiments runs, we selected an audience trace and replayed it every time by automatically connecting or removing video clients from the broadcast, thanks to Docker containers.

The audience profile we chose (Figure 8) is a real trace from a week-long charitable video games event streamed online. The audience used is from 08 July[35] as it contains many typical audience patterns, from 60 000 to 150 000 viewers over 30 hours. We scaled down the number of simultaneous clients to 60 (about 250 unique sessions throughout each experiment) as our experimental infrastructure could not support hundreds of thousands of connections. All clients are desktop with 30-seconds maximum buffer and 8-Mbps download bandwidth.

5.2.3 | Experiments

We perform our experiments using the `Muslin` system as described in Section 4 and the policies explained above. Our experiments consist in a 30-minute live streaming broadcast, rerun five times to aggregate results and reduce noise and outliers impact in the distributions. The used live video content is the Blender Big Buck Bunny video encoded in five video bitrates (see Table 3).

6 | EVALUATION RESULTS

In this section, we evaluate the delivery solutions and various policies in terms of cost, quality of experience, and fairness.

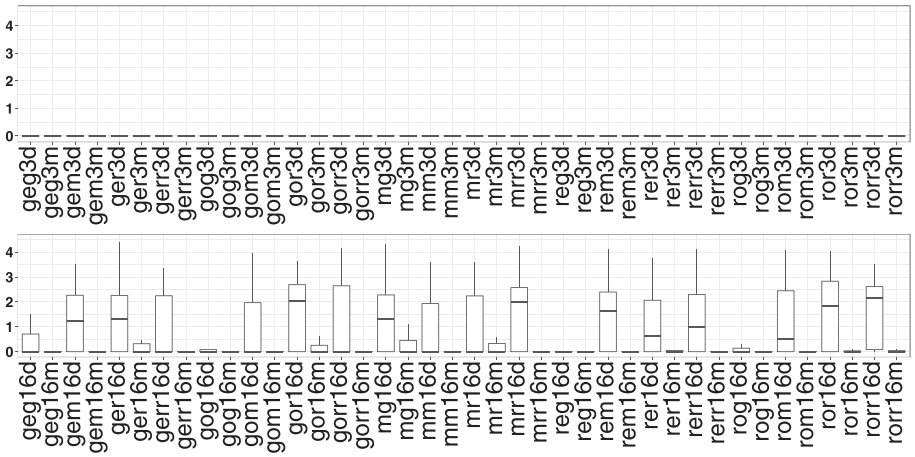

FIGURE 9 Number of rebufferings (per min), three servers test bed (top), 16 servers test bed (bottom)

6.1 | Delivery solutions

In this subsection, we evaluate MS-Stream against single-source DASH streaming. Figure 9 represents the number of rebufferings for each evaluated combination. The X-axes represent experiment setups with IDs in the following form: *provisioning* + *forecast* + *servers selection* + *test bed* + *protocol*. For instance, *geographicaloracle* replication with *random* servers selection on test bed *16* with *MS-Stream* protocol has ID *gor16m*.

The three servers test bed provides all clients with a rebuffering-free streaming session, because of the high servers capacities and capabilities. In the 16-server test bed, the *CDN* experiment performed with the DASH standard instead of MS-Stream results in half the clients suffering at least one rebuffering in their streaming session. But when using MS-Stream instead of DASH, all the clients have a continuous playback. More generally, every MS-Stream setup outperforms its DASH equivalent in terms of rebufferings, as most MS-Stream clients do not experience any rebuffering event at all.

In terms of bitrate, the results are quite similar between the two solutions in the three servers test bed. In the 16 servers test bed, MS-Stream provides a mean bitrate increase up to 4 Mbps over DASH and lowers the number of quality changes for all setups. Besides, the *CDN* setup with MS-Stream provides a more homogeneous distribution as all clients reach a quality higher than 6.2 mbps in both test beds. Oppositely, more than a quarter of DASH clients display less than 6.0 mbps in the 16 servers test bed. Finally, the gains and losses in the number of quality changes for the three servers test bed varies with no distinguishable global trend. In the 16 servers test bed, all clients experience less quality changes when using MS-Stream, with up to four less quality changes per minute.

All these results are explained as MS-Stream was mainly designed to increase the end user's perceived QoE by avoiding rebufferings, providing a smoother playback, and simultaneously utilizing the available bandwidth from multiple paths with heterogeneous characteristics, as previously mentioned.[14] The downside of these QoE improvements is a small CPU overhead, and the compulsory network bandwidth overhead induced by the MS-Stream solution (evaluated in Section 6.5), which corresponds to the percentage of data downloaded by the client but not used. Further details and additional MS-Stream evaluations are available in former works.[11-14] In the rest of this paper, we only consider the MS-Stream delivery solution, as it provides a greater QoE to the end-users for a small CPU and bandwidth overhead.

6.2 | Provisioning cost

`Muslin` aims at providing a high and fairly shared QoE through multisource live streaming, but it also aims at doing so while being cost-efficient when provisioning servers. To compute provisioning cost, we assume a cloud computing service using server time billing. Therefore, we sum the provisioned server time for each experiment run and compute relative values.

As shown in Table 4, the total server time required is lower when using audience estimates and dynamic server provisioning policies. Furthermore, as `Muslin` replication policy also takes into account the actual quality displayed by the

TABLE 4 Total relative cost (server time), 16 servers test bed

Provisioning	Forecast	Server Time
`Muslin`	`Muslin`	100
Geographical	Estimate	109
Geographical	Oracle	118
Random	Estimate	109
Random	Oracle	118

TABLE 5 Selected provisioning, forecast, selection, delivery policies, and test bed

Name ***Muslin***	**Provisioning** `Muslin`	**Forecast** `Muslin`	**Selection** `Muslin`	**Delivery** **MS-Stream**	**Test Bed** **16 servers**
CDN	Geographical	Oracle	CDN	MS-Stream	16 servers
Random	Geographical	Oracle	Random	MS-Stream	16 servers
RR	Geographical	Oracle	Round Robin	MS-Stream	16 servers

clients when dimensioning the delivery system, it does not overprovision if all clients can already obtain the target video quality and effectively lowers the number of servers provisioned when possible. `Muslin` replication is thus the least costly policy for the 16-server test bed, as it allows more flexibility. In the three-server test bed, all policies have roughly the same cost.

For better readability, we identify four relevant combinations, referred to as *Muslin*, *CDN*, *Random*, and *Round Robin* in the text, detailed in Table 5. The *geographical oracle* provisioning and forecast combination is impossible to reach in real life, but it provides a best-case current paradigm comparison.

6.3 | Quality of experience

To evaluate the end users' QoE, three main metrics are considered: the number of rebuffering events on Figure 9, the average video bitrate displayed on the user video player (Figure 10), and the number of quality changes during the session (Figure 11).

`Muslin` clients were able to reach a higher QoE compared with most current setups, as we demonstrate an increase of 100 kbps in median displayed bitrate, 2.5 less quality changes per minute, and almost no rebufferings compared with a best-case *CDN* implementation. The bitrate increase is due to the dynamic provisioning of content servers based on the actual clients demand. The quality changes and rebufferings decreases are a consequence of RS_{sc}-based servers selection, which prioritizes servers with available bandwidth and high-response rates.

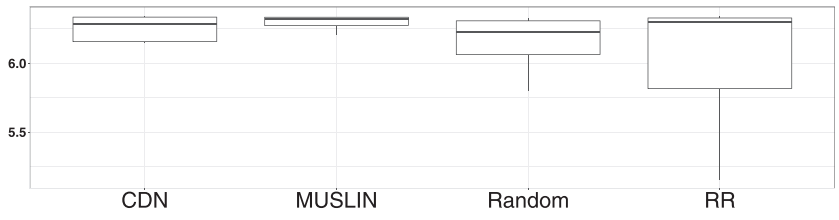

FIGURE 10 Displayed bitrate (Mbps), selected setups

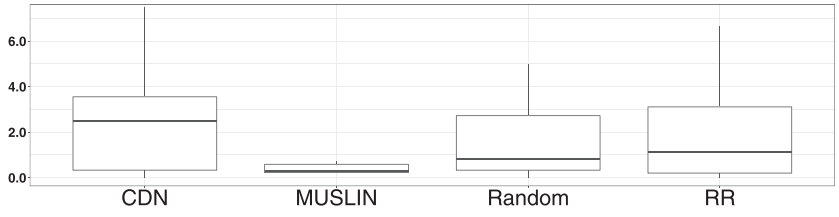

FIGURE 11 Quality changes per minute, selected setups

TABLE 6 QoE fairness (F index), selected setups

QoE metric	CDN	Muslin	Random	RR
Bitrate	0.7727	0.9610	0.5952	0.4685
Quality changes	0.4551	0.9485	0.5408	0.4660
Rebufferings	0.6952	0.9095	0.5179	0.6452

6.4 | QoE fairness

In this subsection, QoE fairness between clients is discussed. As shown above, Muslin median QoE results are better than a best-case *CDN* implementation, and the distributions are less spread than other setups, as the fairness among users is higher.

Indeed, as shown in Table 6, we registered an increase of 19.6% in bitrate fairness, 52% in quality changes fairness, and 23.6% in rebufferings fairness, using the F index (based on standard deviation, see Equation 7) described by T. Hoßfeld et al[36]:

$$F = 1 - \frac{2\sigma}{H - L}. \tag{7}$$

The main reason for such increases is the feedback-based RS_{sc} computation, enabling to advertise the most suitable servers for each client, which are not necessarily the closest ones. It also spreads the load evenly across all servers and avoids starvation that may happen for some clients in a traditional CDN scheme.

6.5 | Network overhead

As stated in Section 2.2, MS-Stream can use some redundancy in subsegments to reduce the number of rebufferings in case of server or network impairment. Figure 12 shows the total network overhead percentage required by MS-Stream clients for a few selected setups.

As Muslin dynamically provisions servers and advertises more suitable content servers to clients, MS-Stream manages to lower the required network overhead. Indeed, the MS-Stream clients detect that most servers are able to reply in time to video segments requests and, thus, lowers the redundancy in subsegments requests. On the contrary, when servers are selected randomly, the network overhead required is higher as the delivery of subsegments is inconsistent.

6.6 | Experiments summary and discussion

Muslin manages to increase QoE and fairness while lowering provisioning costs by combining dynamic provisioning with feedback-based servers selection and multiple-source content delivery. QoE and fairness increases compared with a best-case CDN setup are due to the servers selection taking server load and bandwidth into account and not only distance (thanks to the RS_{sc} ranking score). In our experiments, West coast and California CDN servers are particularly stressed as they are close to large clients pools. In a CDN setup, even if the audience is correctly estimated prior to the streaming session, all clients will contact the nearest server and might top off the maximum capacity of particular CDN servers in specific zones, thus reducing QoE and fairness.

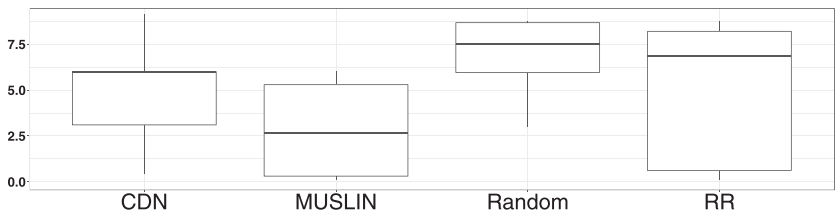

FIGURE 12 Network overhead (%), selected setups

7 | CONCLUSION

We presented Muslin, a multi-source live streaming system, which manages to reach higher QoE and fairness than currently adopted streaming systems. Muslin takes into account clients real-time feedbacks, dynamically replicates content and improves server advertising to clients to enhance users' QoE and fairness while minimizing the required infrastructure scale. We showed in our experiments that thanks to the coupling of MS-Stream with the proposed Muslin system, end users experienced almost no rebufferings, a higher video bitrate, and more evenly shared QoE, compared with existing state-of-the-art streaming systems setups. As future work, we will consider a more complex cost model taking into account scaling and network costs to further improve Muslin benefits towards infrastructure cost and cloud computing capabilities.

ACKNOWLEDGEMENTS

The research leading to these results was partially supported by the CHIST-ERA "DIONASYS" project under contract ANR-14-CHR2-0004 and by the EU Horizon 2020 program towards the Internet of Radio-Light project H2020-ICT 761992. We would like to thank the anonymous reviewers for their exhaustive and helpful remarks and suggestions.

ORCID

Simon Da Silva https://orcid.org/0000-0003-0903-9369

REFERENCES

1. Cisco. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.pdf; 2017.
2. Da Silva S, Bruneau-Queyreix J, Lacaud M, Négru D, Réveillère L. MUSLIN: Achieving High, Fairly Shared QoE Through Multi-Source Live Streaming. In: PV. ACM; 2018; New York, NY, USA; 18:54-59.
3. Da Silva S, Bruneau-Queyreix J, Lacaud M, Négru D, Réveillère L. MUSLIN demo: high QoE fair multi-source live streaming. In: MMSys ACM; 2018; New York, NY, USA; 18:529-532
4. Adobe. https://www.adobe.com/products/hds-dynamic-streaming.html; 2018.
5. Apple. https://developer.apple.com/streaming/; 2018.
6. Microsoft. https://www.microsoft.com/silverlight/smoothstreaming/; 2018.
7. Sodagar I. The MPEG-DASH standard for multimedia streaming over the internet. *IEEE MultiMedia*. 2011;18(4):62-67.
8. Adhikari VK, Yang G, Fang H, et al. Unreeling netflix: Understanding and improving multi-CDN delivery. *IEEE INFOCOM*. 2012.
9. Zhang S, Bo L, Baochun L, et al. Presto: Towards fair and efficient HTTP adaptive streaming from multiple servers. IEEE International Conference on Communications (ICC); 2015.
10. Pu W, Zou Z, Chen CW. Dynamic adaptive streaming over HTTP from multiple content distribution servers. IEEE Global Telecommunications Conference (GLOBECOM); 2011.
11. Bruneau-Queyreix J, Lacaud M, Negru D. A multiple-source adaptive streaming solution enhancing consumers perceived quality; 2017; Las vegas, United States.
12. Bruneau-Queyreix J, Lacaud M, Negru D, Batalla J, Borcoci E. Adding a new dimension to HTTP Adaptive Streaming through multiple-source capabilities. IEEE Multimedia. 2017.
13. Bruneau-Queyreix J, Lacaud M, Negru D, Mongay Batalla J, Borcoci E. MS-Stream: a multiplesource adaptive streaming solution enhancing consumers perceived quality; 2017; Las vegas, United States.
14. Bruneau-Queyreix J, Lacaud M, Negru D, Mongay Batalla J, Borcoci E. QoE Enhancement Through Cost-Effective Adaptation Decision Process for Multiple-Server Streaming over HTTP; 2017.
15. MS-Stream Demonstration: http://msstream.net; 2017.
16. Passarella A. A survey on content-centric technologies for the current Internet: CDN and P2P solutions. *Comput Commun*. 2012.
17. Bestavros A, Rabinovich M. Object replication strategies in content distribution networks. *Web Caching Content Deliv*. 2001;39.
18. Li W, Oteafy SMA, Hassanein HS. StreamCache: Popularity-based caching for adaptive streaming over informationcentric networks; 2016:1-6.
19. Sahoo J, Glitho R. Greedy heuristic for replica server placement in cloud based content delivery networks. In: 2016 IEEE Symposium on Computers and Communication (ISCC); 2016.
20. Lim K, Bang Y, Sung J, Rhee JKK. Joint optimization of cache server deployment and request routing with cooperative content replication; 2014:1790-1795.

21. Hu H, Wen Y, Chua T, Huang J, Zhu W, Li X. Joint content replication and request routing for social video distribution over cloud CDN: A community clustering method. *IEEE Trans Circuits Syst Video Technol*. 2016;26(7):1320-1333.
22. Batalla JM, Beben A, Chen Y. Optimization of the decision process in network and server-aware algorithms; 2012:1-6.
23. Zheng H, Tang X. The server provisioning problem for continuous distributed interactive applications. *IEEE Trans Parallel Distrib Syst*. 2016;27(1):271-285.
24. Puntheeranurak S, Sa-ngarmangkang N. An improvement of video streaming service using dynamic routing over OpenFlow networks; 2015:285-289.
25. Nygren E, Sitaraman Ramesh K, Sun J. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Oper Syst Rev*. 2010;44(3):2-19.
26. Flavel A, Mani P, Maltz David A, et al. *connect*. 2015;27:19.
27. Georgopoulos P, Elkhatib Y, Broadbent M, Mu M, Race N. Towards network-wide QoE fairness using openflow-assisted adaptive video streaming. In: ACM FhMN '13; 2013;15-20.
28. Petrangeli S, Famaey J, Claeys M, Latré S, De Turck F. QoE-driven rate adaptation heuristic for fair adaptive video streaming. *ACM Trans Multimed Comput Commun Appl (TOMM)*. 2016;12(2):28.
29. ITU-T. https://www.itu.int/rec/T-REC-P.1203; 2017.
30. Hobfeld T, Seufert M, Hirth M, Zinner T, Tran-Gia P, Schatz R. Quantification of YouTube QoE via Crowdsourcing; 2011 IEEE International Symposium on Multimedia; 2011:494-499.
31. Seufert M, Egger S, Slanina M, Zinner T, Hobfeld T, Tran-Gia P. A survey on quality of experience of HTTP adaptive streaming. *IEEE Commun Surv Tutorials*. 2015;17(1):469-492.
32. DASH-IF. https://dashif.org/docs/SAND-Whitepaper-Dec13-final.pdf; 2018.
33. CDNPlanet. cdnplanet.com/geo/united-states-cdn; 2018.
34. Google. cloud.google.com/cdn/docs/locations; 2018.
35. Twinge. twinge.tv/gamesdonequick/streams/\#/22233544288; 2018.
36. Hoßfeld T, Skorin-Kapov L, Heegaard Poul E, Varela M. Definition of QoE fairness in shared systems. *IEEE Commun Lett*. 2017;21(1):184-187.

AUTHOR BIOGRAPHIES

Simon Da Silva graduated from Bordeaux INP: ENSEIRB-MATMECA graduate school of engineering with an MSc in Telecommunications, specialized in Software Engineering for Computer Science and Networks. During his studies, he took part in several research and R&I projects, with a focus on experiments and implementation of proposals. He is currently a PhD student at University of Bordeaux (LaBRI). His research topics are video streaming, networks, cloud, blockchain, security, privacy, and trusted execution environments. Contact him at simon.da-silva@labri.fr or contact@simondasilva.fr.

Joachim Bruneau-Queyreix obtained his associate professor position at Bordeaux-INP in September 2018; he joined the CNRS-LaBRI lab as a permanent researcher in the PROGRESS team. His research focuses on systems, distributed systems, and security. Prior to joining CNRS-LaBRI, Joachim enrolled as a research fellow at the National Institute of Telecommunications in Warsaw Poland for 1 year. He received his PhD from the University of Bordeaux in 2017. Contact him at joachim.bruneau-queyreix@labri.fr.

Mathias Lacaud received his MSc in Telecommunications at *ENSEIRB-MATMECA graduate school of engineering*, Bordeaux, in 2016. He worked as a research engineer in the field of optimization for multimedia content delivery for 2 years. He is one of the main contributors of the Multiple-Source Streaming solution. Since 2017, he is pursuing his work at *Joada SAS* and his PhD at *LaBRI/Bordeaux Computer Science Laboratory* on delivering and securing social multimedia contents over the Internet of Things. Contact him at mlacaud@joada.net or mathias.lacaud@labri.fr.

Daniel Negru received his PhD from the *University of Saint Quentin en Yvelines*. In 2007, he became associate professor at *ENSEIRB-MATEMCA/University of Bordeaux*. His research interests include multimedia and networking. From 2010 to 2014, he coordinated the FP7 ALICANTE project tackling networking and multimedia research fields. He participated in more than 10 collaborative research projects, published more than 60 papers. Contact him at daniel.negru@labri.fr.

Laurent Réveillère is professor in computer science at *University of Bordeaux*. He received his PhD in 2001 from *University of Rennes*. He was formerly Associate Professor at *ENSEIRB-MATMECA*, an engineering school of Bordeaux INP. His main research interests fall in the domain of software engineering for distributed systems. He has co-authored 40+ peer-reviewed international publications. He serves on several program committees of major conferences in his field, and he is a member of the steering committee of major conferences such as ACM Middleware and ACM EuroSys. Contact him at laurent.reveillere@labri.fr.

How to cite this article: Da Silva S, Bruneau-Queyreix J, Lacaud M, Negru D, Réveillère L. Muslin: A QoE-Aware CDN resources provisioning and advertising system for cost-efficient multisource live streaming. *Int J Network Mgmt*. 2019;e2081. https://doi.org/10.1002/nem.2081