A Hybrid Model of the Akamai Adaptive Streaming Control System

L. De Cicco * **G. Cofano** * **S. Mascolo** *

* *Politecnico di Bari, Dipartimento di Ingegneria Elettrica e dell'Informazione, Via Orabona 4, Bari, Italy (e-mails: {l.decicco,g.cofano,mascolo}@poliba.it)*

AbstractVideo streaming is becoming the application generating the largest fraction of the Internet traffic. Adaptive video streaming adds to classic video streaming the feature 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. In this paper we focus on the adaptive video streaming control system employed by Akamai, a major CDN 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. Our purpose is to derive and validate a closed loop mathematical model of the control system, which turns out to be a hybrid automaton. The model is then analyzed to derive key properties which can be used to properly tune the controller parameters.

1. INTRODUCTION AND RELATED WORK

Multimedia applications generate an ever increasing fraction of the Internet traffic (Cisco, 2012). Video streaming is the most important application driving this trend. Examples such as YouTube, delivering user-generated video content, or NetFlix, which streams movies and TV series, give evidence of the wide diffusion such applications have reached.

Adaptive video streaming represents a key innovation *wrt* classic progressive download streaming. With adaptive streaming the video bitrate can be throttled on-the-fly to match the time-varying available bandwidth. Moreover, start-up latency can be minimized and video playback interruptions can be avoided.

Today, the leading approach for implementing 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, i.e. 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 servers (Sodagar, 2011).

Recently, several studies have analyzed the issues of client-side adaptive streamers. Typically, such streaming systems are characterized by a on-off traffic pattern Akhshabi et al. (2012): 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. (2012) it is shown that such on-off traffic pattern is the key factor causing unfair bandwidth utilization, server bandwidth underutilization and flickering of the player requested video level. To tackle these issues, several adaptive streaming algorithms have been proposed so far. In Jiang et al. (2012) FESTIVE has been proposed to address the fairness issues arising in a multi-client scenario. In Li et al. (2014) the algorithm PANDA is proposed: a controller dynamically computes the segment inter-request time to address fairness issues and video bitrate oscillations. By using a control theoretical approach, De Cicco et al. (2013) have designed a stream-switching controller which is able to avoid OFF phases without the cooperation of the server.

In this paper 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. Since its source code cannot be inspected, in De Cicco and Mascolo (in press) 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 on the client and the server.

Our purpose is to go further and derive a mathematical model of the closed loop system, which turns out to be a hybrid automaton and allows us to provide rules for the tuning of the controller parameters. The closed loop system is composed of 1) the Akamai stream-switching controller, 2) the actuator, which throttles the sending rate, and 3) the video playout buffer, which represents the plant. The controller and the plant are located at the client, whereas the actuator is placed at the server.

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

2. 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.

In classic progressive download streaming the video content is available only at a bitrate equal to l. Such an approach cannot guarantee both maximum link utilization and the avoidance of buffering events. On the other hand, with adaptive video streaming the video bitrate can be dynamically throttled 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 while providing the highest video quality.

Among the proposed adaptive video streaming approaches, the stream-switching is getting a wide adoption due to its implementation and deployment simplicity (De Cicco et al., 2011). 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 $\mathscr{L} = \{l_0, l_1, ..., l_{N-1}\}$, a set of video level indices $\mathscr{I} = \{0, 1, ..., N-1\}$, and the bijective mapping $l : \mathscr{I} \to \mathscr{L}$ that associates the video level index to the video level bitrate. 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 $l(t) = l(i(t))$. It is important to notice that video level switches can only occur at the beginning of a new segment transmission. Finally, in the following we will use $l(i(t))$ or $l_{i(t)}$ equivalently with some abuse of notation.

3. THE PLAYOUT BUFFER DYNAMICS

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

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

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

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

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

4. THE AKAMAI HYBRID MODEL

4.1 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, 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 based on 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.

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:

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

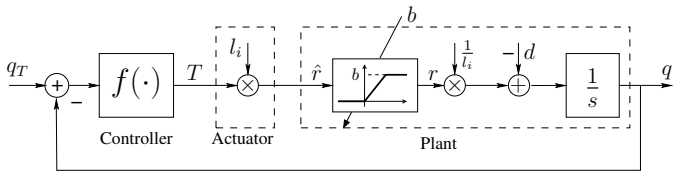

Figure 1. The playout buffer control system

The playout buffer control system

The playout buffer control system is composed of three elements, as shown in Figure 1:

(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, modelling 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 state is left, a periodical switching between Normal and Greedy states is established. To enforce this periodical behaviour a timer is used. 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, on short average, by filling the TCP buffer at the desired sending rate. In practice, since the dynamics of the TCP congestion control is faster than that of the control system, the received rate $r(t)$ can be modelled 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 Figure 1.

Figure 1 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 us to analyze any practical traffic scenario with a bottleneck link (Mascolo, 1999). Thus, we will analyze the system behavior in response to a step input of amplitude B. With respect to (De Cicco and Mascolo, in press) 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.

4.2 The proposed hybrid automaton

In this section we propose a closed loop model of the system described in Section 4.1. The state-dependent and event-triggered dynamics and the discontinuous elements, such as the saturation block, can be properly modelled by means of a hybrid system. In particular, we model the system as a hybrid automaton (Lygeros et al., 2003).

Figure 2 shows the proposed hybrid automaton, which is denoted as $\mathcal{H}$. The state of the system is given by the continuous component:

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

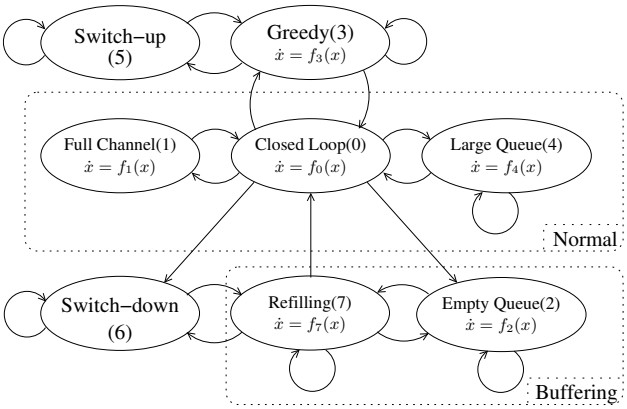

Figure 2. The proposed hybrid automaton $\mathcal{H}$

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 $x \in \mathcal{I} \times [0, B] \times \mathbb{R}_+ \times \mathbb{R}_+ \times \mathbb{R}_+ \times \mathbb{R}_+$.

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.

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 behaviour of the system during the Normal phase described in Section 4.1. It has been necessary to employ three states to deal with the two discontinuities 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 discontinuity 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 4.1. During the Empty Queue state, which is entered when $q(t) = 0$, the video reproduction is interrupted, i.e. $d(t) = 0$, to quickly fill the queue; during Refilling, which is entered from either Switch-down or Empty Queue states, the video reproduction is on, i.e. $d(t) = 1$.
- Greedy (3) models the Greedy phase described in Section 4.1, 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.

The complete model is omitted due to space limitations and is given in the technical report [3] along with a detailed explanation of each element.

5. PROPERTIES

In this Section we derive several properties which can be used to tune the controller parameters. Proofs are omitted due to space limitations.

Proposition 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.

Remark 2. This property must be considered in the design of the video level set $\mathscr{L}$, the throttling T_M and the safety factor S_f.

Proposition 3. A sufficient condition for the boundedness of the playout buffer length $q(t)$ is that:

$$\frac{\tau_2}{\tau_1} < \frac{1 - T_m}{T_M - 1}. \tag{8}$$

Remark 4. In Finamore et al. (2011) 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 (8) holds, the boundedness of the queue length is ensured.

Proposition 5. Let us assume that an actuation time-delay τ_a occurs when a switch-down event is triggered and that $B \geq l_0$. A sufficient condition to avoid buffering when a switch-down event occurs from l_i to any other level is that $q_d > (1 - l_0/l_i)\tau_a$.

Remark 6. In De Cicco and Mascolo (in press) 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 (in press, Section V.F).

6. MODEL VALIDATION

In this section we validate the model of the Akamai adaptive streaming proposed in Section 4.2, that has been implemented through the Matlab *Hybrid Equations (HyEq) Toolbox* Sanfelice et al. (2013). 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. We employ the values that are reported in Table 1.

The experimental results have been obtained by playing the video "*Elephant's Dream*" served by the Akamai server [4] on a Linux PC equipped with the Linux *traffic shaper* `tc` to enable changing the downlink capacity in real time (for more information about the testbed and measurement see De Cicco and Mascolo (in press)). The

[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] http://c3lab.poliba.it/downloads/akamai-model-tr.pdf

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

Table 1. 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
Queue	q_T	setpoint	7s
	q_d	switch-down threshold	4s
	q_h	Refilling threshold	12s
Throttling	T_M	Greedy throttling	5
	T_m	Large Queue throttling	0.1
	T_B	Buffering throttling	2
Timers duration	τ_{NU}	Normal phase after switch-up	10.5s
	τ_{GU}	Greedy phase after switch-up	3.5s
	τ_{ND}	Normal phase after switch-down	6.5s
	τ_{GD}	Greedy phase after switch-down	8.5s

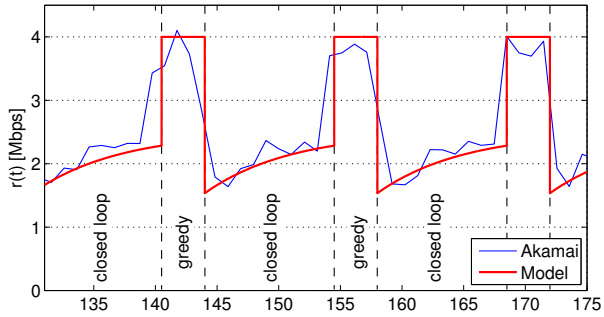

Figure 4. Zoom of the rate evolution

client host was connected to the Akamai server through our campus wired connection.

6.1 Validating the switch-up event

In this experiment we increase the downlink bandwidth from 400kbps to 4000kbps at $t = 0$s with the aim of provoking a video level switch-up event from $l_0 = 300$kbps to $l_3 = 2500$kbps. The initial queue length $q(0)$ has been set to 14s. Figure 3 compares the dynamics of the queue (Figure 3 (left)), the received rate (Figure 3 (center)), and the video level and estimated bandwidth (Figure 3 (right)).

Let us consider Figure 3 (left), which shows the queue evolutions. 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 after the transient: at steady-state the queue oscillates within the range $[7.8, 9.6]$s.

Figure 3 (center) compares the evolution of the received rate. The figure shows that during the Closed Loop state the rate evolution is contained within the same range $[1535, 2585]$kbps (delimited by the dashed lines) for both the model and the real system. To get a further insight, we provide a zoom of Figure 3 (center) in Figure 4, 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 modelled by the Greedy phase; 2) the modelled rate dynamics fits well the measured received rate during the Closed Loop phases.

Finally, we consider Figure 3 (right) that shows the estimated bandwidth and the video level time evolution. It takes around 70s 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 25s. 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.

6.2 Validating the switch-down event

In this experiment the downlink bandwidth experiences a step-like decrease from 4000kbps to 1000kbps at time $t = 0$s to provoke a switch-down event from $l_3 = 2500$kbps to $l_1 = 700$kbps. The initial queue length $q(0)$ has been set to 7s. In Figure 5 a comparison between the experimental result and the simulation is shown. As before, we compare the dynamics of the queue, the received rate and the video level.

Figure 5 (left) shows the queue evolutions. Similarly to what we have described in Section 6.1, 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.

Figure 5 (center) shows the received rate evolutions. This figure confirms that the duration of the timers which trigger the start and the end of the Normal and Greedy phases change after a switch-down event. In particular, the Greedy phase duration increases, whereas the Normal phase decreases. We argue that by increasing the duration of the Greedy phase the queue can be filled more quickly.

Figure 5 (right) shows the estimated bandwidth and video level behavior. The real control system is about 10s slower in the video level switch-down from l_3 to l_1 due to the unmodelled actuation time-delay.

7. 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, we give insights on the tuning of the controller parameters by deriving some key properties. In particular, we show that: 1) video playback interruptions due to actuation delays can be avoided by properly tuning the 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.

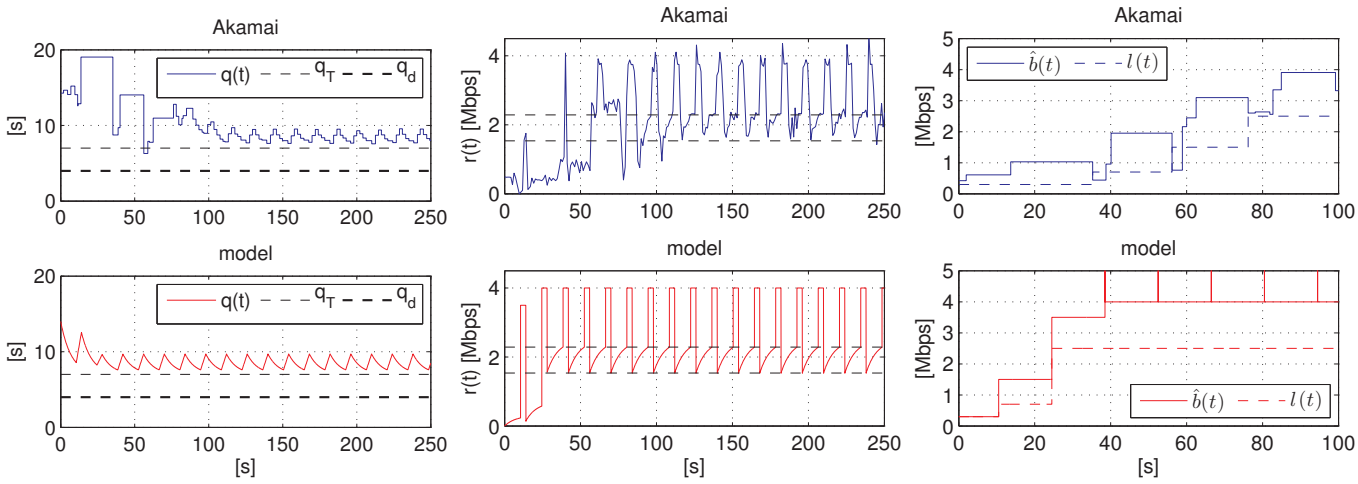

Figure 3. Queue length (left), received rate (center), video level and estimated bandwidth (right) in the case the downlink capacity increases from 400kbps to 4000kbps at $t = 0$s

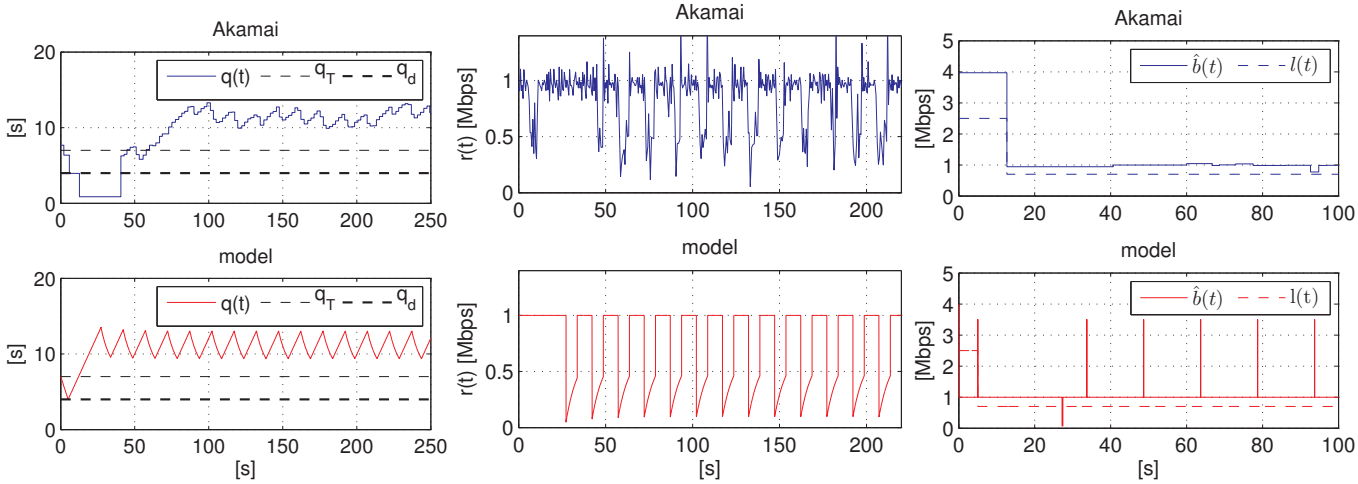

Figure 5. Queue length (left), received rate (center), video level and estimated bandwidth (right) in the case the downlink capacity decreases from 4000kbps to 1000kbps at $t = 0$s

REFERENCES

Akhshabi, S., Anantakrishnan, L., Dovrolis, C., and Begen, A.C. (2012). What happens when http adaptive streaming players compete for bandwidth? In *Proc. of ACM NOSSDAV '12*, 9–14.

Cisco (2012). Cisco visual networking index:forecast and methodology 2012-2017.

De Cicco, L., Caldaralo, V., Palmisano, V., and Mascolo, S. (2013). Elastic: a client-side controller for dynamic adaptive streaming over http (dash). In *Proc. of Packet Video Workshop '13*.

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

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

Finamore, A., Mellia, M., Munafò, M.M., Torres, R., and Rao, S.G. (2011). Youtube everywhere: Impact of device and infrastructure synergies on user experience. In *Proc. of ACM IMC '11*, 345–360.

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

Li, Z., Zhu, X., Gahm, J., Pan, R., Hu, H., Begen, A.C., and Oran, D. (2014). Probe and adapt: Rate adaptation for http video streaming at scale. *IEEE Journal on Selected Areas in Communications*, 32(4), 719–733.

Lygeros, J., Johansson, K.H., Simic, S.N., Zhang, J., and Sastry, S.S. (2003). Dynamical properties of hybrid automata. *IEEE Transaction on Automatic Control*, 48(1), 2–17.

Mascolo, S. (1999). Congestion control in high-speed communication networks using the smith principle. *Automatica*, 35(12), 1921–1935.

Sanfelice, R., Copp, D., and Nanez, P. (2013). A toolbox for simulation of hybrid systems in matlab/simulink: hybrid equations (hyeq) toolbox. In *Proc. of HSCC '13*, 101–106.

Sodagar, I. (2011). The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE MultiMedia*, 18(4), 62–67.