ELASTIC: a Client-side Controller for Dynamic Adaptive Streaming over HTTP (DASH)

Luca De Cicco, *Member, IEEE,* Vito Caldaralo, Vittorio Palmisano, and Saverio Mascolo, *Senior Member, IEEE*

***Abstract*—Today, video distribution platforms use adaptive video streaming to deliver the maximum Quality of Experience to a wide range of devices connected to the Internet through different access networks. Among the techniques employed to implement video adaptivity, the stream-switching over HTTP is getting a wide acceptance due to its deployment and implementation simplicity. Recently it has been shown that the client-side algorithms proposed so far generate an on-off traffic pattern that may lead to unfairness and underutilization when many video flows share a bottleneck. In this paper we propose ELASTIC (fEedback Linearization Adaptive STreamIng Controller), a client-side controller designed using feedback control theory that does not generate an on-off traffic pattern. By employing a controlled testbed, allowing bandwidth capacity and delays to be set, we compare ELASTIC with other client-side controllers proposed in the literature. In particular, we have checked to what extent the considered algorithms are able to: 1) fully utilize the bottleneck, 2) fairly share the bottleneck, 3) obtain a fair share when TCP greedy flows share the bottleneck with video flows. The obtained results show that ELASTIC achieves a very high fairness and is able to get the fair share when coexisting with TCP greedy flows.**

I. Introduction

The Internet traffic due to video applications is increasing thanks to the diffusion of new devices such as tablets, smart phones, Smart TVs which are connected to the Internet through broadband wired and wireless connections. Video streaming, in the form of user-generated video distribution – such as in the case of Youtube – or to deliver movies and TV series – such as in the case of Netflix – is the application that is driving this growth.

Even though the TCP has been regarded in the past as unsuitable to transport video flows, today videos are streamed through HTTP with the TCP, and web browsers implementing the HTML5 standard are now able to reproduce videos without the use of any external plug-in.

The most common approach to distribute video is the *progressive download streaming*: the video content is encoded at a given bitrate and it is sent to the user as any other file through a HTTP connection. The issue with this approach is that, even though TCP connections are elastic, the video content transported through the TCP socket is not elastic; thus, a persistent mismatch between the encoding bitrate and the network available bandwidth may result in playout interruptions. Another drawback of such an approach is that mobile devices, such as tablets or smart phones, may not be able to play a high definition video due to their limited computational resources.

To tackle these issues, the video content must be made adaptive. Among the approaches proposed so far, the stream-switching is gaining momentum due to its deployment and implementation simplicity. With this approach, the video is encoded at different bitrates and resolutions, the *video levels*, and the encoded videos are logically or physically divided into *segments* of fixed durations. The stream-switching *controller* decides, for each video segment, the video level to be streamed.

From the architectural standpoint, two different approaches can be used to implement a stream-switching algorithm: the *client-side*, that places the controller at the client, and the *server-side*, that implements the controller at the server. 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 a bottleneck [1], [2], [9]. Moreover, in [7] it has been established that the adaptive video players of three popular video streaming services were not able to get a fair share when coexisting with a TCP greedy flow. Authors name this issue the "*downward spiral effect*" and ascribe its cause to the on-off traffic pattern described above; authors suggest to increase the segment size and filter bandwidth estimates to tackle this issue.

In this paper we show that it is possible to implement a client-side player that does not generate an on-off traffic pattern and avoids the "downward spiral effect" without increasing the segment size, and without using large player buffers.

The contribution of this paper is two fold: firstly, we propose ELASTIC (*fEedback Linearization AdaptIve STreamIng Controller*), a client-side stream-switching controller designed using feedback control theory that avoids the generation of on-off traffic pattern and is able to fairly share the bottleneck with other videos and TCP greedy flows; secondly, we compare the performances of ELASTIC with 1) a conventional client-side algorithm, 2) PANDA [11], and 3) FESTIVE [8].

To quantitatively evaluate the performances of the considered algorithms, we have used a controlled testbed allowing bandwidth capacity and delays to be set. In particular, we have considered two scenarios: 1) a number of video flows using the same control algorithm sharing a bottleneck; 2) N_v video flows sharing the bottleneck with N_{TCP} TCP greedy flows.

The paper is organized as follows: Section II reviews the state of the art of adaptive video streaming proposals; in Section III ELASTIC is presented; Section IV describes the employed testbed; Section V presents the results of the

L. De Cicco, V. Palmisano, and S. Mascolo are with the Dipartimento di Ingegneria Elettrica e dell'Informazione at Politecnico di Bari, Via Orabona 4, 70125, Bari, Italy Emails: l.decicco@poliba.it, vpalmisano@gmail.com, mascolo@poliba.it

V. Caldaralo is with CRAT Bari, Via Orabona 4, 70125, Bari, Italy, Email: vito.caldaralo@gmail.com

experimental evaluation and Section VI concludes the paper.

II. Related Work

Today, adaptive streaming is employed by major commercial services such as NetFlix, Hulu, Vudu, Akamai, Livestream, and YouTube.

The leading architectural approach to implement adaptive streaming places the controller at the client so that standard HTTP servers can be used for video distribution [15] 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 τ, 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 τ 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 [9], [1], [3].

Recently, several studies have analyzed the issues of client-side adaptive streamers. In [7] it has been shown that three commercial services, Hulu, Vudu, and Netflix, employing client-side adaptation were not able to obtain the fair share when competing with long-lived TCP connections and, as a consequence, videos did not receive the maximum possible video quality. Authors have ascribed this issue to the on-off traffic pattern described above. In [1] authors find that the on-off traffic pattern is the key factor causing the following three issues: unfair bandwidth utilization, server bandwidth underutilization, flickering of the player requested video level. Similar observations regarding the server bandwidth underutilization have been reported in [9].

To tackle those issues, several adaptive streaming algorithms have been proposed so far. FESTIVE has been the first adaptive video streaming algorithm specifically designed to address the fairness issues arising in a multi-client scenario [8]. Another recent proposal can be found in [11], where authors design PANDA, an algorithm that dynamically computes the segment inter-request time to address fairness issues, and video bitrate oscillations. A different approach has been followed in [2], where authors propose to place a traffic shaper at the server with the goal of eliminating the OFF phases when the player is in steady-state.

In [4] 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, such streamer employs a hybrid sender-side/client-side architecture with two controllers running at the client: one for selecting the video level, the other that throttles the sending rate at the server for controlling the playout buffer length at the client. Moreover, the system does not issue many HTTP GET requests to download the segments, but controls the video level to be streamed through commands sent via HTTP POST requests to the server. This feature makes this system interesting and unique *wrt* those which entirely rely on client control.

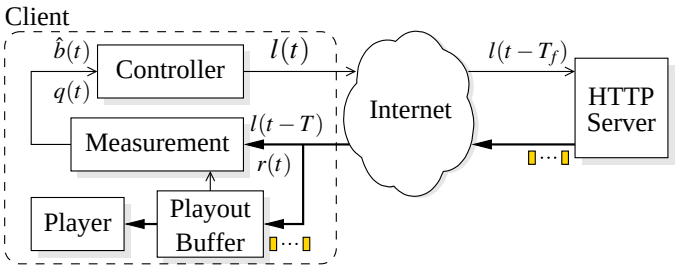


Figure 1. Client-side adaptive video streaming

In [5] a sender-side only video stream switching algorithm is designed by using feedback control theory. The algorithm has been experimentally compared with the adaptive streamer described in [4].

III. Elastic

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

A. Design requirements

The main goal of a stream-switching controller is to dynamically select the *video level* $l(t) \in \mathscr{L} = \{l_0, \ldots, l_{N-1}\}$ for each video segment to achieve the maximum Quality of Experience (QoE). Even though a widely accepted QoE metric for adaptive video streaming is still not available, recently several authors have studied the key factors impairing user engagement [6], [13]. Indeed, re-buffering events, occurring when the player buffer gets empty, have been identified to be one of the major causes impairing user engagement [6]. Moreover, it has been shown that frequent quality switches may be annoying to the user [13], thus limiting video level switches is considered a design requirement by several proposed algorithms [8], [11].

A key factor that may improve the QoE is the video level received by the client: in principle the higher the bitrate and resolution, the better the perceived quality. Indeed, the quantification of how the video level affects the perceived QoE is subject of ongoing research efforts [14], [6], [12].

Summarizing, we consider the following design goals for ELASTIC: 1) minimize the re-buffering ratio; 2) maximize the obtained video level; 3) contain the video level switches; 4) 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 client-side adaptive video streaming system: 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 $r(t)$, bounded

– on average – by $b(t)$, and temporarily stores them in a *playout buffer* that feeds the player decoder. 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 conventional approach to implement a stream-switching system is to design two controllers [3], [4], [7], [11]: the first one throttles the video level $l(t)$ to match the measured available bandwidth $b(t)$, the second one controls the playout buffer length $q(t)$ by regulating the idle period between two segment downloads. In Section II we have reported that the on-off traffic pattern is the key factor causing 1) unfair sharing of bottleneck when multiple video flows coexist, and 2) low channel utilization when in the presence of long-lived TCP flows.

Differently from the conventional approach, ELASTIC only uses one controller that throttles the video level $l(t)$ to drive $q(t)$ to a set-point q_T. This means that we do not require to idle between segment downloads to regulate $q(t)$. 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 $q(t)$ can be mathematically modelled by the following non-linear time-delay differential equation:

$$\dot{q}(t) = \frac{r(t)}{l(t-T)} - d(t) \tag{1}$$

where $d(t)$ is the player *draining rate* (dimensionless) which can be modelled as follows:

$$d(t) = \begin{cases} 1 & \text{playing} \\ 0 & \text{paused or } q(t) = 0 \end{cases} \tag{2}$$

The rationale of (1) is simple: the term $r(t)/l(t-T)$, the ratio between the received rate and the received video level, models the playout buffer filling rate, i.e. how many seconds of video are stored in the buffer per second; the second term $d(t)$ is the draining rate, i.e. how many seconds of video are played per second. When the video is playing the draining rate is 1, since the player drains one second of video per second, whereas when the player is paused, d is zero allowing video segments to be buffered.

The video level $l(t)$ is the output of the controller, $q(t)$ is the output of the control 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 $\mathscr{L}$, 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 (1) 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:

1: On segment download:
2: $\Delta T \leftarrow getDownloadTime()$
3: $S \leftarrow getSegmentSize()$
4: $d \leftarrow isPlaying()$
5: $q \leftarrow getQueueLength()$
6: $r \leftarrow h(S/\Delta T)$
7: $q_I \leftarrow q_I + \Delta T \cdot (q - q_T)$
8: **return** $Quantize(r/(d - k_p q - k_i q_I))$

Figure 2. ELASTIC controller pseudo-code

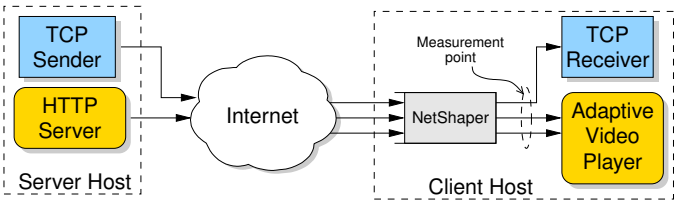


Figure 3. Testbed setup.

$$\dot{q}(t) = -k_p q(t) - k_i q_I(t) \tag{3}$$
$$\dot{q}_I(t) = q(t) - q_T \tag{4}$$

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 (1) and (3), it turns out:

$$l(t-\tau) = \frac{r(t)}{d(t) - k_p q(t) - k_i q_I(t)} \tag{5}$$

that is the control law employed for 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 ΔT (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 (5) (line 8).

IV. Testbed

In this Section we provide the details of the testbed, the scenarios, and the metrics employed to carry out the experimental evaluation of the considered adaptive video streaming algorithms.

A. The testbed

Figure 3 shows the testbed that we have employed to carry out the experimental evaluation: the HTTP, and the TCP servers are installed on a Debian Linux server, whereas a

Table I. DISCRETE SET OF VIDEO LEVELS $\mathscr{L}$.

Video level	l_0	l_1	l_2	l_3	l_4
Bitrate (kbps)	300	700	1500	2500	3500
Resolution	320x180	640x360	640x360	1280x720	1280x720

Ubuntu Linux client machine runs the `Adaptive Video Player (AVP)` and the TCP receiver. We employ Apache as the HTTP server and IPerf as the TCP server and receiver. At the receiver we have used a tool developed by us called `NetShaper` that performs bandwidth shaping and allows propagation delays to be set. This tool uses the `nfqueue` library provided by `Netfilter`[1] in order to capture and redirect the packets arriving at the client host to a user space drop-tail queue, where traffic shaping and measurement are performed.

`AVP` is implemented using the GStreamer[2] libraries and supports the HTTP Live Streaming (HLS) format[3]. We implemented several client-side algorithms as `AVP` plugins, which can be selected from a command line option. The `AVP` has been designed to be as light as possible to allow the same receiving host running a large number of video player instances at the same time. The player logs 1) the playout buffer length $q(t)$, 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 $n_s(t)$.

To run the experiments, we have employed the video sequence "*Elephant's Dream*"[4] encoded at five different bitrates as shown in Table I. The video is encoded at 30 frames per second (fps) using the H.264 codec with a fixed group of picture (GOP) of length 30, so that two consecutive I-frames are 1s apart and time-aligned between different levels; the audio is encoded using MP3 codec at 128 kbps bitrate. Moreover, in order to provide the HLS support for testing the client side algorithms, we have produced a set of fragmented videos for each level using the MPEGTS container, with a fragment duration of 2s, and for each set we have stored the `m3u8` playlist accordingly. All the HLS files (`.ts` fragments and `.m3u8` playlists) are served by the Apache server configured with a `KeepAliveTimeout` of 60 s to support HTTP persistent connections. It is worth mentioning that we have also run the experiments without using persistent connections and we have recovered the poor results presented in [10] particularly when video flows shared the bottleneck with TCP greedy flows. Due to space limitations, we only present the results using HTTP persistent connections in this paper.

B. Scenarios and metrics

We have considered two scenarios: (S1) two total connections (video and TCP) sharing a bottleneck link whose available bandwidth is set to $b = 4$Mbps; this scenario is aimed at showing the dynamic behaviour of the considered algorithms; (S2) a variable number of total connections (video and TCP) over a bottleneck link whose available bandwidth is set to $b = 40$Mbps. For each scenario the round-trip propagation delay is set to 50 ms.

Table II. METRICS DEFINITION

Symbol	Name	Definition
r	Received rate	$\frac{\text{downloaded bytes}}{\text{time interval}}$
U	Channel utilization	$\overline{r}/b$
JFI	Jain Fairness Index	$\left(\sum_{i=1}^{n}\overline{r}_i\right)^2 / \left(n\sum_{i=1}^{n}\overline{r}_i^2\right)$
RB	Re-buffering ratio	$\frac{\text{total re-buffering time}}{\text{experiment duration}}$
n_s	video level switches	

For the scenario (S1) we will show the dynamics of the following variables: the received video level $l(t)$ and the received video bitrate $r(t)$. In the scenario (S2) we employ the metrics defined in Table II to compare the performances of the considered algorithms. In order to evaluate the fairness in the case n video flow share the bottleneck, we measure the Jain Fairness Index (JFI):

$$JFI = \frac{\left(\sum_{i=1}^{n}\overline{r}_i\right)^2}{n\sum_{i=1}^{n}\overline{r}_i^2}$$

where $\overline{r}_i$ is the average received rated obtained by the i-th video flow.

C. The considered algorithms

To carry out the experimental evaluation, we have implemented the following algorithms in the Adaptive Video Player:

1) ELASTIC according to the description given in Section III with $k_p = 1/100$,$k_i = 1/1000$;
2) FESTIVE according to [8];
3) PANDA according to the implementation details given in [11, Table II];
4) the conventional controller as described in [11]; the parameters have been set with the values specified in [11, Table II];

All the players employed a threshold for the playout buffer length equal to 15s.

V. RESULTS

A. Two total connections sharing a 4 Mbps link

In this scenario we investigate the dynamic behaviour of the considered algorithms when sharing a 4 Mbps bottleneck link in two cases: 1) two videos using the same control algorithm sharing the bottleneck; 2) one video and one TCP flow share the bottleneck. The experiment duration is 300s. The fair share in this experiment is 2 Mbps, that corresponds to a video level between l_2 (1.5 Mbps) and l_3 (2.5 Mbps).

Figure 4 shows the dynamics of the received video bitrate for each algorithm in the case two video flows share the bottleneck. Both the video connections start at $t = 0$ s. The figure shows that the Conventional player and PANDA exhibit similar dynamics: in both the cases the two video flows obtain the same video level and exhibit very few video level switches; however, the steady-state received rate obtained by the two competing flows is below the fair share, indicating that the

[1] http://www.netfilter.org/
[2] http://gstreamer.freedesktop.org/
[3] http://tools.ietf.org/html/draft-pantos-http-live-streaming-07
[4] http://orange.blender.org/

channel is not fully utilized. On the other hand, the other two controllers provide a received video rate that oscillates around the fair share, but with an increased number of video level switches.

Let us now consider the case of a video flow sharing the bottleneck with a long-lived TCP flow starting at $t = 100$s. Figure 5 shows the dynamics of the received video bitrate and the TCP bitrate for each of the considered algorithms. The figure clearly shows that Conventional, PANDA, and FESTIVE are not able to obtain the fair share when coexisting with a TCP flow showing the "downward spiral effect" shown in [7]. In particular, the steady-state video levels obtained by Conventional, PANDA, and FESTIVE are respectively l_0, l_1 and l_1. On the other hand, ELASTIC obtains a bitrate very close to the fair share, with a video level oscillating between l_2 and l_3.

The ELASTIC flow is able to get the fair share when coexisting with a long-lived TCP flow because it does not produce an on-off traffic pattern thus behaving as a TCP flow.

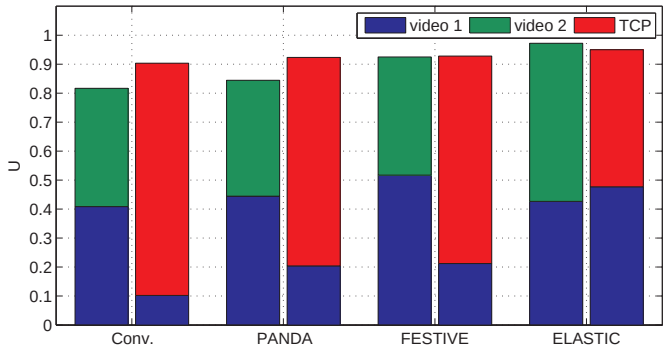


Figure 6. Channel utilization in the case of two flows (either two videos or one video and one TCP) sharing a 4Mbps link

Figure 6 shows the channel utilization for each flow of all the experiments presented above. ELASTIC obtains the best results in both the scenarios, obtaining the highest channel utilization (0.95) and a fair utilization of the bottleneck. In the case two video flow share the bottleneck, Conventional, PANDA, and FESTIVE obtain a total channel utilization respectively equal to 0.82, 0.85, and 0.93. When a *conventional* flow shares the bottleneck with a TCP flow, it obtains only 10% of the bottleneck capacity. Finally, PANDA and FESTIVE obtain roughly 20% of the bottleneck capacity when coexisting with a TCP flow.

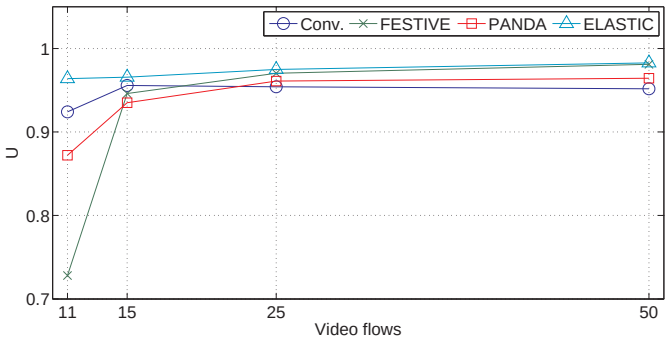
To summarize, this scenario has shown that the proposed controller is able to get the fair share when coexisting with a concurrent TCP flow due to the fact that it does not generate an on-off traffic pattern.

B. N_v videos sharing a 40Mbps link with N_{TCP} long-lived TCP flows

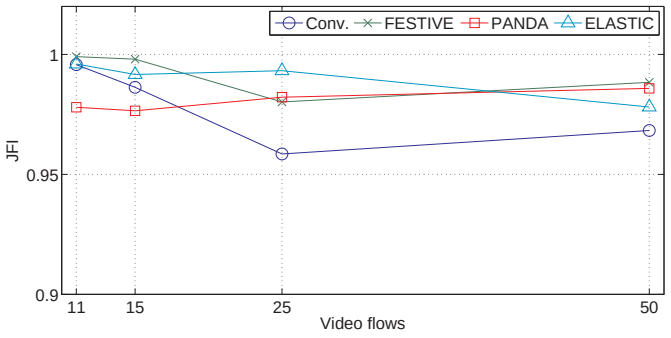
In this scenario we investigate the behaviour of the considered algorithms when a 40 Mbps bottleneck link is shared among a variable number of video flows and TCP connections.

$N=11$ (l_4)		$N=15$ (l_3)		$N=25$ (l_2)		$N=50$ (l_1)		
N_v	N_{TCP}	N_v	N_{TCP}	N_v	N_{TCP}	N_v	N_{TCP}	$\frac{N_{TCP}}{N}$
11	0	15	0	25	0	50	0	**0**
8	3	11	4	18	7	38	12	**1/4**
5	6	7	8	12	13	25	25	**1/2**
3	8	4	11	6	19	12	38	**3/4**

Table III. EXPERIMENTS SETUP WITH A VARIABLE NUMBER OF CONNECTIONS



(a) Channel utilization



(b) Jain Fairness Index

Figure 7. A variable number of video flows sharing a 40Mbps link ($N_{TCP} = 0$)

The video flows start at $t = 0$s, whereas the TCP flows start at 100s. The total duration of the experiment is 300s. The total number N of flows varies in the set $\mathcal{N} = \{11, 15, 25, 50\}$ so that for each $N \in \mathcal{N}$ the fair share can be computed as $F = 40\text{Mbps}/N$.

For each $N \in \mathcal{N}$ we vary the number of TCP flows N_{TCP} such that the fraction of TCP flows N_{TCP}/N is equal to 0, $1/4$, $1/2$, $3/4$. Table III shows the values of N_v and N_{TCP} employed in each experiment. The table also shows for each $N \in \mathcal{N}$ the optimal video level that each flow should get. It is important to notice that all the reported metrics have been evaluated for $t > 100$s, after all the TCP flows are started.

We start by considering a variable number of videos N_v without concurrent TCP flows ($N_{TCP} = 0$) with the aim of investigating the fairness and the efficiency in utilizing the bottleneck.

Figure 7 (a) shows the channel utilization U function of N_v. ELASTIC provides the best results: the channel utilization does not depend on N_v and it is roughly equal to 0.96. On

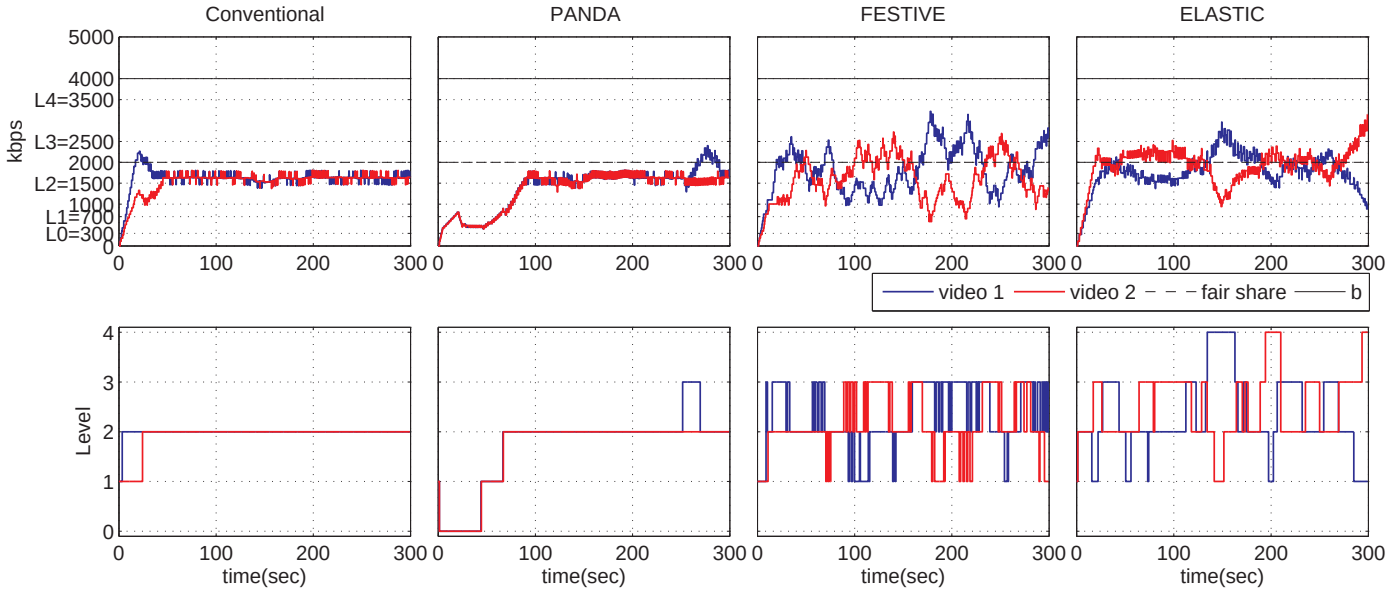


Figure 4. Two video flows sharing a 4Mbps bottleneck link.

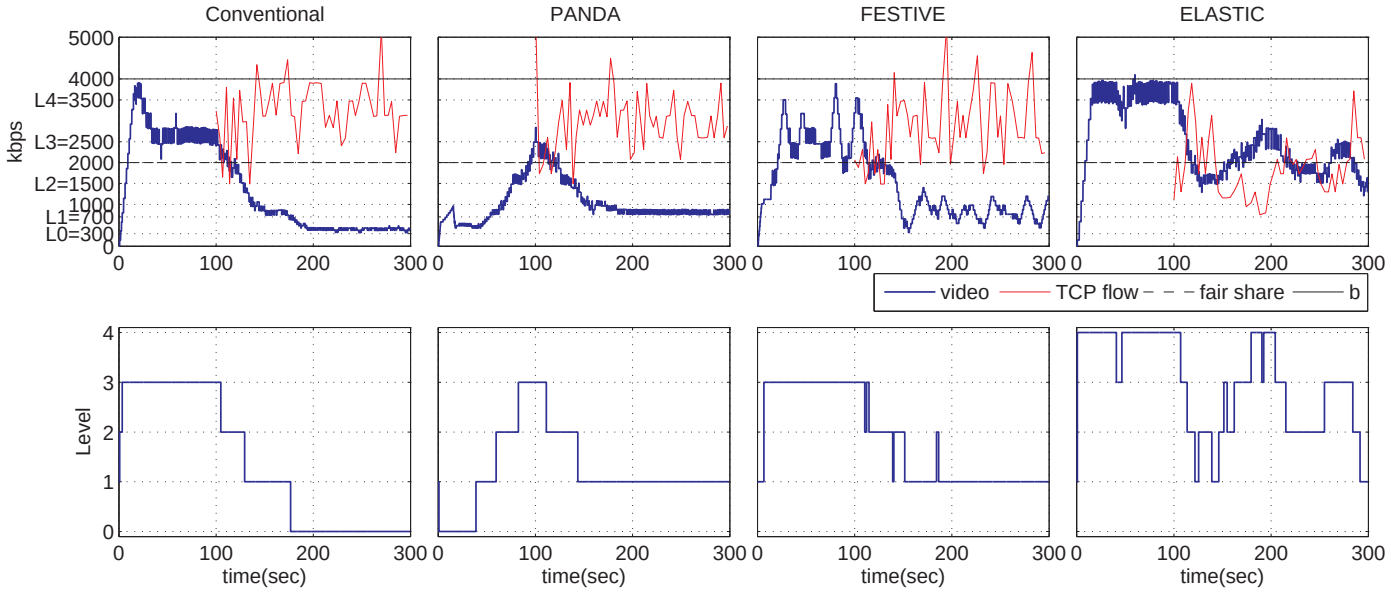


Figure 5. One video flow sharing a 4Mbps bottleneck link with a TCP flow

the other hand, the other algorithms exhibit a lower channel utilization for $N_v = 11$ that increases with N_v. Figure 7 (b) shows that the JFI obtained by all the considered algorithms is in the range $[0.96, 1]$.

Figure 8 shows the average video rate function of $N \in \{11, 15, 25, 50\}$ and of $N_{TCP}/N \in \{0.25, 0.5, 0.75\}$ along with the fair share drawn with a dashed line.

The figure shows that, regardless of the total number of flows N and of the fraction of TCP flows N_{TCP}/N, ELASTIC always obtains an average video rate that is very close to the fair share and to the optimal video level.

The average video rate obtained by FESTIVE does not depend on the fraction of TCP flows but it is affected by the total number of connections. In particular when $N = 11$, FESTIVE obtains an average video rate of around 2500kbps which corresponds to roughly 70% of the fair share, whereas when N increases the average rate approaches to the fair share.

Let us now consider Conventional and PANDA that achieve similar results: Figure 8 clearly shows that the efficiency in sharing the bottleneck decreases when the fraction of TCP

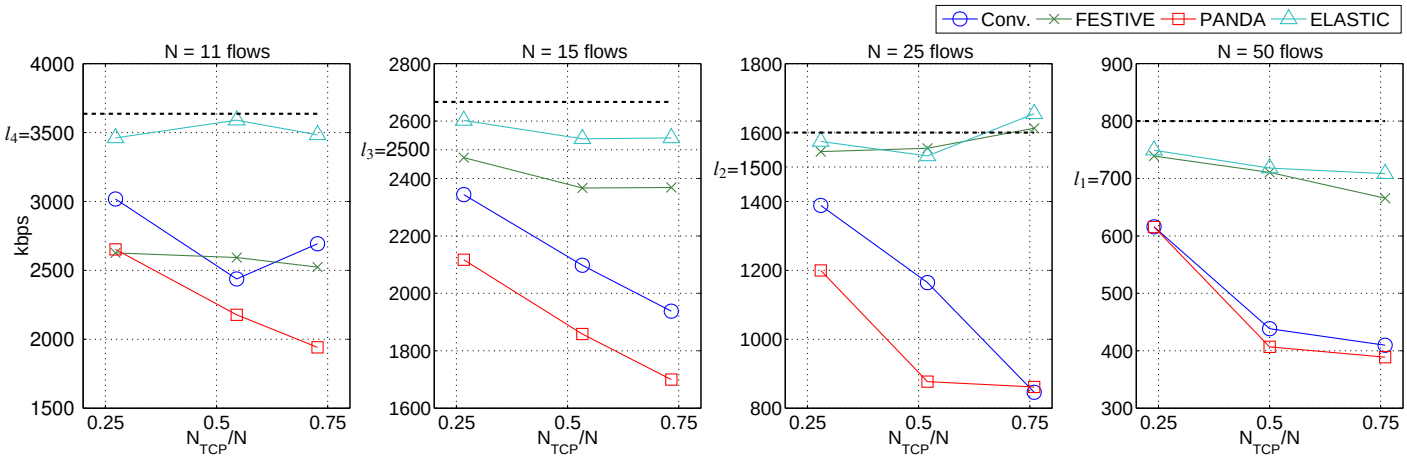


Figure 8. Average video rate when N_v videos share a 40 Mbps bottleneck with N_{TCP} TCP flows

flows increases, meaning that each video flow leaves to the TCP flows an increasing share of bandwidth when N_{TCP} is large. When $N = 25$ and $N_{TCP}/N = 0.75$ both the algorithms obtain an average video rate of 800 kbps which is close to $l_1 =$ 700kbps and well below the fair share 1600kbps that would allow a video level $l_2 = 1500$kbps.

Figure 9 shows the average number of switches provoked by the considered algorithms. PANDA always provides the best results, provoking on average less than 2 switches per video flow. ELASTIC produces roughly the same number of video switches in all the considered experiments. A maximum of 8 switches per flow is obtained for $N = 15$ flows. The number of switches exhibited by FESTIVE increases with N but does not depend on the number of concurrent TCP flows. The worst performance is obtained for $N = 25$ where FESTIVE provokes around 20 level switches. The number of level switches provided by the Conventional algorithm is lower than the one provoked by ELASTIC for $N \geq 15$ and it decreases with the number of concurrent TCP flows.

To conclude the performance evaluation, we consider the re-buffering ratio. Figure 10 only shows the re-buffering ratio in the case $N = 50$, since for $N < 50$ we have measured re-buffering ratios always less than 1% for all the considered algorithms. The figure shows that both ELASTIC and Conventional provide re-buffering ratios that are less than 5%. On the other hand, FESTIVE and PANDA exhibit increasing re-buffering ratios when the number of TCP flows increases. In particular, when $N_{TCP}/N = 0.75$ FESTIVE and PANDA provoke respectively 17% and 12% rebuffering ratios.

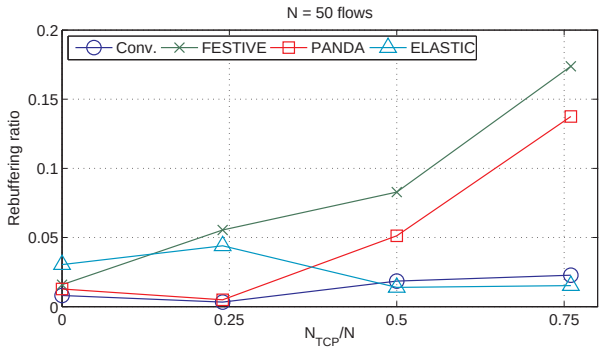


Figure 10. Rebuffering ratio in the case of a variable number of video flows sharing a 40 Mbps link.

VI. Conclusions

In this paper we have proposed ELASTIC, a client-side stream-switching controller for dynamic adaptive streaming over HTTP designed using feedback control theory. Differently from the conventional approach, which employs two controllers, one to throttle the video level, the second to regulate the playout buffer, ELASTIC only uses one controller that computes the video level $l(t)$ to drive the playout buffer to a set-point. With the conventional approach an on-off traffic pattern is generated which is known to be the cause of underutilization when coexisting with long-lived TCP flows [7], whereas ELASTIC always generates a traffic pattern that is identical to any long-lived TCP flow.

We have experimentally compared the performance of ELASTIC with three client-side algorithms in a controlled testbed: 1) FESTIVE [8], 2) PANDA [11], and 3) a conventional player. We have found that all the considered algorithms provide high channel utilization and fairness when multiple video flows share a bottleneck in the absence of concurrent TCP flows. However, when a mix of video flows and TCP flows share the bottleneck we have found that FESTIVE, PANDA, and the conventional player are not able to always get the fair share. On the other hand, the proposed controller has been able to obtain the fair share in all the considered scenarios, while providing a continuous reproduction of the video and mitigating the video level switches.

Our future work will aim at carrying out a sensitivity analysis of the performance of the control system *wrt* the chosen video levels set $\mathscr{L}$, on the controller parameters k_p and k_i, and on the duration of the video segment. Finally, we aim at

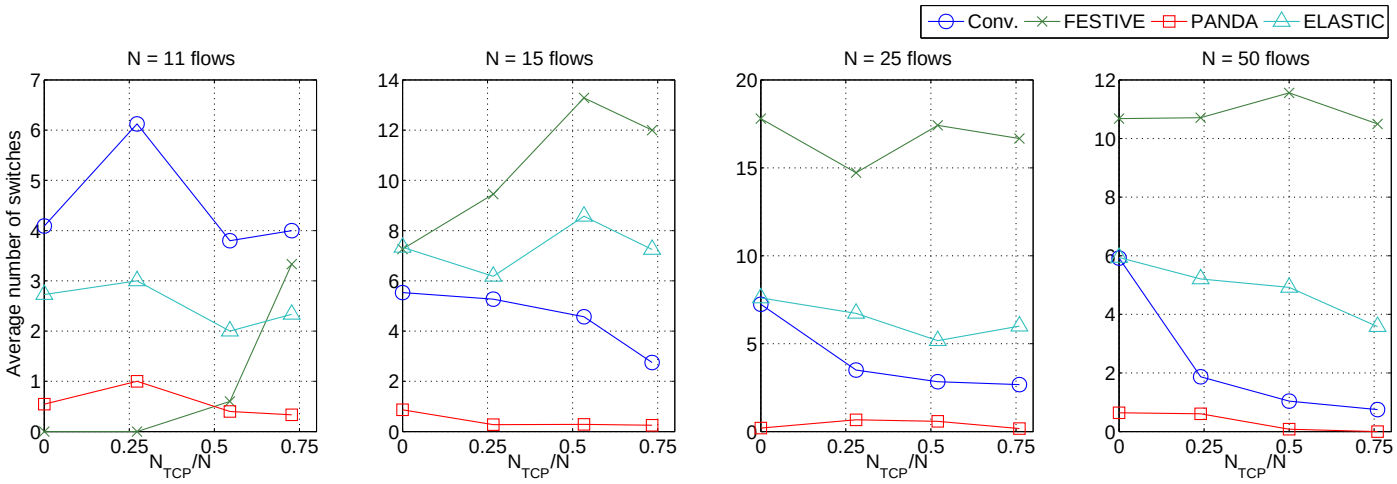


Figure 9. Average number of switches in the case N_v videos share a 40 Mbps bottleneck with N_{TCP} TCP flows

deriving fundamental properties of the control system, such as the stability of the equilibrium as function of the parameters.

VII. Acknowledgements

This project has been made possible in part by the gift CG #574954 from the Cisco University Research Program Fund, a corporate advised fund of Silicon Valley Community Foundation. This work has been also partially supported by the Italian Ministry of Education, Universities and Research (MIUR) through the PLATINO project (PON01 01007).

References

[1] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. C. Begen. What happens when http adaptive streaming players compete for bandwidth? In *Proc. of ACM NOSSDAV '12*, 2012.

[2] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. C. Begen. Server-based traffic shaping for stabilizing oscillating adaptive streaming players. In *Proc. of ACM NOSSDAV '13*, 2013.

[3] S. Akhshabi, A. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. *Proc. of ACM MMSys 2011*, pages 157–168, 2011.

[4] L. De Cicco and S. Mascolo. An adaptive video streaming control system: Modeling, validation, and performance evaluation. *IEEE/ACM Transactions on Networking*, in press.

[5] L. De Cicco, S. Mascolo, and V. Palmisano. Feedback control for adaptive live video streaming. In *Proc. of ACM MMSys 2011*, pages 145–156, 2011.

[6] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the impact of video quality on user engagement. In *Proc. of ACM SIGCOMM 2011*, pages 362–373, 2011.

[7] T. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari. Confused, timid, and unstable: picking a video streaming rate is hard. In *Proc. of ACM Internet Measurement Conference*, pages 225–238, 2012.

[8] J. Jiang, V. Sekar, and H. Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proc. of CoNEXT '12*, pages 97–108, 2012.

[9] T. Kupka, P. Halvorsen, and C. Griwodz. Performance of On-Off Traffic Stemming From Live Adaptive Segmented HTTP Video Streaming. In *Proc. of IEEE Conference on Local Computer Networks*, pages 405–413, Oct. 2012.

[10] S. Lederer, C. Müller, and C. Timmerer. Dynamic adaptive streaming over http dataset. In *Proc. of ACM MMSYS '12*, pages 89–94, 2012.

[11] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran. Probe and adapt: Rate adaptation for http video streaming at scale. *arXiv preprint arXiv:1305.0510*, 2013.

[12] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang. A case for a coordinated internet video control plane. In *Proc. of ACM SIGCOMM 2012*, SIGCOMM '12, pages 359–370, 2012.

[13] P. Ni, R. Eg, A. Eichhorn, C. Griwodz, and P. Halvorsen. Flicker effects in adaptive video streaming to handheld devices. In *Proc. of 19th ACM international conference on Multimedia*, pages 463–472. ACM, 2011.

[14] A. B. V. Sekar, A. Akella, S. S. I. Stoica, and H. Zhang. Developing a predictive model of quality of experience for internet video. In *Proc. of ACM SIGCOMM '13*, 2013.

[15] I. Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE MultiMedia*, 18(4):62–67, 2011.