An Adaptive Video Streaming Control System: Modelling, Validation, and Performance Evaluation

Luca De Cicco and Saverio Mascolo

***Abstract*—Adaptive video streaming is a relevant advancement with respect to classic progressive download streaming *a-la* YouTube. Among the different approaches, the video stream-switching technique is getting wide acceptance, being adopted by Microsoft, Apple and by popular video streaming services such as Akamai, Netflix, Hulu, Vudu, and Livestream. In this paper, we present a model of the automatic video stream-switching employed by one of these leading video streaming services along with a description of the client-side communication and control protocol. From the control architecture point of view, the automatic adaptation is achieved by means of two interacting control loops having the controllers at the client and the actuators at the server: one loop is the buffer controller, which aims at steering the client playout buffer to a target length by regulating the server sending rate; the other one implements the stream-switching controller and aims at selecting the video level. A detailed validation of the proposed model has been carried out through experimental measurements in an emulated scenario.**

***Index Terms*—Adaptive video streaming, stream-switching, modelling, performance evaluation**

I. Introduction

VIDEO traffic will account for more than 90% of the global Internet traffic in 2014 according to a recent report published by Cisco [1]. Such a tremendous growth is fed by video streaming applications such as YouTube, which delivers user-generated video content, or NetFlix, which streams movies and already accounts for more than 20% of the USA Internet traffic. Another ongoing trend that is feeding this growth is the ever increasing number of smart-phones and tablet devices accessing the Internet by using 3G/4G wireless mobile connections [1].

Today, the content producer has to undertake the challenging task of providing the user with a seamless multimedia experience at the maximum obtainable Quality of Experience (QoE) given the user device heterogeneity. To this purpose, multimedia content is required to be adaptive in order to match a wide set of variables such as user screen resolution, CPU load, network available bandwidth, power consumption.

Adaptive video streaming represents a key innovation *wrt* classic progressive download streaming such as the one employed by YouTube. In fact, with progressive streaming, the video is encoded at a constant quality or bit-rate and it is delivered as any other file over HTTP. The received video is temporarily stored in a playout buffer before the playing is started so that the short-term mismatches between the video bitrate and the available bandwidth can be absorbed and video interruptions can be mitigated. Nonetheless, in the case of a persistent mismatch, the buffer could eventually get empty with the consequence of playback interruptions and the need of re-buffering. On the other hand, with adaptive streaming the video bitrate is throttled on-the-fly in order to match the time-varying available bandwidth and get the best video quality while minimizing start-up latency and avoiding video playback interruptions.

The leading approach for implementing adaptivity is the *stream-switching* (or multi bit-rate): the server encodes the video content at different bit-rate levels and an adaptation algorithm selects the video level to be served based on measurements such as the available bandwidth and the player buffer length [2]. The stream-switching technique is today employed, among others, by Apple HTTP-based streaming, Microsoft IIS Smooth Streaming, Adobe Dynamic Streaming, Akamai, Move Networks, Hulu, NetFlix, and Livestream. Moreover, this approach is also adopted by the *Dynamic Adaptive Streaming over HTTP* (DASH), a new MPEG standard pursuing the interoperability between devices and servers of various vendors [3].

Recently, a great deal of attention in the literature has been devoted to adaptive streaming and to its commercial implementations [4], [5], [6], [7], [8], [9], [10] (see Section II). In this paper we consider a major CDN operator whose interesting and unique adaptive streaming architecture operates, according to our findings, using two controllers, one for selecting the video level that matches the Internet best-effort bandwidth and the other for controlling the playout buffer length. In particular, the considered video streaming system employs a server that actively throttles the sending rate to control the playout buffer. This makes the considered control system different from those which entirely rely on client control [4], [3].

The main goal of this paper is to mathematically model and analyze, in a control theoretic framework, the mentioned automatic video streaming system that is based on both server and client side control.

The paper is organized as follows: Section II summarizes the related works dealing with adaptation techniques for video streaming along with the experimental evaluation of several stream-switching approaches; in Section III the model of the automatic stream-switching controller is presented; Section IV describes the client-server communication and control protocol employed to implement the stream-switching control algo-

Luca De Cicco is research assistant at Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari, Via Orabona 4, Italy (e-mail: l.decicco@poliba.it), Phone: +390805963851, Fax: +390805963410

Saverio Mascolo is full professor at Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari, Via Orabona 4, Italy (e-mail: mascolo@poliba.it), Phone: +390805963621, Fax: +390805963410

This work has been partially supported by the project "Platform for Innovative services in the Future Internet" (PLATINO - PON 2007IT161PO006) funded by Italian Ministry of Education, Universities and Research (MIUR).

rithm; in Section V the proposed model is validated in a wired controlled testbed; finally, Section VI draws the conclusions of the paper.

II. Related Work

In the last two decades a vast literature regarding video streaming has been produced with main topics ranging from the design of transport protocols specifically tailored for video streaming to the video quality adaptation techniques.

Concerning the first topic, several transport protocols for video streaming have been proposed, such as the TCP Friendly Rate Control (TFRC) [11], the Real-Time Transport Protocol (RTP) [12], the Microsoft Media Services (MMS), the Real Time Messaging Protocol (RTMP) [13]. Some of the mentioned streaming protocols have been employed in commercial products such as RealNetworks, Windows Media Player, Flash Player. Even though TCP has been initially considered as unsuitable for the transport of videos, recently it is getting a wider acceptance and it is being used with the HTTP. This is mainly due to the following reasons: i) HTTP-based streaming is cheaper to deploy since it employs standard HTTP servers [2]; ii) TCP has built-in NAT traversal functionalities; iii) it is easy to be deployed within Content Delivery Networks (CDN) [2]; iv) TCP delivers most part of the Internet traffic and it guarantees the stability of the network with its efficient congestion control algorithm [14], [15].

For what concerns video quality adaptation techniques, the proposed techniques can be classified into three main categories: 1) transcoding-based, 2) scalable encoding-based, 3) stream-switching (or multiple-bitrate - MBR).

The *transcoding-based* [16] approach consists in adapting the video content to match a specific bitrate by means of on-the-fly transcoding of the raw content. These algorithms can achieve a very fine granularity by throttling frame rate, compression, and video resolution. Nevertheless, this comes at the cost of increased processing load and poor scalability, due to the fact that transcoding has to be done on a per-client basis. Another important issue is that such algorithms are difficult to be deployed in CDNs.

An important class of adaptation algorithms employs *scalable codecs* such as H264/MPEG-4 SVC [17], [18]. Both spatial and temporal scalability can be exploited to adapt picture resolution and frame rate without having to re-encode the raw video content. With respect to transcoding-based approach, scalable codecs reduce processing costs since the raw video is encoded once and adapted on-the-fly by exploiting the scalability features of the encoder. However, this approach is difficult to be used with CDNs since the adaptation logic requires to be run on specialized servers and content cannot be cached in standard proxies. Another issue is that the adaptation logic depends on the employed codec, thus restricting the content provider to use only a limited set of codecs.

Stream-switching, or Multiple Bit-Rate (MBR) streaming, is gaining momentum through its adoption in leading streaming systems. Such algorithms encode the raw video content at increasing bitrates resulting into N versions of the video that form the discrete set $\mathscr{L} = \{l_0, \ldots, l_{N-1}\}$ of video levels; an algorithm automatically selects a video level $l_i \in \mathscr{L}$ that matches the user's available bandwidth; these algorithms minimize the processing costs since, once the video is encoded, no further processing is required in order to adapt the video to the variable bandwidth. Another important advantage of such algorithms is that they do not rely on particular functionalities of the employed codec, i.e. they are codec-agnostic. Moreover, the encoder can operate at the best efficiency for each of the video levels. Drawbacks of this approach are the increased storage requirements and the fact that adaptation is characterized by a coarser granularity since video bitrates can only belong to a discrete set of levels.

Among the streaming systems that employ the stream-switching technique we cite: 1) the *Microsoft IIS Smooth Streaming* [2]; 2) the *Adobe HTTP Dynamic Streaming* (HDS) [19]; 3) the *Apple HTTP Adaptive Live Streaming* (HLS) ; 4) *Move Networks* that provides live adaptive streaming service to several TV networks [20]; 5) *Netflix* that implements stream switching in its popular video on demand streaming platform [21]. With the aim of promoting the standardization and interoperability of stream switching systems, the MPEG Dynamic Adaptive Streaming over HTTP (MPEG-DASH) [3] has been recently standardized within the ISO to deliver multimedia content over the HTTP using existing infrastructures.

A great deal of attention has been also devoted to the performance evaluation of adaptive streaming algorithms in different scenarios.

In [4] an experimental evaluation of Microsoft IIS Smooth Streaming, Netflix, and Adobe OSMF has been carried out by testing the three adaptive algorithm under variable network available bandwidth. The paper has shown that the three considered algorithms implement client-side adaptation algorithms, which continuously request audio and video fragments at different bitrate via GET HTTP requests in order to match the changing available bandwidth. In particular, results have shown that the IIS Smooth streaming and the Netflix clients can be in two different phases: the *buffering phase* and the *steady phase*. During the buffering phase, the client requests new audio/video fragments right after the previous audio/video fragment has been downloaded in order to build up the player queue; during the steady phase, the player requests video segments every τ seconds (around 2s for IIS and around 3s for Netflix) in order to maintain the client buffer at a desired level. It has been shown that IIS Smooth Streaming buffers around 30s of video, whereas Netflix buffers up to 300s of video. In [9] it has been shown that the traffic pattern generated by such on/off fragments downloads makes the HTTP servers unable to fully utilize the network bandwidth. In [10], the impact of such on/off traffic pattern on the fairness in bandwidth utilization between two video flows is investigated.

In [5] different adaptive video streaming services have been tested in a mobile 3G network scenario. Results have shown that the Apple HTTP Dynamic Streaming was very conservative and less prone to change video level to match variable network conditions at the cost of a reduced bandwidth utilization. The Adobe Dynamic Streaming was quite responsive to network variability thus achieving a good bandwidth utilization even though frequent re-buffering events occur due

to short buffering at the client. Finally, Microsoft IIS Smooth Streaming achieved better quality *wrt* Apple HTTP Dynamic Streaming but suffering more re-buffering events.

In [6] Microsoft IIS Smooth Streaming, Adobe Dynamic Streaming, Apple HTTP Dynamic Streaming, and an implementation of DASH have been tested in a controlled testbed by using bandwidth traces obtained in a vehicular scenario. The results have shown that Microsoft IIS performed well minimizing the video level switches while maintaining the buffer at around 20s while avoiding re-buffering events. The Adobe Dynamic Streaming has shown unsatisfactory results with very frequent switches between the lowest and the highest video level and several re-buffering events. The Apple HTTP Dynamic streaming has exhibited a very conservative dynamics with very few video level switches and very large buffering (up to 200 seconds). Finally, the DASH implementation performed well, maintaining the buffer full and avoiding re-buffering events, but with more frequent switches *wrt* to Microsoft IIS [6].

Recently, bandwidth utilization, stability, and fairness issues have been addressed in major existing video streaming servers. In particular, in [7] it has been shown that NetFlix, Hulu, and Vudu are not able to estimate the actual network available bandwidth and, as consequence, to serve videos at the best possible quality. In [8] a suite of techniques is presented for trading off efficiency, fairness and stability.

III. The Automatic Stream-Switching Controller

In this section we propose a mathematical model of the automatic stream-switching controller of a major CDN operator aiming at describing the dynamic behavior of the controlled systems, i.e. the mathematical relation between inputs, such as measured bandwidth, playout buffer length and outputs, such as input rate, video level. In the following, we use a top-down approach for the description, i.e. we first describe the model and then validate it, because it revealed to be the most effective way to present the algorithm from the point of view of conciseness and clarity. The model has been the result of hypothesis based on a careful interpretation of experimental investigations. After the model has been derived, it has been validated through other experiments run in completely different scenarios. The results of the model validation are reported in Section V. The remainder of this section is organized as follows: in Section III-A we describe the overall control architecture showing the components of the systems and their interconnections; in Section III-B we propose the model of the playout buffer controller, whereas in Section III-C we propose the model of the stream-switching controller.

A. The overall control architecture

The primary goal of the automatic stream-switching controller is to dynamically select the video level $l_i \in \mathscr{L}$ in order to match the network available bandwidth $b(t)$, which is time-varying and unpredictable. It is reasonable to start by assuming that the system controllers are driven by the available bandwidth estimate $\hat{b}(t)$ and the client playout buffer duration $q(t)$ measured in seconds. In fact, this has been validated in our experiments by throttling the available bandwidth $b(t)$ using NetEm [22], which is a network emulation tool that allows the available bandwidth $b(t)$ to be set, in order to affect the controller decisions. In particular, the main investigation method has been to let the available bandwidth $b(t)$ to vary as a step function and evaluate key system response indices such as sending rate, video level switching, settling time, overshoots, and the time constants of the system dynamic response [23]. Using a step function to test the dynamic response of a system is a typical approach in control and identification [24].

Figure 1 shows the overall control and communication architecture of the automatic switching video streamer. The client and the server communicate over the Internet through the HTTP protocol; each video streaming session employs two sockets: the *control socket* (CS), which is used by the client to send commands to the server via `HTTP POST` requests to drive the stream-switching algorithm; the *data socket* (DS), which is used by the server to send the video stream to the client.

The client is made by the following entities:

1) the *player buffer* that stores the video frames and feeds the video player;
2) the *buffer controller* that, based on the player buffer queue length $q(t)$ expressed in seconds computes a control variable $T(t)$ that is sent to the buffer controller actuator at the server using a `HTTP POST` request;
3) the *stream switching controller* that, based on the estimated bandwidth $\hat{b}(t)$ and the queue length $q(t)$, computes a control variable $c(t)$ that is sent to the stream switching actuator at the server using a `HTTP POST` request.
4) a *measurement* module that supplies an array $\boldsymbol{F}(t)$ of measurements to the controllers. Among the values supplied by $\boldsymbol{F}(t)$ there are: the measured values of the buffer length $q(t)$, the received goodput $r(t-\tau_b)$, the estimated bandwidth $\hat{b}(t)$, and the decoded frame rate $f(t)$; all the components of $\boldsymbol{F}(t)$ will be examined in detail in Section IV-A2.

At the server, the following components are present:

1) The *stream-switching actuator* that, based on the control variable $c(t)$ received from the stream-switching controller at the client, selects the proper video level l_i;
2) the *buffer controller actuator* that, based on the controller variable $T(t)$ received from the buffer controller and the video level requested by the switching controller at the client, throttles the video streamer rate $\overline{r}(t)$.

Before going into a detailed description of the controllers, it is worth noting that the control loops between the client and the server are closed over the Internet, which is a time delayed communication channel characterized by a forward delay τ_f (from the client to the server) and a backward delay τ_b (from the server to the client).

In the following subsections we will present the models of the two control loops.

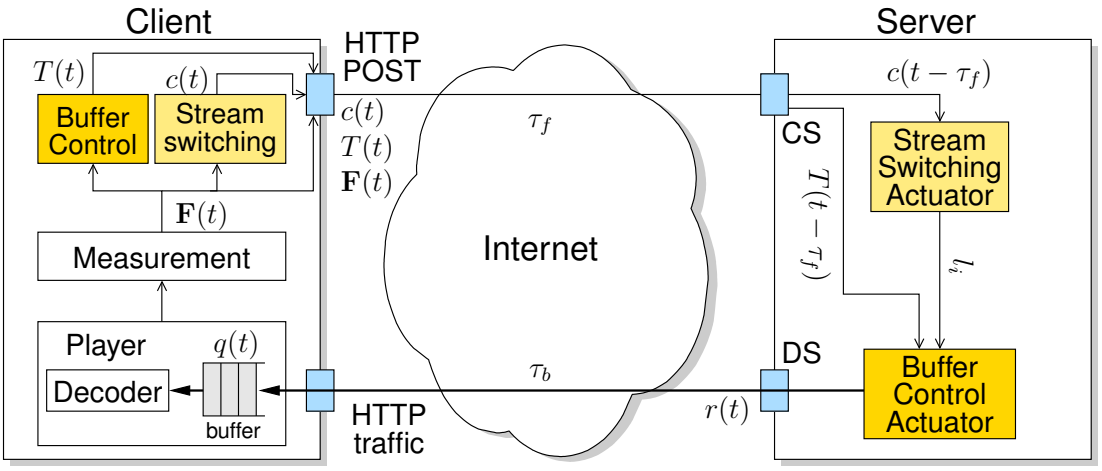


Fig. 1. The overall control and communication architecture of the automatic video level switching system: thick lines represent data flow, thin lines represent control signals

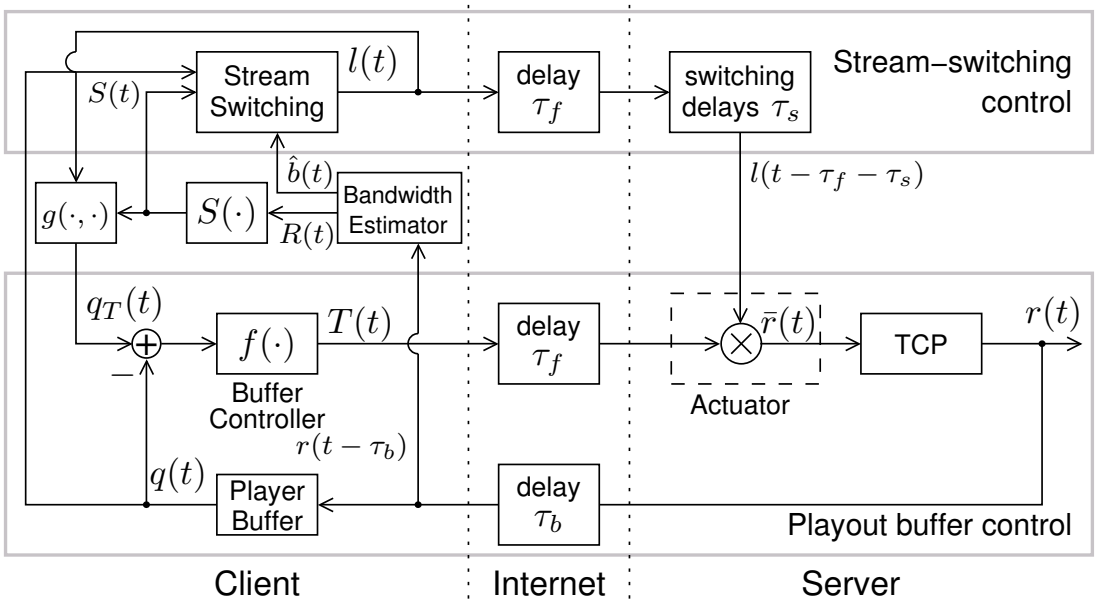


Fig. 2. Automatic video level switching system containing: 1) a playout buffer control; 2) a stream switching controller; 3) a bandwidth estimator

B. The playout buffer level controller and the bandwidth estimator

The aim of the *playout buffer level controller* is to drive the player buffer length $q(t)$ to a target length $q_T(t)$ in order to guarantee a continuous reproduction of the video. A block diagram of this control loop is shown in Figure 2.

The output of the nonlinear controller block $f(\cdot)$ shown in Figure 2 is the *throttle percentage* $T(t) \in [10, 500]$. It aims at steering to zero the difference between the target buffer length $q_T(t)$ and the buffer length $q(t)$.

We have observed that the buffer controller works in one the following three phases: 1) *re-buffering*; 2) *bandwidth probing*; 3) *normal*.

The *re-buffering* phase is triggered when the player buffer gets empty and lasts until a sufficient amount of video is stored in the playout buffer: during this phase the throttle percentage is set to 200.

During the *bandwidth probing* phase, $T(t)$ it is set to 500 and the client sends to the server a `rtt-test` command on average every 11s (see Section IV), to trigger a bandwidth probing phase which lasts, on average, 5s: during this short interval the received video rate reaches a peak which is close to the available bandwidth $b(t)$. At the end of this phase, the server sends to the client the RTT estimate $R(t)$. Thus, it can be argued that, when the throttle signal is 500, the video flow acts as a greedy flow. At the end of this phase, the received rate $r(t-\tau_b)$ is measured to get an estimate of the available bandwidth $\hat{b}(t)$, to be used as input to the stream switching controller. Moreover, we have found that, based on the round trip time estimate $R(t)$, the client computes a *safety factor*

$S(t)$ according to the following equations:

$$S(R(t)) = \begin{cases} 0.2 & 0 < R(t) < 0.02\text{s} \\ 2.5R(t) + 0.15 & 0.02 \text{ s} < R(t) < 0.1\text{s} \\ 0.4 & R(t) > 0.1\text{s} \end{cases} \quad (1)$$

Eq. (1) shows that, when the RTT is less than 20ms, the safety factor is set to 0.2, whereas when it is larger than 100ms, it is set to 0.4. For values between 20ms and 100ms the safety factor grows linearly. In the following we will use $S(t)$ instead of $S(R(t))$, for sake of conciseness. In Section III-C we will show how the safety factor is used by the stream-switching controller.

During the *normal* phase the throttling percentage is set as follows:

$$T_N(t) = \max\left((1 + \frac{q_T(t) - q(t)}{q_T(t)})100, 10\right) \quad (2)$$

The client sends the throttle percentage $T(t)$ to the server as an argument of the `throttle` command. In order to actuate the controller decision, the server produces a video rate $\overline{r}(t)$ that fills a sender buffer with the currently selected video level $l(t - \tau_f)$ according to the following equation:

$$\overline{r}(t) = l(t - \tau_f)\frac{T(t - \tau_f)}{100}. \quad (3)$$

The rationale of controlling $\overline{r}(t)$ is to steer $q(t)$ to $q_T(t)$. The block "TCP", shown in Figure 2, models the TCP congestion control dynamics [25]: briefly, the TCP sender rate $r(t)$ drains the video streamer buffer that is filled at rate $\overline{r}(t)$; a key feature of the TCP congestion control is that it generates a best effort traffic that matches the available bandwidth $b(t)$ when the buffer length $B(t) > 0$; in particular, the following relation holds:

$$r(t) = \begin{cases} \overline{r}(t) & \text{iff } \overline{r}(t) \leq b(t) \wedge B(t) = 0 \\ b(t) & \text{iff } \overline{r}(t) > b(t) \vee B(t) > 0 \end{cases} \quad (4)$$

Moreover, if we define the error as $e(t) = \overline{r}(t) - r(t)$, the integral error is always bounded by the maximum sender buffer size B_M:

$$\int_0^t e(\xi)d\xi = \int_0^t (\overline{r}(\xi) - r(\xi))d\xi = B(t) \leq B_M \quad (5)$$

Eq. (5) says that, on "*short time average*", $r(t)$ always matches $\overline{r}(t)$.

When the throttle percentage $T(t - \tau_f)$ is above 100%, the server streams the video at a rate that is greater than the current video level bitrate and the player buffer is filled. It is important to remark that, in the live streaming case, it is not possible for the server to supply a video at a rate that is greater than the encoding bitrate for a long period, since the video source is not pre-encoded.

By looking at (2), when the buffer length $q(t)$ matches the target buffer length $q_T(t)$, the throttle percentage $T(t)$ is equal to 100% and $\overline{r}(t)$ matches $l(t)$ (see (3)). On the other hand, when the error $q_T(t) - q(t)$ increases, $T(t)$ increases to allow $\overline{r}(t)$ to fill the buffer.

It is important to notice that the buffer control loop is coupled with the stream-switching control loop (see Figure 2) since the output $l(t)$ of the stream-switching algorithm, i.e. the selected video level, is an input through $g(\cdot, \cdot)$ to the buffer controller loop.

Figure 2 shows that the queue threshold $q_T(t)$ is function of the current video level $l(t)$ and the safety factor $S(t)$. In particular, the queue threshold is set as follows:

$$q_T(t) = g(S(t), l(t)) = q_T^S(S(t)) + q_T^0(l(t)) \quad (6)$$

where $q_T^S(S(t)) = 15(S(t) - 0.2)$ and $q_T^0(t)$ is function of the video level $l_i \in \mathscr{L}$:

$$q_T^0(l_i) = \begin{cases} 7 & i = 0, \ldots, 3 \\ 20 & i = 4 \end{cases}$$

The rationale of having $q_T^0(l_i)$ equal to 7s, when the video level is less than l_4, and equal to 20s, when equal to l_4, is that when an HD video (l_4) is streamed a larger amount of video is stored in the playout buffer in order to better cope with sudden bandwidth drops. Moreover, $q_T^S(S(t)) \in [0, 3]$s adds a contribute which is proportional to the safety factor $S(t)$, in order to buffer more video when the RTT of the connection is large.

A second threshold $q_L(t)$ is used by the stream-switching algorithm and is computed similarly to (6):

$$q_L(t) = h(S(t), l(t)) = q_L^S(S(t)) + q_L^0(l(t)) \quad (7)$$

where $q_L^S(S(t)) = 15(S(t) - 0.2)$ and $q_L^0(t)$ is function of the video level $l_i \in \mathscr{L}$:

$$q_L^0(l_i) = \begin{cases} 4 & i = 0, \ldots, 3 \\ 16 & i = 4 \end{cases}$$

In Section III-C we will describe how $q_L(t)$ is used by the stream-switching controller.

To the best of our knowledge, the considered adaptive streamer is the only one that controls the playout buffer length. In fact, client-side adaptive streaming systems such as[4], [7], [26] employ two phases, the *buffering phase*, during which the buffer is filled, and the *steady state,* during which the client aims at keeping the buffer length constant (see Section II). In [6] it has been shown that control strategies in [4], [7], [26] do not provide a stable playout buffer length.

C. The automatic stream-switching controller

The stream-switching controller aims at selecting the video level to be sent by the server to the client so that: 1) the best video quality is perceived given the current available bandwidth and 2) re-buffering at the player is avoided. The stream-switching controller is event-based, i.e. its decisions are triggered when particular events occur.

For each video level $l_i \in \mathscr{L}$ a *high threshold* L_i^H and a *low threshold* L_i^L are maintained:

$$L_i^H(t) = l_i \cdot (1 + S(t)) \; ; \; L_i^L = l_i \cdot 1.2 \quad (8)$$

A switch up command (`SWITCH_UP`) is triggered when $\hat{b}(t) > L_i^H(t)$. This means that, in order to switch up to level l_i, the estimated bandwidth must cross the $L_i^H(t)$ curve from below. It is worth noting that $L_i^H(t)$ is between 1.2 and 1.4

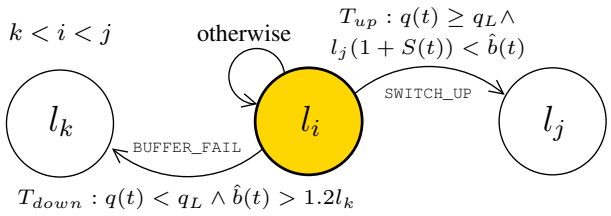


Fig. 3. Finite state machine modelling the stream-switching algorithm: l_i is the current level played by the client; if T_{up} (T_{down}) is true the level l_j (l_k) is selected by sending the SWITCH_UP (BUFFER_FAIL) command to the server.

times l_i depending on $R(t)$. This conservative approach may lead to network underutilization and reduced QoE.

A switch down command (BUFFER_FAILURE) is sent when $q(t) < q_L(t)$ occurs, where $q_L(t)$ is the threshold (7) computed as function of the safety factor $S(t)$ and the video level l_i. The BUFFER_FAILURE command triggers the selection of the highest video level l_i such that $L_i^L < \hat{b}(t)$. In other terms, in order to select a level l_i, the currently estimated bandwidth $\hat{b}(t)$ must be at least equal to $1.2l_i$.

Let us now consider the case of a re-buffering phase that occurs when the playout buffer becomes empty: assuming that the average available bandwidth does not further decrease during the re-buffering phase, it will take less than $q_T(t)$ seconds to reach again the queue target $q_T(t)$ regardless of the value of the chosen level l_i. In order to prove this result, we recall that, when a re-buffering event occurs at time $\bar{t}$, the video gets paused and the video level is set to $l_i < \hat{b}(\bar{t})/1.2$, according to the estimated available bandwidth $\hat{b}(\bar{t})$, and it is kept constant during the re-buffering phase. In order to buffer q_T seconds of video, it is necessary to download $q_T \cdot l_i$ bytes. The time required to download such an amount of data is equal to $T_{rb} = q_T l_i / \hat{b}$. Thus, to have $T_{rb} \leq q_T$ it is sufficient that $l_i \leq \hat{b}$, which under setting $l_i < \hat{b}/1.2$ is always satisfied. This result is valid regardless of the value of l_i.

Figure 3 shows the finite state machine which models the automatic stream-switching algorithm executed at the client. The current video level l_i is maintained until one of the two transition conditions, or events, occur:

$$T_{up} : q(t) \geq q_L \wedge l_j(1 + S(t)) < \hat{b}(t) \quad (9)$$

$$T_{down} : q(t) < q_L \wedge \hat{b}(t) > 1.2l_k \quad (10)$$

Moreover, we have found that the server actuates a SWITCH_UP command after a median switch-up delay τ_{su} of around 6.5s and a BUFFER_FAIL command with a median switch-down delay τ_{sd} of around 10.5s. These actuation delays are shown in Figure 2 through the block "switching delays".

It is worth noting that these delays τ_s are at least one order of magnitude larger than the round trip delays $\tau_f + \tau_b$, which makes the latter negligible in the loop dynamics that is dominated by τ_s [23], [15], [27].

We conclude this section by briefly summarizing the effects of time-varying network conditions (the available bandwidth $b(t)$ and the RTT) on the stream-switching algorithm. The variable that mainly affects the adaptation algorithm is the estimated available bandwidth $\hat{b}(t)$. In fact, depending on the value of $\hat{b}(t)$, the stream-switching controller selects the video level l_i to be sent to the client (see Figure 3). Regarding the effect of the connection RTT, it can be noted that the larger the measured RTT $R(t)$, the larger the safety factor $S(t)$ is, which makes the stream switching algorithm more conservative, i.e. less prone to switch-up (see equation (8) in Section III). Moreover, since the video stream is sent over a TCP connection, the connection RTT has an influence on the obtainable throughput [28].

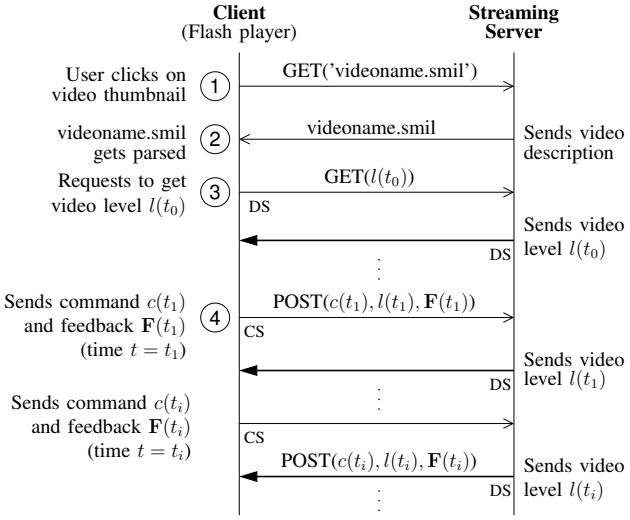


Fig. 4. Client-server time sequence graph: thick lines indicate video data transfer, thin lines represent HTTP requests sent from client to server

IV. The Client-Server Communication and Control Protocol

A. The client-server communication and control protocol

With the purpose of analyzing the client-server communication and control protocol we have carried out an experiment in which the client is connected to the Internet through our campus wired connection and receives the video sequence "*Elephant's Dream*" from the demo server. By analyzing the dump file of the traffic received by the client, we were able to reconstruct the time sequence graph shown in Figure 4.

At first, the client connects to the server through a web browser, then a Flash application is loaded and a number of videos are made available to the client. When the user clicks on the thumbnail of the video he is willing to play (point ① in Figure 4), a GET HTTP request is sent to the server pointing to a SMIL compliant file [29] (an excerpt of this file is shown in Figure 5). The SMIL file provides the base URL of the video (httpBase), the video levels filenames, and the corresponding encoding bit-rates (system-bitrate). By looking at the SMIL file, it can be seen that the videos available on the demo website are encoded at five different bitrates $l_i \in \{l_0, \ldots, l_4\}$ at any given time instant t. Table I shows the video resolution for each of the five video levels l_i, that ranges from 320×180 up to high definition 1280×720.

Then, the client parses the SMIL file (point ② in Figure 4) so that it can reconstruct the complete URLs of the available video levels and it can request the corresponding video level

```
<head>
 <meta name="title" content="Elephants Dream" />
 <meta name="httpBase" content=
"http://baseurl"/>
 <meta name="rtmpAuthBase" content="" />
</head>
<body>
 <switch id="Elephants Dream">
  <video src="ElephantsDream2_h264_3500@14411"
system-bitrate="3500000"/>
  <video src="ElephantsDream2_h264_2500@14411"
system-bitrate="2500000"/>
  <video src="ElephantsDream2_h264_1500@14411"
system-bitrate="1500000"/>
  <video src="ElephantsDream2_h264_700@14411"
system-bitrate="700000"/>
  <video src="ElephantsDream2_h264_300@14411"
system-bitrate="300000"/>
 </switch>
</body>
```

Fig. 5. Excerpt of the SMIL file

TABLE I
THE SET OF AVAILABLE VIDEO LEVELS $\mathscr{L}$

Video level	Bitrate (kb/s)	Resolution (width×height)
l_0	300	320x180
l_1	700	640x360
l_2	1500	640x360
l_3	2500	1280x720
l_4	3500	1280x720

based on the stream-switching algorithm.We were able to individually download each video level by using standard HTTP GET requests pointing to the URLs reconstructed using the SMIL file. This suggests that the server does not segment the video as in the case of the Apple HTTP adaptive streaming, but it encodes the original raw video source into N different files, one for each available level. This is the same approach used today by Microsoft IIS which logically partitions a video level in segments which are physically stored into a single file. Using one physical file for each video level, instead of having many files such as in the case of Apple HLS, reduces the number of objects that a CDN have to handle.

Furthermore, we have analyzed the downloaded files and we have found out that all the video levels are encoded at 30 frames per second (fps) using the H.264 codec with a FLV container and a group of picture (GOP) length equal to 36, which means that two consecutive I frames are 1.2 s apart. This also implies that, since a video switch can occur only at the GOP boundaries, the video level can at most be changed every 1.2 s. It is worth mentioning that a stream switch can also occur between video levels characterized by different video resolutions. The audio is encoded with Advanced Audio Coding (AAC) at 128 kb/s bitrate.

After the SMIL file has been parsed, at time $t = t_0$ (point ③ in Figure 4) the client opens a TCP socket, which is the *data socket* (DS), and sends a HTTP GET request that specifies in the URL one of the video level filenames contained in the SMIL file. The server starts to stream the video level specified in the GET request on the data socket. By analyzing the dump-file, it can be noticed that this socket is kept open until the reproduction of the video is stopped. This indicates that, when a stream-switch occurs, the new video level is sent to the client using the same socket.

POST /control/levelFname?**cmd**=c_i,$a_1,\ldots,a_n$ &**v**=1.0.7.23&**r**=ABCDE&**g**=token&**lvl1**=$F_1,\ldots,F_{12}$

Fig. 6. HTTP POST requests from the client to the server over the control socket

TABLE II
COMMANDS SENT BY THE CLIENT TO THE STREAMING SERVER VIA CMD, ARGUMENTS ARRAY $\boldsymbol{a}$, AND OCCURRENCE PERCENTAGE (× INDICATES AN UNIDENTIFIED ARGUMENT)

cmd	**arguments array - $\boldsymbol{a}$**	**Occ.(%)**
throttle	$T(t)$	~80%
rtt-test		~15%
SWITCH_UP	$\hat{b}(t), l_k, 1+S(t), k$, oldLevelFname	~2%
BUFFER_FAIL	$\times, \times, k, l_k, \hat{b}(t), 1+S(t)$, newLevelFname	~2%
log	$\times, \times$	<1%

At time $t = t_1$ (point ④ in Figure 4) the client opens a second socket, the *control socket* (CS), and it sends a POST request to the server specifying five parameters (see Figure 6).

The first POST parameter is cmd and specifies a command c_i containing an array $\boldsymbol{a}$ of n comma separated arguments $a_1,\ldots,a_n$; the parameter v specifies the HDCore Library of the client; the parameter r is a 5 letters string that seems to be encrypted; the parameter g is constant throughout all the connection and is a session identifier; the parameter lvl1 is a comma separated array $\boldsymbol{F}$ of feedback variables that will be described later.

At time $t = t_1$ (point ④ in Figure 4), the quality adaptation algorithm starts and at a generic time instant $t_i > t_1$, the client issues commands to the server via HTTP POST requests in order to drive the automatic stream-switching algorithm discussed in Section III-C.

Thus, the considered system employs a different approach with respect to other adaptive video streaming systems. Typically, when a player wants to switch the video level to follow a changing available bandwidth, it sends a new HTTP GET request specifying 1) the new video level URL, 2) the timestamp pointing to the frame in the stream starting from which the server should send the new video level [6], [4], [7].

Differently from this approach, the system considered in this paper, sends a HTTP GET request only at the beginning of the video session, whereas it sends HTTP POST requests (SWITCH_UP/BUFFER_FAIL commands) in order to change the video level to the server through the control socket. In the following, we will discuss the commands and the feedback variable sent by the client to the server.

1) The cmd parameter: By parsing all the HTTP POST requests sent by the client to the server through the control socket, we have identified five different commands c_i, each one with a different number of arguments. Table II reports all the command names along with the arguments array, the HTTP reply code sent back from the server to the client, and the occurrence percentage.

The first two commands, i.e. throttle and rtt-test, are issued periodically, whereas the remaining three are event-based. The periodicity of throttle and rtt-test com-

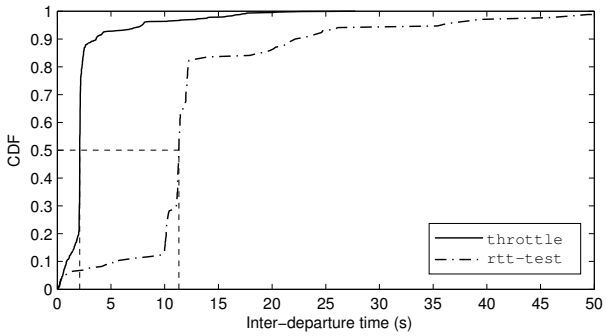


Fig. 7. CDF of the inter-departure time between two consecutive `throttle` or `rtt-test` commands

mands can be inferred by looking at Figure 7, which shows the cumulative distribution functions (CDF) of the inter-departure times of two consecutive `throttle` or `rtt-test` commands. It is worth noticing that the upper tails of both the commands are due to the re-buffering phases: in fact, during this phase, the `rtt-test` command is inhibited, whereas the `throttle` command is sent at a lower frequency. Figure 7 shows that the `throttle` command is issued with a median inter-departure time of about 2 seconds, whereas the `rtt-test` command is issued with a median inter-departure time of about 11 seconds.

The `throttle` command is the most frequently issued (80% of occurrence) and specifies a single argument, i.e. the *throttle percentage* $T(t)$, which is used to control the received buffer length and estimate the available bandwidth $\hat{b}(t)$ as we have already described in Section III-B.

The `rtt-test` command is issued by the client to the server to ask for sending video in greedy mode and to measure the round trip time (RTT) under loaded network. When this command is received, the server sends back to the client a HTTP reply with the measured RTT in milliseconds.

The `SWITCH_UP` command is issued by the client to trigger the switching from the current video level l_j to a video level $l_k > l_j$. Five parameters are supplied to the server: 1) the estimated bandwidth $\hat{b}(t)$; 2) the bitrate l_k of the video level that the client wants to receive; 3) $1 + S(t)$, where $S(t)$ is the safety factor; 4) the identifier k of the video level that the client wants to receive; 4) the filename of the video level l_j that is currently playing.

The `BUFFER_FAIL` command is issued by the client to the server to switch from the current video level l_j to a video level $l_k < l_j$. We have identified five out of the seven parameters supplied with this command: 1) the identifier k of the video level that the client wants to receive; 2) the bitrate l_k of the video level the client wants to receive; 3) the estimated bandwidth $\hat{b}(t)$; 4) $1 + S(t)$, where $S(t)$ is the safety factor; 5) the filename of the video level l_k that is requesting to switch to.

The last command is `log`, it takes two arguments and it is used to log anomalous events occurred during the video streaming.

2) The `lvl1` parameter: The `lvl1` parameter is an array $\boldsymbol{F}$ made of 12 feedback variables $F_1, \ldots, F_{12}$ separated by commas. We have identified 10 out of the 12 variables as follows:

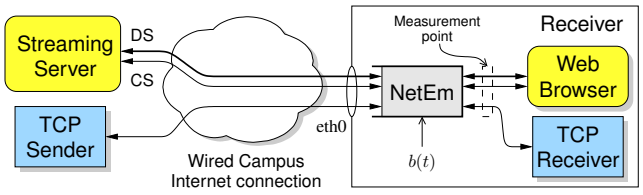


Fig. 8. The testbed employed for the model validation

- F_1 - **Playoutbuffer length** $q(t)$: it represents the number of seconds stored in the playout buffer;
- F_2 - **Playout buffer target** $q_T(t)$: it represents the desired size of the playout buffer size measured in seconds;
- F_3 - unidentified parameter;
- F_4 - **Received video frame rate** $f(t)$: it is the decoded video frame rate, measured in frames per second;
- F_5 - unidentified parameter;
- F_6 - **Estimated RTT** $R(t)$: the estimated RTT measured in ms after an `rtt-test` command is sent;
- F_7 - **Estimated bandwidth** $\hat{b}(t)$: it is measured in kb/s;
- F_8 - **Received goodput** $r(t)$: it is the received rate (kb/s) measured at the client;
- F_9 - **Current video level identifier**: it represents the identifier of the video level that is currently received by the client. It belongs to the set $\{0, 1, 2, 3, 4\}$.
  - F_{10} - **Current video level bitrate** $l(t)$: it is the video level bitrate measured in kb/s that is currently received by the client. It belongs to the set $\mathscr{L} = \{l_0, l_1, l_2, l_3, l_4\}$ (see Table I).
- F_{11} - **Playing time**: it represent the playing time expressed in seconds.
- F_{12} - **Timestamp** t_i: it represents the Unix timestamp of the client.

V. Model Validation and Performance Evaluation

In this section we validate the model presented in Section III by employing the bandwidth-controlled testbed described in Section V-A. Moreover, we evaluate to what extent the stream-switching algorithm is able to maintain a continuous reproduction when the bandwidth suddenly drops or when the bottleneck is shared with TCP flows.

A. The controlled testbed

Figure 8 shows the *bandwidth-controlled testbed scenario* that we have built to infer and validate the proposed model.

The receiving host is an Ubuntu Linux machine connected to the Internet via our campus wired connection. The receiver is equipped with NetEm, a Linux kernel module that, along with the traffic control tools available on Linux, allows downlink channel bandwidth $b(t)$ and delays to be set [22]. In order to perform traffic shaping on the downlink, we have used the Intermediate Functional Block pseudo-device IFB[1]. The receiving host also runs an `iperf` server (TCP Receiver) in

[1]http://www.linuxfoundation.org/collaborate/workgroups/networking/ifb

order to receive TCP greedy flows sent by a remote `iperf` client (TCP Sender). We use `tcpdump`[2] to capture all the ingress traffic at the receiver and we use a python script to post-process the dumpfile and log: 1) all the commands $c(t)$ and the feedback array $\boldsymbol{F}(t)$ sent by the client to the server through the control socket (CS); 2) the rate of the video streaming flow received over the data socket (DS).

The server provides a number of videos through a demo website that can be accessed using a web browser. We have used the video sequence "*Elephant's Dream*" for the experiments, since its duration allows a complete performance evaluation.

With the aim of validating the model proposed in Section III, we employ step-like downlink available bandwidth increases and decreases and we evaluate: 1) the transient times required by the video switching algorithm to match the time-varying available bandwidth with the highest possible video level, 2) the ability to avoid playout interruptions when a sudden decrease of the available bandwidth occurs, 3) the efficiency of the algorithm in terms of bandwidth utilization, 4) the fairness when sharing the bottleneck with one concurrent TCP connection.

We have considered the following four experiments: 1) one video flow over a bottleneck link whose bandwidth capacity changes following a step function with minimum value 400 kb/s and maximum value 4000 kb/s; 2) one video flow over a bottleneck link whose bandwidth capacity varies as a square wave with a period of 200 s, a minimum value of 400 kb/s and a maximum value of 4000 kb/s; 3) one video flow sharing a bottleneck, whose capacity is fixed to 4000 kb/s, with one concurrent TCP flow; 4) one video flow over a bottleneck link whose bandwidth capacity drops from a value $A_M^{(i)} \in \{1000, 2000, 3200, 5000\}$kb/s to a value $A_m^{(j)} \in \{500, 1000, 2000, 3200\}$kb/s, with $A_M^{(i)} > A_m^{(j)}$.

It is worth to mention that, before running each experiment, we have carefully checked that the end-to-end available bandwidth was well above 5000 kb/s, which was the maximum value of the bandwidth we set in the traffic shaper. The measured RTT between our client and the server was in the order of 10 ms. Finally, in order to obtain a careful validation of the model, we have repeated the experiments for each of the considered scenarios 50 times unless otherwise specified. The time-behaviours reported in this section are the typical ones.

B. Efficiency indices

For what concerns the evaluation of the Quality of Experience (QoE), we note that, even though metrics based on PSNR are still widely used in the literature, in many cases they are not well correlated with subjective quality measures such as the Mean Opinion Score (MOS) [30]. Even though several objective video quality metrics have been proposed to assess the QoE, the design of a video quality metric which is suitable for any video application is still an open issue and, to the best of authors knowledge, it has not been proposed to assess the QoE of adaptive HTTP video streaming. In this paper we propose to evaluate the following three efficiency indices:

1) the *network utilization index* $\eta_n = \hat{l}/C \in [0, 1]$, which assesses to what extent the stream-switching algorithm is able to use a known available bandwidth b, where $\hat{l}$ is the average value of the received video level $l(t)$ and $C = \min(l_M, b)$ where l_M is the maximum video level; the index η_n is 1 when the average value of the received video level is equal to C, i.e. when the video level exactly matches the network available bandwidth.
2) the *video continuity index* $\eta_c \in [0, 1]$, or *complementary re-buffering ratio*, which measures to what extent the controller is able to avoid re-buffering pauses and it is defined as follows:
$$\eta_c = 1 - \frac{\Delta T_{rb}}{\Delta T} \tag{11}$$
where ΔT_{rb} is the total time the client remains paused due to re-buffering events and ΔT is the duration of the experiment. This efficiency index is 1 when no re-buffering pauses are experienced by the client. It is important to notice that ΔT_{rb} does not include start-up time. In [31], [32] it has been shown that the re-buffering ratio $\Delta T_{rb}/\Delta T$ is an important factor impairing user engagement.
3) the transient time required for the video level $l(t)$ to match the available bandwidth $b(t)$ when a step like variation of $b(t)$ occurs.

C. Model validation

This section aims at validating: 1) the model of the stream-switching proposed in Section III; 2) the controller dynamics of the received video level $l(t)$ and of the receiver buffer length $q(t)$; 3) the explanation of the commands that we have described in Section IV.

To the purpose, the available bandwidth follows a step-like change. In particular, $b(t)$ suddenly increases at $t = 50$s from a value of $A_m = 400$ kb/s to a value of $A_M = 4000$ kb/s. By letting the step vary from 400kb/s $> l_0$ to 4000kb/s $> l_4$, we are able to excite the complete dynamics of the controller. Since for $t > 50$s the available bandwidth is greater than the maximum video level l_4, we expect a steady state video level equal to l_4.

1) Validation of the stream-switching logic: We first focus on the effects of the commands issued by the client to the server in order to drive the stream-switching algorithm described in Section III-C.

Figure 9 shows the dynamics of the video level $l(t)$ and the estimated bandwidth $\hat{b}(t)$ as reported by the `lvl1` parameter via the HTTP POST requests (see Section IV-A2). In order to show the effects of `BUFFER_FAIL` and `SWITCH_UP` commands on $l(t)$, Figure 9 also reports the time instants when these commands are issued.

The figure shows that, every time the client sends a `SWITCH_UP` command, the video level $l(t)$ increases after a *switch-up delay* τ_{su} , whereas, when `BUFFER_FAIL` commands are sent, $l(t)$ decreases after a *switch-down delay* τ_{sd}. We have collected τ_{su} and τ_{sd} and we have found that the

[2]http://www.tcpdump.org/

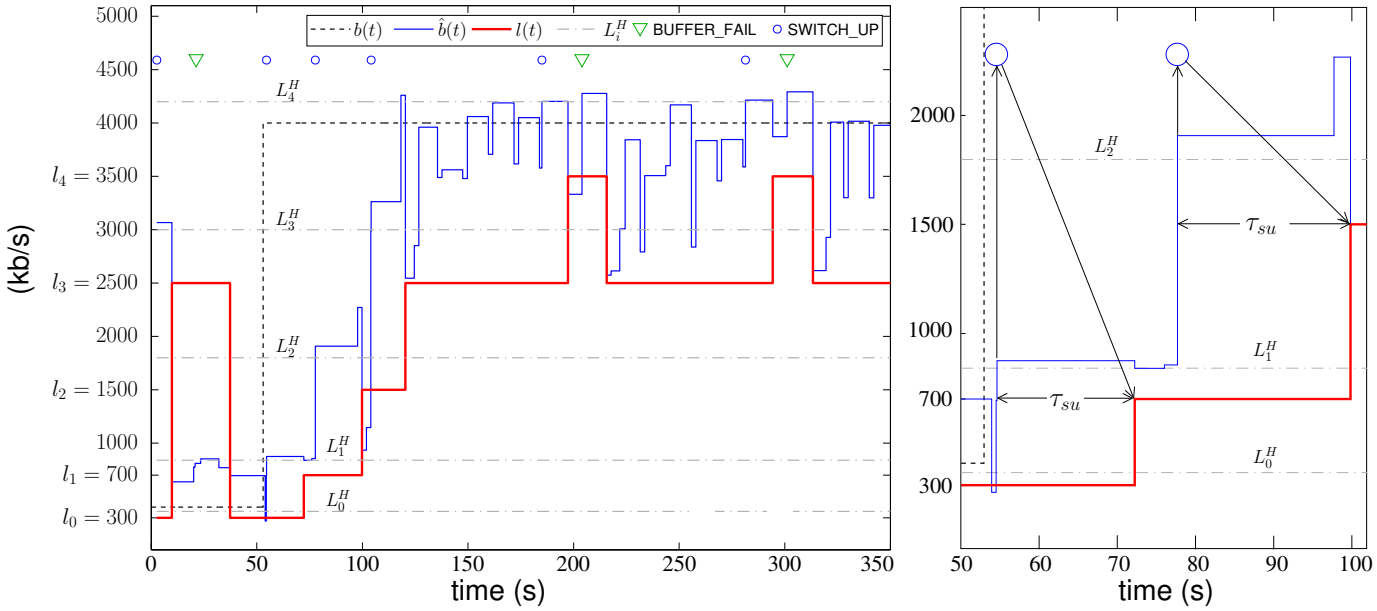


Fig. 9. Step variation of the available bandwidth $b(t)$, estimated bandwidth $\hat{b}(t)$, received video level $l(t)$, BUFFER_FAIL/SWITCH_UP events, and switch-up thresholds L_i^H ; a zoom of the time interval $t \in [50, 100]$ shows the τ_{su} delay

average value of switch-up delays is $\tau_{su} \simeq 14$s, whereas the average switch-down delay is $\tau_{sd} \simeq 7$s. This confirms that the stream-switching logic is distributed in part at the client, which executes the stream-switching control by means of SWITCH_UP and BUFFER_FAIL commands, and in part at the server, which receives and actuates those commands after a switching delay. We conjecture that the commands are executed only if their trigger conditions hold for the time τ_{su} and τ_{sd} respectively. This aims at avoiding persistent switching between video levels (*chattering*) which has been shown to negatively affect the quality of experience [33].

By parsing the debug information provided by the client, we have found that, every time a rtt-test command is sent and a new value of $R(t)$ is reported by the server, the client updates the safety factor $S(t)$ according to (1). With the aim of validating the safety factor equation (1), Figure 11 shows the measured pairs $(R(t), S(t))$ drawn as black dots and the static model $S(R(t))$ obtained by using equation (1). Figure 10 (d) shows the round trip time $R(t)$ estimated after a rtt-test command is sent. This value is used to compute the thresholds $L_i^H(t)$ that are shown in Figure 9.

In order to validate the stream-switching model discussed in Section III-C, let us look at the zoom in the time interval $[50, 100]$s shown in Figure 9: in addition to the dynamics of the video level $l(t)$ and the estimated bandwidth $\hat{b}(t)$, the figure shows, in dash-dotted lines, the high thresholds $L_i^H(t)$ computed using (8). It can be seen that, after the available bandwidth increases at $t = 50$s, the estimated bandwidth $\hat{b}(t)$ increases to 875 kb/s and, at $t = 54.61$s, it crosses from below the threshold $L_1^H = 840$ kb/s. Since $\hat{b}(t) > L_1^H(t)$, a switch-up event, marked with a $\circ$ in Figure 9, is triggered and a SWITCH_UP command is immediately sent at $t = 54.62$s. The video level is actually increased by the server at $t = 72.25$s, i.e. after a switch-up delay of 17.63 s. The same behaviour can be observed in Figure 9 every time the estimated available bandwidth $\hat{b}(t)$ surpasses a threshold L_i^H.

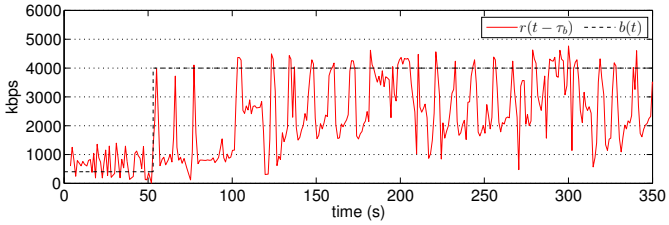
2) Validation of the playout buffer control model: Figure 10 (a) and (b) show the received goodput $r(t)$ and the throttle percentage signal $T(t)$ respectively, along with the time instants, marked with a $\times$, at which rtt-test commands are issued. Figure 10 (b) clearly shows that every time a rtt-test command is sent, the throttle percentage $T(t)$ is set to 500% for around 5s: during this short interval the received rate $r(t)$ reaches a peak which is close to the available bandwidth $b(t)$. The time behaviour of the received video rate confirms that, when the throttle signal is 500%, the video flow acts as a greedy flow.

Figure 12 shows the measured received video rate $r(t)$ and the sender buffer filling rate $\bar{r}(t)$ given by (3). The figure shows that the error between $\bar{r}(t)$ and $r(t)$ is bounded as dictated by (5).

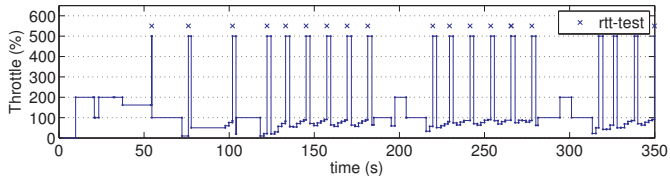
Figure 13 compares the measured throttle signal with the time behaviour obtained by using the conjectured control law (2). Eq. (2) fits very well the measured throttle signal: in fact, the difference is only in correspondence of the rtt-test commands, when the throttle is equal to 500, or re-buffering phases when the throttle is equal to 200.

Finally, Figure 10 (c) shows that the playout buffer controller is able to drive the playout buffer length $q(t)$ to the target $q_T(t)$.

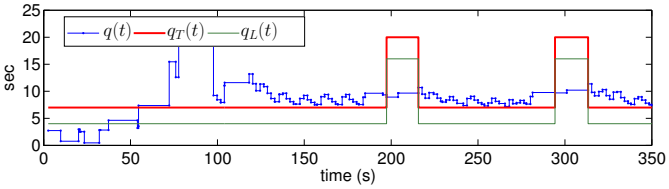
3) Investigation of the dynamic behaviour of the stream-switching algorithm: Figure 9 shows that the video level is initialized at the lowest available video level l_0. Nevertheless, at time $t = 0$ the estimated bandwidth $\hat{b}(t)$ is erroneously estimated at the value of 3067 kb/s which is above the threshold $L_3^H = 3000$ kb/s. Thus, a SWITCH_UP command is sent to the server and the video level $l_3 = 2500$ kb/s is streamed after a switch-up delay of 7.16s. By sending the video level l_3, which is above the channel bandwidth $A_m = 500$ kb/s, the



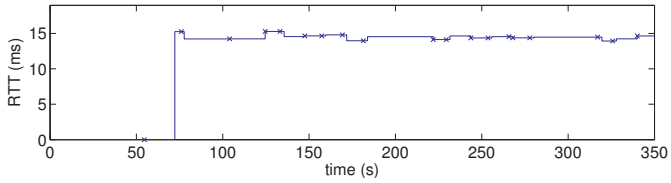
(a) Received video rate



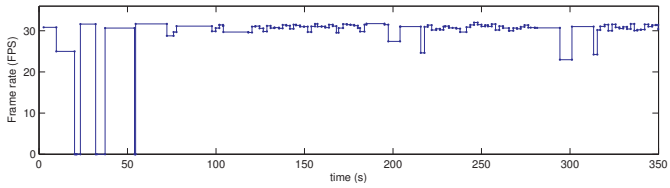
(b) Throttle $T(t)$ and `rtt-test` commands



(c) Playout buffer length $q(t)$ and target buffer length $q_T(t)$



(d) Round trip time $R(t)$ estimated via the `rtt-test` command



(e) Frame rate $f(t)$

Fig. 10. Response to a step change of available bandwidth at $t = 50$s

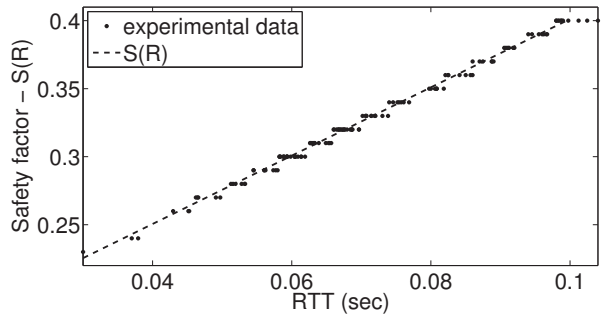


Fig. 11. Safety factor $S(t)$ vs round trip time $R(t)$

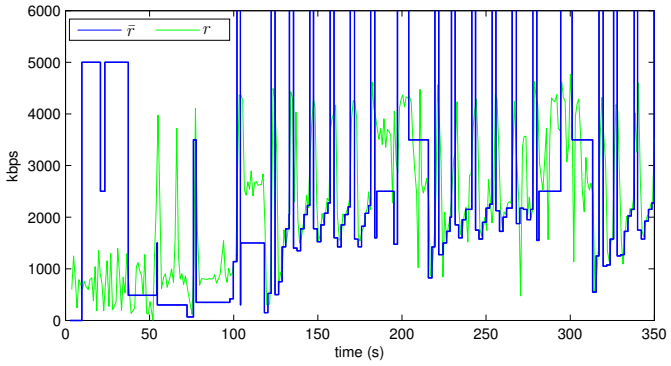


Fig. 12. Sender buffer filling rate $\bar{r}$ and received video rate r dynamics when the available bandwidth varies as a step function

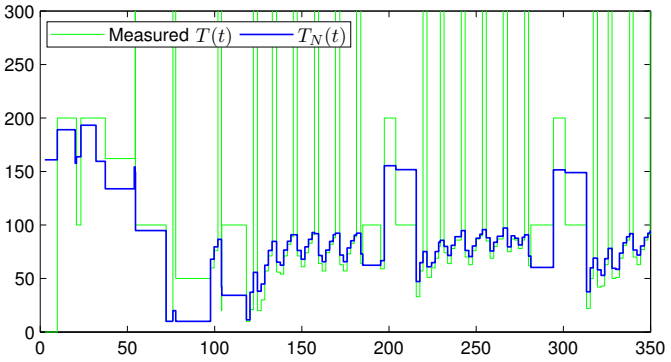


Fig. 13. Measured throttle signal $T(t)$ compared to the conjectured control (2)

received buffer length $q(t)$ starts to decrease and it eventually gets empty at $t = 17.5$s. Figure 10 (e) shows that the playback frame rate decreases to zero, i.e. the video is paused, in the time interval $[17.5, 20.8]$s. At time $t = 18.32$s, a `BUFFER_FAIL` command is finally sent to the server. After a delay of about 16s , the server switches down to the video level $l_0 = 300$ kb/s that is below the available bandwidth A_m.

Figure 10 (e) shows that the adaptation algorithm does not change the frame rate to adapt the video content to the network available bandwidth.

Let us now look at the dynamics of the estimated bandwidth $\hat{b}(t)$ shown in Figure 9. When the bottleneck capacity increases to $A_M = 4000$ kb/s, $\hat{b}(t)$ slowly increases and, after a transient time of 75s, it correctly estimates the bottleneck capacity A_M. This large transient can be explained by considering the stream switching algorithm and the playout buffer controller described in Section III. In fact, a `SWITCH_UP` to a level l_i can occur only if $\hat{b}(t) > (1 + S(t))l_i = L_i^H$. When the available bandwidth increases from 400 kb/s to 4000 kb/s the video level is $l_0 = 500$kb/s. Then, at $t = 51$s, a `rtt-test` event is triggered and consequently the throttling percentage is set to 500% to enter the bandwidth probing phase during which the sender buffer filling rate is set to $\bar{r}(t) = 500/100 l_0 = 5l_0 = 1500$kb/s according to (3). This means that the sender buffer is filled at 1500 kb/s so that the sending rate cannot exceed this rate on average. Being the sending rate at 1500kb/s, it is not possible to estimate an available bandwidth greater than 1500kb/s and, as a consequence, the video level cannot quickly

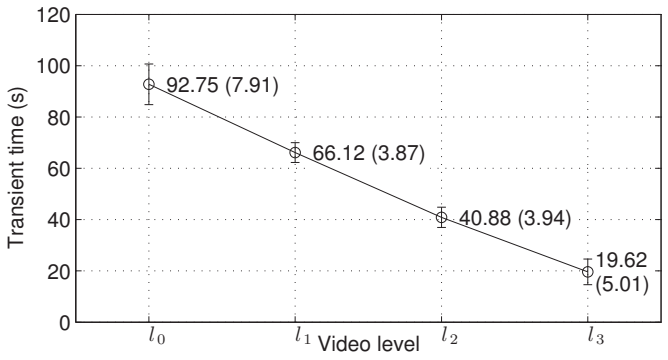


Fig. 14. Average transient time $E[T_{i\to 4}]$ and standard deviation $\sigma_{T_{i\to 4}}$ (shown in parentheses) of the transient time $T_{i\to 4}$ required to switch from level l_i to l_4

increase to the maximum level. Thus, due to the large transient time of $\hat{b}(t)$ and to the switch-up delay τ_{su}, the transient time required for $l(t)$ to reach the maximum video level l_4 is around 150s. The safety factor $S(t) \in [0.2, 0.4]$ also adds further conservativeness to the algorithm since, in order to switch up to level l_i, the estimated bandwidth must be greater than $(1 + S(t))l_i$.

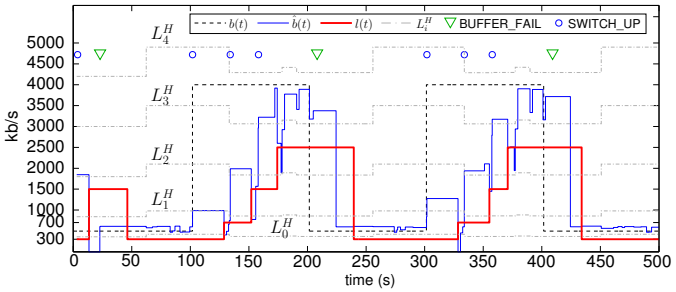
Finally, in order to assess the performance of the quality adaptation algorithm, we have measured the network efficiency $\eta_n = 0.676$ and the video continuity efficiency $\eta_c = 0.91$.

4) Transient times: To conclude this section we investigate the transient time $T_{i\to 4}$ required to switch from a video level l_i, with $i = 0, 1, 2, 3$, to the maximum video level l_4. With this purpose, we let the bottleneck available bandwidth $b(t)$ to change as a step function with an initial value $A_m^{(i)}$ belonging to the set $\mathscr{A} = \{500, 1000, 2000, 3200\}$kb/s and a final value $A_M = 5000$ kb/s, i.e. $b^{(i)}(t) = A_m^{(i)} \cdot 1(t) + (A_M - A_m^{(i)}) \cdot 1(t - 100)$ for $i = 0, 1, 2, 3$, where $1(t)$ is the unitary step function which is 0 for $t < 0$ and 1 for $t \geq 0$. For each of the four considered available bandwidths $b^{(i)}(t)$, we have carried out ten experiments and we have measured the average value $E[T_{i\to 4}]$ and the standard deviation $\sigma_{T_{i\to 4}}$ that are shown in Figure 14. The figure clearly shows that the average transient time decreases roughly linearly when the required number of levels to switch-up decreases.

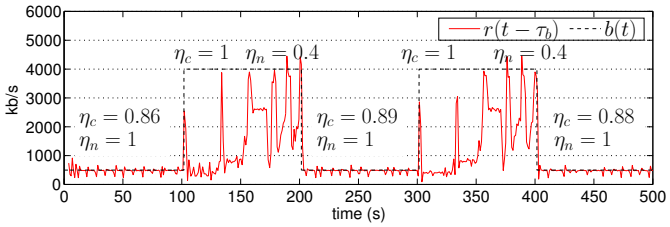
D. Investigation of the dynamic behaviour of the stream switching in the presence of a square-wave time-varying bottleneck capacity

In this experiment the bottleneck available bandwidth $b(t)$ is set to a square-wave function with a period of 200s, a minimum value $A_m = 500$ kb/s and a maximum value $A_M = 4000$ kb/s. The aim of this experiment is to assess the responsiveness of the stream-switching algorithm in shrinking the video level $l(t)$ in response to a significant drop of the available bandwidth while guaranteeing a continuous reproduction of the video content.

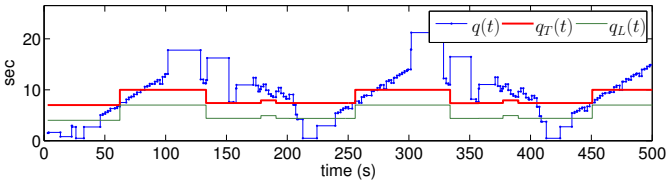
By looking at the time behaviour of the video level $l(t)$ shown in Figure 15(a), two main facts can be noticed: 1) when the available bandwidth increases to A_M, the video level is increased to l_3, which is less than the maximum video



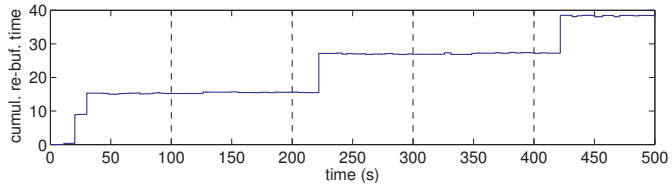
(a) Available bandwidth $b(t)$, estimated bandwidth $\hat{b}(t)$, video level $l(t)$, BUFFER_FAILURE/SWITCH_UP events, and high thresholds L_i^H



(b) Received video rate $r(t - \tau_b)$



(c) Playout buffer length $q(t)$, target buffer length $q_T(t)$, switch-down threshold $q_L(t)$

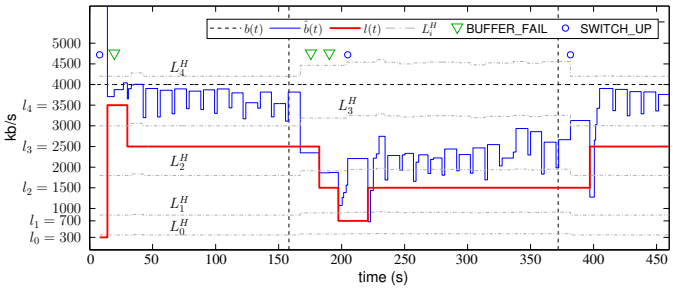


(d) Cumulative re-buffering time

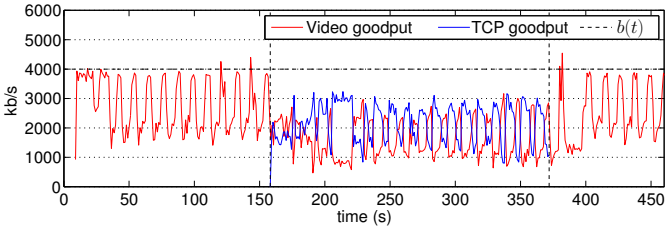
Fig. 15. Response to a square-wave available bandwidth with period 200 s

level l_4, after around 75s; 2) when bandwidth drops occur, the playback is affected by interruptions (see Figure 15(c), which shows the client buffer length, and Figure 15 (d), which shows the cumulative re-buffering time). In particular, when the first bandwidth drop occurs at $t = 200$s, the buffer length $q(t)$ gets below the switch-down threshold $q_L(t)$ after roughly 7s and a BUFFER_FAILURE command is sent to switch down the video level from l_3 to l_0 (see Figure 15 (c) and 15 (a)). Finally, the video level is switched to l_0 after a switch-down delay τ_{sd} of 30s. Therefore, it takes a total delay of 37s for the video level to match the new value of the available bandwidth. Due to this large delay, the receiver buffer gets empty at $t \simeq 220$s and the reproduction of the video is blocked for around 12s. This is confirmed by considering Figure 15 (d) which shows a step of 12s in the cumulative re-buffering time occurring at $t \simeq 220$s.

The same happens when the second bandwidth drop takes place at 400s. In this case, the total delay required to correctly set the video level is 34s. Again, 21s after the second



(a) Available bandwidth $b(t)$, estimated bandwidth $\hat{b}(t)$, video level $l(t)$, BUFFER_FAILURE/SWITCH_UP events, and high thresholds L_i^H



(b) Video goodput and TCP goodput

Fig. 16. Adaptive video streaming when sharing the bottleneck with a TCP flow

bandwidth drop, the video reproduction is paused for 11s.

Finally, Figure 15 (b) shows the video continuity index η_c and the network efficiency index η_n computed during each half-cycle of the square wave. In particular, when the available bandwidth is maximum ($b(t) = A_M$) no pauses due to re-buffering occur ($\eta_c = 1$), but the network efficiency is only 0.4, meaning bandwidth underutilization and reduced QoE. Conversely, when the available bandwidth is minimum ($b(t) = A_m$) the network efficiency is 1, whereas the video continuity index is roughly 0.9, meaning that for roughly 10% of the time the video was paused.

E. One video flow versus one concurrent TCP flow

This experiment investigates the performance of the stream-switching algorithm when one video flow shares the bottleneck with one greedy TCP flow. The bottleneck available bandwidth is kept constant at 4000 kb/s throughout the whole duration of the experiment. A video streaming session is started at $t = 0$ and a greedy TCP flow is injected at time $t = 150$ s and stopped at time $t = 370$s.

Figure 16 (a) shows the video level time behaviour $l(t)$ and the estimated bandwidth $\hat{b}(t)$. Vertical dashed lines divide the experiment in three parts.

During the first part of the experiment ($t < 150$ s), except during the short time interval $[6.18, 21.93]$s when $l(t)$ is equal to $l_4 = 3500$ kb/s, the video level is set to $l_3 = 2500$ kb/s and the network efficiency η_n is 0.74.

The second part of the experiment starts at $t = 150$s when the TCP flow joins the bottleneck and grabs the fair bandwidth share of 2000 kb/s. However, it is worth noting that the estimated bandwidth $\hat{b}(t)$ decreases to the correct value only after 9s. After an additional delay of 8s a BUFFER_FAIL command is sent at $t = 167$s (see Figure 16 (a)). Then, after the switch-down delay τ_{sd}, the video level is shrunk to the correct value $l_2 = 1500$ kb/s. Therefore, it took a total delay of 24s to select the proper video level. In this case, the actuation delay does not affect the video reproduction. At time $t = 182$s, a second BUFFER_FAIL command is issued and the video level is shrunk at time $t = 189$s, after a switch-down delay of $\tau_{sd} \simeq 7$s. At time $t = 193$s, an rtt-test command is issued so that for a short amount of time the video flow becomes greedy (see Subsection V-C). At time $t = 196$s, a 2200 kb/s bandwidth is estimated so that a SWITCH_UP command is sent and, at $t = 212$s, the video level is switched up to the suitable value of $l_2 = 1500$ kb/s. In this part of the experiment, the network efficiency index was equal to 0.76, the average goodput of the greedy TCP flow was 2170 kb/s, whereas the goodput obtained by video flow was 1643 kb/s, showing network underutilization. TCP gets a larger share of bandwidth since the streaming session is not always greedy using its throttling control (see equations (2), (3)). In the third part of the experiment, after the TCP flow leaves the bottleneck at time $t = 370$s, the level is switched up to $l_3 = 2500$ kb/s after a delay of 26s. In this part of the experiment the network utilization was $\eta_n = 0.69$.

TABLE III
AVERAGE AND STANDARD DEVIATION (IN PARENTHESIS) OF RE-BUFFERING TIME $\Delta T_{rb}^{(i,j)}$ IN SECONDS FOR EACH CONSIDERED BANDWIDTH DROP FROM $A_M^{(i)}$ TO $A_m^{(i)}$ IN KBPS

$A_M^{(i)}$ \ $A_m^{(j)}$	3200 (l_3)	2000 (l_2)	1000 (l_1)	500 (l_0)
5000 (l_4)	0 (0)	0 (0)	0 (0)	0.51 (1.1)
3200 (l_3)	/	0 (0)	2.7 (2.2)	8.5 (2.8)
2000 (l_2)	/	/	0(0)	6.1 (3.9)
1000 (l_1)	/	/	/	0 (0)

F. Investigating re-buffering events due to bandwidth drops

In this section we investigate to what extent the stream-switching is able to avoid re-buffering phases when switching down from l_i to l_j with $l_i > l_j$. With this purpose we have considered the available bandwidths $b^{(i,j)}(t) = A_M^{(i)} 1(t) - (A_M^{(i)} - A_m^{(j)}) \cdot 1(t - 100)$ exhibiting a sudden bandwidth decrease at $t = 100$s from $A_M^{(i)} \in \{A_M^{(1)} = 1000, A_M^{(2)} = 2000, A_M^{(3)} = 3200, A_M^{(4)} = 5000\}$kb/s to $A_m^{(j)} \in \{A_m^{(0)} = 500, A_m^{(1)} = 1000, A_m^{(2)} = 2000, A_m^{(3)} = 3200\}$kb/s with $A_M^{(i)} > A_m^{(j)}$. We have run 10 experiments for each $b^{(i,j)}(t)$ and we have measured the average and standard deviation of the re-buffering time $\Delta T_{rb}^{(i,j)}$ that are shown in Table 3. The table shows that the stream-switching algorithm performs well when switching down from l_4 to a lower video level. In fact, when $l(t) = l_4$ the queue threshold $q_T(t)$ is greater than 20s and the switch-down threshold $q_L(t)$ is greater than 16s (see equations (6) and (7)): thus, when the bandwidth decreases a BUFFER_FAIL command is sent when the buffer length is equal to $q_L(t) \in [16, 19]$s giving enough time to avoid a buffer underrun episode.

When switching down to the lowest level l_0 the player exhibited re-buffering phases due to buffer underruns: in particular, the most critical case was with a switch down from

level l_3 to level l_0 ($A_M = 3200$ kb/s, $A_m = 500$ kb/s) during which the re-buffering phase lasted on average 8.5s with a standard deviation of 2.8s. This suggests that increasing $q_T^0(l_i)$ for $i = 0, \ldots, 3$ could help to mitigate re-buffering phases.

VI. Conclusions

In this paper we have investigated the control system of a leading adaptive video streaming service. The contribution of this paper is twofold: 1) we have modelled and validated the automatic stream-switching controller; 2) we have carried out an extensive performance evaluation in a controlled environment.

For what concerns the modelling, we have found that the adaptive streaming algorithm employs two controllers: the first one controls the length of the client buffer and the second one selects the video level. The dynamics of the two interacting controllers have been described and validated in a controlled testbed scenario.

Regarding the performance evaluation, we have mainly found that: 1) a video flow takes 21s on average to switch up between two adjacent video levels in response to a bandwidth increase; 2) due to the conservativeness of the stream-switching algorithm, a low bandwidth utilization may be obtained; 3) when the available bandwidth suddenly decreases, interruptions of the video playback may occur due to a large actuation delay.

Acknowledgments

We thank the Associate Editor and the anonymous reviewers for the constructive comments that allowed us to improve the overall quality of the paper.

References

[1] Cisco, "Cisco Visual Networking Index:Forecast and Methodology 2012-2017," *White Paper*, 2012.

[2] A. Zambelli, "IIS smooth streaming technical overview," *Microsoft Corporation*, 2009.

[3] I. Sodagar, "The mpeg-dash standard for multimedia streaming over the internet," *IEEE MultiMedia*, vol. 18, no. 4, pp. 62–67, 2011.

[4] S. Akhshabi, A. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP," *Proc. of ACM MMSys '11*, pp. 157–168, 2011.

[5] H. Riiser, H. Bergsaker, P. Vigmostad, P. Halvorsen, and C. Griwodz, "A comparison of quality scheduling in commercial adaptive HTTP streaming solutions on a 3G network," in *Proc. of the 4th Workshop on Mobile Video*, 2012, pp. 25–30.

[6] C. Müller, S. Lederer, and C. Timmerer, "An evaluation of dynamic adaptive streaming over HTTP in vehicular environments," in *Proc. of the 4th Workshop on Mobile Video*, 2012, pp. 37–42.

[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 IMC '12*, 2012, pp. 225–238.

[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*, 2012, pp. 97–108.

[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*, Oct. 2012, pp. 405–413.

[10] 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.

[11] M. Handley, S. Floyd, and J. Pahdye, "TCP Friendly Rate Control (TFRC): Protocol Specification," *RFC 3448, Proposed Standard*, Jan. 2003.

[12] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," *RFC 3550, Standard track*, 2003.

[13] "Real-Time Messaging Protocol (RTMP) Specification," Available at: http://www.adobe.com/devnet/rtmp.html, Apr. 2009.

[14] V. Jacobson, "Congestion avoidance and control," in *Proc. of ACM SIGCOMM '88*, 1988, pp. 314–329.

[15] S. Mascolo, "Congestion control in high-speed communication networks using the Smith principle," *Automatica*, vol. 35, no. 12, pp. 1921–1935, 1999.

[16] M. Prangl, I. Kofler, and H. Hellwagner, "Towards QoS Improvements of TCP-Based Media Delivery," in *Proc. of ICNS '08*, 2008, pp. 188–193.

[17] C. Krasic, J. Walpole, and W. Feng, "Quality-adaptive media streaming by priority drop," in *Proc. of NOSSDAV 2003*, 2003.

[18] R. Kuschnig, I. Kofler, and H. Hellwagner, "An evaluation of TCP-based rate-control algorithms for adaptive internet streaming of H. 264/SVC," in *Proc. of ACM MMSys '10*, 2010, pp. 157–168.

[19] D. Hassoun, "Dynamic streaming in Flash Media Server 3.5 - Part 1: Overview of the new capabilities," Jan. 2009. [Online]. Available: http://www.adobe.com/devnet/flashmediaserver/

[20] "Move networks hd adaptive video streaming," http://www.movenetworkshd.com.

[21] "Netflix," http://www.netflix.com.

[22] S. Hemminger, "Network emulation with NetEm," in *Linux Conference Au*, 2005, pp. 18–23.

[23] G. Franklin, J. Powell, and A. Emami-Naeini, *Feedback control of dynamic systems*. Addison-Wesley, 2002.

[24] L. De Cicco and S. Mascolo, "A Mathematical Model of the Skype VoIP Congestion Control Algorithm," *IEEE Transactions on Automatic Control*, vol. 55, no. 3, pp. 790–795, Mar. 2010.

[25] T. Ott, J. Kemperman, and M. Mathis, "The Stationary Behavior of Ideal TCP Congestion Avoidance," Aug. 1996. [Online]. Available: ftp://ftp.research.telcordia.com/pub/tjo/TCPwindow.ps

[26] A. Rao, A. Legout, Y.-s. Lim, D. Towsley, C. Barakat, and W. Dabbous, "Network characteristics of video streaming traffic," in *Proc. of CoNEXT '11*, 2011, pp. 25:1–25:12.

[27] L. De Cicco, S. Mascolo, and S.-I. Niculescu, "Robust stability analysis of Smith predictor-based congestion control algorithms for computer networks," *Automatica*, vol. 47, no. 8, pp. 1685 – 1692, 2011.

[28] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," in *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 4, 1998, pp. 303–314.

[29] D. Bulterman, "SMIL 2.0 Part 1: Overview, Concepts and Structures," *IEEE Multimedia*, vol. 8, no. 4, pp. 82–89, 2001.

[30] S. Winkler and P. Mohandas, "The evolution of video quality measurement: From PSNR to hybrid metrics," *IEEE Transactions on Broadcasting*, vol. 54, no. 3, pp. 660–668, 2008.

[31] 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*, 2011, pp. 362–373.

[32] S. S. Krishnan and R. K. Sitaraman, "Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs," in *Proc. of ACM IMC '12*, 2012, pp. 211–224.

[33] P. Ni, R. Eg, A. Eichhorn, C. Griwodz, and P. Halvorsen, "Flicker effects in adaptive video streaming to handheld devices," in *Proc. of ACM International Conference on Multimedia '11*, 2011, pp. 463–472.