`MUSLIN`: Achieving High, Fairly Shared QoE Through Multi-Source Live Streaming

Simon Da Silva[1], Joachim Bruneau-Queyreix[23], Mathias Lacaud[12], Daniel Négru[4], Laurent Réveillère[1]
[1] Univ. Bordeaux, LaBRI, UMR 5800, F-33400 Talence, France, [2] Joada SAS, Bordeaux, France, [3] National Institute of Telecommunications, Warsaw, and [4] Univ. Bordeaux, Bordeaux INP, LaBRI, Talence

ABSTRACT

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. Currently, content delivery networks provide numerous servers hosting replicas of the video content, and consuming clients are re-directed to the closest server. Then, the video content is streamed using adaptive streaming solutions. However, some servers become overloaded, and clients may experience a poor or unfairly distributed quality of experience.

In this paper we propose `Muslin`, a streaming solution supporting a high, fairly shared end-users quality of experience for live streaming. `Muslin` leverages on MS-Stream, a content delivery solution in which a client can simultaneously use several servers. `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 one-day video-games event, with hundreds of clients and several test beds. Our results shows that our approach outperforms traditional content delivery schemes by increasing the fairness and quality of experience at the user side without requiring a greater underlying content delivery platform.

CCS CONCEPTS

- **Networks**;

KEYWORDS

live streaming, multi-source adaptive streaming, fairness, QoE

1 INTRODUCTION

End-users' Quality of Experience (QoE) is a crucial factor for the success of the increasing number of video streaming services. According to Cisco [3], video traffic will experience a tremendous growth and is expected to exceed 80% of the total Internet traffic by 2020. Most of the time, such traffic increase forecasts are not followed by the necessary upgrade of core networks capacity due to the important costs it incurs and major issues arise with respect to the Quality of Experience of such services. Therefore, the design of current and future content delivery solutions needs to consider such aspects.

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 re-directed 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 over-scaled incurring unnecessary expenditures, or under-sized and thus delivering poor QoE to end-users.

In addition to the CDN-based infrastructure, 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 the consuming client 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 streaming service, the nearest server may become rapidly overloaded. As a consequence, some users may suffer throughput degradation or content unavailability, and experience a poor or unfairly shared QoE as they compete for network and server resources.

We introduce `Muslin`, a streaming solution supporting a high, fairly shared end-users quality of experience for live streaming services over the Internet. `Muslin` leverages on MS-Stream, based on the DASH standard, in which a client can simultaneously use several servers with heterogeneous capacities to aggregate network throughput on multiple communication channels. `Muslin` periodically estimates the required throughput to adjust the service infrastructure scale. `Muslin` then assigns content servers to clients based on periodic feedbacks from `Muslin` clients during streaming sessions.

2 RELATED WORK AND BACKGROUND

Video streaming is a trending topic in research as consumers demand is continuously growing. Many video streaming architectures and techniques have been proposed [23] [24]. HAS

Packet Video'18, June 12–15, 2018, Amsterdam, Netherlands
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5773-9/18/06...$15.00
https://doi.org/10.1145/3210424.3210432

protocols have seen important interest from the industry and research, mainly due to their capabilities to render smooth video playback to the consumers, hence a better QoE.

HTTP Adaptive Streaming and DASH. The MPEG-DASH standard, widely adopted in the industry, aims at delivering uninterrupted multimedia content through the network via conventional HTTP traffic [29]. 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* (GoP). 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.

Multiple-Source Adaptive Streaming. The Multiple-Source Adaptive Streaming over HTTP (MS-Stream) [8] [11] [9] [10] 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. In MS-Stream, for a given video segment, each considered server delivers a video sub-segment to the client.

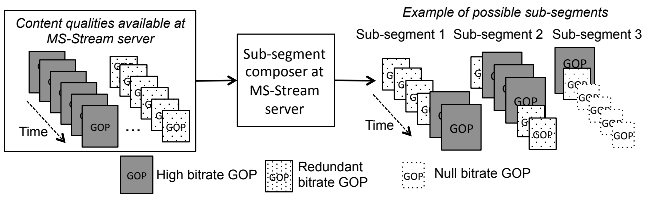
Content qualities available at MS-Stream server
Time
GOP
Sub-segment composer at MS-Stream server
Example of possible sub-segments
Sub-segment 1
Sub-segment 2
Sub-segment 3
High bitrate GOP
Redundant bitrate GOP
Null bitrate GOP

Figure 1: Sub-segment generation and composition

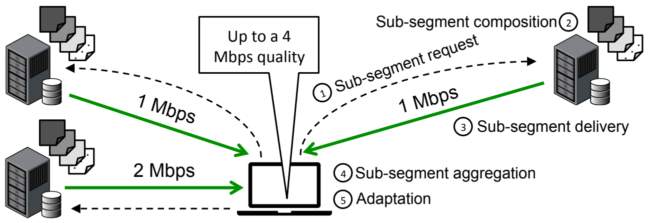

Figure 2: MS-Stream content delivery overview

As shown in Fig.1, sub-segments are generated by interleaving GoPs at different bitrates for the same segment: a high desired bitrate, a critically low bitrate (redundant bitrate), or an emtpy GoP. The redundant bitrate is set to low values (e.g. 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 sub-segments at client-side. Should some sub-segments be missing, the content is still playable by relying on the redundant GoPs, hence displaying a sub-optimal visual quality but providing reliability and less rebufferings in fluctuating network conditions.

An overview of the MS-Stream functioning is depicted in Fig. 2. A MPD file containing the available MS-Stream servers and video segments is periodically delivered to the client. The client instructs MS-Stream servers to generate and deliver sub-segments composed of video GoPs from the representations available (listed in the MPD file). Then, the MS-Stream client merges the received sub-segments to reconstruct a playable video segment with the highest possible visual quality. The client adapts the number of simultaneously used servers according to the observed network conditions and to the targeted bitrate. The client also attempts to minimize the bandwidth consumption overhead ($O\%$) resulting from GoP redundancy. This redundancy adds about 6.5% network overhead on average. It ought to be noted that the generation and aggregation of sub-segments have very low processing footprints [10] as they only require to assemble already encoded GoPs available at different bitrates. A demonstration of MS-Stream is available online [1].

The work of Adhikari et al. [20] 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 [17] [22], to the best of our knowledge, none of the existing other approaches provide both redundancy between independent sub-segments (to avoid rebufferings) and bandwidth aggregation (to reach a higher visual quality).

Content replication policies. The most widespread video caching and replication technique is based on greedy heuristic algorithms. Indeed, iteratively caching content with global system knowledge to try to reach an optimum has been shown to be an efficient way to distribute video content [4]. It can be done by maximizing a utility function [21] or minimizing a cost function [28] [13] for instance. Other policies consider social relationships between users and forecast the trending videos [7]. Our work is also based on a greedy iterative algorithm, however it differs from these propositions. First, the live content is only stored for a short time, as opposed to on-demand streaming where caching policies are often applied on a per-segment basis for each video content. In our case, the popularity of each content only corresponds to the current number of viewers. Besides, some works use network awareness [12] and QoS metrics to route requests or to select servers, but do not consider end-users QoE. Zheng et al. [31] base their approach on complex path latency optimization through multiple servers, but not bandwidth or system scale. Similarly, Puntheeranurak et al. [27] only take into account latency, delay and jitter inside the network. As opposed to these approaches, `Muslin` takes into account live clients feedbacks to provision servers.

Servers selection and QoE fairness. Although CDN operators keep their strategies secret [26], the usual paradigm is to estimate the audience for an event, and to provision enough servers near end-users to withstand the demand. Then, when clients request video content, the CDN strategy is to route their requests to the nearest server thanks to DNS [6] or IP anycast [5], and use HAS protocols for delivery. This behavior minimizes network-induced latency, and lowers the probability to encounter congestion. For instance, Adhikari et al. [20] 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. Therefore, 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. [15] use Software Defined Networks to allocate bandwidth to each link, and Petrangeli et al. [16] 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.

3 MUSLIN: HIGH, FAIRLY SHARED QOE IN MULTI-SOURCE LIVE STREAMING

As previously-mentioned, Muslin goal is to provide a high and fairly shared QoE for live streaming services. To do so, it tackles the main reasons why end-users are not satisfied with their streaming experience, which are the number of rebuffering events, considered the main negative impact on perceived QoE [18], the average video bitrate displayed on the user video player and the number of resolution changes during the session, as both have a significant influence on QoE in adaptive streaming [14]. Muslin intends to solve the root causes for such QoE degradation, the two main reasons being (1) the server load and (2) 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 a Muslin overlay to handle feedbacks and provisioning. Indeed, Muslin clients send periodic feedbacks to the Muslin server, including the observed bandwidth from each server, the video sub-segment 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, re-allocates servers, and re-advertises content servers to Muslin clients to provide a better QoE to end-users.

As illustrated in Fig.3, (**1**) the Muslin server dynamically provisions content servers and replicates content to available MS-Stream content delivery servers, which then register

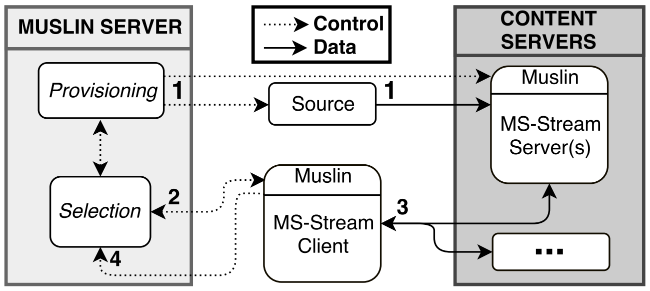

Figure 3: Muslin system architecture overview

themselves to the selection module; (**2**) when a client requests a 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.

3.1 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 seconds).

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.

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. Due to MS-Stream specification, the sub-segments redundancy adds a network bandwidth overhead percentage O (up to 10%). 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 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 (that is to say the number of late replies over the total number of requests).

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

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:

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

Provisioning decision. When the total throughput D is known, 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 for the provisioning decision. The RS_s is based on clients and servers proximity, and on feedbacks gathered periodically from all clients. For each server, the number of clients for which this would be the closest content server is computed as N_s. Also, the `Muslin` clients detect when servers fail to deliver a sub-segment in time. This measurement is aggregated into a failure rate FR_s. It represents the ratio of delivery failures detected over the total number of clients that requested a sub-segment 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 thus compute the average observed bandwidth estimate OBW_s for each server s. As shown in equation 4, the RS_s thus 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.

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

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, in a greedy heuristic-like fashion. Else, if the system is over-provisioned, the servers are deprovisioned according to their RS_s in ascending order.

3.2 Selection module

The `Muslin` selection module goal is to advertise a subset of available content servers to each client, based on a Ranking Score RS_{sc}, in order to reach a high and fairly shared QoE. Then, `Muslin` clients decide how many servers they use, based on MS-Stream adaptation strategies. As illustrated in Fig. 4, if the closest content server is 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.

First, the selection module returns an ordered list of servers when a client requests to discover available content servers. To order the list of servers, the selection module uses a client-specific Ranking Score (labeled RS_{sc}) for each 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 (20000 kilometers), the geographical distance GD_{sc} using geoIP data inferred from IP addresses, the video sub-segment delivery failure rate FR_s of server s (i.e. 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.

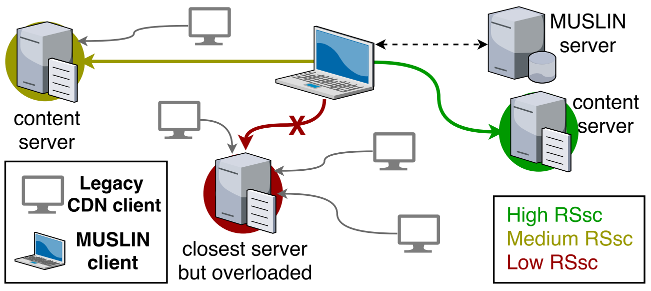

Figure 4: `Muslin` RS_{sc}-based servers selection example

$$RS_{sc} = ((20000 - GD_{sc}) * (1 - FR_s) * OBW_s)^{\frac{1}{3}} \tag{5}$$

The selection module computes the client-specific Ranking Score RS_{sc} between each client c and each currently provisioned server s, and returns the MPD file containing servers sorted by descending RS_{sc} order.

3.3 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` 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). We thus ensure statistically that at most N clients will send a feedback for every period T, depending on the current audience v_t.

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

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).

4 EXPERIMENTAL SETUP

In order to evaluate our approach, `Muslin` was deployed and compared with various strategies that are commonly used.

Servers provisioning, advertising, and content delivery strategies. 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. Therefore, we implement a *Geographical oracle* provisioning policy, which is aware of the exact amount of viewers and their locations. The system replicates content to the optimum number of servers near end-users locations. This is a scenario impossible to reach in real-life, but it provides a best-case current paradigm comparison.

We then 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.

We perform our experiments using the `Muslin` system as described in Section 3, the policies explained above, and the MS-Stream solution. We do not detail in this paper the evaluation of MS-Stream against traditional HAS solutions based on DASH since it has already been done [10].

Servers and clients setup. We set up 16 Points of Presence (PoP) geographically distributed in the US on a local network, by computing the latency and bandwidth between each client and server according to the geographical distance. We chose 16 locations as most CDN providers have between 10 and 30 PoP [2], and Google provides 16 locations [25]. 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 (e.g. California: 13%, Texas 10%, Florida 8%, etc.) and re-used the same toss in all the experiments. Figure 5 shows the location of PoP and clients pools.

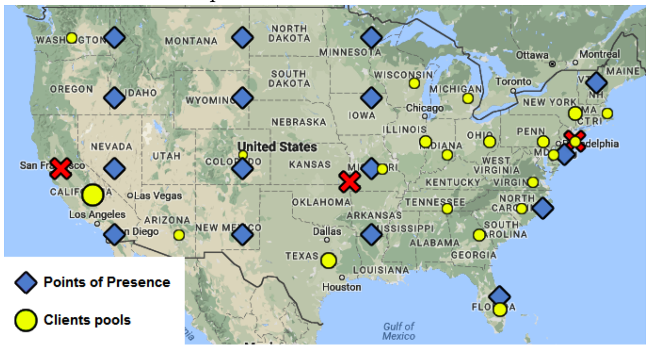

Figure 5: US map with points of presence and clients

Audience trace. The used live video content is the Blender Big Buck Bunny video encoded in five video bitrates: 205 kbps, 1 012 kbps, 2 029 kbps, 4 086 kbps and 6 391 kbps. The audience profile is a real trace from a week-long charitable videogames event streamed online. The audience used is from day 6, July 08 2016 [30], 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.

Experiments. Our experiments consist in a 30 minutes live streaming broadcast, re-run 5 times to aggregate results and reduce noise and outliers impact in the distributions. To remain realistic given the number of clients, we set servers bandwidth to 30 Mbps, and the provisioning policies can select up to 13 servers. Each Point of Presence can host multiple servers simultaneously.

5 EVALUATION RESULTS

This section presents our results. The fairness and QoE results are based on three main metrics: the number of rebuffering events, which is considered the main negative impact on perceived QoE [18], the average video bitrate displayed on the user video player and the number of resolution changes during the session, as both have a significant influence on QoE in adaptive streaming [14].

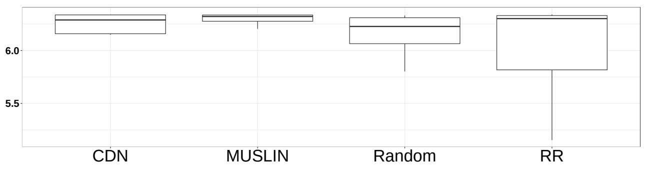

Figure 6: Displayed bitrate (Mbps)

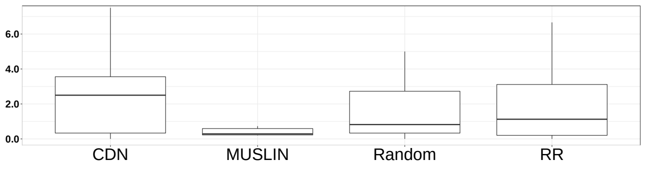

Figure 7: Quality changes per minute

`Muslin` clients were able to reach a higher QoE compared to 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 to 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.

Table 1: QoE fairness (F index)

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

Furthermore, Muslin median results are not only better than a best-case CDN implementation, but also the distributions are less spread than other setups, as the fairness among users is higher. We thus 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) described by T. Hoßfeld et al. [19]. The main reason for such increases is the feedback-based RS_{sc} computation, enabling to advertise the most suitable servers for each client, 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.

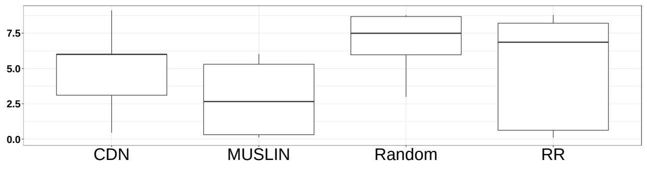

Figure 8: Network overhead (%)

Besides, 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 client detects that most servers are able to reply in time to video segments requests, and thus lowers the redundancy in sub-segments requests.

6 CONCLUSION

We presented Muslin, a resource-efficient multi-source live streaming system which manages to reach higher QoE and fairness than currently adopted streaming systems, while minimizing the required infrastructure scale. By taking into account clients real-time feedbacks, Muslin dynamically replicates content and improves server advertising to clients to enhance users' QoE and fairness. 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 to 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.

ACKNOWLEDGMENTS

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.

REFERENCES

[1] 2017. MS-Stream Demonstration: http://msstream.net. (2017).
[2] CDNPlanet. 2018. (2018). cdnplanet.com/geo/united-states-cdn
[3] Cisco. 2016. VNI. (2016). cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf
[4] A. Bestavros et al. 2001. Object Replication Strategies in Content Distribution Networks. *Web Caching and Content Delivery* (2001).
[5] A. Flavel et al. 2015. Fastroute: A scalable load-aware anycast routing architecture for modern cdns. *connections* 27 (2015).
[6] E. Nygren et al. 2010. The Akamai Network: A Platform for High-performance Internet Applications. *SIGOPS Oper. Syst. Rev.* (2010).
[7] H. Hu et al. 2016. Joint content replication and request routing for social video distribution over cloud CDN: A community clustering method. *IEEE Transactions on Circuits and Systems for Video Technology* (2016).
[8] J. Bruneau-Queyreix et al. 2017. 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.
[9] J. Bruneau-Queyreix et al. 2017. MS-Stream: A multiple-source adaptive streaming solution enhancing consumer's perceived quality. In *IEEE Consumer Communications and Networking Conference (CCNC).* Las vegas, United States.
[10] J. Bruneau-Queyreix et al. 2017. QoE Enhancement Through Cost-Effective Adaptation Decision Process for Multiple-Server Streaming over HTTP. In *IEEE International Conference on Multimedia and Expo (ICME).*
[11] J. Bruneau-Queyreix et al. 2018. Adding a new dimension to HTTP Adaptive Streaming through multiple-source capabilities. In *IEEE Multimedia Magazine.*
[12] J. M. Batalla et al. 2012. Optimization of the decision process in network and server-aware algorithms. In *International Telecommunications Network Strategy and Planning Symposium.*
[13] K. Lim et al. 2014. Joint optimization of cache server deployment and request routing with cooperative content replication. In *IEEE International Conference on Communications (ICC).*
[14] M. Seufert et al. 2015. A Survey on Quality of Experience of HTTP Adaptive Streaming. *IEEE Communications Surveys and Tutorials* (2015).
[15] P. Georgopoulos et al. 2013. Towards Network-wide QoE Fairness using OpenFlow-assisted Adaptive Video Streaming. *ACM SIGCOMM Workshop on Future Human-Centric Multimedia Networking* (2013).
[16] S. Petrangeli et al. 2015. QoE-Driven Rate Adaptation Heuristic for Fair Adaptive Video Streaming. *ACM Trans. Multimedia Comput. Commun. Appl.* (2015).
[17] S. Zhang et al. 2015. Presto: Towards fair and efficient HTTP adaptive streaming from multiple servers. *IEEE International Conference on Communications (ICC)* (2015).
[18] T. Hobfeld et al. 2011. Quantification of YouTube QoE via Crowdsourcing. In *IEEE International Symposium on Multimedia.*
[19] T. Hoßfeld et al. 2017. Definition of QoE Fairness in Shared Systems. *IEEE Communications Letters* (2017).
[20] V. K. Adhikari et al. 2012. Unreeling netflix: Understanding and improving multi-CDN delivery. *IEEE INFOCOM* (2012).
[21] W. Li et al. 2016. StreamCache: Popularity-based caching for adaptive streaming over information-centric networks. In *IEEE International Conference on Communications (ICC).*
[22] W. Pu et al. 2011. Dynamic Adaptive Streaming over HTTP from Multiple Content Distribution Servers. *IEEE Global Telecommunications Conference (GLOBECOM)* (2011).
[23] X. Zhang et al. 2005. CoolStreaming/DONet: A Data-driven Overlay Network for P2P Live Media Streaming. In *IEEE Infocom.*
[24] Z. Li et al. 2014. Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale. *IEEE Journal on Selected Areas in Communications* (2014).
[25] Google. 2018. (2018). cloud.google.com/cdn/docs/locations
[26] A. Passarella. 2012. A survey on content-centric technologies for the current Internet: CDN and P2P solutions. *Computer Communications* (2012).
[27] S. Puntheeranurak and N. Sa-ngarmangkang. 2015. An improvement of video streaming service using dynamic routing over OpenFlow networks. In *International Conference on Information Technology and Electrical Engineering (ICITEE).*
[28] J. Sahoo and R. Glitho. 2016. Greedy heuristic for replica server placement in Cloud based Content Delivery Networks. In *IEEE Symposium on Computers and Communication (ISCC).*
[29] I. Sodagar. 2011. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE MultiMedia* (2011).
[30] Twinge. 2018. (2018). twinge.tv/gamesdonequick/streams/#/22233544288
[31] H. Zheng and X. Tang. 2016. The Server Provisioning Problem for Continuous Distributed Interactive Applications. *IEEE Transactions on Parallel and Distributed Systems* (2016).