2011 IEEE International Symposium on Computer-Aided Control System Design (CACSD)
Part of 2011 IEEE Multi-Conference on Systems and Control
Denver, CO, USA. September 28-30, 2011

An Experimental Evaluation of Akamai Adaptive Video Streaming over HSDPA networks

Luca De Cicco, Saverio Mascolo, and Chaouki T. Abdallah

Abstract— **Adaptive video streaming is a relevant advancement with respect to classic progressive download streaming such as the one employed by YouTube. Building upon its content delivery network (CDN), Akamai recently started to offer High Definition (HD) adaptive video streaming using HTTP. Nowadays, not only the amount of Internet video traffic is always increasing but also the number of users accessing the Internet using wireless links. In this paper we experimentally investigate the switching algorithm employed by Akamai to implement video quality adaptation over a High Speed Downlink Data Packet Access (HSDPA) link. In order to assess the Quality of Experience we measure goodput, TCP friendliness, and video reproduction continuity. Main results are: 1) Akamai flows are not able to achieve the fair share when competing with a TCP greedy flow due to the conservativeness of the stream-switching algorithm; 2) when the link is shared with a greedy TCP connection in 50% of the experiments the video reproduction was paused for more than 19% of the experiment duration.**

I. Introduction

Nowadays the Internet is becoming the most important platform to deliver audio/video delay-sensitive traffic. According to a recent report published by Cisco, the traffic generated by video applications will account in 2014 for 91% of the global traffic [3]. Important applications that feed this trend are YouTube, which delivers user-generated video content, and Skype audio/video conference over IP.

In this paper we focus on adaptive streaming that represents a key advancement *wrt* classic progressive download streaming such as the one employed by YouTube. With download streaming, the video is a static file that is delivered as any data file using greedy TCP connections. The receiver employs a player buffer that allows the file to be stored in advance *wrt* the playing time in order to mitigate video interruptions. On the other hand, with adaptive streaming, the video source is adapted on-the-fly to the network available bandwidth so that live video content can be delivered in real-time and users can watch videos at the maximum bit rate that is allowed by the time-varying available bandwidth.

Mobile multimedia delivery is becoming more and more ubiquitous thanks to the development of broadband wireless technologies such as IEEE 802.11 for local access and 3G-4G for large area coverage and due to the spreading of smart phones with powerful CPUs. In particular, the High Speed Downlink Packet Access (HSDPA) is an upgrade of UMTS that is getting worldwide deployment and that can provide downlink peak rates of several Mbps, which is more than one order of magnitude improvement with respect to the 100kbps offered by GSM EDGE few years ago [8].

In this paper we investigate the performance of the adaptive streaming service provided by Akamai, which is the worldwide leading Content Delivery Network (CDN), over a HSDPA link. The service is called Akamai High Definition Video Streaming (AHDVS) and aims at delivering HD videos over Internet connections using the Akamai CDN. The Akamai system employs the *stream-switching* technique: the server encodes the video content at different bit rates and it switches from one video version to another based on client feedbacks such as the measured available bandwidth. It can be said that the Akamai approach is the leading commercial one since it is employed by the Apple HTTP-based streaming, the Microsoft IIS server, the Adobe Dynamic Streaming, and Move Networks.

In [4] we focused on the dynamic response of Akamai switch-stream algorithm in response to abrupt variations of the available bandwidth in a controlled network environment. We have found that the control logic employed by AHDVS to dynamically select the suitable video level is implemented at the client and it is based on an estimate of the available bandwidth. If, on one hand, executing the adaptation logic at the client-side has the advantage of keeping the streaming server implementation simple, on the other hand, from a control theoretic point of view, this has the drawback of introducing a variable time delay in the control loop which makes the design of an effective controller more challenging.

We have shown that the adaptation algorithm implemented by Akamai is not able to avoid short playout interruptions when abrupt decreases of the available bandwidth occur and in other cases the conservativeness of the algorithm leads to network underutilization [4].

In this paper we show the results of a measurement campaign we have carried out over the downlink of a HSDPA network. In order to assess the Quality of Experience (QoE) the following metrics have been measured for each experiment: 1) goodput, 2) Jain Fairness Index (JFI) to estimate the TCP friendliness, and 3) paused time percentage to estimate the video playback continuity.

The rest of the paper is organized as follows: in Section II a brief review of adaptive streaming algorithms proposed in the literature is presented; in Section III we discuss the main characteristics of the quality adaptation logic employed

Luca De Cicco is post-doc at Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari, Via Orabona 4, Italy (email: ldecicco@gmail.com)

Saverio Mascolo is with the Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari, Via Orabona 4, Italy (email: mascolo@poliba.it)

Chaouki T. Abdallah is with the Electrical and Computer Engineering Department, MSC01 1100 1 University of New Mexico, Albuquerque, NM, USA (email: chaouki@ece.unm.edu)

978-1-4577-1067-4/11/$26.00 ©2011 IEEE

by Akamai HD Video Streaming; in Section IV the obtained experimental results are discussed; finally, Section V summarizes the major findings and concludes the paper.

II. Related Works

Even though TCP has been considered not suitable for the transport of video streaming, 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 [10]; ii) TCP has built-in NAT traversal functionalities; iii) it is easy to be deployed within Content Delivery Networks (CDN) [10]; iv) TCP delivers most part of the Internet traffic and it is able to guarantee the stability of the network by means of an efficient congestion control algorithm [9].

Stream-switching algorithms encode the raw video content at increasing bitrates resulting into N versions, i.e. *video levels*; an algorithm dynamically chooses the video level that matches the user's available bandwidth. Stream-switching 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 [10], [1], [7], [2], [6]. Another important advantage of such algorithms is that they do not rely on particular functionalities of the employed codec and thus can be made codec-agnostic. The disadvantages 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. From the point of view of the control algorithm this means that the actuation variable is quantized.

Recently, the stream-switching approach is being adopted by several adaptive streaming commercial products. In particular we cite: 1) the *IIS Smooth Streaming* [10] which is provided by Microsoft; 2) the *Adobe Dynamic Streaming* [6] developed by Adobe and recently included by Amazon in the EC2 platform; 3) the *HTTP Adaptive Live Streaming* solution [7] which has been proposed for standardization within IETF by Apple; 4) *Move Networks* that provides live adaptive streaming service [2] to several TV networks; 5) *Netflix* that implements stream switching in its video on demand streaming platform.

In spite of the wide acceptance of this quality adaptation paradigm in the multimedia industry, the literature regarding the stream-switching approach is not exhaustive. In particular, the few published papers that focus on designing a stream-switching control algorithm often rely on heuristic-based arguments [5]. This has the clear drawback of making the system dynamics difficult to be predicted and mathematically analyzed.

III. Akamai Switch-Stream Control Algorithm

Before presenting the details of the control algorithm employed by Akamai to dynamically switch among the video levels, we briefly discuss the client-server protocol used by this algorithm.

AHDVS employs HTTP connections to stream data from the server to the client. The adaptation algorithm is executed at the client in a Flash application. By capturing and analyzing the traffic between the Akamai server and the client we have observed that the client issues a number of HTTP requests to the server throughout all the duration of the video streaming carrying feedbacks and commands to the server using a separate TCP socket.

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

TABLE I
Set of available video levels $\mathscr{L}$

At first, the client connects to the Akamai server [1], 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, a GET HTTP request is sent to the server which points to a SMIL compliant file. In the SMIL file the base URL of the video, the available video levels, and the corresponding encoding bit-rates are provided.

After that, the client parses the SMIL file to reconstruct the complete URLs of the available video levels and selects the corresponding video level based on the quality adaptation algorithm. All the videos available on the demo website are encoded at five different bitrates as shown in Table I. In particular, the *video level* bitrate $l(t)$ can assume values in the discrete *set of available video levels* $\mathscr{L} = \{l_0, \ldots, l_4\}$. All the video levels are encoded at 30 frames per second (fps) using H.264 codec with a group of picture (GOP) of length 36, so that two consecutive I frames are 1.2s apart. This means that, since a video switch can occur only at the boundaries of GOPs, video levels can change only each 1.2s. Finally, the audio is encoded with Advanced Audio Coding (AAC) at 128 kbps bitrate.

After the SMIL file gets parsed, at time $t = t_0$, the client issues the first POST request specifying several parameters. Among those, the most important parameters are `cmd`, that specifies a command the client issues on the server, and `lvl1`, that specifies several feedback variables $\mathbf{F}(t)$ such as: 1) the receiver buffering time $q(t)$ measured in seconds, 2) the receiver buffer target $q_T(t)$ measured in seconds, 3) the received video frame rate $f(t)$ measured in frames per second (fps), 4) the estimated bandwidth $B(t)$ measured in kilobit per second (kbps), 5) the received goodput $r(t)$ measured in kbps, 6) the current received video level bitrate $l(t)$ measured in kbps; 7) the playing time $t_p(t)$ measured in seconds; 8) the estimated round trip time $R(t)$.

The quality adaptation algorithm starts at $t = t_0$. For a generic time instant $t_i > t_0$ the client issues commands via HTTP POST requests to the server in order to select the suitable video level.

Table II reports all the possible commands c_i that the client

	Command	Args	Occurrence (%)
c_1	throttle	1	~80%
c_2	rtt-test	0	~15%
c_3	SWITCH_UP	5	~2%
c_4	BUFFER_FAILURE	7	~2%
c_5	log	2	~1%

TABLE II

COMMANDS ISSUED BY THE CLIENT TO THE STREAMING SERVER VIA THE CMD PARAMETER

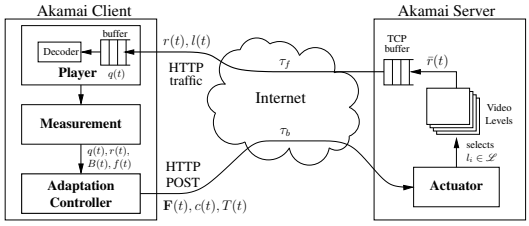


Fig. 1. A block diagram of the control architecture employed by AHDVS

can issue on the servers along with the number of arguments and the occurrence percentage. The first two commands are issued periodically, throttle with a median inter-departure time of about 2s and rtt-test with a median inter-departure time of roughly 11s. On the other hand, log, SWITCH_UP and BUFFER_FAILURE are event triggered commands.

In [4] we have shown that the throttle command specifies a single argument, the *throttle percentage* $T(t)$, that it is used to control the receiver buffering time $q(t)$ as we will discuss in Section III-A. The rtt-test command is issued to periodically actively probe for the available bandwidth and to measure the round trip time $R(t)$ (RTT) of the connection.

Finally, the two event-based commands SWITCH_UP and BUFFER_FAILURE are sent from the client to ask the server to respectively switch up or down the video level $l(t)$.

Figure 1 shows a block diagram of the overall control architecture employed by AHDVS. The server is connected to the client through an Internet connection characterized by a forward connection delay τ_f and a backward connection delay τ_b. Figure 1 shows that the three main components of the control loop, i.e. measurement, adaptation controller, and actuator, are connected through the Internet so that the control loop is affected by an overall delay $\tau = \tau_f + \tau_b$.

The client receives the video flow at level $l(t) \in \mathscr{L}$ over an HTTP connection at a rate $r(t)$. The received video is stored in a playout buffer, whose instantaneous length expressed in seconds is $q(t)$, which is drained by the decoder at the current received video level. A *measurement* module feeds the values of the buffer length $q(t)$, the received goodput $r(t)$, the bandwidth $B(t)$, and the decoded frame rate $f(t)$ to the adaptation controller.

The adaptation controller is made of two modules: 1) a *playout buffering time controller* whose goal is to drive the buffering time to a target value; 2) a *stream-switching logic* that selects the appropriate video level to be streamed by the server.

A. The playout buffering time controller

In [4] we have shown that the control law implemented by Akamai to regulate the buffer length $q(t)$ is a proportional controller that takes the error $q_T(t) - q(t)$ as the input and whose output is the throttle percentage $T(t)$:

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

The throttle percentage $T(t)$ is used to set the rate $\overline{r}(t)$ at which the Akamai server feeds the TCP socket buffer with the current video level $l(t)$ as follows:

$$\overline{r}(t) = l(t)\frac{T(t)}{100} \quad (2)$$

The rationale of controlling $\overline{r}(t)$ is to induce, on average, a TCP sending rate that is equal to $\overline{r}(t)$. This means that when the throttle percentage is above 100% the server can stream the video at a rate that is above the encoding bitrate $l(t)$. It is important to stress that, in the case of live streaming, it is not possible for the server to supply a video at a rate that is above the encoding bitrate for a long period, since the video source is not pre-encoded.

By looking at (1) we find that 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 $l(t)$. On the other hand, when the error $q_T(t) - q(t)$ increases, $T(t)$ increases accordingly in order to allow $\overline{r}(t)$ to increase so that the buffer can be filled quickly.

B. The stream switching logic

Let us now focus on the *stream-switching logic,* the heuristic-based controller that dynamically decides which video level $l(t) \in \mathscr{L}$ has to be sent by the server, based on the estimated bandwidth, the current video level, the playout buffer length, and the frame rate. In particular, and based on the debug information provided by the Akamai Client and on the experiments we have run, the stream-switching heuristic works as follows.

The client periodically issues rtt-test commands that have the effect of setting at the server a throttling percentage of 500%, thus asking the server to periodically send the video in greedy mode. In this way Akamai actively probes for extra available bandwidth and estimates the RTT $R(t)$ under congestion. Based on the estimated value of the RTT, the client computes a *safety factor* S. By parsing the debug information in order to collect the pairs $(R(t), S(t))$ it was possible to run a linear regression over the dataset which yielded to the following static linear model ($R(t)$ is expressed in seconds):

$$f(R(t)) = 2.5R(t) + 0.15$$

By considering the collected samples we found that when $R(t) > 0.1$s the safety factor remains set to 0.4, whereas when $R(t) < 0.02$s, it is set to 0.2. Thus, we can conclude

that the complete model for $S(R(t))$ is the following:

$$S(R(t)) = \begin{cases} 0.2 & 0 < R(t) < 0.02\text{s} \\ 2.5R(t) + 0.15 & 0.02\text{s} \leq R(t) \leq 0.1\text{s} \\ 0.4 & R(t) > 0.1\text{s} \end{cases} \quad (3)$$

It is worth noting that in the case of HSDPA connections round trip times are likely to be higher than 0.1s so that we can assume in this specific scenario a fixed safety factor S equal to 0.4.

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

$$L_i^H(t) = l_i \cdot (1 + S(t)) \; ; \; L_i^L = l_i \cdot 1.2 \quad (4)$$

A switch up to a higher video level l_i occurs when $B(t) > L_i^H(t)$, which means that if, for instance, the RTT is above 0.1 s and thus $S(R(t)) = 0.4$, to allow a switch up to level l_i the estimated bandwidth must be at least 40% higher than l_i. This seems to be a conservative approach that leads to network underutilization and, as a consequence, to a reduced QoE.

On the other hand, the switch down event occurs when:

$$q(t) < q_L(t) \quad (5)$$

where $q_L(t)$ is another threshold that is smaller than the queue target[1] $q_T(t)$. When (5) holds, a `BUFFER_FAILURE` is sent and the new video level $l_i < l(t)$ is selected. In particular, the highest video level $l_i \in \mathscr{L}$ satisfying the following condition:

$$B(t) > 1.2 \cdot l_i = L_i^L$$

is selected. Thus, in order to select the level l_i, the currently estimated bandwidth $B(t)$ must be at least 20% above l_i. Moreover, in [4] we have shown that when `SWITCH_UP` and `BUFFER_FAILURE` commands are sent from the client, the *actuator*, which is located at the server, takes an average delay of $\tau_{su} \simeq 14$s and $\tau_{sd} \simeq 7$s respectively, to actuate these commands.

Finally, it is worth noting that the overall system exhibits a very complex dynamics due to the interaction of two closed-loop dynamics: the stream-switching logic, which has been designed using heuristic arguments, and the playout buffering time controller. As a consequence, it is very complex to develop a mathematical analysis as well as to tune control variables to satisfy key design requirements such as settling times and steady state errors.

IV. Experimental results

In this Section we show the results obtained by an experimental evaluation of AHDVS over a commercial HSDPA network. In order to carry out the experimental evaluation we have employed the video sequence "*Elephant's Dream*", available on the Akamai demo website [1], since its duration is long enough for a careful experimental evaluation. The receiving host is a laptop equipped with an Ubuntu Linux running a 2.6.32 kernel.

[1]The identification of $q_L(t)$ has not been carried out.

A total of 125 experiments have been carried out at different hours of the day during a two months period. We have considered the following QoE metrics:

1) the *received goodput* which is computed as:

$$g(t_k) = \frac{d(t_k) - d(t_{k-1})}{t_k - t_{k-1}}$$

where $d(t_k)$ is the total amount of data received up to the k-the sampling time t_k. The sampling interval is equal to 1s.

2) In scenarios where two flows share the HSDPA downlink it has been evaluated the *Jain Fairness Index* (JFI) to estimate the TCP friendliness as follows:

$$JFI = \frac{(\sum_{i=1}^{N} g_i)^2}{N \sum_{i=1}^{N} g_i^2}$$

where g_i is the average goodput obtained by the i-th flow. The minimum fairness is obtained when one link grabs all the available bandwidth and the others obtain no bandwidth. In this case the JFI is $1/N$. The maximum fairness, i.e. $JFI = 1$, is obtained when all the flows obtain the same bandwidth share.

3) The *paused time percentage* P estimates the video playback continuity is defined as follows:

$$P = \frac{\Delta P}{\Delta T} 100 = \frac{\Delta T - t_p(\Delta T)}{\Delta T} 100$$

where ΔP is the total time the player is paused and ΔT is the duration of the experiment. The paused time ΔP is computed as the difference between the duration of the experiment ΔT and the playing time evaluated at the end of the experiment $t_p(\Delta T)$.

In the following we report the metrics described above in the form of cumulative distribution functions. Moreover, the dynamics of a subset of the feedback variables provided in the POST commands via the `lvl1` parameter are shown. In this paper we have considered two scenarios: 1) one video streaming flow and 2) one streaming flow versus one TCP greedy flow.

A. One video streaming flow

In this Section we present the results obtained when a single video streaming flow accesses the HSDPA downlink for 300s with no concurrent traffic.

We have chosen one experiment that is able to show the typical behaviour of the control algorithm employed by AHDVS and that let us validate the proposed model discussed in Section III. Figure 2 shows the dynamics of the received video level $l(t)$, the estimated bandwidth $B(t)$, the `SWITCH_UP` and `BUFFER_FAIL` events, and the high thresholds $L_i^H(t)$ evaluated by using[2] (4). Figure 3 shows the dynamics of the buffer length $q(t)$ along with the target

[2]In order to evaluate $S(t)$ we employ the equation (3) which expresses $S(t)$ as a function of the measured RTT $R(t)$. The RTT sample $R(t)$ is taken from the sixth parameter of the `lvl1` variable that contains all the feedback information to the server.

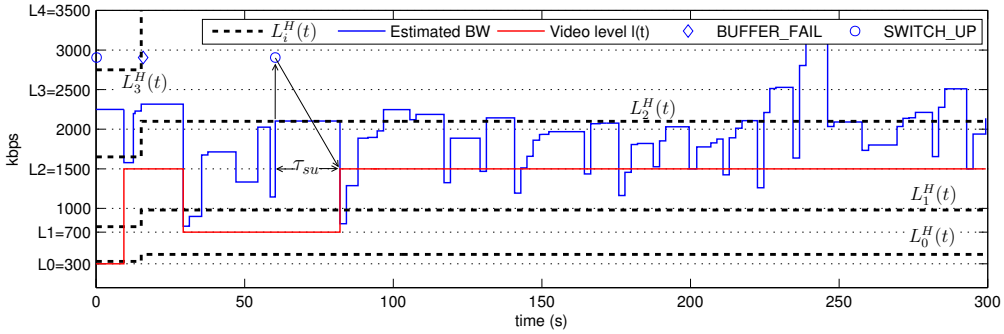
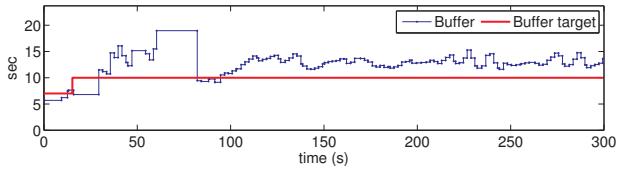
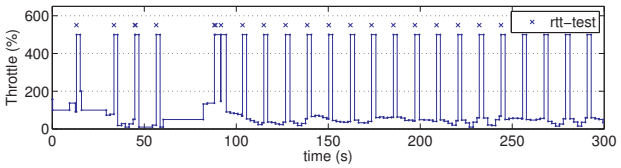


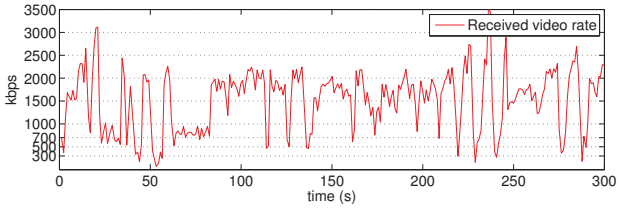
Fig. 2. Video level $l(t)$, estimated bandwidth $B(t)$ dynamics and `BUFFER_FAIL`, `SWITCH_UP` events in the case of one Akamai video streaming flow is received over the HSDPA downlink



(a) Buffer length $q(t)$ and buffer target length $q_T(t)$

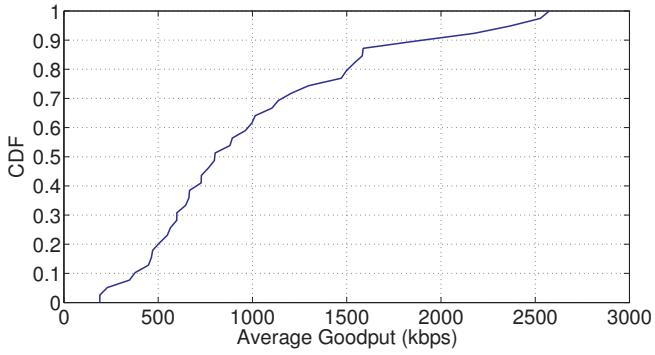


(b) Throttle percentage $T(t)$ and `rtt-test` events

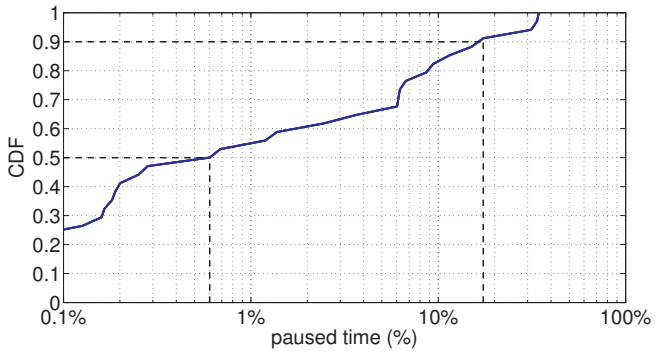


(c) Goodput

Fig. 3. The case of one Akamai video streaming flow received over the HSDPA downlink



(a)



(b)

Fig. 4. Cumulative distribution functions of (a) goodput and (b) paused time percentage (semilogarithmic scale) in the case a single flow is received on the HSDPA downlink

buffer length $q_T(t)$, the throttle percentage $T(t)$ along with `rtt-test` events, and the goodput measured at the receiver.

When the experiment starts the video level $l(t)$ is set to l_0, however, since the estimated bandwidth is above $L_2^H(t)$, according to (4), a `SWITCH_UP` command is immediately sent to the server asking to set the video level to l_2. The video level is actually switched up after a delay of 9.4s. Let us now look at the dynamics of the buffer length $q(t)$ and the target buffer length $q_T(t)$ shown in Figure 3(a). When the video level is switched up, the threshold $q_T(t)$ is increased from 7s to 10s. Nevertheless, when the buffer length $q(t)$ drops to less than 7s at time $t = 15.9$s a `BUFFER_FAIL` command is sent asking the server to set the video level to l_1. The video level is kept to l_1 until the estimated bandwidth $B(t)$ becomes slightly higher than $L_2^H(t)$ at time $t = 60$s so that a `SWITCH_UP` command is sent and it is actuated after a delay τ_{su} of around 24s. By looking at the dynamics of $q(t)$ shown in Figure 3(a) we see that at the steady state an offset is present since a proportional controller is used to control the queue length. Finally, Figure 3(c) shows the received goodput that exhibits remarkable oscillations due to the fact that AHDVS periodically switches between two states: in the *normal* state the video sending rate is bounded by the maximum sending rate $\overline{r}(t)$ given by (2), whereas each time a `rtt-test` command is issued AHDVS enters the *greedy-mode* state and for a short time interval of around 5s the sending rate is limited by the available bandwidth.

Figure 4 shows the cumulative distribution functions of the average received goodput and of the paused time percentage

in the considered scenario. The CDF of the average received goodput (Figure 4(a)) shows a median value of around 700 kbps corresponding to the level l_1 with a resolution of 640x360 and a 90th percentile of around 2000kbps which supports a video level l_2 with a resolution of 640x360. Let us now consider the paused time percentage CDF shown in Figure 4(b): the median value of P is 0.6% which means that in half the experiments the video has been paused for less than the 0.6% of the experiment duration. However, in 20% of the cases the video is paused for about 10% of the experiment duration, resulting in roughly 30s of interruption. Finally, the standard deviation of P in this scenario is 9.7%.

In conclusion this basic scenario shows that AHDVS is not able to deliver high definition video over the HSDPA downlink and even if in half the experiments the video interruptions are negligible, in more than 20% of the experiments the video reproduction is affected by interruptions of more than 30s.

B. One video streaming flow with one concurrent TCP flow

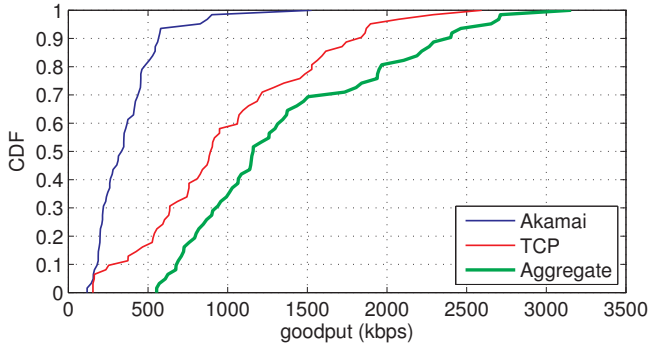
In this scenario we investigate the performance of AHDVS stream-switching algorithm when sharing the HSDPA downlink with one greedy TCP flow, such as in the case of a parallel download session. The video streaming session is started at $t = 0$, a greedy TCP connection is started at $t = 100$s and it is stopped at $t = 250$s. Due to space constraints we do not show the dynamics of the stream-switching controller and we show the CDFs of goodput and paused time percentage.

Figure 5(a) shows the CDF of the TCP goodput, the goodput obtained by AHDVS and the aggregate goodput. The CDF of the aggregate goodput shows that the median value is 1160 kbps, meaning that the median fair share is around 600 kbps. Nevertheless, the median value of the goodput obtained by the Akamai flow is only 334 kbps, whereas the median goodput obtained by the TCP flow is 892 kbps.

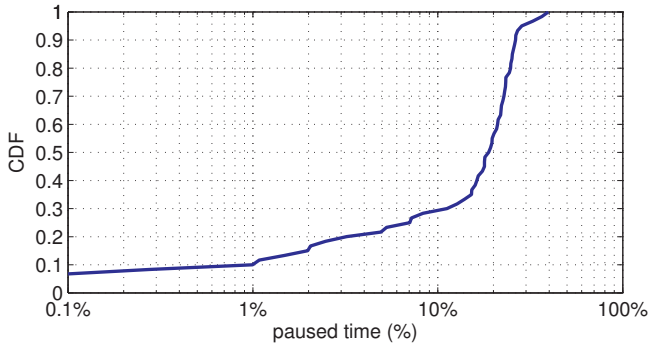
In order to measure the fairness we computed the JFI and we obtained a low median fairness of 0.77 with a standard deviation of 0.1. This indicates that, due to the conservativeness of Akamai stream-switching algorithm, in 50% of the experiments the Akamai flow is not able to obtain the fair share and it can only support the lowest video level $l_0 = 300$ kbps.

Finally, Figure 5(b) shows the CDF of the paused time percentage achieved in this scenario. The figure shows that the median value of the paused time percentage is 18.8%, corresponding to a total playout interruption of 56 s due to re-buffering.

The results obtained in this scenario clearly show that AHDVS is not able to correctly adapt the video level in order to obtain the fair-share and to avoid playout interruptions due to the conservativeness of its stream-switching algorithm.



(a)



(b)

Fig. 5. (a) CDF of Akamai, TCP and aggregate goodput; (b) CDF of the paused time percentage (semilogarithmic scale) in the case one video flow shares the HSDPA downlink with a TCP connection

V. Conclusions

In this paper we have shown the results of an experimental evaluation of the Akamai HD Video Streaming (AHDVS) service over a HSDPA link in two different scenarios. The main findings are: 1) the stream-switching algorithm employs a conservative safety factor to trigger video level switch-ups; 2) AHDVS flows are not able to achieve the fair share when competing with a TCP greedy flow due to the conservativeness of the stream-switching algorithm; 3) when the link is shared with a greedy TCP connection in 50% of the experiments the video reproduction was paused for more than 18.8% of the experiment duration.

References

[1] Akamai HD Network Demo. http://wwwns.akamai.com/hdnetwork/demo/flash.

[2] Move Networks HD adaptive video streaming. http://www.movenetworkshd.com.

[3] Cisco Inc. Cisco Visual Networking Index:Forecast and Methodology 2009-2014. *White Paper*, June 2010.

[4] L. De Cicco and S. Mascolo. An Experimental Investigation of the Akamai Adaptive Video Streaming. In *Proc. of USAB 2010*, Nov. 4–5, 2010.

[5] K. Dovstam, T. Einarsson, W. Eklof, and M. Kampmann. Stream-switching with in-stream transmission rate probing for adaptive mobile multimedia streaming. In *Proc. of 17th IEEE International conference on Image Processing*, pages 2889–2892, 2010.

[6] D. Hassoun. Dynamic streaming in flash media server 3.5. Available: http://www.adobe.com/devnet/flashmediaserver/.

[7] R. Pantos and W. May. HTTP Live Streaming. *IETF Draft*, June 2010.

[8] M. Sauter. *Beyond 3G - Bringing Networks, Terminals and the Web Together*. John Wiley & Sons, 2008.

[9] V. Jacobson. Congestion avoidance and control. In *Proc. of ACM SIGCOMM '88*, pages 314–329, 1988.

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