Dynamic Adaptive Streaming over HTTP (DASH) using feedback linearization: a comparison with a leading Italian TV operator

Vito Caldaralo, Luca De Cicco, Saverio Mascolo, and Vittorio Palmisano

Abstract—Dynamic Adaptive Streaming Over HTTP (DASH) is a video streaming standardization effort that aims at building video content delivery systems that dynamically adapt video bitrates to match the time-varying bandwidth of a conventional HTTP connection over the Internet Protocol. In particular, the video content is segmented into a sequence of chunks, each one containing a short interval of playback time, for instance 6 seconds. Chunks are encoded at different bit rates and a DASH client, at the receiver side, automatically selects the next chunk to download based on current Internet available bandwidth. The control goal is to provide the best possible video quality while avoiding playback video interruptions, i.e. rebuffering events, in the presence of an unpredictable bandwidth. In this paper we design a new DASH controller using feedback linearization. A real implementation of the controller is tested and compared with the automatic video switching control of a leading Italian TV operator.

I. INTRODUCTION AND RELATED WORK

Recent developments in Future Internet platform development are opening several possibilities for new multimedia applications [4]. The video part of Internet traffic is booming due to the ever increasing availability of video content from site such as YouTube (video sharing), Netflix (movie on-demand), Livestream (live streaming) and the pervasive diffusion of Tablets, Smart-Phones and SmartTVs which access the Internet through broadband wired and wireless links. The Cisco Visual Networking Index predicts that video will be 69 percent of all consumer Internet traffic in 2017 excluding peer-to-peer (P2P) video file sharing, whereas the sum of all forms of video (TV, video on demand and P2P) will be in the range of 80 to 90 percent of global consumer traffic by 2017 [5].

The key technological choice that ignited the start of video distribution over the Internet at large scale was the use of the HTTP protocol over the TCP. This choice was done by YouTube in 2005. The first approach to video streaming was the progressive download streaming where a video was encoded at a given quality and sent to the user as any other file using a HTTP connection. This approach has the following main problem: the video is encoded at a given bitrate, which is not elastic, but it is transported through the TCP that, on the contrary, is designed for elastic traffic over best-effort Internet. As a consequence, a persistent mismatch between the video bitrate and the best-effort Internet available bandwidth may result in an empty video playout buffer with video interruptions.

For this reason it is necessary to make the video content adaptive at least to some extent. The leading proposed approach consists of encoding a video at different bitrates and resolutions, the video levels, and video levels are divided into segments or chunks of fixed durations. A stream-switching controller at the client selects the next chunk to be downloaded at the best possible quality given the available bandwidth with the constraint of avoiding video interruptions. Standard HTTP servers can be used for video distribution [13] and scalability can be easily obtained using CDNs. Typically, adaptive players work as follows: at the beginning of the connection the player requests the video segments, of fixed duration $\tau$, through consecutive HTTP GET requests in order to build the buffer; then, when a certain amount of video is stored in the playout buffer, the buffering-phase is completed and the player enters in the steady-state phase; while in this state, the player strives to maintain the playout buffer level constant by issuing the HTTP requests each $\tau$ seconds. Thus, the player generates a on-off traffic pattern during the steady-state: the video segments are downloaded during the ON phase and then, during the OFF phase, the player remains idle until the next download is started [10], [1], [3].

It has been shown that the client-side algorithms proposed so far generate an on-off traffic pattern at steady-state that can lead to unfairness when many video flows share the same bottleneck [1], [2], [10]. Moreover, [8] shows that three popular video on demand streaming services in the US, Netflix, Vudu and Hulu, are not able to obtain their fair share of bandwidth in the presence of coexisting TCP greedy flows. This phenomenon was called “downward spiral effect” and was explained as an effect of the on-off traffic pattern generated at the sender side.

To overcome the before mentioned issues, several adaptive streaming algorithms have been proposed. FESTIVE has been proposed to provide fairness in a multi-client scenario [9]. PANDA [11] computes the chunk inter-request time to provide fairness and cut video bitrate oscillations. [2] proposes to introduce a traffic shaper at the server to eliminate the OFF phases when the player is in steady-state.

In [6] it has been shown that the automatic stream-switching system of a major CDN operator employs a different approach wrt the classic client-side architecture. In particular, it employs a hybrid sender-side/client-side architecture with two controllers running at the client: one for
selecting the video level, the other for throttling the sending rate at the server in order to control the playout buffer length at the client. Moreover, the system does not issue a lot of HTTP GET requests to download the segments, but sends HTTP POST requests to the server to select the video level to be streamed.

In this paper we propose a novel client side controller, named ELASTIC (Feedback Linearization Adaptive Streaming Controller), that has been designed using feedback linearization. The proposed control algorithm is able to avoid the on-off traffic pattern at the sender and to get its fair share of the bottleneck bandwidth when coexisting with TCP greedy flows. Finally we develop an experimental comparison of ELASTIC with a leading Italian TV operator. In particular, to compare the considered algorithms, we have set up a controlled testbed that allows bandwidth and delays to be set.

The paper is organized as follows: in Section II ELASTIC is presented; Section III describes the employed testbed; Section IV presents the results of the experimental evaluation and Section V concludes the paper.

II. ELASTIC

In this section we propose ELASTIC, a client-side adaptive streaming algorithm designed using feedback control theory. In Section II-A we present the design requirements; Section II-B describes the control system model; in Section II-C the control algorithm is presented, and Section II-D provides the controller implementation details.

A. Design requirements

The main goal of a stream-switching controller is to dynamically select the video level \( l(t) \in \mathcal{L} = \{ l_0, \ldots, l_{N-1} \} \) for each video segment to achieve the maximum Quality of Experience (QoE) while avoiding video interruptions that happen when the receiver playout buffer gets empty. Re-buffering events, occurring when the player buffer gets empty, have been identified to be one of the major causes impairing user engagement [7]. Moreover, it has been shown that frequent quality switches may be annoying to the user [12], thus limiting video level switches is considered a design requirement by several proposed algorithms [9], [11].

Summarizing, we consider the following design goals for ELASTIC: 1) minimize the re-buffering ratio; 2) maximize the obtained video level; 3) provide fair sharing of the bottleneck when coexisting with other video or long-lived TCP flows.

B. The control system model

Figure 1 shows the block diagram of a DASH streaming system with the controller at the client-side. The HTTP server sends the video to the client through an Internet connection with an end-to-end bandwidth \( b(t) \) and a round-trip-time (RTT) equal to \( T \). The client receives the video segments at a rate \( r(t) < b(t) \), and temporarily stores them in a playout buffer that feeds the video player. The controller dynamically decides, for each video segment, the video level \( l(t) \) to be downloaded sending a HTTP GET request to the HTTP server. The measurement module feeds the controller with measurements such as the estimated bandwidth \( \hat{b}(t) \) and the playout buffer level \( q(t) \).

C. The adaptive streaming controller

The typical approach to implement a stream-switching system is to design two controllers [3], [6], [8], [11]: one throttles the video level \( l(t) \) to match the measured available bandwidth \( b(t) \), the other regulates the playout buffer length \( q(t) \) by controlling the idle period between two segment downloads.

Differently from the currently used approaches, ELASTIC uses a unique controller that selects the video level \( l(t) \) to drive \( q(t) \) to a set-point \( q_T \). This eliminates the idle periods between segment downloads. Indeed, by reaching the controller goal, i.e. \( q(t) \to q_T \), the video level \( l(t) \) also matches the available bandwidth \( b(t) \), i.e. the maximum possible video level is obtained.

The playout buffer length \( q(t) \), i.e. the seconds of video stored in the playout buffer, can be modelled as an integrator:

\[
\dot{q}(t) = f(t) - d(t),
\]

where \( f(t) \) is the filling rate and \( d(t) \) is the draining rate.

Let us focus on the filling rate, which is equal by definition to \( dt_v/dt \), where \( dt_v \) is the amount of video duration received by the client in a time \( dt \). The video encoding bitrate is defined as \( l(t) = dD/dt_v \), where \( dD \) is the amount of bytes required to store a portion of video of duration \( dt_v \). It is important to notice that \( l(t) \) is always strictly greater than zero by definition. The received rate \( r(t) \) is defined as \( r(t) = dD/dt \), i.e. the amount of bytes received in a time interval \( dt \). Thus, since \( f(t) = dt_v/dt = (dt_t/dD) \cdot (dD/dt) \), it turns out that:

\[
f(t) = \frac{r(t)}{l(t)}
\]

We now derive the model of the draining rate \( d(t) \). The playout buffer is drained by the player: when the video is playing, \( dt_p \) seconds of video are played in \( dt = dt_v \) seconds, i.e. \( d(t) = 1 \); on the other hand, when the player is paused the draining rate is zero. Thus, \( d(t) \) can be modelled as follows:

\[
d(t) = \begin{cases} 1 & \text{playing} \\ 0 & \text{paused} \end{cases}
\]
Finally, by combining (1) and (2) we obtain the playout buffer length model:

$$\dot{q}(t) = \frac{r(t)}{l(t)} - d(t).$$

(3)

The video level $l(t)$ is the control variable, $q(t)$ is the output of the controlled system, whereas $r(t)$ can be modelled as a disturbance. It is important to notice that $l(t)$ can only assume values in the discrete set $\mathcal{Z}$, i.e. the output of the controller is quantized.

In the following we employ the feedback linearization technique to compute a control law that linearizes (3) and that steers $q(t)$ to the set-point $q_T$. To this end, we impose the following linear closed-loop dynamics for the queue:

$$\dot{q}(t) = -k_p q(t) - k_i q_I(t)$$

(4)

$$q_I(t) = q(t) - q_T$$

(5)

where $q_I$ is an additional state that holds the integral error, $k_p \in \mathbb{R}_+$ and $k_i \in \mathbb{R}_+$ are the two parameters of the controller.

Now, by equating the right-hand sides of (3) and (4), it turns out:

$$l(t) = \frac{r(t)}{d(t) - k_p q(t) - k_i q_I(t)}$$

(6)

that is the control law employed by the stream-switching controller.

D. Implementation

Figure 2 shows the pseudo-code of the controller. When a segment is downloaded, the following quantities are measured: 1) the time spent to download the segment $\Delta T$ (line 2); the last downloaded segment size $S$ in bytes (line 3); state of the player $d$ (line 4); the playout buffer length (line 5); the received rate $r$ is estimated by passing the last segment download rate $S/\Delta T$ through a harmonic filter $h(\cdot)$ over the last 5 samples of $r$ (line 6). Then, the integral error $q_I$ is updated (line 7) and the control law is computed using (6) (line 8).

III. TESTBED

In this Section we describe the testbed, the experimental scenarios, and the metrics employed to evaluate and compare ELASTIC with the automatic video switching control of a leading Italian TV operator.

A. The testbed

Figure 3 shows the testbed that we have employed to carry out the experimental evaluation: the receiving host, or client, is a Debian Linux machine connected to the Internet via our 100 Mbps campus wired connection. The Adaptive Video Player (AVP), which we have developed in Python, runs on it. AVP is implemented using the GStreamer library and supports playback of MP4-encoded videos under the control of ELASTIC as adaptive streaming algorithm. Moreover, one or more Google Chrome Web Browser instances runs on the client with an embedded Video logger extension developed by us to measure the adaptive streaming metrics of the automatic video switching control considered in the comparison. The Video logger extension parses the video playlists, captures the video chunks download requests and measures both the playout buffer length and the portion of buffered video. We have always used the web browser in incognito mode to prevent from interference with caching techniques by removing data stored in the browser after previous views.

In order to set the bottleneck bandwidth capacity and propagation delays we have developed NetShaper. This tool employs the nfqueue library provided by Netfilter to capture and redirect to a user space drop-tail queue the packets arriving at the client. The traffic shaping policies are performed on this queue.

Before running each experiment, we have carefully checked that the end-to-end available bandwidth between the TV operator’s server and the client was well above the bottleneck capacity set by the traffic shaper. It is worth noting that all the measurements we report in the paper have been performed after the traffic shaper.

In each experiment we have used the same video sequence, encoded at two different bitrates as shown in Table I, with resolutions 700x394 and 1024x574. The duration of each chunk is 6s.

For each video, both the player and the Video logger extension are able to log: 1) the playout buffer length $q(t)$ measured in seconds, 2) the video level $l(t)$, 3) the cumulative downloaded bytes $D(t)$, 4) the cumulative re-buffering time $T_{rb}(t)$, 5) the number of re-buffering events $n_{rb}(t)$, 6) the number of level switches $n_I(t)$.

B. Scenarios and metrics

We have considered three scenarios: (S1) one video over a bottleneck link whose available bandwidth is set to

For each scenario we will show the dynamics of the following variables: 1) the chunk download bitrate \( c(t) \); 2) the received video bitrate \( r(t) = D(t)/\Delta T \) with \( \Delta T = 10s \); 3) the received video level \( l(t) \); 4) the playout buffer length \( q(t) \) measured in seconds. 

IV. RESULTS

In this Section we provide the details of the results obtained for both the considered algorithms.

A. One video over a 1.6 Mbps link

In this scenario we investigate the dynamic behaviour of an ELASTIC client and an automatic video switching control used by a leading Italian TV operator client to stream a video over a 1.6Mbps bottleneck link. This capacity is lower than the bitrate of the maximum level \( l_1 \) and higher than the bitrate of the minimum level \( l_0 \). The experiment duration is 600s.

Figure 4 shows the dynamics for each considered metric. The video connections start at \( t = 0 \) s. The figure shows that ELASTIC obtains a bitrate very close to the maximum possible, with a video level oscillating between \( l_0 \) and \( l_1 \) since the available bitrate is in the range \([l_0, l_1]\) . It is worth noting that the average video level obtained in this experiment is equal to the available bandwidth. On the other hand, in the case of the Italian TV operator the video level is always \( l_0 \), indicating that the maximum quality is not achieved.

B. The case of a square-wave varying bottleneck capacity

In this scenario we investigate how the quality adaptation algorithm reacts in response to abrupt drops/increases of the bottleneck capacity. Towards this end, we let the bottleneck capacity to vary as a square-wave with a period of 200 s, with a minimum value \( A_m = 1 \text{ Mbps} \) and a maximum value \( A_M = 4 \text{ Mbps} \). The aim of this experiment is to assess if the considered players are able to quickly change the video level when an abrupt drop/increase of the bottleneck capacity occurs in order to guarantee continuous reproduction of the video content at the maximum quality.

In the first case we have considered an initial bottleneck capacity set to \( A_M \). As shown in Figure 5a, the first result is that the Italian TV player starts with the maximum level due to the fact that the available bandwidth is well above the bitrate of the highest quality level \( l_1 \). It makes consecutive HTTP GET requests and fill the playout buffer up to 96 s until \( t = 200 \) s. At this moment, the available bandwidth goes down, the algorithm reacts immediately with a switch down at level \( l_0 \), but it produces an off period while the buffer decrease at 60 s, which corresponds to the upper limit for the buffer when the level is \( l_0 \). Finally, when there is a new bandwidth increase at \( t = 400 \) s the controller reacts immediately with another switch up at the highest quality \( l_1 \) and produces consecutive chunk requests, as shown by the requests estimated bandwidth \( cr \), until the playout buffer reach 146 s of video and the test is stopped.

In the figure 5b we can see the dynamic behaviour of the ELASTIC algorithm. The player enters in the steady-state phase when the playout buffer is up to 60 s. When the available bandwidth goes down at \( t = 200 \) s the player does not react immediately with a switch down although the estimated bandwidth is close to \( A_m \) and lower than the bitrate of \( l_1 \). This is due to the fact that the playout buffer is sufficient to avoid re-buffering events while the download at \( l_1 \) is completed. In the interval between \( t = 200 \) s and

---

**TABLE I**

<table>
<thead>
<tr>
<th>Video level</th>
<th>( l_0 )</th>
<th>( l_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>bitrate (kbps)</td>
<td>844</td>
<td>1845</td>
</tr>
<tr>
<td>Resolution</td>
<td>700x394</td>
<td>1024x576</td>
</tr>
</tbody>
</table>

**FIGURE 4**

One video over a 1.6Mbps bottleneck link.
In this paper we have proposed ELASTIC, a novel controller for adaptive video streaming obtained by using feedback linearization. Differently from current existing proposals, ELASTIC uses a unique controller that selects the video level \( l(t) \) that drives the playout buffer length to a set-point. ELASTIC eliminates the on-off traffic pattern which causes underutilization and unfairness when video flows coexist with long-lived TCP flows [8].

We have experimentally compared the performance of ELASTIC with the automatic video switching control of a leading Italian TV operator in a controlled testbed. Results have shown that Italian TV player is responsive in changing the video level to match the available bandwidth, but it is not able to grab the fair share when in the presence of coexisting TCP traffic. On the other hand, ELASTIC is always able to get the fair share and to match the available bandwidth on average in all the considered scenarios.

V. CONCLUSIONS

In this paper we have proposed ELASTIC, a novel controller for adaptive video streaming obtained by using feedback linearization. Differently from current existing proposals, ELASTIC uses a unique controller that selects the video level \( l(t) \) that drives the playout buffer length to a set-point. ELASTIC eliminates the on-off traffic pattern which causes underutilization and unfairness when video flows coexist with long-lived TCP flows [8].

We have experimentally compared the performance of ELASTIC with the automatic video switching control of a leading Italian TV operator in a controlled testbed. Results have shown that Italian TV player is responsive in changing the video level to match the available bandwidth, but it is not able to grab the fair share when in the presence of coexisting TCP traffic. On the other hand, ELASTIC is always able to get the fair share and to match the available bandwidth on average in all the considered scenarios.

VI. ACKNOWLEDGEMENT

This work has been partially supported by the Italian Ministry of Education, Universities and Research (MIUR) through the PLATINO project (PON01 01007).

REFERENCES


Fig. 5. One video flow sharing a square-wave varying bottleneck link starting with $A_M = 4$Mbps

(a) Italian TV operator

(b) ELASTIC

Fig. 6. One video flow sharing a square-wave varying bottleneck link starting with $A_m = 1$Mbps

(a) Italian TV operator

(b) ELASTIC


