Nonlinear Analysis: Hybrid Systems 21 (2016) 139–154

Contents lists available at ScienceDirect

Nonlinear Analysis: Hybrid Systems

journal homepage: www.elsevier.com/locate/nahs

A hybrid model of the Akamai adaptive streaming control system☆

Luca De Cicco*, Giuseppe Cofano, Saverio Mascolo

Politecnico di Bari, Dipartimento di Ingegneria Elettrica e dell'Informazione, Via Orabona 4, Bari, Italy

ARTICLE INFO

Article history:
Available online 29 January 2016

Keywords:
Hybrid systems
Hybrid model validation
Adaptive video streaming

ABSTRACT

Video streaming is the application generating the largest fraction of the Internet traffic. Adaptive video streaming adds to classic video streaming the possibility of dynamically adapting the video bitrate to track the time-varying network available bandwidth, avoid playback interruptions and ensure the delivery of the best video quality. This work focuses on the adaptive video streaming control system employed by Akamai, a major Content Delivery Network operator whose video delivery system is used by several video streaming platforms, including Livestream. Differently from the typical client-side control, Akamai employs an interesting and unique hybrid client/server control architecture. In this paper we propose and experimentally validate a closed loop mathematical model of the control system in the form of a hybrid automaton. The model is analyzed to derive key properties which can be used to properly tune the controller parameters.

1. Introduction

Today, multimedia applications produce an ever increasing fraction of the Internet traffic [1]. Video streaming is the most important application driving this trend. Examples such as YouTube and Netflix, which together account for more than half of US traffic during peak hours,[1] give evidence of the wide diffusion such applications have reached.

Adaptive video streaming represents a key innovation with respect to classic progressive download streaming. In fact, adaptive streaming systems can change the video bitrate to match the time-varying and unpredictable available bandwidth. Moreover, start-up latency can be minimized and video playback interruptions can be avoided.

Today, the leading approach to implement adaptivity, which is used – among the others – by YouTube, Netflix, and Akamai, is the *stream-switching*: the server encodes the video content at different bitrates, namely the video levels, and a control algorithm dynamically selects the video level to be sent based on measurements such as the available bandwidth and the player buffer length.

From the control architecture point of view, the mainstream approach is the one employed by the *Dynamic Adaptive Streaming over HTTP* (DASH) standard, which places the controller at the client and streams the video through standard HTTP (HyperText Transfer Protocol) servers [2].

In this paper, which is an extended version of [3], we focus on the proprietary adaptive streaming control system employed by Akamai, which is used by several video streaming services to implement a massive video distribution platform.

☆ This is an extended version of the paper De Cicco et al. (2014) [3] presented at the IFAC World Congress 2014.

* Corresponding author.
E-mail addresses: luca.decicco@poliba.it (L. De Cicco), giuseppe.cofano@poliba.it (G. Cofano), mascolo@poliba.it (S. Mascolo).

[1] Sandvine, "Global Internet Phenomena Report, 2014", 2014, https://www.sandvine.com/trends/global-internet-phenomena/.

http://dx.doi.org/10.1016/j.nahs.2015.12.007
1751-570X/

Since its source code is not available and cannot be inspected, in De Cicco and Mascolo [4] an experimental testbed has been used to investigate its control system. It has been shown that the Akamai control system employs an interesting and unique hybrid architecture that is distributed at the client and the server. In De Cicco and Mascolo [4] the key features of the Akamai adaptive video streaming algorithm were identified and experimentally validated. However, a rigorous mathematical model of the control system was missing. In this paper we make the following contributions: (1) we propose a model of the client playout buffer; (2) we design a hybrid automaton rigorously modeling the Akamai control system, which shows a tight interaction between time-discrete and time-continuous dynamics; (3) we derive several control system properties that can be used for tuning the controller parameters; (4) an experimental validation of the proposed hybrid automaton is given.

By analyzing the model we are able to provide tuning rules for the controller parameters to ensure that: (1) the optimal video level can be reached; (2) the playout buffer does not grow unbounded; (3) the interruption of the video reproduction due to the presence of an actuation time-delay is avoided.

The paper is organized as follows. Section 2 contains a review of the literature studying adaptive video streaming systems and an overview of the related work on hybrid models for computer network applications; Section 3 introduces the stream-switching approach; in Section 4 the dynamics of the playout buffer is derived; the model of the Akamai control system is proposed in Section 5 and several properties of the system are derived in Section 6; finally, Section 7 provides the validation of the model.

2. Related work

The study of client-side adaptive streaming algorithms is a hot topic in the networking community. Recently, several authors have analyzed the issues of client-side adaptive streaming systems. Typically, such systems produce an on–off traffic pattern [5,6]: the video segments are downloaded during an ON phase and then, during the OFF phase, the player is kept idle until the next download is started. In Akhshabi et al. [7] it is shown that such on–off traffic pattern is the key factor causing unfair bandwidth utilization, server bandwidth underutilization and oscillation of the player requested video level. To tackle these issues, several adaptive streaming algorithms have been proposed so far. In Jiang et al. [8] FESTIVE has been proposed to address the fairness issues arising in a multi-client scenario. In Li et al. [9] the algorithm PANDA is proposed: a controller dynamically computes the segment inter-request time to address fairness issues and video bitrate oscillations. A different approach has been taken in Akhshabi et al. [10], where authors place a traffic shaper at the server with the purpose of ruling out the OFF phases when the player is in steady-state. In Huang et al. [11] a heuristic control approach to control the playout buffer avoiding OFF phases is taken at the client. The proposed algorithm does not require any bandwidth estimate. An extensive experimental section is presented, in which performance obtained by Netflix clients employing the proposed algorithm is shown. In Zhu et al. [12] a PI controller is employed to control the playout buffer. Instead, by using a feedback linearization approach De Cicco et al. [13] have designed a stream-switching controller which is able to avoid OFF phases without requiring server cooperation. Experimental results carried out in a controlled testbed are provided. In Yin et al. [14] an optimization problem is formulated to take a Model Predictive Control approach to control the playout buffer. The objective function to maximize expresses the user perceived Quality of Experience, which is related to the number of times the playout buffer gets empty.

Regarding the topic of hybrid modeling for computer networking applications, in Lee et al. [15] a hybrid model of both the buffer routers and the TCP (Transmission Control Protocol) congestion control algorithm is provided in the form of a hybrid automaton. The model is validated by comparing simulations of the hybrid models against packet-level simulations. Another relevant application is considered in Hespanha [16], which proposes a stochastic hybrid framework to model the TCP congestion control. Both the congestion-avoidance and slow-start modes for on–off TCP flows are considered in the model, which takes into account the distribution of the transmitted bytes. It is shown that, for transfer-size distributions reported in the literature, the standard deviation of the sending rate is much larger than its average and that the average sending rate varies slightly with the probability of packet drop. In Savkin et al. [17] the Medium Access Control (MAC) of a class of wireless communication networks is modeled through a hybrid framework. Some stabilizability conditions, which guarantee that any data packet in the wireless network will reach its destination within finite time, are provided. In Albea-Sanchez et al. [18] a hybrid model of a cluster of three servers has been proposed to design a load balancing hybrid controller. In particular, the controller aims at adapting the number of active servers to the total load of incoming requests in order to minimize energy consumption.

This paper enriches the state of the art by proposing, at the best of our knowledge, the first hybrid model of a real adaptive video streaming system.

3. The stream-switching approach

A video streaming system allows a client to reproduce the video that is sent by a remote server through an Internet connection. The client employs a playout buffer to absorb the instantaneous mismatches between the encoding bitrate and the network available bandwidth, which in best effort Internet is unpredictable and time-varying. However, if the bandwidth gets below the video bitrate for a sufficiently long time, the buffer will eventually get empty and a buffering phase will be triggered: the player gets paused for a time interval, the *buffering time*, allowing the buffer to reach a safety threshold before the playing can be resumed again.

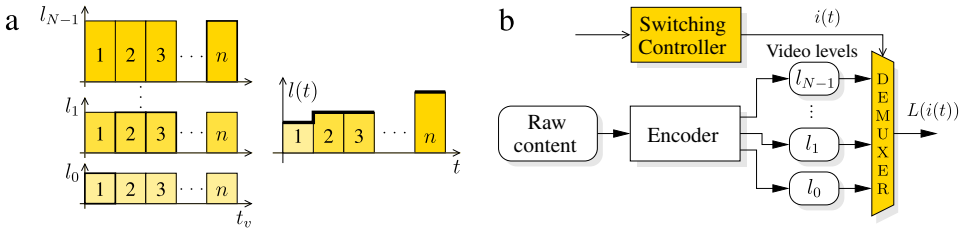

Fig. 1. Video segments and video levels (a) and the video level selection (b).

In classic progressive download streaming the video content is available only at a bitrate equal to l. Such an approach cannot guarantee both maximum channel utilization and the avoidance of buffering events. On the other hand, with adaptive video streaming the video bitrate can be dynamically throttled, at least to some extent, to match the available bandwidth $b(t)$. Hence, an adaptive video streaming system can leverage the video encoding bitrate to implement a controller whose aim is to ensure that the player buffer length $q(t)$ does not get empty and that the highest video quality is provided.

Among the proposed adaptive video streaming approaches, the stream-switching is getting a wide adoption due to its implementation and deployment simplicity [19]. Fig. 1(a) shows that the video is encoded at bitrates $l_0 < l_1 < \cdots < l_{N-1}$ resulting into N versions, i.e. the *video levels*. Each video level is then divided logically, or physically, into n segments of fixed duration. We define the discrete set of video levels $\mathscr{L} = \{l_0, l_1, \ldots, l_{N-1}\}$, a set of video level indices $\mathscr{I} = \{0, 1, \ldots, N-1\}$, and the bijective mapping $L : \mathscr{I} \to \mathscr{L}$ that associates the video level index to the video level bitrate. Fig. 1(b) shows how the switching controller changes the video level: at any given time t, based on the controller input, it computes the video level index $i(t)$ to select. As a consequence, the client will receive a video with a variable encoding bitrate $l(t) = L(i(t))$. Fig. 1(a) depicts a possible dynamics of $l(t)$ which also shows that video level switches can only occur at the beginning of a new segment transmission.

4. The playout buffer dynamics

In this section we derive a fluid flow model of the client playout buffer. The fluid flow assumption has been employed in the literature to model several computer network systems (Hollot et al. [20], Michiels et al. [21], Mascolo [22], Srikant [23], Paganini et al. [24]) and to obtain mathematically tractable models used to derive properties useful for system parameters sizing. In the case of an adaptive video streaming system the fluid flow assumption consists in assuming that the video segments are infinitesimal in size.

As any storage element, the playout buffer length $q(t)$, i.e. the total duration of video stored in the playout buffer, can be modeled as an integrator:

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

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

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

$$f(t) = \frac{r(t)}{l(t)}. \tag{1}$$

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

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

Finally, by combining (1) and (2) we obtain the playout buffer length model:

$$\dot{q}(t) = \frac{r(t)}{l(t)} - d(t). \tag{3}$$

Table 1
Dynamic variables symbols.

Symbol	Description
i	Video level index
l	Video level bitrate
l_{opt}	Optimal video level bitrate
q	Playout buffer level
r	Received rate
$\hat{r}$	Server sending rate
T	Throttling signal
b	Available bandwidth
$\hat{b}$	Estimated available bandwidth
s	Hybrid automaton state
τ	Timer
τ_1	Timer duration for Normal phase
τ_2	Timer duration for Greedy phase

Table 2
System parameters symbols.

Symbol		Description	Value
	$\mathscr{L}$	Video level set	{0.3, 0.7, 1.5, 2.5, 3.5} Mbps
	S_f	Safety factor	0.2
	B	Constant available bandwidth	No fixed value
	q_T	Setpoint	7 s
Queue	q_d	Switch-down threshold	4 s
	q_h	Refilling threshold	12 s
	T_M	Greedy throttling	5
Throttling	T_m	Large Queue throttling	0.1
	T_B	Buffering throttling	2
	τ_{NU}	Normal phase after switch-up	10.5 s
Timers	τ_{GU}	Greedy phase after switch-up	3.5 s
duration	τ_{ND}	Normal phase after switch-down	6.5 s
	τ_{GD}	Greedy phase after switch-down	8.5 s

5. The Akamai hybrid model

In this section we give a complete mathematical model of the Akamai adaptive video streaming control system. The features of the control system have been derived in De Cicco and Mascolo [4] by means of an experimental investigation, since the control system is proprietary and its source code cannot be inspected. It is worth noticing that such findings have been validated in De Cicco and Mascolo [4] through a large set of experiments, but a model of the playout buffer is not provided in the same work. In this paper, by leveraging the playout buffer model, we formulate a mathematical model of the overall control system in the form of an hybrid automaton.

In Section 5.1 we present the main features of the Akamai adaptive stream-switching controller, which are discussed in detail in De Cicco et al. [13]. Then, in Section 5.2 a model of the system is derived in the form of a hybrid automaton. Tables 1 and 2 provide a list of the symbols employed in the model.

5.1. The Akamai adaptive stream-switching control system

The video is sent to the client by a HTTP server over a TCP connection. The typical behavior of a Akamai video streaming session can be summarized as follows. At the beginning of the video session, a *Buffering* phase is entered to quickly fill the queue until a threshold is reached. Then, a periodical switching between two phases is established: during the *Normal* phase the control system throttles the server sending rate to steer the playout buffer to a target length; during the *Greedy* phase, instead, the server sends at a higher sending rate to probe and estimate the available bandwidth. During the periodical switching the following events can occur: (1) the control system decides to either switch up or switch down the video level; (2) the playout buffer gets empty. The *Buffering* phase is entered after a switch-down or when the playout buffer gets empty.

The control system is composed of two controllers both executed at the client: (1) the *stream-switching controller*, which dynamically selects the appropriate video level and (2) the *playout buffer level controller*, whose goal is to avoid that the playout buffer gets empty.

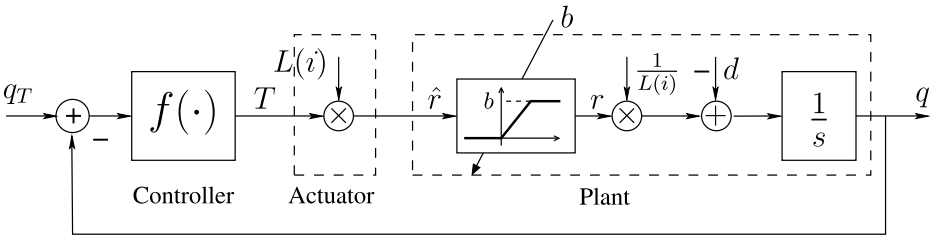

Fig. 2. The playout buffer control system.

The stream-switching control system

The stream-switching controller is event-based, i.e. its decisions are triggered when particular events occur. Assuming the current video level is l_i, a video level switch-up is triggered when the estimated bandwidth $\hat{b}(t)$ is at least greater than the video level l_{i+1} by a safety factor $S_f > 0$, i.e. when $\hat{b}(t) > (1 + S_f)l_{i+1}$. Then, the algorithm selects the maximum video level $l_j > l_i$ such that $\hat{b}(t) > (1 + S_f)l_j$. On the other hand, a video level switch-down is triggered when the playout buffer length $q(t)$ is lower than the switch-down threshold q_d. In this case the algorithm selects the maximum video level $l_j < l_i$ such that $\hat{b}(t) > (1 + S_f)l_j$. Hence, the *optimal video level* $l_{opt}(t)$ is given by:

$$l_{opt}(t) = \max_{l \in \mathscr{L}} l \quad \text{s.t. } \hat{b}(t) > (1 + S_f)l. \tag{4}$$

The playout buffer control system

The playout buffer control system is composed of three elements, as shown in Fig. 2:

1. the controller, which sets a throttling signal $T(t)$ to drive the server sending rate $\hat{r}(t)$;
2. the actuator, placed at the server, which computes $\hat{r}(t)$ based on the throttling signal $T(t)$ and the current video level $L(i(t))$;
3. the plant, which is made of a saturation block, modeling the bottleneck link of bandwidth $b(t)$, and the playout buffer $q(t)$.

The controller computes the throttling signal $T(t)$ according to the current phase of the system, i.e. Buffering, Normal, or Greedy.

During the Buffering phase, $T(t)$ is set equal to $T_B > 1$. This phase is left when $q(t)$ gets above a threshold $q_h > q_d$. It is important to notice that switch-up events are inhibited during this phase.

As mentioned above, when the Buffering phase is left, a periodical switching between Normal and Greedy phases is established. This periodical behavior is enforced by using a timer. Its duration depends on the last video level switching command: after a switch-up (switch-down) the Normal phase lasts τ_{NU} (τ_{ND}), whereas the Greedy phase lasts τ_{GU} (τ_{GD}).

During the Normal phase $T(t)$ is varied to implement a closed loop control system that aims at steering $q(t)$ to a set-point q_T. To the purpose, $T(t)$ is set as a function of the error $e(t) = q_T - q(t)$ according to:

$$T(t) = \max\left(1 + \frac{q_T - q(t)}{q_T}, T_m\right). \tag{5}$$

The minimum throttling T_m avoids the sending rate to be zero. By considering (5) it turns out that $T(t) = T_m$ when $q(t) > (2 - T_m)q_T$, i.e. in case the playout buffer is large.

The Greedy phase is employed to make the video saturate the bottleneck link and measure the available bandwidth. During this phase $T(t)$ is set to $T_M > 2$, which is the maximum value of $T(t)$ during the Normal phase.

The throttling signal $T(t)$ is then used by the actuator to produce the sending rate according to:

$$\hat{r}(t) = T(t)L(i(t)). \tag{6}$$

Actually, since Akamai employs the TCP to send the video content, the sending rate is controlled by the TCP congestion control algorithm, which generates a best-effort traffic quickly matching the available bandwidth $b(t)$. However, the Akamai video server can indirectly control the sending rate by filling the TCP buffer at the desired sending rate. In De Cicco and Mascolo [4] it is shown that the integral error between the desired sending rate and the sending rate obtained with this technique is bounded by the TCP buffer size, i.e. the desired sending rate and the sending rate match on short average. In

practice, since the dynamics of the TCP congestion control is much faster than that of the control system, the received rate $r(t)$ can be well approximated by the following equation:

$$r(t) = \begin{cases} \hat{r}(t) & \hat{r}(t) < b(t) \\ b(t) & \text{otherwise} \end{cases} \tag{7}$$

which corresponds to the saturation block shown in Fig. 2.

Fig. 2 shows that controlling the sending rate through the throttling signal $T(t)$ multiplied by $L(i(t))$ is equivalent to a feedback linearization of the plant. In fact, in the absence of saturation, it turns out that $\dot{q}(t) = T(t) - d(t)$. At steady state, $T(t)$ is equal to $d(t) = 1$. By considering (5) it turns out that $q(t) = q_T$. Thus, during the Normal phase the control signal $T(t)$ drives the queue length to the set-point.

The client estimates the available bandwidth $\hat{b}(t)$ by measuring and filtering the received rate $r(t)$. By neglecting the filter, since its dynamics is dominated by that of the system, we can assume $\hat{b}(t) = r(t)$. Hence, in the hybrid model we will employ $r(t)$ in place of $\hat{b}(t)$ when formulating the video level switching conditions.

Assumptions

In this paper we assume a piecewise constant available bandwidth input function, which allows to analyze any practical traffic scenario with a bottleneck link [22]. Thus, we will analyze the system behavior in response to a step input of amplitude B. With respect to [4] in the model presented above we have made the following simplifying assumptions:

- A1: the communication forward and backward delays from the client to the server τ_f and τ_b and the actuation time-delay τ_a have been neglected;
- A2: the time-varying safety factor $S(t)$, which varies in [0.2, 0.4] depending on the Round Trip Time, has been assumed to be constant and equal to S_f;
- A3: the queue thresholds $q_d(t)$ and $q_h(t)$ and the set-point $q_T(t)$, which depend on the current video level and on the safety factor $S(t)$, have been assumed to be constant.

Regarding communication delays, even though they play an important role in several computer network domains such as TCP congestion control (Mascolo [22]), it has been experimentally shown that such delays have a negligible impact on the performance of adaptive video streaming control systems (Akhshabi et al. [5], Huang et al. [6], Jiang et al. [8], Akhshabi et al. [7]). In Section 7 we show through an experimental validation that this modeling assumption is reasonable and the proposed model is able to nicely model the real system that contains the Internet delays.

5.2. The proposed hybrid automaton

In this section we propose a closed loop model of the system described in Section 5.1. The state-dependent and event-triggered dynamics and the nonlinear elements, such as the saturation block, can be properly modeled by means of a hybrid system. In particular, we model the system as a hybrid automaton [25].

Fig. 3 shows the proposed hybrid automaton, which is denoted as $\mathcal{H}$. The state of the system is given by the continuous component:

$$x = [i(t), r(t), q(t), \tau(t), \tau_1(t), \tau_2(t)]^\intercal$$

and the logical component $s \in S = \{0, 1, \ldots, 7\}$, where S is the *set of logical modes* of the automaton. The six state variables of x are: (1) the current video level index $i(t)$ decided by the stream-switching controller, (2) the received rate $r(t)$ in kbps, (3) the playout buffer length $q(t)$ in seconds, (4) a timer $\tau(t)$, (5) and (6) timer 1 and timer 2 durations $\tau_1(t)$ and $\tau_2(t)$. The initial conditions for each state are such that $x \in \mathcal{L} \times [0, B] \times \mathbb{R}_+ \times \mathbb{R}_+ \times \mathbb{R}_+ \times \mathbb{R}_+$.

It is worth noting that only $q(t)$ and $\tau(t)$ are characterized by a dynamic evolution, whereas the remaining variables can change only during jumps. The timer dynamics is given by:

$$\dot{\tau} = \begin{cases} 1 & \text{on}, \\ 0 & \text{off}. \end{cases}$$

Thus, to characterize its behavior it is sufficient to specify in which states it is on.

Before giving the complete model, we recall that: (1) Dom(s) is the *domain set* of x associated to the logical mode $s \in S$; (2) $f_s : \mathbb{R}^6 \to \mathbb{R}^6$ is the *flow map* which describes the continuous evolution of x for each mode $s \in S$; (3) G_{ij} is the guard condition to be matched to switch from mode i to j; (4) R_{ij} is the *reset map* that is used to set the state x^+ after a jump from mode i to j, i.e. $x^+ = R_{ij}(x)$.

Now we are ready to present the complete model. We define the following conditions to simplify the notation of the domain sets:

$$C_1 = x_3 > q_d \wedge x_3 < (2 - T_m)q_T \wedge x_4 < x_5$$
$$C_2 = x_3 > 0 \wedge x_3 \leq q_d \wedge (x_1 = 0 \vee x_2 \geq (1 + S_f)L(x_1))$$
$$C_3 = x_3 \geq \left(2 - \frac{B}{L(x_1)}\right) q_T.$$

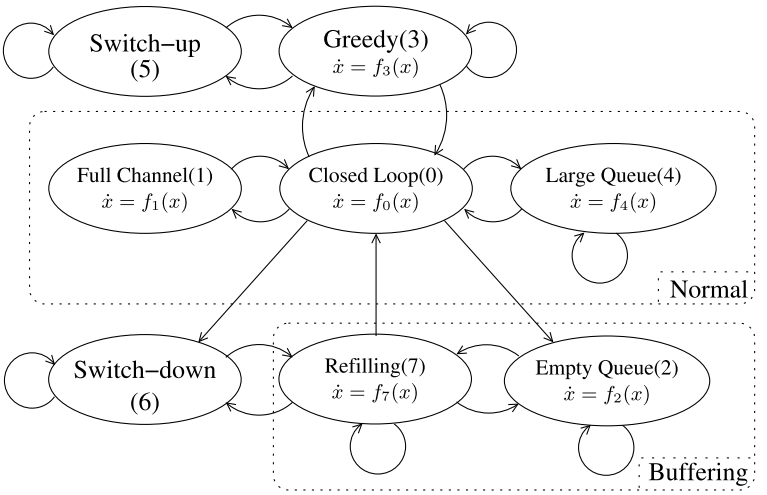

Fig. 3. The proposed hybrid automaton $\mathcal{H}$.

The domain sets are defined as follows:

$$\begin{aligned}
&\text{Dom}(0) = \{x : (C_1 \vee C_2) \wedge C_3\}\\
&\text{Dom}(1) = \{x : (C_1 \vee C_2) \wedge \neg C_3\}\\
&\text{Dom}(2) = \{x : x_2 \leq B \wedge x_3 < q_d\}\\
&\text{Dom}(3) = \{x : (x_1 = N-1 \vee x_2 < (1+S_f)L(x_1+1)) \wedge x_2 \leq B \wedge x_3 > q_d \wedge x_4 < x_6\}\\
&\text{Dom}(4) = \{x : x_2 \leq B \wedge x_3 > (2-T_m)q_T \wedge x_4 < x_5\}\\
&\text{Dom}(5) = \text{Dom}(6) = \emptyset\\
&\text{Dom}(7) = \{x : x_2 \leq B \wedge (C_2 \vee (x_3 > q_d \wedge x_3 < q_h))\}.
\end{aligned}$$

The flow maps $f_s(x)$ are given by:

$$\begin{aligned}
f_0(x) &= \left[0; 0; -\frac{x_3}{q_S} + 1; 1; 0; 0\right]\\
f_1(x) &= \left[0; 0; \frac{B}{L(x_1)} - 1; 1; 0; 0\right]\\
f_2(x) &= \left[0; 0; \frac{x_2}{L(x_1)}; 0; 0; 0\right]\\
f_3(x) &= f_4(x) = \left[0; 0; \frac{x_2}{L(x_1)} - 1; 1; 0; 0\right]\\
f_7(x) &= \left[0; 0; \frac{x_2}{L(x_1)} - 1; 0; 0; 0\right].
\end{aligned}$$

The flow maps f_5 and f_6 are not defined since the states 5 and 6 have no continuous dynamics.

The guard conditions are:

$$\begin{aligned}
G_{01} &= \{x : x_3 < (2 - B/L(x_1))q_T \wedge x_3 > 0 \wedge x_4 < x_5 \wedge (x_3 > q_d \vee x_1 = 0 \vee x_2 \geq (1+S_f)L(x_1))\}\\
G_{02} &= \left\{x : (x_1 = 0 \vee x_2 \geq (1+S_f)L(x_1)) \wedge x_3 \leq 0\right\}\\
G_{03} &= \{x : x_3 > q_d \wedge x_4 \geq x_5\}\\
G_{04} &= \{x : x_3 > (2-T_m)q_T \wedge x_4 < x_5\}\\
G_{06} &= \left\{x : x_1 > 0 \wedge x_2 < (1+S_f)L(x_1) \wedge x_3 \leq q_d\right\}\\
G_{10} &= G_{02} \cup G_{03} \cup G_{04} \cup G_{06} \cup \{x : x_3 \geq (2 - B/L(x_1))q_T\}\\
G_{22} &= G_{33} = G_{44} = G_{77} = \{x : x_2 > B\}\\
G_{27} &= \{x : x_2 \leq B \wedge x_3 \geq q_d\}\\
G_{30} &= \{x : x_2 \leq B \wedge (x_3 \leq q_d \vee x_4 \geq x_6)\}\\
G_{35} &= \{x : x_1 < N-1 \wedge x_2 \geq (1+S_f)L(x_1+1) \wedge x_2 \leq B \wedge x_3 > q_d \wedge x_4 < x_6\}
\end{aligned}$$

$$G_{40} = \{x : x_2 \leq B \wedge (x_3 \leq (2 - T_m)q_T \vee x_4 \geq x_5)\}$$
$$G_{53} = \{x : x_1 = N - 1 \vee x_2 < (1 + S_f)L(x_1 + 1)\}$$
$$G_{55} = \{x : x_1 < N - 1 \wedge x_2 \geq (1 + S_f)L(x_1 + 1)\}$$
$$G_{66} = \{x : x_1 > 0 \wedge x_2 < (1 + S_f)L(x_1)\}$$
$$G_{67} = \{x : x_1 = 0 \vee x_2 \geq (1 + S_f)L(x_1)\}$$
$$G_{70} = \{x : x_2 \leq B \wedge x_3 \geq q_h\}$$
$$G_{72} = \{x : x_2 \leq B \wedge x_3 \leq 0\}$$
$$G_{76} = \{x : x_1 > 0 \wedge x_2 < (1 + S_f)L(x_1) \wedge x_3 \leq q_d\}.$$

The reset maps are:

$$R_{01} = R_{10} = R_{33} = R_{40} = R_{44} = R_{77} = [x_1; B; x_3; x_4; x_5; x_6]$$
$$R_{02} = R_{27} = [x_1; T_B L(x_1); x_3; 0; x_5; x_6]$$
$$R_{03} = R_{67} = [x_1; T_M L(x_1); x_3; 0; x_5; x_6]$$
$$R_{30} = R_{72} = [x_1; x_2; x_3; 0; x_5; x_6]$$
$$R_{53} = R_{70} = [x_1; x_2; x_3; x_4; x_5; x_6]$$
$$R_{04} = [x_1; T_m L(x_1); x_3; x_4; x_5; x_6]$$
$$R_{06} = [x_1 - 1; x_2; x_3; x_4; \tau_{ND}; \tau_{GD}]$$
$$R_{22} = [x_1; B; x_3; 0; x_5; x_6]$$
$$R_{35} = [x_1 + 1; x_2; x_3; x_4; \tau_{NU}; \tau_{GU}]$$
$$R_{55} = [x_1 + 1; x_2; x_3; x_4; x_5; x_6]$$
$$R_{66} = [x_1 - 1; x_2; x_3; x_4; x_5; x_6]$$
$$R_{76} = [x_1 - 1; x_2; x_3; x_4; \tau_{ND}; \tau_{GD}].$$

In the following we describe the 8 logical modes[2] of $\mathcal{H}$:

- Closed Loop (0), Full Channel (1), and Large Queue (4) model the behavior of the system during the Normal phase described in Section 5.1. It has been necessary to employ three states to deal with the two nonlinearities that can occur in this phase (see (5) and (7)). The Closed Loop state holds by default during Normal phase. The Large Queue state holds only if $q(t) > (2 - T_m)q_T$, i.e. this state models the nonlinearity in (5). Full Channel holds when the bottleneck is saturated (see (7)) and prevents the received rate from going above the bottleneck bandwidth B.
- Empty Queue (2) and Refilling (7) model the Buffering phase described in Section 5.1. During the Empty Queue state, which is entered when $q(t) = 0$, the video playback is stopped ($d(t) = 0$) to quickly fill the queue; during Refilling, which is entered from either Switch-down or Empty Queue states, the video playback is active, i.e. $d(t) = 1$.
- Greedy (3) models the Greedy phase described in Section 5.1, where the system probes the available bandwidth;
- Switch-up (5) and Switch-down (6) are logical modes and implement the stream-switching controller; in these states only jumps can occur and a continuous evolution is forbidden.

In the following we give an explanation of the typical discrete and continuous dynamics of the proposed model. As a modeling choice, we have decided to minimize the amount of guard conditions in order to improve the readability of the model. Following this design choice, in some cases we have decided to replace the direct transition between two states with an indirect transition through another state. For instance, the transition from the Full Channel state to the Empty Queue state, which should be triggered when the condition G_{02} (the queue becomes empty, i.e. $x_3 = 0$) is met while being in the Full Channel state, has been handled in the following way. If G_{02} is met when the state is Full Channel, G_{10} is met and a transition to the Closed Loop state is triggered, without changing x_3 (see R_{10}). Once in the Closed Loop state, G_{02} is met and the transition from the Closed Loop state to the Empty Queue state is triggered.

Let us focus on the Normal phase, which is described by means of three states: Closed Loop, Large Queue and Full Channel. During this phase the timer $\tau(t)$ is on and is not reset during transitions between these three states.[3] As shown in Fig. 3, the Normal phase is entered either from Greedy, when the Greedy timer fires or when the queue gets below q_d (G_{30}), or from Refilling, when the queue gets above q_d (G_{70}). The Normal phase can be entered and left only through the Closed Loop state. Large Queue and Full Channel can be entered from Closed Loop if either G_{04} or G_{01} guard conditions are matched. The Normal phase is left when: (1) the timer fires (G_{03}), (2) $q(t)$ gets below q_d (G_{06}) or (3) the queue gets empty (G_{02}). If one of these conditions is matched during Large Queue or Full Channel, a transition to Closed Loop first occurs and then the Normal phase is left. Let us now analyze the queue dynamics and the guard conditions of these three states.

2 The hybrid system states names are written with a different font. For instance Greedy refers to the logical mode, whereas Greedy refers to the phase.

3 According to the reset maps $R_{01}, R_{10}, R_{04}, R_{40}$ it turns out that $x_4^+ = x_4$.

During Closed Loop the queue dynamics can be derived by combining (3) and

$$T(t) = 1 + (q_T - q(t))/q_T \tag{8}$$

that is obtained by (5), giving:

$$\dot{q}(t) = \frac{r(t)}{L(i(t))} - 1 = \left(2 - \frac{q(t)}{q_T}\right)\frac{L(i(t))}{L(i(t))} - 1 = -\frac{q(t)}{q_T} + 1, \tag{9}$$

which turns out to be a first order linear system.

Full Channel is employed to model the bottleneck link saturation nonlinearity (7). Hence, a transition to Full Channel occurs when the rate $\hat{r}(t)$ gets above B. Thus, by imposing $\hat{r}(t) > B$ we obtain the guard condition G_{01}:

$$\hat{r}(t) = T(t)L(i(t)) > B \Rightarrow \left(2 - \frac{q(t)}{q_T}\right)L(i(t)) > B \Rightarrow q(t) < \left(2 - \frac{B}{L(i(t))}\right)q_T.$$

Then in this state, the queue dynamics is described by:

$$\dot{q}(t) = \frac{B}{L(i(t))} - 1.$$

Thus, the queue is open-loop and is either filled or drained according to the ratio $B/L(i(t))$. The Full Channel state is left when G_{10} is matched, i.e. if either $q(t)$ gets above $(2 - \frac{B}{L(i(t))})q_T$ or one of the guard conditions to leave Closed Loop is matched.

The Large Queue state is entered only when $q(t)$ gets above $(2 - T_m)q_T$ according to G_{04}. During this state, the playout buffer dynamics is given by (3) with $d(t) = 1$ and $r(t) = T_m L(i(t))$ (see (5)). When $B > T_m L(i(t))$, G_{44} is matched and $r(t)$ is set equal to B according to R_{44}.

Let us now focus on the Buffering phase, which is described by means of the states Empty Queue and Refilling. During this phase the timer $\tau(t)$ is off. The Buffering phase is entered either from Switch-down, after a switch-down event (G_{67}), or from Closed Loop, when the queue gets empty (G_{02}). This phase can be left only during the Refilling state either when $q(t)$ gets above q_h (G_{70}) or when a switch-down event is triggered (G_{76}).

Let us now analyze the queue dynamics and the guard conditions of Empty Queue and Refilling.

Empty Queue is entered from Closed Loop or Refilling when the queue gets empty (respectively, G_{02} and G_{72}). The queue dynamics is given by (3) with $d(t) = 0$ and $r(t) = T_B L(i(t))$. When $T_B L(i(t)) > B$, G_{22} is matched and $r(t)$ is set equal to B according to R_{22}.

Refilling is entered when $q(t)$ gets above q_d (G_{27}). The queue dynamics is given by (3) with $d(t) = 1$ and $r(t) = T_B L(i(t))$. When $T_B L(i(t)) > B$, G_{77} is matched and $r(t)$ is set equal to B. Refilling is left when: (1) the queue gets empty due to the fact that $r(t) < L(i(t))$ and a transition to Empty Queue occurs (G_{72}); (2) $r(t) < (1 + S_f)L(i(t))$ and a transition to Switch-down (G_{76}) is triggered to select the optimal video level (4); (3) $q(t)$ gets above q_h (G_{70}) and a transition to Closed Loop is triggered.

Let us focus on the Greedy state. In this state the timer $\tau(t)$ is on. This state is entered either from Closed Loop, when the timer fires (G_{03}), or from Switch-up, after a switch-up event is triggered (G_{53}). The Greedy state is left either when the timer fires or $q(t)$ gets below q_d (G_{03}) or when a switch-up event is triggered (G_{35}). The queue dynamics is given by (3) with $d(t) = 1$ and $r(t) = T_M L(i(t))$. When $T_M L(i(t)) > B$, G_{33} is matched and $r(t)$ is set equal to B according to R_{33}.

The Switch-up state is reached from Greedy when a switch-up event is triggered (G_{35}) and $i(t)$ is increased by one according to the reset map R_{35}. When in Switch-up a continuous evolution is not allowed and there are two alternatives: if the selected video level is not yet equal to l_{opt}, G_{55} is matched and the level is increased again by one according to R_{55}; otherwise, the selected video level is l_{opt} and a jump from the Switch-up state to Greedy is triggered (G_{53}). In other terms, this mechanism of repeated jumps implements the video level optimization in (4).

The Switch-down state is reached from Closed Loop or Refilling when a switch-down event is triggered (G_{06} or G_{76}). A similar mechanism of repeated jumps is employed to select l_{opt}. After the selection, a jump to Refilling is triggered (G_{67}).

6. Properties

In this section we derive several properties which can be used to tune the controller parameters.

Property 1. *A necessary condition to allow a switch-up between two adjacent levels l_i and l_{i+1} is that their relative difference $(l_{i+1} - l_i)/l_i$ is less than or equal to T_M/S_f.*

Proof. If a switch-up event has been triggered, during Greedy the guard condition G_{35} has been matched and it holds $r(t) = \min(T_M L(i(t)), B)$. In order to prove necessity, the guard condition has to be matched in both cases, $r(t) = T_M L(i(t))$ and $r(t) = B$. In the case $B > T_M L(i(t))$, $r(t) = T_M L(i(t))$, thus it turns out that $T_M L(i(t)) \geq (1 + S_f)L(i(t) + 1)$, from which follows the thesis. In the case $B < T_M l(i(t))$, $B < T_M L(i(t))$ and $T_M L(i(t)) > B > (1 + S_f)L(i(t) + 1)$ again.

Remark 1. This property must be considered in the design of the video level set $\mathscr{L}$, the maximum throttling signal T_M and the safety factor S_f.

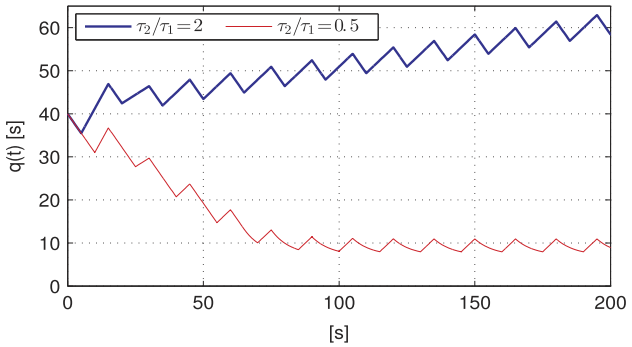

Fig. 4. Queue evolution when (10) is either true (red line) or false (blue line). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Property 2. *A sufficient condition for the boundedness of the playout buffer length $q(t)$ is that:*

$$\frac{\tau_2}{\tau_1} < \frac{1 - T_m}{T_M - 1}. \tag{10}$$

Proof. We prove it by contradiction, i.e. we suppose that $\tau_2/\tau_1 < (1 - T_m)/(T_M - 1)$ and $\lim_{t\to\infty} q(t) = +\infty$ are both true. Hence, it must hold:

$$\forall Q > 0,\ \exists t_Q > 0 \text{ s.t. } \forall t > t_Q :\ q(t) > Q. \tag{11}$$

Let us choose $Q > (2 - T_m)q_T$ and t_Q such that (11) holds. It is trivial to show that, with this choice of Q, the dynamics of the queue is driven by $f_4(x)$ and $f_3(x)$ which alternate according to the timer mechanism for all $t > t_Q$. Without loss of generality we choose $t_0 > t_Q$, which corresponds to the beginning of a Greedy state. Now we have to distinguish two cases. If $l_i > B$, the queue decreases in both states. Thus, there exists $t > t_Q$ such that $q(t) < Q$ and (11) is false. In the case $l_i \le B$, during the Greedy state the queue does not decrease. We can compute its evolution as follows. At the end of the Greedy phase, at time $t_0 + \tau_2$, the queue is equal to:

$$q(t_0 + \tau_2) = q(t_0) + \int_{t_0}^{t_0+\tau_2} \sigma\, dt = q(t_0) + \sigma\tau_2,$$

where $\sigma = \min(T_M - 1, B/l_i - 1) \ge 0$ and $S_f \le \sigma \le T_M - 1$. At the end of the Large Queue phase, at time $t_0 + \tau_2 + \tau_1$, the queue length is:

$$\begin{aligned} q(t_0 + \tau_2 + \tau_1) &= q(t_0 + \tau_2) + \int_{t_0+\tau_2}^{t_0+\tau_2+\tau_1} (T_m - 1)dt \\ &= q(t_0) - (T_m - 1)\tau_1 + \sigma\tau_2. \end{aligned}$$

By iterating this computation, after the kth Greedy-Large Queue period the queue will be equal to:

$$q(t_0 + k(\tau_1 + \tau_2)) = q(t_0) - ((T_m - 1)\tau_1 - \sigma\tau_2)k.$$

From (10) it follows that $(T_m - 1)\tau_1 - \sigma\tau_2 > 0$. Thus, it exists $\bar{k}$ such that $q(t_0 + \bar{k}(\tau_1 + \tau_2)) < Q$, which makes (11) false and proves the property.

Remark 2. In Finamore et al. [26] it has been shown that in 80% of the Youtube video sessions users do not watch the entire video. Thus, from the point of view of the content distribution network it is important to prevent large buffering which causes a waste of network resources. Nevertheless, when (10) holds, the boundedness of the queue length is ensured. Fig. 4 compares the queue dynamics when $\tau_2/\tau_1 = 0.5$ and $\tau_2/\tau_1 = 2$. In this simulation $T_m = 0.1$ and $T_M = 5$. If the initial value is very large ($q(t_0) = 40$ s), the queue length is bounded only in the case $\tau_2/\tau_1 = 0.5$, i.e. when (10) is true, whereas for $\tau_2/\tau_1 = 2$ the queue grows unbounded.

Property 3. *Let us assume that an actuation time-delay τ_a occurs when a switch-down event is triggered and that $B \ge l_0$. A sufficient condition to avoid buffering when a switch-down event occurs from l_i to any other level is that $q_d > (1 - l_0/l_i)\tau_a$.*

Proof. Let us assume that at time $t = 0$ s the current video level is equal to l_i and that the queue length $q(0)$ crosses the threshold q_d from above so that a switch-down event is triggered. To avoid a buffering event, the queue must be large enough to ensure video reproduction at the current level l_i for a time that is greater than the actuation delay τ_a. Let us assume the

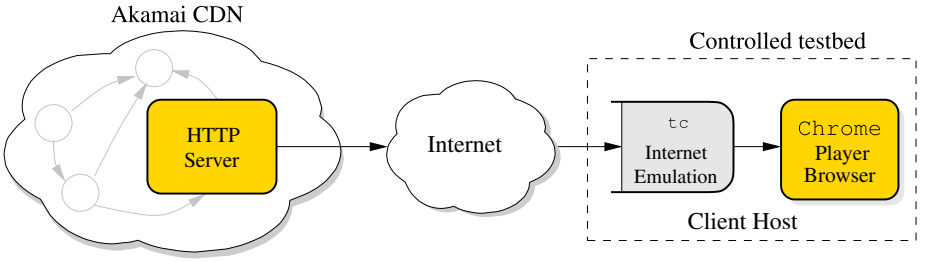

Fig. 5. The experimental testbed.

worst case, i.e. B is equal to the lowest level l_0. Since the filling rate in (3) is $B/l_i = l_0/l_i$, in τ_a seconds the queue drains of a quantity Δq that is equal to:

$$\Delta q = -\int_0^{\tau_a} \left(\frac{l_0}{l_i} - 1\right) dt = \left(1 - \frac{l_0}{l_i}\right) \tau_a. \tag{12}$$

Thus, it turns out that to avoid buffering it must hold $q(0) = q_d > \Delta q$, which proves the property.

Remark 3. In De Cicco and Mascolo [4] it has been shown that the real system is affected by an actuation delay when switch-up/down events are triggered: the delay may cause buffering events in some cases. This property may be used to properly tune the threshold q_d, which is considered an important open issue in De Cicco and Mascolo [4, Section V.F].

7. Model validation

In this section we validate the model of the Akamai adaptive streaming proposed in Section 5.2. The model has been implemented through the Matlab *Hybrid Equations (HyEq) Toolbox* (Sanfelice et al. [27]). The validation has been carried out by comparing the dynamics of key variables of the proposed hybrid automaton model with the ones of the real system obtained by means of Internet experiments. To the purpose the video level set $\mathscr{L}$, the throttling signals T_m, T_B and T_M, the timers τ_{NU}, τ_{GU}, τ_{ND} and τ_{GD} assume the values which have been experimentally found in De Cicco and Mascolo [4] and reported in Table 2.

The experimental results have been obtained by streaming the video "Elephant's Dream", which is served by a HTTP server[4] in the Akamai CDN over the Internet, on a client host connected to the Internet through our campus wired connection (see Fig. 5). The client host is a Linux PC equipped with the Linux traffic shaper *tc* to enable changing the downlink capacity in real time. The *tcpdump* tool has been employed to capture the data traces from the network interface.

We point out that the purpose of the validation is not to quantitatively assess how far the model is from the real system, since we run our experiments over the Internet and cannot neither access the Akamai server nor run experiments in a fully controlled testbed. Our aim is to give a qualitative validation of all the components of the control system in its tight interaction between time-discrete and time-continuous dynamics.

We have considered abrupt step-like downlink bandwidth increases and decreases to validate the behavior of the model when respectively switch-up and switch-down events are triggered. In both cases we have measured the evolution of the key variables of the streaming session before and after the bandwidth variation, which occurs at $t = 150$ s. The experiment lasts 400 s to allow the validation of both the transient and the steady state.

7.1. Validating the switch-up event

We have performed two different experiments to fully characterize the switch-up dynamics.

In the first experiment, we have increased the downlink bandwidth from 1000 to 2500 kbps at $t = 150$ s to provoke a video level switch-up event between the two adjacent levels $l_1 = 700$ kbps and $l_2 = 1500$ kbps.

Fig. 6 compares the dynamics of the queue (Fig. 6 (top)), the received rate (Fig. 6 (middle)), the video level and estimated bandwidth (Fig. 6 (bottom)).

Let us consider Fig. 6 (top), which shows the queue evolutions. When the bandwidth is equal to 1000 kbps the figure shows two alternating dynamics: during the first phase, which in our model corresponds to the Closed Loop state, the queue follows an exponential decrease dynamics, whereas in the second phase, corresponding to the Greedy state, the queue increases linearly. The figure shows that the simulated system models accurately the real system, except for the different ranges within which the queue oscillates. This is due to the fact that in our model we have assumed a constant setpoint q_T,

[4] http://wwwns.akamai.com/hdnetwork/demo/flash/default.html.

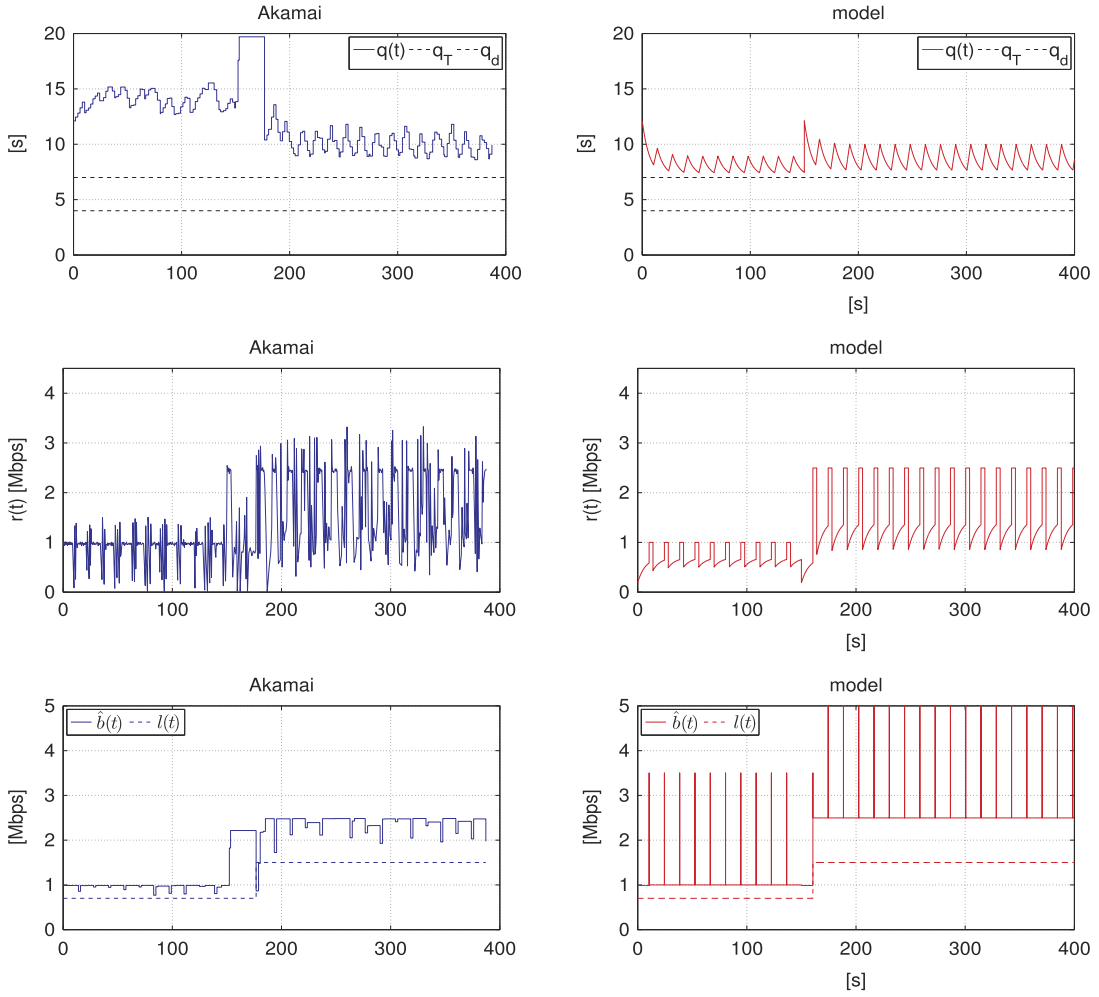

Fig. 6. Queue length (top), received rate (middle), video level and estimated bandwidth (bottom) in the case the downlink capacity increases from 1000 to 2500 kbps at $t = 150$ s.

differently from the real system (see assumptions in Section 5.1). After the bandwidth variation, the figure shows again the two alternating dynamics: the simulated system models accurately the real system after the transient.

Fig. 6 (middle) compares the evolution of the received rate. This figure confirms the experimental findings provided in De Cicco and Mascolo [4] in which an oscillatory received rate was measured. In fact, the real system shows the two alternating dynamics modeled by the Greedy and Normal phases. The proposed model dynamics matches the dynamics of the real system.

Finally, we consider Fig. 6 (bottom) that shows the estimated bandwidth and the video level dynamics. It takes around 30 s to the system to switch up from l_1 to l_2. The proposed model selects the same video level l_2 in response to the sudden increase of the available bandwidth with a shorter transient time of 20 s. This difference is due to the presence of the actuation delay τ_a in the real system, which has been neglected in the proposed model.

In the second experiment we have increased the downlink bandwidth from 400 to 4000 kbps at $t = 150$ s to provoke a video level switch-up event from $l_0 = 300$ kbps to $l_3 = 2500$ kbps. We have considered this experiment to show the dynamics of the system in the case of a switch-up between two non adjacent video levels.

Fig. 7 compares the dynamics of the queue (Fig. 7 (top)), the received rate (Fig. 7 (middle)), and the video level and estimated bandwidth (Fig. 7 (bottom)).

In Fig. 7 (top) the queue evolutions are shown. In the first part of the experiment, when the bandwidth is equal to 400 kbps, we are not able to show the complete evolution of the queue, since the video player reported only few measurement samples. The bandwidth set with the traffic shaper is barely sufficient to allow the playback with the minimum level $l_0 = 300$ kbps. After the bandwidth step variation, occurring at $t = 150$ s, the figure shows two alternating dynamics in a similar way to what we have described in the first experiment: during the first phase, which in our model corresponds to the Closed Loop state, the queue follows an exponential decrease dynamics, whereas in the second phase, corresponding to the Greedy

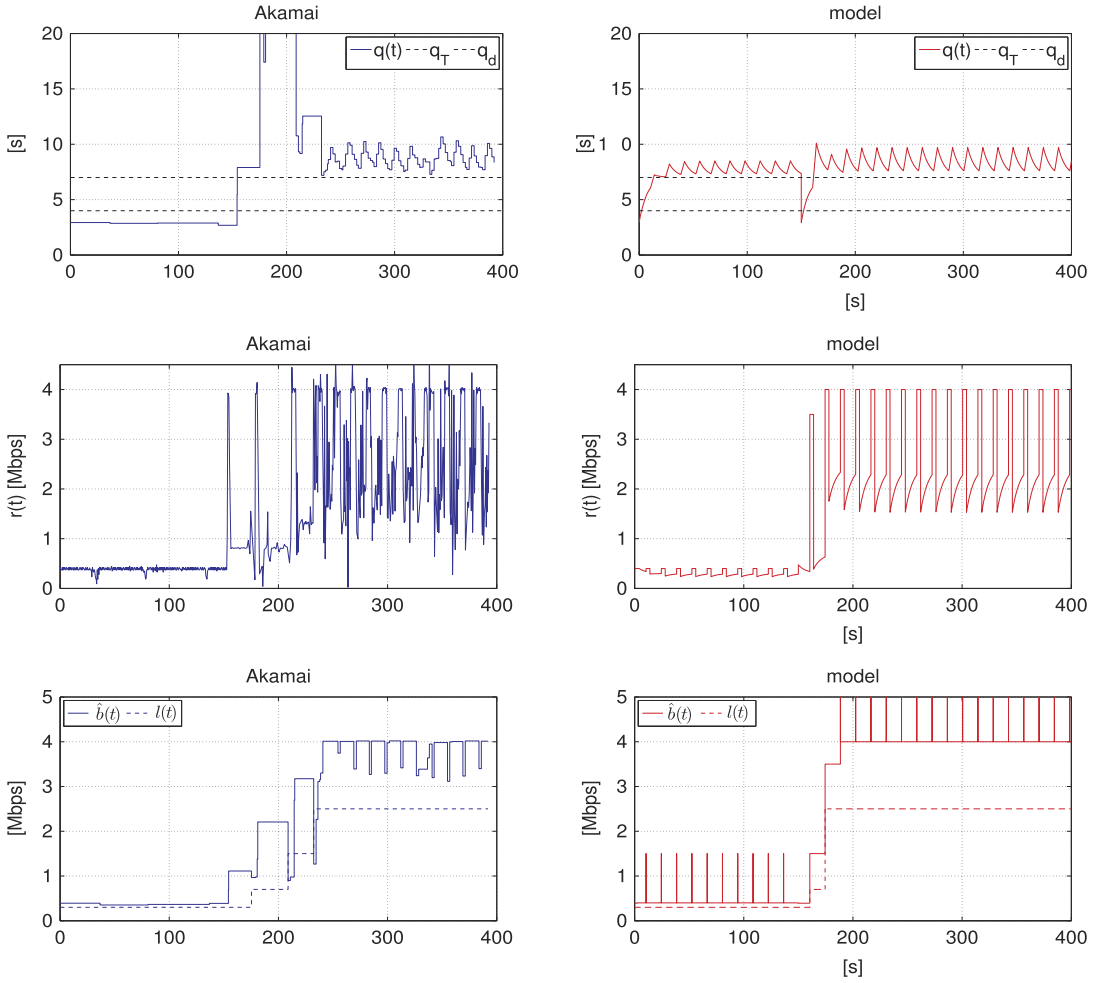

Fig. 7. Queue length (top), received rate (middle), video level and estimated bandwidth (bottom) in the case the downlink capacity increases from 400 to 4000 kbps at $t = 150$ s.

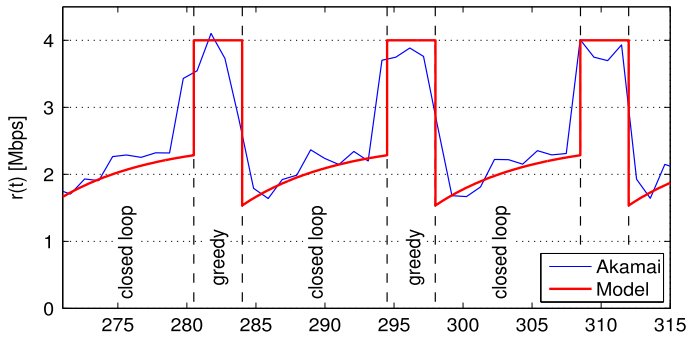

Fig. 8. Zoom of the rate evolution.

state, the queue increases linearly. The figure shows that the simulated system models accurately the real system after the transient: at steady-state the queue oscillates within the range [7.8, 9.6] s.

Fig. 7 (middle) compares the evolution of the received rate. The figure shows that during the Closed Loop state the rate evolution is roughly contained within the same range [1500, 2500] kbps for both the model and the real system. To get a further insight, Fig. 8 provides a zoom of Fig. 7, which shows that the rate dynamics changes according to the logical state. In particular, the figure confirms that: (1) the spikes of the measured received rate are well modeled by the Greedy phase; (2) the modeled rate dynamics fits well the measured received rate during the Closed Loop phases.

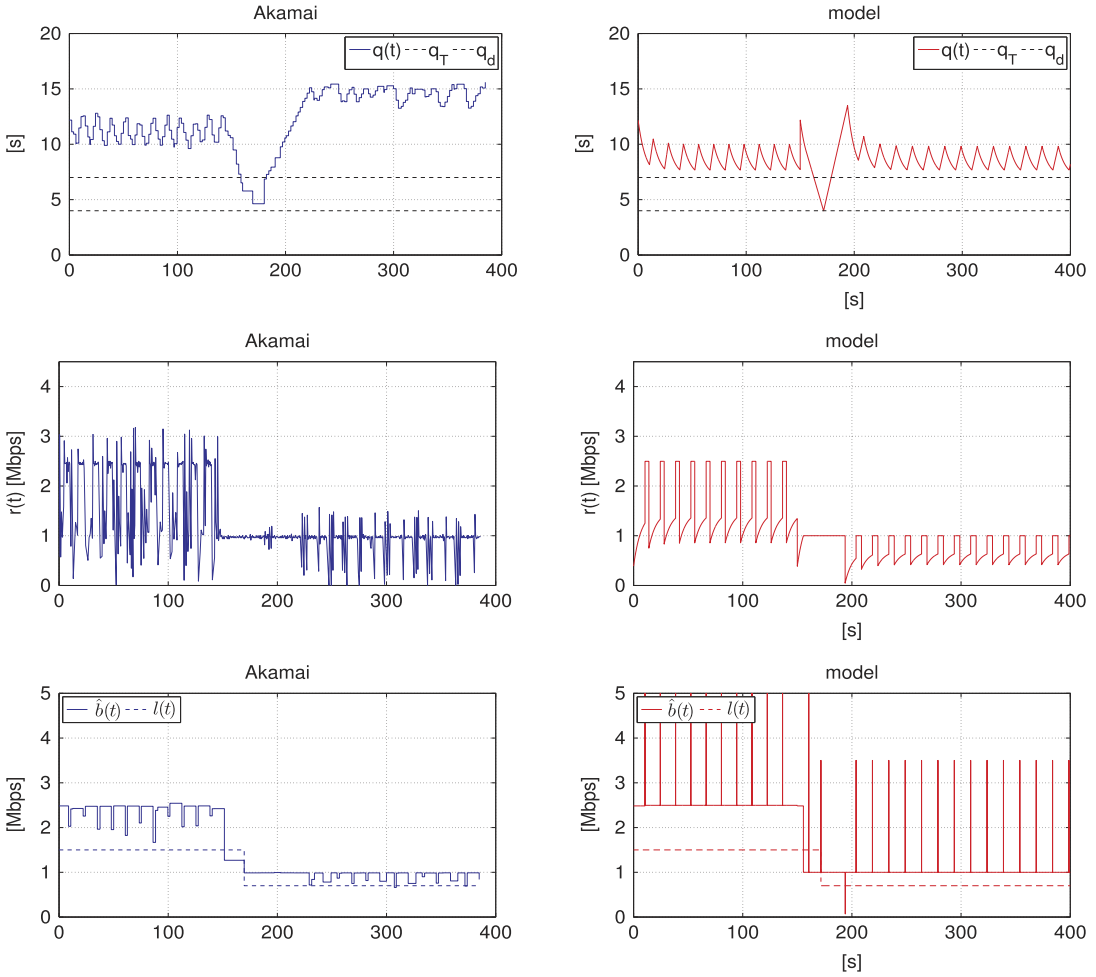

Fig. 9. Queue length (top), received rate (middle), video level and estimated bandwidth (bottom) in the case the downlink capacity decreases from 2500 to 1000 kbps at $t = 150$ s.

Finally, we consider Fig. 7 (bottom) that shows the estimated bandwidth and the video level time evolution. It takes around 70 s to the system to switch up from l_0 to l_3. The proposed model selects the same video level l_3 in response to the sudden increase of the available bandwidth, with a shorter transient time of 25 s. This difference is again due to the presence of the actuation delay τ_a in the real system, which has been neglected in the proposed model. In this case the difference is larger with respect to the first experiment due to the fact that several level switches have to be executed.

7.2. Validating the switch-down event

Similarly to the switch-up event, we have performed two different switch-down experiments.

In the first experiment the downlink bandwidth experiences a step-like decrease from 2500 to 1000 kbps at time $t = 150$ s to provoke a switch-down event from $l_2 = 1500$ kbps to $l_1 = 700$ kbps. In Fig. 9 a comparison between the experimental results and the simulation is shown. As we have done before, we compare the dynamics of the queue, the received rate, the estimated bandwidth, and the video level.

Fig. 9 (top) shows the queue evolutions. Similarly to what we have described in Section 7.1, two alternating dynamics are displayed at steady state: a phase of exponential decrease, which corresponds to Closed Loop, and a phase of linear increase, which corresponds to Greedy. Also in this scenario, the simulation results confirm that the proposed hybrid automaton models the real system accurately, except for the range of oscillation. The difference is due to the queue setpoint q_T, which in the real case changes after the switch-down, unlike what is assumed in the model.

Fig. 9 (middle) shows the received rate evolutions: even though the measurement is quite noisy, the real system and the model exhibit the same pattern. This figure confirms that the duration of the timers which trigger the start and the end of the Normal and Greedy phases changes after a switch-down event. In particular, the Greedy phase duration increases,

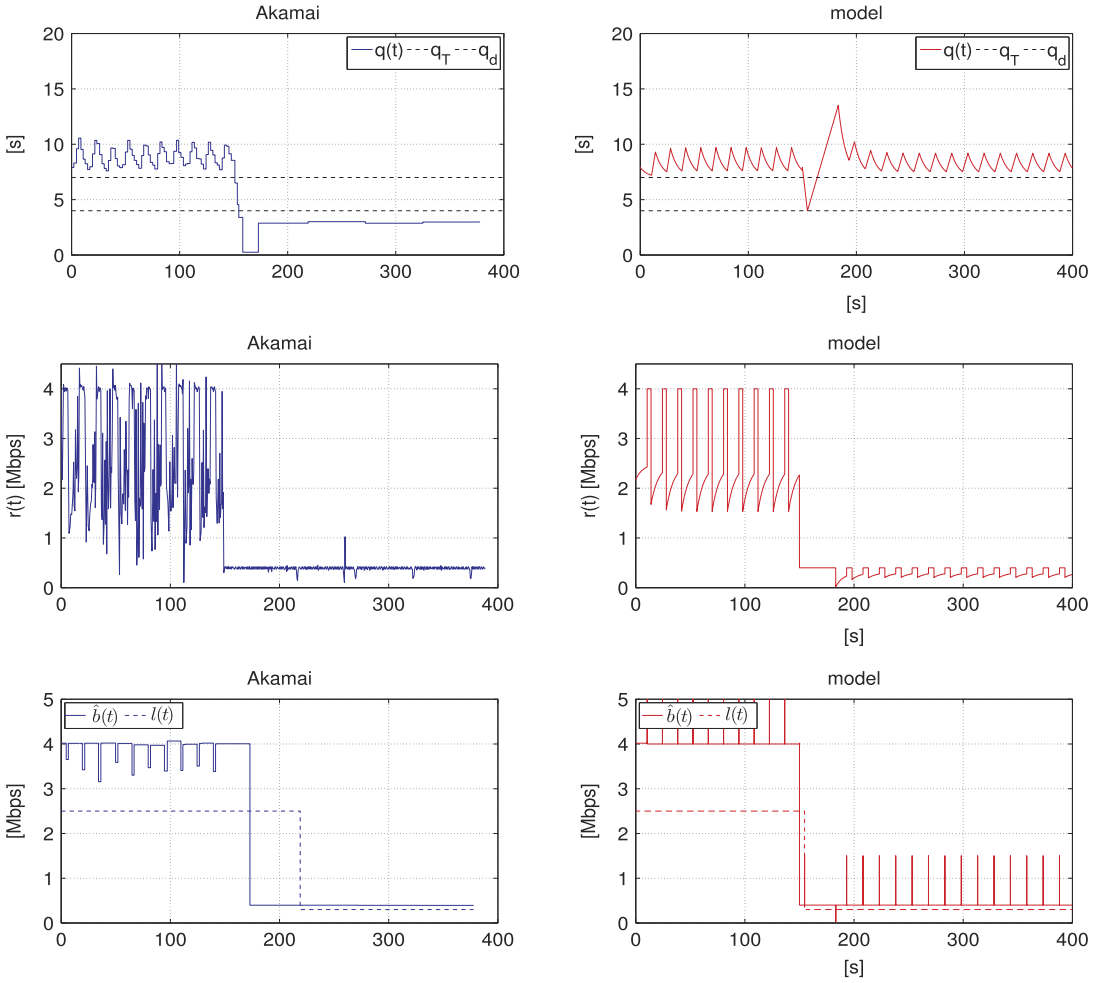

Fig. 10. Queue length (top), received rate (middle), video level and estimated bandwidth (bottom) in the case the downlink capacity decreases from 4000 to 400 kbps at $t = 150$ s.

whereas the Normal phase decreases. We argue that by increasing the duration of the Greedy phase the queue can be filled more quickly. Finally, Fig. 9 (bottom) validates the model regarding the estimated bandwidth and video level evolutions.

In the second experiment the downlink bandwidth experiences a step-like decrease from 4000 to 1000 kbps at time $t = 0$ s to provoke a switch-down event from $l_3 = 2500$ kbps to $l_0 = 300$ kbps.

Fig. 10 (top) shows the comparison between the dynamics of the two queues. Also in this case, similarly to the switch-up case between the same bandwidth values, the client provides only few measurement samples when the bandwidth decreases to 400 kbps. However, the results of the first phase of the experiment confirm that the proposed hybrid automaton models the real system accurately.

Fig. 10 (middle) shows the received rate evolutions which again validate the switching between the Normal and Greedy phases.

Fig. 10 (bottom) shows the estimated bandwidth and video level behavior. The real control system is about 60 s slower in triggering the video level switch-down from l_3 to l_0 due to the unmodeled actuation time-delay.

8. Conclusion

In this paper we have proposed a closed loop hybrid model of the adaptive streaming system employed by Akamai. The model has been validated by comparing simulations and experimental results obtained in a bandwidth-controlled testbed scenario. Moreover, based on the proposed model, we have derived some key properties which allow us to provide some insights into the tuning of the controller parameters. In particular, we show that: (1) video playback interruptions due to actuation delays can be avoided by properly tuning a queue threshold employed by the controller; (2) large buffering, which wastes network resources, can be avoided by properly setting the ratio between the duration of Greedy and Normal phases;

(3) a condition on the relative distance between two adjacent video levels has to be satisfied to ensure the reachability of all the video levels.

Acknowledgment

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

References

[1] Cisco, Cisco visual networking index:forecast and methodology 2012-2017, 2012.
[2] I. Sodagar, The MPEG-DASH Standard for Multimedia Streaming Over the Internet, IEEE MultiMedia 18 (4) (2011) 62–67.
[3] L. De Cicco, G. Cofano, S. Mascolo, A hybrid model of the Akamai adaptive streaming control system. in: Proc. of the 19th IFAC World Congress. Vol. 19. 2014, pp. 3321–3326.
[4] L. De Cicco, S. Mascolo, An adaptive video streaming control system: Modeling, validation, and performance evaluation, IEEE/ACM Trans. Netw. 22 (2) (2014) 526–539.
[5] S. Akhshabi, A. Begen, C. Dovrolis, An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP, in: Proc. of the ACM MMSys. 2011, pp. 157–168.
[6] T. Huang, N. Handigol, B. Heller, N. McKeown, R. Johari, Confused, timid, and unstable: picking a video streaming rate is hard, in: Proc. of the ACM IMC. 2012, pp. 225–238.
[7] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, A.C. Begen, What happens when HTTP adaptive streaming players compete for bandwidth? in: Proc. of the ACM NOSSDAV. 2012, pp. 9–14.
[8] J. Jiang, V. Sekar, H. Zhang, Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with FESTIVE, in: Proc. of the ACM CoNEXT. 2012, pp. 97–108.
[9] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A.C. Begen, D. Oran, Probe and adapt: Rate adaptation for http video streaming at scale, IEEE J. Sel. Areas Commun. (2014) 719–733.
[10] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, A.C. Begen, Server-based traffic shaping for stabilizing oscillating adaptive streaming players, in: Proc. of the ACM NOSSDAV. 2013, pp. 19–24.
[11] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, M. Watson, A buffer-based approach to rate adaptation: Evidence from a large video streaming service, in: Proc. of the ACM SIGCOMM, 2014.
[12] X. Zhu, Z. Li, R. Pan, J. Gahm, H. Hu, Fixing Multi-Client Oscillations in HTTP-based Adaptive Streaming: A Control Theoretic Approach, in: Proc. of the IEEE MMSP, 2013.
[13] L. De Cicco, V. Caldaralo, V. Palmisano, S. Mascolo, ELASTIC: a Client-side Controller for Dynamic Adaptive Streaming over HTTP (DASH), in: Proc. of the Packet Video Workshop, December 2013.
[14] X. Yin, V. Sekar, B. Sinopoli, Toward a principled framework to design adaptive streaming algorithms over HTTP, in: Proc. of the ACM Hotnets, 2014.
[15] J. Lee, S. Bohacek, J.a.P. Hespanha, K. Obraczka, Modeling communication networks with hybrid systems, IEEE/ACM Trans. Netw. 15 (3) (2007) 630–643.
[16] J.P. Hespanha, A model for stochastic hybrid systems with application to communication networks, Nonlinear Anal. TMA 62 (8) (2005) 1353–1383.
[17] A.V. Savkin, A.S. Matveev, P. Rapajic, The medium access control problem for wireless communication networks modelled as hybrid dynamical systems, Nonlinear Anal. TMA 62 (8) (2005) 1384–1393.
[18] C. Albea-Sanchez, A. Seuret, L. Zaccarian, Hybrid control of a three-agent network cluster, in: IEEE Conference on Decision and Control, 2014, pp. 5302–5307.
[19] L. De Cicco, S. Mascolo, V. Palmisano, Feedback control for adaptive live video streaming, in: Proc. of the ACM MMSys. 2011, pp. 145–156.
[20] C.V. Hollot, V. Misra, D. Towsley, W. Gong, Analysis and design of controllers for AQM routers supporting TCP flows, IEEE Trans. Automat. Control 47 (6) (2002) 945–959.
[21] W. Michiels, D. Melchior-Aguilar, S.I. Niculescu, Stability analysis of some classes of TCP/AQM networks, Internat. J. Control 79 (9) (2006) 1136–1144.
[22] S. Mascolo, Congestion control in high-speed communication networks using the Smith principle, Automatica 35 (12) (1999) 1921–1935.
[23] R. Srikant, The Mathematics of Internet Congestion Control, Springer, 2004.
[24] F. Paganini, Z. Wang, J. Doyle, S. Low, Congestion control for high performance, stability, and fairness in general networks, IEEE/ACM Trans. Netw. 13 (1) (2005) 43–56.
[25] J. Lygeros, K.H. Johansson, S.N. Simic, J. Zhang, S.S. Sastry, Dynamical properties of hybrid automata, IEEE Trans. Automat. Control 48 (1) (2003) 2–17.
[26] A. Finamore, M. Mellia, M.M. Munafò, R. Torres, S.G. Rao, Youtube everywhere: Impact of device and infrastructure synergies on user experience, in: Proc. of the ACM IMC. 2011, pp. 345–360.
[27] R. Sanfelice, D. Copp, P. Nanez, A toolbox for simulation of hybrid systems in Matlab/Simulink: Hybrid Equations (HyEq) toolbox, in: Proc. of the HSCC. 2013, pp. 101–106.