Cloud-assisted Buffer Management for HTTP-based Mobile Video Streaming

Stefania Colonnese[*]
DIET at University of Roma "Sapienza"
Rome, ITALY
colonnese@infocom.uniroma1.it

Francesca Cuomo
DIET at University of Roma "Sapienza"
Rome, ITALY
francesca.cuomo@uniroma1.it

Tommaso Melodia
Dept. of Electrical Engineering
State University of New York
Buffalo, NY, USA
tmelodia@eng.buffalo.edu

Raffaele Guida
University of Roma "Sapienza"
Rome, ITALY
raffaeleguida1984@gmail.com

ABSTRACT

This paper studies a cloud-assisted procedure to improve the user's Quality of Experience (QoE) in HTTP Adaptive Streaming (HAS) services. HAS delivers video streaming services following a client-server architecture and requires the client to originate repeated HTTP requests to download chunks of encoded video. In state-of-the-art systems, the client selects the actual chunk to be downloaded from a finite set of differently encoded video versions available at the server site, according to a client-based buffer management procedure.

In a multimedia cloud framework, HAS can leverage knowledge of the characteristics of the encoded video available at the server side. Therefore, we propose a cloud-assisted HAS procedure that exploits information on the encoded video content available at the cloud side to control the client-originated download requests.

The proposed approach balances client-related quality issues, which would require intensive video chunks download to avoid playout stalls, with cloud related system constraints, which require the average download rate not to overcome the average video encoding rate. Finally, this approach procedure alleviates the computational load at the client, since the downloading strategy is computed at the cloud side.

We demonstrate that significant QoE improvements are achievable through the proposed cloud-assisted buffer management procedure.

Categories and Subject Descriptors

C.2 [Computer-Communication Networks]: Network Protocols

[*]Corresponding author

Keywords

HTTP adaptive streaming, cloud computing

1. INTRODUCTION

Video streaming services have experienced a major boost in the last years and are expected to grow in both mobile and wired networks [1]. HTTP Adaptive Streaming (HAS) is a widely deployed client-server architecture for delivering streaming video services in current video networking systems. HAS has become popular in commercial deployments. Streaming platforms such as Apple's HTTP Live Streaming, Microsoft's Smooth Streaming, and Adobe's HTTP Dynamic Streaming all use HTTP streaming concept [2]. MPEG and 3GPP standardization activities have also provided a standard framework for HAS [3].

With respect to alternative video streaming protocol architectures, HTTP-based systems are fully compatible with network devices and middleboxes (e.g., firewalls, NATs) currently used in the Internet. Furthermore, the client can adaptively select among different versions of the same video available at the server side, based on rate adaptation strategies that take into account the current network conditions (e.g., available bandwidth). Thereby, the effective video streaming rate depends on both the encoding settings and on the actual end-to-end channel experienced between the server and the client terminal [4].

In HAS, the underlying reliable protocol (TCP) prevents packet losses and recovers channel errors at the price of introducing random end-to-end packet transmission delays. Hence, underflow events may occur at the client buffer, causing playout interruptions (stalls) during which the client enters a state of rebuffering until the buffer is loaded to a preset threshold prior to restarting the playout. Such rebuffering events clearly affect the user perceived quality. Indeed, TCP-based media streaming suffers significantly on wireless IP networks due to TCP throughput variations [5]. Experimental evaluations of different commercial HAS systems [2][6] also show that client-based rate adaptation strategies may result in fluctuations of the Quality of Experience (QoE) at the client side. Even when the end-to-end bandwidth is relatively stable, the encoded media may present rate variations leading to different transmission time for different video fragments. Therefore, even under constant-bandwidth conditions, the local fluctuations of the encoding rate can cause non-negligible QoE degradation.

Within this context, the objective of this work is to propose enhancements to state-of-the-art HTTP-based video streaming protocols specialized for Mobile Cloud Computing systems (MCC), where mobile devices experience large network fluctuations and processing tasks should migrate from the client to the cloud. The HAS architecture provides an effective starting point to develop video streaming techniques that can match these specific challenges of MCC.

In this paper, we propose a cloud-assisted HTTP streaming strategy, where the cloud server providing the content optimally controls the intervals between consecutive video segment requests performed by the client. The proposed strategy has several advantages:

- it reduces the computational load at the client, since the rate adaptation strategy is computed at the cloud side;
- it exploits information on the encoded video content available at the cloud side and not at the client;
- it balances client-related quality issues, which would require intensive download to avoid playout stalls, with cloud related system constraints, which require the average download rate not to overcome the average video encoding rate.

2. HAS SYSTEM MODEL

In HAS systems the service is provided to the client via a Multimedia Presentation Description (MPD). The contents are then fetched by the client in the form of fragments by means of HTTP GET requests. Specifically, in HTTP video streaming systems like DASH [3], a video content is encoded into multiple versions, each with a different video quality. Each encoded video is then divided into small video "chunks", which may encompass one or more GOPs (Group of Pictures). Video chunks can be addressed by a URL and are served to the client using HTTP servers. For example, the server may store L versions of the same encoded video at different rates $R_i, i = 1, \dots L$. Each sequence is encoded and parsed in fragments beginning with a random access frame and corresponding to a playback interval of τ_f s.

Let us consider the transmission of a video bitstream encoded at a given average rate R_1. The encoded fragment size in bits varies in a random way depending on the sequence activity and on the encoding parameters. We denote by x_n the size in bit of the n-th sequence fragment. The streaming session takes place when a sequence of HTTP-GET messages are sent by the client to begin the download of consecutive sequence fragments. After an HTTP-GET, the fragment is downloaded in a time $\tau_n = x_n/r_n$, being r_n the net throughput available at the application layer.

In this framework, we propose cloud-assisted HTTP streaming, where the servers providing the contents in the cloud have the possibility to drive the rate of the HTTP GETs of the clients in order to improve their QoE. A qualitative representation of the considered system architecture is in Fig.1.

A Video Agent (VA) is activated at the cloud side for the HTTP session of the client in order to provide an better adaptive streaming service across the wired/wireless net-

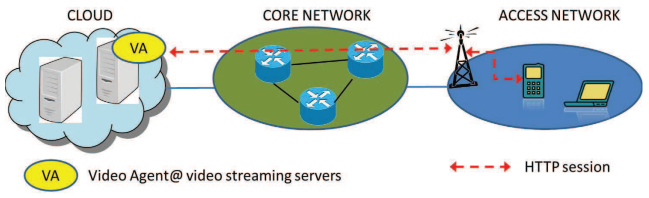

Figure 1: Architectural model.

works interconnecting the mobile client. The VA is able to virtually monitor the QoE of the client by measuring the amount of streamed bytes and the amount of bytes that are needed by the client to perceive a given quality. A key point is that the VA, being at the server side, has:

1. a complete knowledge of the amount of bytes that constitute a group of segments of the video at a given quality;
2. the possibility to run optimization algorithms on behalf of the mobile client, thus reducing the amount of processing resources spent at the client side in accordance with a cloud computing paradigm.

In the direction of achieving an optimization of the video streaming, the point 1) assures that, once the client select one of the available R_i, the server knows the whole set x_n, for $n = 1, \dots k$, of bytes that constitute the video sequence at that rate. Thereby, at the VA side, a cross-layer interactions between application and TCP layer may allow the VA to estimate the time required to transmit the n-th set of bytes x_n. Specifically, this can be computed by using the estimated RTT used by the TCP (that is a parameter known at the served side) and the x_n.

The point 2) instead includes two advantages, on the one hand there is the possibility to shift processing complexity from the client side to the server side (this may have benefits in mobile devices) and on the other hand the server may run optimization rules involving multiple video streams that are simultaneously downloaded by a set of clients. This latter aspect can allow also server resource optimizations.

In the existing literature there have been several papers attempting to improve the video streaming quality by introducing network devices (e.g., proxies) able to interact with the users on one side and with the servers on the other side to assure prompt reactions to bandwidth variations [7]. In our scenario, a Video Agent is activated for the HTTP session and has the advantage of being at the server side and it is aware of the content selected by the client via access to the MDP. Management of the video quality at the server side has been considered in the InSite approach [8] where video delivery from cloud data centers is controlled to assure the QoE of multiple clients of a given video service provider being aware of the bandwidth bottleneck that may arise in the network. The InSite control relies on the TCP advertised receiver window, and controls the rates provided to each flow so as to reduce the number of playout stalls and to distribute the stalls across clients fairly. Through experiments with different TCP variants, the authors show the achieved balance between QoE maintenance and bandwidth wastage.

The work in [9] characterizes the bandwidth consumption of a widely deployed DASH-based system, Netflix. The results show that different clients have a similar basic response to network congestion, when they behave differently in case of sustained congestion. The study suggests that the Netflix adaptation algorithm and the TCP congestion control are intertwined during period of volatile network conditions, but the latter is dominant in case of of heavy congestion.

Similarly, we propose a short-term rate adaptation scheme based on a look-ahead mechanism operated at the VA side. We expect that this scheme will be able to reduce playout buffer stalls and oscillations that characterize the system in Fig. 1.

3. CLOUD-ASSISTED HAS

We consider a server storing L versions of the same encoded video at different rates $R_i, i = 1, \dots L$. Each sequence is encoded using a fixed Group of Pictures (GOP) structure beginning with a random access frame. One or few consecutive GOPs constitutes a video fragment to be downloaded during one HTTP-GET. Each fragment corresponds to a playback interval of τ_f s. We consider an HTTP session where the number of downloaded fragments is N.

Let us consider the transmission of a video bitstream encoded at a given average rate $\overline{R}$. Let us denote by x_n the size in bits of the n-th fragment. x_n randomly varies depending on the sequence activity and of the encoding parameters. When the client requests the fragment using an HTTP-GET, the fragment is downloaded in a time

$$\tau_n = \frac{x_n}{r_n}$$

where r_n is the net bandwidth available at the application layer, i.e., the average TCP throughput measured during the fragment download time. In HAS, the client does not request the $n+1$-th fragment unless the download of the n-th one is completed. Besides, in typical HTTP adaptive streaming systems, the client waits for at least τ_f s before requesting the next fragment, except when in buffering or rebuffering state. Fig.2 presents the timing of the server transmissions after HTTP GETs sent by the client.

However, given the information available in the cloud on the video fragments sizes and given an estimate of the net bandwidth, for the inter-arrival time can be set in accordance to an optimal strategy. Herein, we discuss how the interval between consecutive HTTP-GETs, denoted as δ in the following, can be optimized at the cloud side. The optimal scheduling is then communicated to the client, which then generates the actual HTTP-GET requests. We refer to this strategy as cloud-assisted streaming.

With these premises, we look for an optimal HTTP-get scheduling. Specifically, we aim at identifying a set of N intervals δ_n such that the intervals between consecutive HTTP-GETs,

$$t_n^{GET} - t_{n-1}^{GET} = \max(\tau_{n-1}, \delta_n), n = 0, \dots N-1$$

optimizes a suitable QoE metric.

The selected QoE metric considers the number of stalls and the rebuffering events duration. With this aim, let us now evaluate the buffer load at the receiver size.

We denote by t_n^W the time instant at which the n-th fragment is fully downloaded in the receiver buffer, also known as the playout buffer (see the lower axis of Figs.2). Besides,

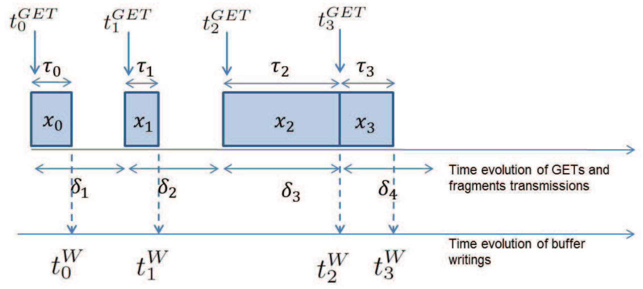

Figure 2: Time evolution of the HTTP GETs sent by the client, the transmission of requested fragments and their writing in the playout buffer.

we denote by t_n^R the time instant at which the n-th fragment is read for playback. The initial time for fragment reception is assumed to be $t_0^W = 0$; the playback begins with an initial buffering delay of δ_B s, that is $t_0^R = \delta_B$.

Once the playback starts, fragments are read at a regular rate; nonetheless, due to the delay and jitter in fragment reception, caused by both bandwidth and encoding rate local fluctuations, the buffer may underflow. In this case, playback is stalled for a rebuffering interval δ_R, during which fragments are loaded but not played out. After the rebuffering period, the playout restarts at a rate $1/\tau_f$.

Hence, accurate modeling of the buffer behavior needs to take into account the occurrences of stalls. We then introduce a binary sequence s_n, defined as follows: $s_n = 1$ when a stall occurs corresponding to the n-th fragment and $s_n = 0$ otherwise. Then, the buffer evolution is characterized by the sequences t_n^W, s_n, t_n^R defined as

$$\begin{cases} t_0^{GET} = 0 \\ t_0^W = t_0^{GET} + \tau_0 \\ s_0 = 0 \\ t_0^R = t_0^W + \delta_B \end{cases} \quad (1)$$

and

$$\begin{cases} t_n^{GET} = t_{n-1}^{GET} + \max(\tau_{n-1}, \delta_n) \\ t_n^W = t_n^{GET} + \tau_n \\ s_n = 1/2 - 1/2 \operatorname{sign}(t_{n-1}^R + \tau_f - t_n^W) \\ t_n^R = t_{n-1}^R + (1 - s_n)\tau_f + s_n\, \delta_R \end{cases} \quad (2)$$

for any $n \neq 0$.

We can then compute the number of stalls as

$$S = \sum_0^{N-1} s_n$$

If we assume that each rebuffering period is deterministcally set to δ_R[1], the overall re-buffering time equals to

$$D = \delta_R \times \sum_0^{N-1} s_n$$

[1]As an alternative, it could be assumed that each rebuffering period may vary so as to correspond to a fixed amount of downloaded data as well as to a fixed video playout interval, leading to a random characterization of the oveall rebuffering time D.

We can now express the buffer occupancy based on the above expressions of the writing and reading instants. Specifically, let us introduce the writing process $w(t)$ as follows:

$$w(t) = \sum_n x_n u_{-1}(t - t_n^W)$$

and the reading process as

$$r(t) = \sum_n x_n u_{-1}(t - t_n^R)$$

We can then express the buffer occupancy in bits as

$$\begin{aligned} B(t) &= w(t) - r(t) \\ &= \sum_n x_n \mathrm{rect}_{t_n^R - t_n^W}\left(t - \frac{t_n^R + t_n^W}{2}\right) \end{aligned}$$

and the buffer occupancy in seconds of video as

$$Q(t) = \sum_n \mathrm{rect}_{t_n^R - t_n^W}\left(t - \frac{t_n^R + t_n^W}{2}\right) \times \tau_f$$

A suitable QoE metric for HTTP-GET scheduling is the minimization of the number of stalls, which has direct implications in terms of the duration of rebuffering events. On the other hand, when operating on a set of consecutive fragments, there is a finite probability that the client can unpredictably end the streaming session. Therefore, we consider the following QoE-related objective,

$$\mathcal{C}\left(\delta_0, \delta_{N-1}\right) = \min_{\delta_n, n=0,\dots N-1} \sum_{0}^{N-1} \rho_{SW}^n s_n \qquad (3)$$

where $\rho_{SW} \in (0, 1]$ is a discount factor that takes into account the fact that the user can tear down the streaming session.

To avoid overflows at the receiver buffer and to limit the net throughput required by a single client to the cloud, we also require that downloading rate averaged on any window of W fragments, does not deviate significantly from the average video encoding rate $\overline{R}$. Therefore, we consider the following additional rate constraint,

$$\frac{\sum_{k=0}^{W-1} x_{n+k}}{t_{n+W-1}^W - t_n^W} \le \overline{R}\,(1 + \alpha) \qquad (4)$$

where α is a system design parameter.

3.1 Proposed approach for cloud-assisted HAS

In its general formulation, our optimal strategy accounts for all the stalls occurring over N fragments, requiring a rate-matching inequality over a sliding window of length W. This requires knowledge of the N values $\tau_n, n = 0, \dots N-1$, which in turn implies knowledge of x_n and $r_n, n = 0, \dots N-1$. While the sequence $x_n, n = 0, \dots N-1$ is known at the server, the sequence of the TCP throughputs $r_n, n = 0, \dots N-1$ should be predicted. The prediction accuracy bounds the values of N of practical interest. Therefore, we restrict our analysis to the case $N = W$ and we design a suboptimal procedure leading us to a solution $\delta_0 \le \delta_1 \le \dots \le \delta_{N-1}$ matching the rate inequality (4) with $\alpha = 0$.

The algorithm is as follows: if $\sum_{k=0}^{W-1} \max(\tau_k, \tau_f) \le W\tau_f$, δ_n is kept equal to τ_f; otherwise, δ_n is set equal to 0. Besides, if $\sum_{k=0}^{W-1} \max(\tau_k, 0) \ge W\tau_f$, i.e. consecutive downloading the W fragments has exceeded the expected window W, the client immediately restarts the downloading on the following window. Otherwise, it waits for $W\tau_f - \sum_{k=0}^{W-1} \tau_k$ and then resumes downloading on the next window.

4. SIMULATION RESULTS

Table 1: Video traces and statistics

Sequence	Avg. frame size [kbit]	frame size std dev.	Avg. bitrate $\overline{R}$[Mbps]
Alice	136	19478	3.26
Monster	114	7222	1.57
SpaceStation	199	9717	4.79
Titans	158	3857	3.79

The numerical simulations of the HAS session have been carried out using Matlab, in order to analyze different performance parameters, such as the buffer occupancy vs. time, and the relevant QoE parameters, including the number of underflow events (which is also the number of stalls), and their durations. More into detail, the server is equipped with a set of video sequence traces, namely the video traces in [10] analyzed in [11] and [12]. Specifically, we have considered four videos, at a spatial resolution of 1920x1080 pixels, 24 Hx, using an encoding GoP pattern of G16B1 (16 frames per GoP, 1 B frame in between I/P key pictures) of duration 35.544 min each. Each video presents intrinsic frame size fluctuations, detailed in Tab.1, resulting in x_n varying around the average size. The client requests the fragments corresponding to τ_f video seconds in accordance to the selected (optimized or not) HTTP-GET sheduling strategy and it begins the play out phase after an initial delay δ_B. The net bandwidth r_n at the application layer, always larger than the net video rate, is kept constant throughout the session simulation. When a stall occurs, the client continues downloading fragments and restarts the play out after a rebuffering duration δ_R. In the following, we analyse the HAS session performance by varying different system parameters such as the bandwidth r_n, the rebuffering duration δ_R, the sliding window length W.

Firstly, we present the results of a set of simulations where we have assumed a constant bandwidth equal to 150% of the video average bit-rates, i.e. $r_n = 1.5\overline{R}$. Remarkably, even in case of constant bandwidth, the intrinsic fluctuations of the encoded video can lead the system to very low performance as shown in Fig.3. In this case each playback can interrupt from $S = 2$ up to $S = 10$ times, which gives a total duration of the interruptions that goes from about $D = 10\ s$ to more than $D = 70\ s$ (Fig.3).

Although the number of stalls S may differently affect the client perceived QoE, depending on several characteristics, such as the rebuffering time duration, the video clip content and length, or even the subjective end-user expectation, the parameter S indeed accounts for the main cause of decoded video quality degradation in a HAS session where no video data losses are experienced.

Increasing the available bandwidth reduces the number of stalls during the playback, but it cannot completely avoid them. Fig.4 considers the case when the ratio between the average video rate and the available bandwidth decreases.

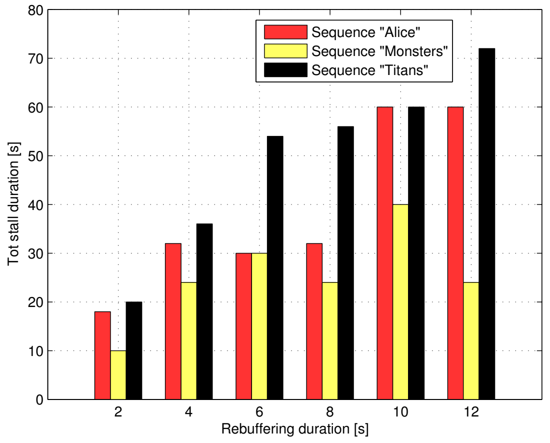

Figure 3: Total stall duration for three videos at different rebuffering duration.

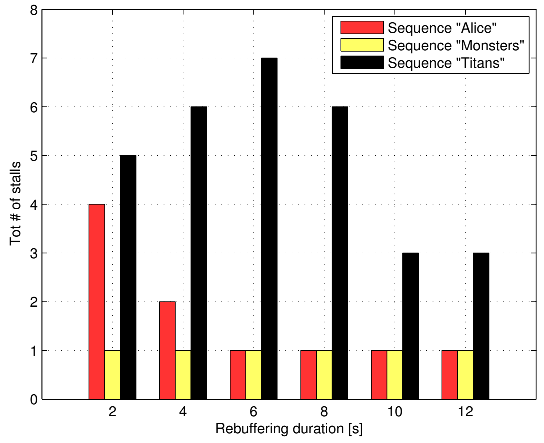

Figure 5: Number of stalls varying the re-buffering duration ($W = 5$).

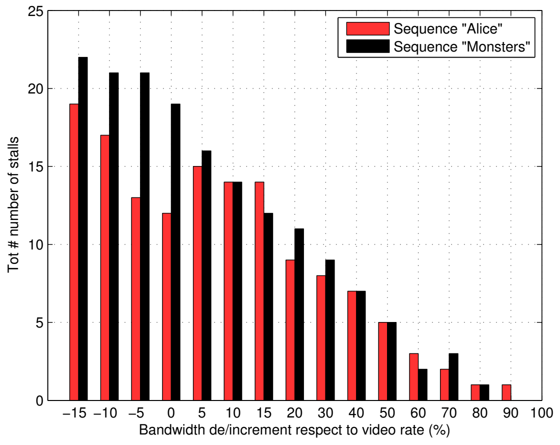

Figure 4: Relationship between stalls and video rate/bandwidth ratio reduction.

Remarkably, even when the bandwidth is twice the average video encoding rate, uninterrupted video playback cannot be assured.

Based on these results, we assume as an operating point for our simulations $r_n = 1.48\overline{R}$, since this assures a reduced number of stalls. The resulting bandwidth occupancy is tabulated in Tab.2.

Table 2: Video rates and assigned bandwidths

Sequence	Avg. bitrate $\overline{R}$ [Mbps]	Bandwidth [Mbps]
Alice	3.26	6.85
Monster	1.57	3.31
SpaceStation	4.79	9.79
Titans	3.79	7.97

Fig.5 shows the number of stalls S vs the rebuffering duration δ_R. An example of the stall duration D observed on the sequence Alice at $W = 6$ is reported in Tab.3.

From this Section analysis it stems out that the user QoE, coarsely characterized by the number of stalls S and by the rebuffering events' duration D depends on the pattern of download time τ_n, in turn depending on the GOP sizes and the available bandwidth. If the videos are HD encoded and high bit-rates are involved, bandwidth of several Mbit/s are required to avoid QoE flaws. Therefore, the performance in terms of QoE depends not only on the ratio between the available bandwidth and the encoding/playback rate, but also on the fluctuations of the video that has to be transferred.

We now show how adoption of a cloud assisted HAS strategy can improve the afore-described QoE metrics. Specifically, we discuss the QoE improvements achieved by adopting the algorithm in Section 3.1.

In Fig.6, we compare the number of stalls observed using the cloud-assisted procedure in Section 3.1 on the sequence Alice using $W = 6$, corresponding to $T_W = W\tau_f = 12s$, and $\delta_R = 6s$, under different encoding rate vs bandwidth ratios. We appreciate the significant reduction of the number of stalls achieved by the cloud assisted procedure.

Now we evaluate how many times the cloud-assisted procedure in Section 3.1 effectively activates an optimal scheduling. Specifically, in Fig.7 we show the percentage of windows of duration T_W over which the inter HTTP-GET period is effectively modified with respect to the total number of encoded sequence windows. Clearly, the activation is required more often when the bandwidth to encoding rate ratio is lower, and more rarely for high ratio values. Despite being rarely used, the cloud assisted adjustment of the inter HTTP-GET period, it completely avoids stalls (see Fig.6).

In Fig.8 we show the total number of stalls S vs the window size W for different values of the encoding rate to bandwidth ratio. As expected, increasing the window size improves the QoE performance. On the other hand, Fig.9 clearly shows that the advantages of using the window of

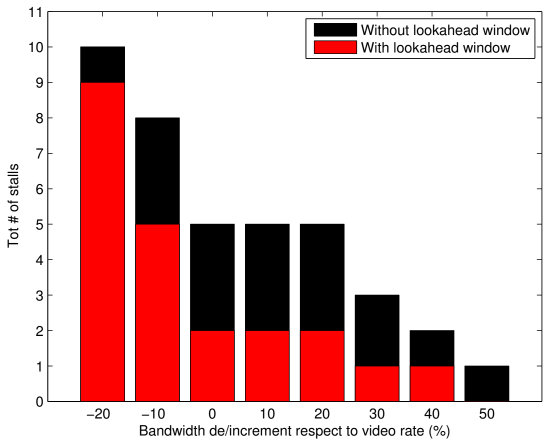

Figure 6: Number of stalls S with and without the cloud assisted procedure, for different encoding rate vs bandwidth ratios.

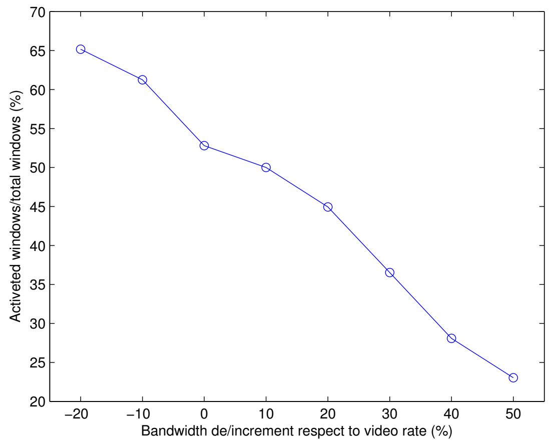

Figure 7: Percentage of the windows of duration T_W over which the inter HTTP-GET period is actually modified vs the bandwidth to encoding rate ratio.

size W stand for different values of the rebuffering duration δ_R.

Finally, Fig.10 shows the percentage of windows of duration T_W over which the inter HTTP-GET period is actually modified vs the window size W. We observe that increasing the window size W facilitates activation of the cloud assisted procedure nonetheless it requires accurate bandwidth prediction over a longer period. This definitely limits the feasibility of large window sizes.

Table 3: Rebuffering and stalls.

Rebuffering δ_R (s)	# of stalls S	Stalls' duration D (s)
2	4	8
4	3	12
6	3	18
8	3	24
10	3	30
12	2	24

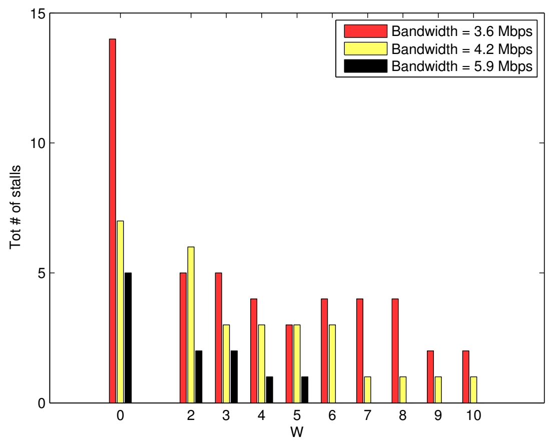

Figure 8: Total number of stalls S vs window size W.

5. CONCLUDING REMARKS

In this paper, we have presented a cloud-assisted procedure that is able to improve the Quality of Experience (QoE) in HTTP Adaptive Streaming (HAS) services. The cloud assisted HAS procedure exploits information on the encoded video content available at the cloud side to drive the client originated encoded video fragments download requests. The proposed approach leverages knowledge of characteristics of the encoded video available at the server side. Furthermore, the procedure balances client-related quality issues, which would require intensive video chunk downloads to avoid playout stalls, with cloud related system constraints, which require the average download rate not to overcome the average video encoding rate. Numerical simulation results show the significant QoE improvement achievable by performing HAS in accordance with the proposed cloud-assisted buffer management procedure.

The proposed approach can be generalized to take into account joint management of multiple users requiring HAS at the same cloud server or servers' cluster. The download request rate of different users as well as the user's selected video quality level can be optimally selected at the cloud side to assure suitable QoE constraints at each user's side. This generalization of the herein presented approach is currently under investigation.

6. REFERENCES

[1] Cisco, "Cisco Visual Networking Index: Forecast and Methodology, 2012-17." [Online]. Available: http://www.cisco.com

[2] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP," in Proceedings of the second annual ACM conference on Multimedia systems, ser. MMSys '11, 2011, pp. 157–168.

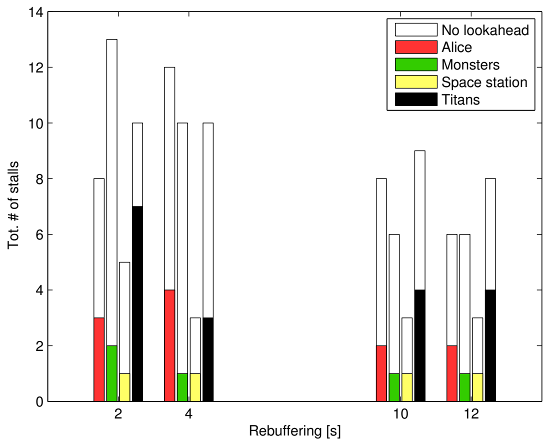

Figure 9: Total number of stalls S vs rebuffering duration δ_R.

[3] I. Sodagar, "The MPEG-DASH Standard for Multimedia Streaming Over the Internet," MultiMedia, IEEE, vol. 18, no. 4, pp. 62–67, 2011.

[4] S. Colonnese, P. Frossard, S. Rinauro, L. Rossi, and G. Scarano, "Joint source and sending rate modeling in adaptive video streaming," Signal Processing: Image Communication, vol. 28, no. 5, pp. 403 – 416, 2013.

[5] M. Gorius, Y. Shuai, and T. Herfet, "Dynamic media streaming over wireless and mobile ip networks," in Consumer Electronics - Berlin (ICCE-Berlin), 2012 IEEE International Conference on, 2012, pp. 158–162.

[6] L. De Cicco and S. Mascolo, "An experimental investigation of the akamai adaptive video streaming," in Proceedings of the 6th international conference on HCI in work and learning, life and leisure: workgroup human-computer interaction and usability engineering, ser. USAB'10, 2010, pp. 447–464.

[7] J.-S. Leu and C.-W. Tsai, "Practical design of a proxy agent to facilitate adaptive video streaming service across wired/wireless networks," Journal of Systems and Software, vol. 82, no. 11, pp. 1916 – 1925, 2009.

[8] V. Gabale, P. Dutta, R. Kokku, and S. Kalyanaraman, "Insite: Qoe-aware video delivery from cloud data centers," in Quality of Service (IWQoS), 2012 IEEE 20th International Workshop on, 2012, pp. 1–9.

[9] J. Martin, Y. Fu, N. Wourms, and T. Shaw, "Characterizing Netflix bandwidth consumption," in Consumer Communications and Networking Conference (CCNC), 2013 IEEE, 2013, pp. 230–235.

[10] "Video traces." [Online]. Available: http://trace.eas.asu.edu/videotraces2/3d/MultiviewJMVC/

[11] A. Pulipaka, P. Seeling, M. Reisslein, and L. Karam, "Traffic and statistical multiplexing characterization of 3-d video representation formats," Broadcasting, IEEE Transactions on, vol. 59, no. 2, pp. 382–389, 2013.

[12] P. Seeling and M. Reisslein, "Video transport evaluation with h.264 video traces," Communications Surveys Tutorials, IEEE, vol. 14, no. 4, pp. 1142–1165, 2012.

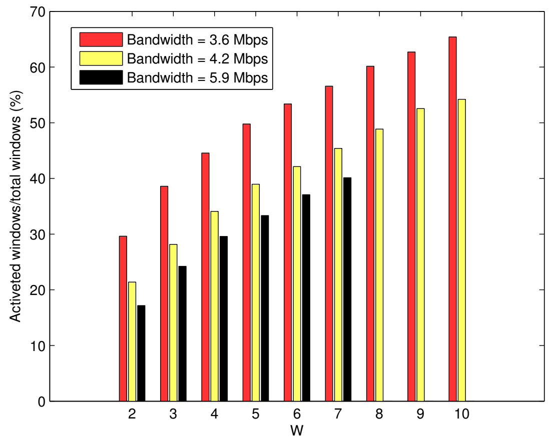

Figure 10: Percentage of the windows of duration T_W over which the inter HTTP-GET period is actually modified vs the window size W.