Adaptation Algorithm for Adaptive Streaming over HTTP

Konstantin Miller*, Emanuele Quacchio†, Gianluca Gennari† and Adam Wolisz*
*Technische Universität Berlin, Germany
Email: {konstantin.miller, adam.wolisz}@tu-berlin.de
†STMicroelectronics, Milan, Italy
Email: emanuele.quacchio@st.com, gennarone@fastwebnet.it

***Abstract*—Internet video makes up a significant part of the Internet traffic and its fraction is constantly growing. In order to guarantee best user experience throughout different network access technologies with dynamically varying network conditions, it is fundamental to adopt technologies enabling a proper delivery of the media content. One of such technologies is adaptive streaming. It allows to dynamically adapt the bit-rate of the stream to varying network conditions. There are various approaches to adaptive streaming. In our work, we focus on the receiver-driven approach where the media file is subdivided into segments, each of the segments is provided at multiple bit-rates, and the task of the client is to select the appropriate bit-rate for each of the segments. With this approach, the challenges are (i) to properly estimate the dynamics of the available network throughput, (ii) to control the filling level of the client buffer in order to avoid underflows resulting in playback interruptions, (iii) to maximize the quality of the stream, while avoiding unnecessary quality fluctuations, and, finally, (iv) to minimize the delay between the user's request and the start of the playback. During our work, we designed and implemented a receiver-driven adaptation algorithm for adaptive streaming that does not rely on cross-layer information or server assistance. We integrated the algorithm with a prototype implementation of a streaming client based on the MPEG DASH (Dynamic Adaptive Streaming over HTTP) standard. We evaluated the implemented prototype in real-world scenarios and found that it performes remarkably well even under challenging network conditions. Further, it exhibits stable and fair operation if a common link is shared among multiple clients.**

I. Introduction

Fast home broadband connections, the growing popularity of connected TV sets, and the spread of WiFi/3G enabled mobile terminals are among the key drivers for the growing popularity of Internet video. In addition to faster connections, technological improvements in video coding and video delivery are making it possible for more consumers to watch video content online. In order to guarantee best user experience throughput different network access technologies with dynamically varying network conditions, it is fundamental to adopt technologies enabling a proper delivery of the media content. One of such technologies is adaptive video streaming.

The concept of adaptive video streaming is based on the idea to adapt the bandwidth required by the video stream to the throughput available on the network path from the stream source to the client. The adaptation is performed by varying the quality of the streamed video and thus its bit-rate, that is, the number of bits required to encode one second of playback. One of the approaches here is to divide the video stream into segments and to encode each of the segments in multiple quality levels, called representations. Based on his estimation of the available throughput, a client might request subsequent segments at different quality levels in order to cope with varying network conditions. The algorithmic process of deciding the optimal representation for each of the segments in order to optimize the viewing experience is a key element and one of the major challenges in adaptive streaming systems. In particular, the challenges for the client are (i) to properly estimate the dynamics of the available throughput, (ii) to control the filling level of the local buffer in order to avoid underflows resulting in playback interruptions, (iii) to maximize the quality of the stream, while avoiding unnecessary quality fluctuations, and, finally, (iv) to minimize the delay between the user's request and the start of the playback.

Recent years have seen the advent and widespread of streaming technologies based on the HTTP/TCP protocols. The use of HTTP/TCP offers several advantages. It is cost-effective since standard web servers and caches can be used. Furthermore, it is firewall-friendly since almost all firewalls are configured to support HTTP connections. Finally, HTTP is stateless and the streaming session is managed by the client, reducing the load on the server and thus allowing for better scalability. On the other hand, operation on top of HTTP/TCP reveals additional challenges since adaptation on top of TCPs congestion control algorithm creates nested control loops.

In our study, we designed an algorithm for dynamic selection of segment bit-rate based on the network conditions. We implemented the algorithm as a platform independent software library. Further, based on the GStreamer framework [1], we implemented a client for the recently adopted standard for adaptive streaming over HTTP: MPEG DASH [2]. We integrated the developed algorithm with the client and evaluated its performance in real-world scenarios.

The outline of the paper is as follows. Section II contains references to related work. In section III we describe our system model, assumptions and notation. Section IV describes the adaptation algorithm including adaptation goals, input arguments and configuration parameters. It also provides the pseudo-code of the algorithm. Section V presents the results of the evaluation. Finally, section VI concludes the paper.

II. Related Work

Liu et al. [3] propose an adaptation algorithm for adaptive streaming over HTTP and evaluate it using the network simulator ns-2 [4] in the presence of exponential and constant bit-rate background traffic. The results show that the algorithm occasionally fails to select the best possible representation. Further, it exhibits significant fluctuations of the selected video quality.

Jarnikov el al. [5] proposed to calculate the adaptation strategy using a Markov decision process. With this approach, an optimal strategy is calculated off-line for a given distribution function of segment download times. The objective function is a linear function giving constant penalty to playback interruptions and changes of video quality, and a reward proportional to the selected video quality. We argue that, despite of the attractiveness of this approach, selection of a fixed distribution function and performing the calculation off-line does not appropriately account for the dynamics of the throughput. The authors performed a numerical evaluation of the approach using fixed, uniform and normal distributions of the available bandwidth.

Akhshabi et al. [6] experimentally evaluated two major commercial players and one open source player that use the technology of adaptive streaming over HTTP. In their study, the authors focus on three aspects: reaction to persistent or short-term throughput changes, the ability of two players to properly operate on a shared network path, and if the player is able to sustain a short playback delay and thus perform well with live content. The authors identified significant inefficiences in each of the studied players.

Finally, Zink et al. [7] empirically investigate the impacts of video quality variations on human perception.

III. System model

We assume that we are given a video stream consisting of n segments, each containing τ seconds of playback. We call τ the duration of a segment. (In order to simplify the presentation, we assume that all segments have equal duration. The presented mechanisms can, however, easily be extended to the general case.) Each segment is available in different representations. We denote by $\mathcal{R}$ the set of representations available for the given video stream. The representations in $\mathcal{R}$ might differ in various aspects. They might contain 2D or 3D video, different video quality levels, different spatial resolutions, etc. With MPEG DASH, the set of available representations is obtained by the client prior to the begin of the streaming session by downloading the XML-based Media Presentation Description (MPD) file.

Assume that a user would like to watch the video by streaming it from a given server to a given client over some network path. We denote by $\mathcal{R}_f$ the set of feasible representations for a given client. On a device lacking 3D capabilities $\mathcal{R}_f$, e.g., does not include representations providing 3D video. Further, $\mathcal{R}_f$ does not include representations providing a spacial resolution that is not supported by the client, etc.

In the presented work, we focus on dynamically selecting a representation from $\mathcal{R}_f$ based on its average bit-rate. Additionally, it might be reasonable to include further metrics into the selection process. A client supporting 3D might decide to prefer a 3D representation to a 2D representation with a similar bit-rate. We do not consider this feature here. Note, however, that some features like, e.g., adapting the spacial resolution of the stream after the user resized the player window can easily be implemented by dynamically adapting the set of feasible representations $\mathcal{R}_f$.

In the following, we denote the average bit-rate of a representation $r \in \mathcal{R}_f$ by $\hat{\rho}_r$. We write $r_{\max} \in \mathcal{R}_f$ ($r_{\min} \in \mathcal{R}_f$) for the representation with the highest (lowest) bit-rate in $\mathcal{R}_f$. We write $r^{\uparrow} \in \mathcal{R}_f$ ($r^{\downarrow} \in \mathcal{R}_f$) for the representation with the next higher (lower) bit-rate with respect to r, if $r \neq r_{\max}$ ($r \neq r_{\min}$).

We assume that after the user requests to watch the video stream, (i) the client downloads the segments in chronological order, (ii) each of the segments is downloaded in exactly one representation, and that (iii) the downloads are non-preemptive (that is, the download of segment $i-1$ must be completed before the start of the download of segment i). We denote by $t_i^b \in \mathbb{R}_+$ ($t_i^e \in \mathbb{R}_+$) the time (in seconds) of the begin (end) of the download of segment i. W.l.o.g., we assume $t_1^b = 0$. We denote by t_{start} the time when the first bits of the first segment arrive at the client.

After the data is downloaded, it is decoded and presented to the user. It is not required that a segment is downloaded completely before it can be decoded and played. Although the decoder requires a certain minimum amount of bytes to trigger decoding, it is usually less than the duration of a segment. During the time between arriving at the client and being played, the data is locally buffered. We denote by $\beta(t)$ the buffer level in seconds of playback at time t. We denote by $\beta_{\min}(t)$ the minimum buffer level observed during a time interval with duration Δ_β, where Δ_β is a configuration parameter: $\beta_{\min}(t) = \min\left\{\beta(t') \,|\, t' \in \left[\left\lfloor \frac{t}{\Delta_\beta}\right\rfloor \Delta_\beta, \left\lfloor \frac{t}{\Delta_\beta}\right\rfloor \Delta_\beta + \Delta_\beta\right]\right\}$. Here, $\lfloor\cdot\rfloor$ denotes the largest previous integer.

For a downloaded segment i, we denote by $r_i \in \mathcal{R}_f$ the representation that was selected for the download. We denote by $\tilde{\rho}_i = \left(\hat{\rho}_{r_i} \cdot \tau\right) / \left(t_i^e - t_i^b\right)$ the segment throughput of segment i. Note that if defined in this way, the segment throughput depends on the round-trip delay. We denote by

$$\tilde{\rho}(t_1, t_2) = \frac{\sum_{i=1}^{n(t_2)} \tilde{\rho}_i \cdot \left|[t_i^b, t_i^e] \cap [t_1, t_2]\right|}{\sum_{i=1}^{n(t_2)} \left|[t_i^b, t_i^e] \cap [t_1, t_2]\right|}$$

the average segment throughput during the time interval $[t_1,\, t_2]$. Here, $n(t_2)$ is the number of segments that have been fully downloaded until time t_2, while $\left|[a,b]\right| = b - a$ is the length of the interval $[a, b]$.

We denote by $\sigma_i = \left(r_i,\, t_i^b,\, t_i^e\right)$ a tuple holding information about the download of a segment i.

IV. Adaptation

The challenge for an adaptation algorithm arises from the fact that the available throughput on an Internet path

considerably fluctuates due to multiple reasons. (Throughout the paper we use the term *available throughput* to name the throughput a given TCP flow would exhibit on a given network path at a given point in time.) The reasons include on the one hand cross-traffic, interference, fading, etc. On the other hand, performing adaptation on top of TCP, which performs retransmissions and deploys congestion control mechanisms adapting its throughput, increases the complexity, since it results in nested control loops.

A. Adaptation goals

The goal of adaptation is to optimize viewing experience subject to throughput dynamics of the TCP flow on the network path from the server to the client. We identified the following optimization goals. (i) Avoid interruptions of playback due to buffer underruns: $\beta(t) > 0$ for all $t > t_{\text{start}}$. (ii) Maximize the minimum and the average video quality. (iii) Minimize the number of video quality shifts. (iv) Minimize the time after the user requests to view the video and before the start of the playback, which is equivalent to minimizing t_{start}.

Note that goals (ii) and (iii) constitute a trade-off since downloading each segment in highest possible representation results in frequent changes of playback quality whenever the dynamics of the available throughput exhibit strong fluctuations. Also, goals (ii) and (iv) constitute a trade-off since t_{start} is minimized if the first segment is downloaded at lowest representation.

In our work, we assigned the first goal highest priority, whereas we tried to achieve a balance between goals (ii) and (iii). We resolve the trade-off between goals (ii) and (iv) by downloading the first segment at lowest representation but introducing a fast start phase that increases video quality in a more aggressive way. By introducing various configuration parameters, we provide the possibility to adjust the behavior of the adaptation to individual viewing preferences.

B. Description of the algorithm

The algorithm's pseudo-code is provided in Algorithm 1.

We assume that the algorithm we describe here is invoked at time t, immediately after the download of segment $n(t)$ is completed. In order to efficiently adapt the video quality to the dynamics of available throughput the algorithm takes two input arguments: (i) information about the dynamics of the available throughput in the past: $(\sigma_i)_{i=1,\ldots,n(t)}$, (ii) buffer level $\beta(t')$, $t' \in [0,t]$. Although, strictly speaking, buffer level is a function of throughput, in practice it is easier to monitor the buffer level separately since the computation of buffer level as a function of throughput is imprecise due to several issues like, e.g., bit-rate variation over the duration of a segment or playback interruptions due to a slow decoder.

The algorithm has two output arguments: (i) the representation to be selected for the download of the next segment: $r_{n(t)+1}$, and (ii) the minimum buffer level in seconds of playback when the download must be started: B_{delay}. The purpose of the second output argument B_{delay} is to be able to delay subsequent downloads in order to reduce the buffer level if we already arrived at highest representation or if the available throughput does not allow to select a higher representation. The download of segment $n(t)+1$ shall not be started before the buffer level $\beta(t)$ falls below B_{delay}.

Additionally to the two input arguments, the algorithms can be configured using several configuration parameters: $0 \leq B_{\text{min}} < B_{\text{low}} < B_{\text{high}}$, $\Delta_\beta > 0$, $\Delta_t > 0$, $0 < \alpha_1, \ldots, \alpha_5 \leq 1$. They will be described in more details in the following.

The algorithm always selects the lowest representation for the first segment to be downloaded. The benefit of this approach is that it minimizes t_{start} or, equivalently, the delay

Algorithm 1: ADAPTATION ALGORITHM

Input: $(\sigma_i)_{i=1,\ldots,n(t)}$
Output: $r_{n(t)+1}$, B_{delay}

static runningFastStart := true;
$B_{\text{delay}} := 0$;
$r_{n(t)+1} := r_{n(t)}$;
if runningFastStart ...
  $\wedge\ r_{n(t)} \neq r_{\max}$ ...
  $\wedge\ \beta_{\min}(t_1) \leq \beta_{\min}(t_2)\, \forall t_1 < t_2 \leq t$ ...
  $\wedge\ r_{n(t)} \leq \alpha_1 \cdot \tilde{\rho}\big(t - \Delta_t, t\big)$ **then**
    if $\beta(t) < B_{\text{min}}$ **then**
      if $r^{\uparrow}_{n(t)} \leq \alpha_2 \cdot \tilde{\rho}\big(t - \Delta_t, t\big)$ **then**
        $r_{n(t)+1} := r^{\uparrow}_{n(t)}$;
    else if $\beta(t) < B_{\text{low}}$ **then**
      if $r^{\uparrow}_{n(t)} \leq \alpha_3 \cdot \tilde{\rho}\big(t - \Delta_t, t\big)$ **then**
        $r_{n(t)+1} := r^{\uparrow}_{n(t)}$;
    else
      if $r^{\uparrow}_{n(t)} \leq \alpha_4 \cdot \tilde{\rho}\big(t - \Delta_t, t\big)$ **then**
        $r_{n(t)+1} := r^{\uparrow}_{n(t)}$;
      if $\beta(t) > B_{\text{high}}$ **then**
        $B_{\text{delay}} := B_{\text{high}} - \tau$;
else
    runningFastStart := false;
    if $\beta(t) < B_{\text{min}}$ **then**
      $r_{n(t)+1} := r_{\min}$;
    else if $\beta(t) < B_{\text{low}}$ **then**
      if $r_{n(t)} \neq r_{\min} \wedge r_{n(t)} \geq \tilde{\rho}_{n(t)}$ **then**
        $r_{n(t)+1} := r^{\downarrow}_{n(t)}$;
    else if $\beta(t) < B_{\text{high}}$ **then**
      if $r_{n(t)} = r_{\max} \vee r^{\uparrow}_{n(t)} \geq \alpha_5 \cdot \tilde{\rho}\big(t - \Delta_t, t\big)$ **then**
        $B_{\text{delay}} := \max\left(\beta(t) - \tau,\, B_{\text{opt}}\right)$;
    else
      if $r_{n(t)} = r_{\max} \vee r^{\uparrow}_{n(t)} \geq \alpha_5 \cdot \tilde{\rho}\big(t - \Delta_t, t\big)$ **then**
        $B_{\text{delay}} := \max\left(\beta(t) - \tau,\, B_{\text{opt}}\right)$;
      else
        $r_{n(t)+1} := r^{\uparrow}_{n(t)}$;

after the user requests to watch the stream until the playback starts. The disadvantage is, clearly, that the first seconds of the video are downloaded at lowest quality. In order to mitigate this effect, we introduce a *fast start* phase at the beginning of playback, with the goal to increase the quality of the downloaded segments in a more aggressive manner.

In the following, we first describe the operation of the algorithm after the fast start phase. We define three thresholds for the buffer level, measured in seconds of playback: $0 \leq B_{\text{min}} < B_{\text{low}} < B_{\text{high}}$. We call $\mathcal{B}_{\text{tar}} = [B_{\text{low}}, B_{\text{high}}]$ the target interval. Further, we define $B_{\text{opt}} = 0.5(B_{\text{low}} + B_{\text{high}})$ as the middle of the target interval. The algorithm tries to keep the buffer level close to B_{opt}. Note that the only way to increase $\beta'(t)$, that is, the rate at which the buffer level is changing, is to switch to a lower representation, while there are two ways to decrease $\beta'(t)$. The first is to switch to a higher representation, while the second is to delay subsequent segment downloads.

While $\beta(t) \in \mathcal{B}_{\text{tar}}$, we never change to a different representation. The reason behind it is not to react to short-term variations of the available throughput. If we observe a positive or negative spike lasting for few seconds on an otherwise constant channel, we do not want to switch to a different representation since we would be immediately forced to switch back. This behavior decreases the viewing experience by introducing quality fluctuations. Thus, we configure the sensitivity of the algorithm to throughput spikes by adjusting the size of the target interval $\mathcal{B}_{\text{tar}}$.

Now, assume that we observe a decrease of the available throughput followed by a decrease of the buffer level below B_{low}. We react to this change by selecting a lower representation. We continue to switch to a lower representation as long as we are below B_{low} *and* the segment throughput of the last downloaded segment is smaller than the bit-rate of the currently selected representation. The second condition prevents us from undershooting too much below the available throughput – if we are below B_{low} but the buffer level is increasing, there is no need to change the representation.

Next, assume that we observe an increase of the available throughput followed by an increase of the buffer level above B_{high}. In this case, we have two options: to stay with the current representation and to delay the subsequent download, or to select the next higher representation. Our choice here depends on the throughput measured during the last Δ_t seconds. If the bit-rate of the next higher representation is below a certain percentage of the throughput: $r^{\uparrow}_{n(t)} < \alpha_5 \cdot \tilde{\rho}\big(t-\Delta_t, t\big)$, we prefer to keep the current representation and to delay the request. Otherwise, we step up. Here, α_5 is a configuration parameter. Selecting $\alpha_5 < 1$ (as opposed to $\alpha_5 = 1$) increases the robustness of the algorithm to measurement uncertainties. If we decide to keep the bit-rate, we need to delay the subsequent download in order to bring the buffer level back to the target interval. We achieve this by delaying the next download until the buffer level falls below $\beta(t) - \tau$.

The presented mechanisms already allow to efficiently avoid playback interruptions by keeping the buffer level within the target interval. However, instead of staying close to the optimum value B_{opt}, the buffer level constantly fluctuates around B_{high}, thus unnecessarily increasing the sensitivity of the client to positive spikes. In fact, if the buffer level is above B_{opt} and rising but we know that the conditions for increasing the representation are not fulfilled, there is no reason to wait until $\beta(t)$ reaches B_{high} before we start to delay the downloads. Instead we add the following mechanism. Whenever the buffer level is in $[B_{\text{opt}}, B_{\text{high}}]$ but the conditions for increasing the representation are not fulfilled, we start to delay the subsequent downloads until the buffer level falls back to B_{opt}. By keeping the buffer level close to B_{opt} we achieve a similar sensitivity to both positive and negative spikes.

Here, we would like to point out that although we could bring the buffer level to its optimum value much faster by delaying a download until the buffer level falls below B_{opt}, this approach has a strong disadvantage that can be observed if multiple clients share a common link. In fact, if one of the clients delays a request by a certain time, the TCP congestion control of the other client will be able to double its throughput, eventually triggering an increase of the video quality. Once, however, the first client resumes download, both instances of TCP will again converge to the fair share of the available bandwidth forcing the second client to adjust his representation once again. Thus, in order to avoid these unnecessary fluctuations, it is required to minimize the delays between subsequent downloads.

Whenever the buffer level is below B_{min}, we immediately switch to the lowest available bit-rate. The reason is that we assigned the goal to avoid playback interruptions highest priority. If $\beta(t) < B_{\text{min}}$, there is a high probability of a buffer underrun in the presence of throughput fluctuations.

As already mentioned above, the algorithm starts by selecting the lowest representation for the first segment in order to minimize the delay between the user's request to watch the video and the actual start of the playback. In order to quickly ramp up to best quality that is feasible with the current dynamics of the path throughput, we introduce a more aggressive fast start phase. Without this phase, we would have to wait until the buffer level reaches B_{high} before we increase the quality of the video for the first time. Setting B_{high} to, e.g., 50 seconds would imply that the user will watch the first 50 seconds of the stream in lowest quality even on a high-speed link.

During the fast start phase we proceed as follows. For each subsequent download we select the next higher representation as long as its bit-rate is below a certain percentage of the throughput measured over the last Δ_t seconds. The percentage varies in three steps depending on the current buffer level. For $\beta(t) < B_{\text{min}}$, we switch to a higher representation if $r^{\uparrow}_{n(t)} \leq \alpha_2 \cdot \tilde{\rho}\big(t-\Delta_t, t\big)$. For $\beta(t) \in [B_{\text{min}}, B_{\text{low}})$ we use $\alpha_3 \geq \alpha_2$. Finally, for $\beta(t) \geq B_{\text{low}}$ we use $\alpha_4 \geq \alpha_3$. Thus, the algorithm becomes more and more aggressive the higher the buffer level.

The fast start phase terminates, when one of the following conditions is violated: (i) $r_{n(t)} \neq r_{\max}$, (ii) $\beta_{\min}(t_1) \leq \beta_{\min}(t_2)\, \forall t_1 < t_2 \leq t$, or (iii) $r_{n(t)} \leq \alpha_1 \cdot \tilde{\rho}\big(t-\Delta_t, t\big)$.

Condition (i) lets us terminate the fast start phase after we converged to the highest representation available. Condition (ii) ensures that we proceed to the regular mode of operation if the buffer level is not monotonically increasing. Finally, condition (iii) forces us to quit the fast start phase when the bit-rate of the selected representation is too close to the measured throughput.

V. Evaluation

In order to evaluate the proposed algorithm in real-world scenarios, we implemented it as a platform independent software library written in C++. The library provides an API that can be used to interface it with HTTP streaming clients. For the evaluation of the algorithm, we integrated the library in one of the first DASH client prototypes [8], which is based on the media framework GStreamer [1].

We evaluated the implemented prototype in several artificial and real-world scenarios. On the one hand, we run experiments streaming over the local loop without cross-traffic, artificially limiting the throughput using the open source tool DummyNet [9]. On the other hand, we run experiments in a busy domestic WiFi with a high level of interference and heavy cross-traffic. We observed that the algorithm performs remarkably well even under extremely challenging network conditions. Further, it exhibits a stable and fair behavior if multiple clients share a common network path.

In all experiments we used the following values for configuration parameters. We set the minimum buffer level, below which we immediately switch to the lowest representation, to $B_{\min} = 10$ s. We set the target interval to $[20, 50]$ s. The discretization parameter for the buffer level was set to $\Delta_\beta = 1$ s. The available throughput was measured and averaged over the past $\Delta_t = 10$ s. The safety margins were set to $(\alpha_1, \ldots, \alpha_5) = (0.75, 0.33, 0.5, 0.75, 0.9)$.

The used video sequence was generated by an open source DASH content generation tool [10] and downloaded from *http://www-itec.uni-klu.ac.at/ftp/datasets/mmsys12/BigBuckBunny/bunny_2s/*. It has a duration of approximately 600 seconds. The segment size is $\tau = 2$ s. The available bit-rates are 45, 89, 131, 178, 221, 263, 334, 396, 522, 595, 791, 1033, 1245, 1547, 2134, 2484, 3079, 3527, 3840, and 4220 kbit/s.

Figure 1 shows the convergence properties of the algorithm on a link with "unlimited" bandwidth (local loop). The upper subfigure shows the selected bit-rate r_i, the lower subfigure the dynamics of buffer level $\beta(t)$. The playback started approximately 250 ms after the begin of the download. After 1.25 s, the algorithm reached the maximum available bit-rate.

Figure 2 shows the reaction of the algorithm to persistent throughput changes. The upper subfigure shows the artificial throughput limitation, the measured segment throughput $\tilde{\rho}_i$ and the selected bit-rate r_i. The lower subfigure shows the dynamics of buffer level $\beta(t)$. After the initial phase, where the throughput was limited to 1000 kbit/s, the throughput limitation was shifted to 2000 kbit/s for 200 seconds, and then back to 1000 kbit/s. The algorithm reacted by adapting the bit-rate of the stream after a moderate delay.

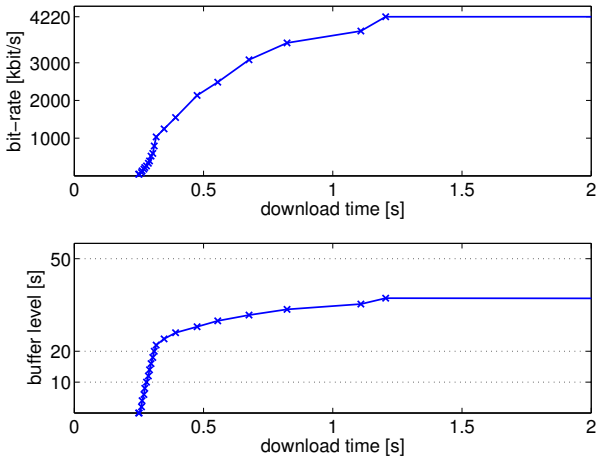

Fig. 1. Single client, local loop, "unlimited" bandwidth

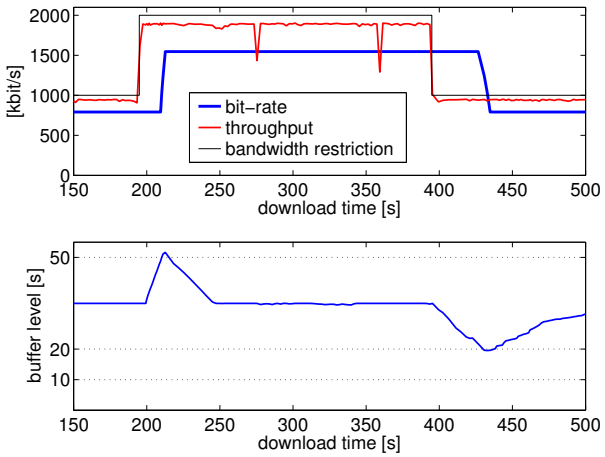

Fig. 2. Single client, local loop, persistent bandwidth change

Figure 3 shows the results of a run, where the client was exposed to periodic short-term throughput spikes lasting for 5 s each. The desired behavior here is not to follow the individual spikes but rather to stay with a bit-rate that is feasible with the available average throughput. This is also the behavior exhibited by the algorithm.

Figure 4 shows an experiment in a domestic WiFi network with a high level of interference and heavy cross-traffic. We observe that the algorithm is able to follow the dynamics of the average available throughput in a robust manner.

The last two figures 5 and 6 show an artificial and a real-world scenario where two players share a common link. In the first scenario, the test is run over the local loop with total throughput restricted to 2000 kbit/s. In the second scenario, the clients stream over the domestic WiFi. In both scenarios, we observe that the clients are able to share the available bandwidth in a stable and fair manner.

Finally, we remark that in all the runs, no buffer underruns occured.

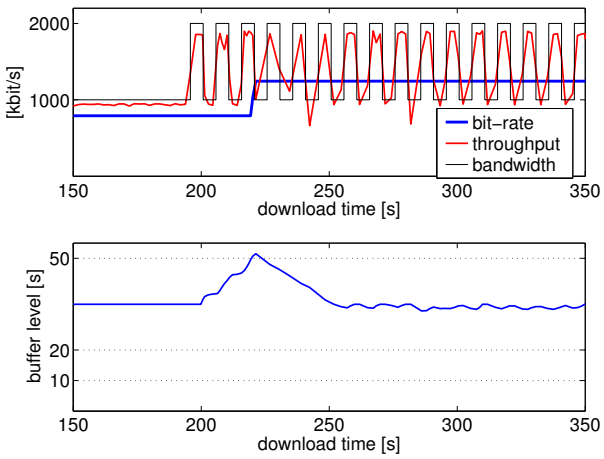

Fig. 3. Single client, local loop, spikes $5s$

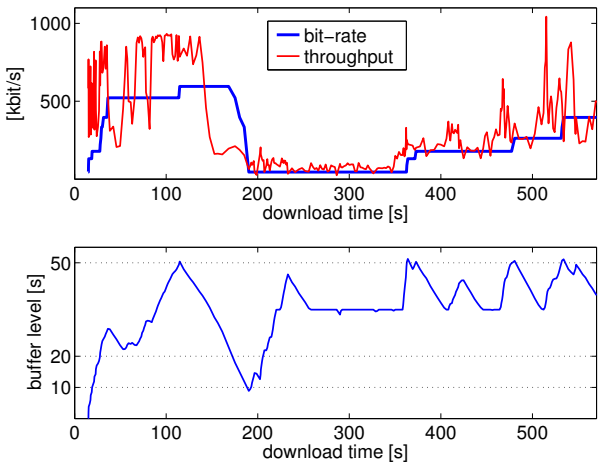

Fig. 4. Single client, busy domestic WiFi

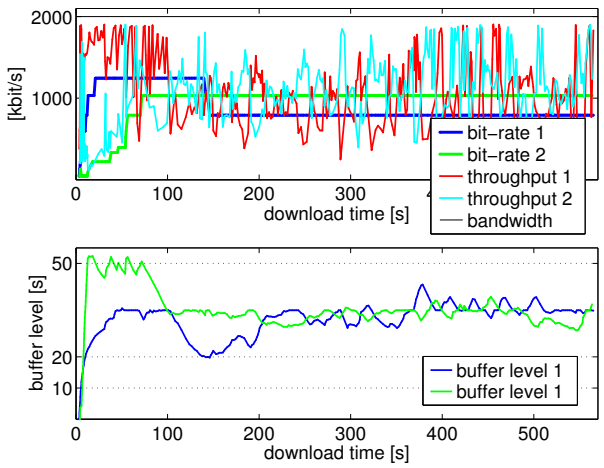

Fig. 5. Concurrent clients, local loop, restricted bandwidth

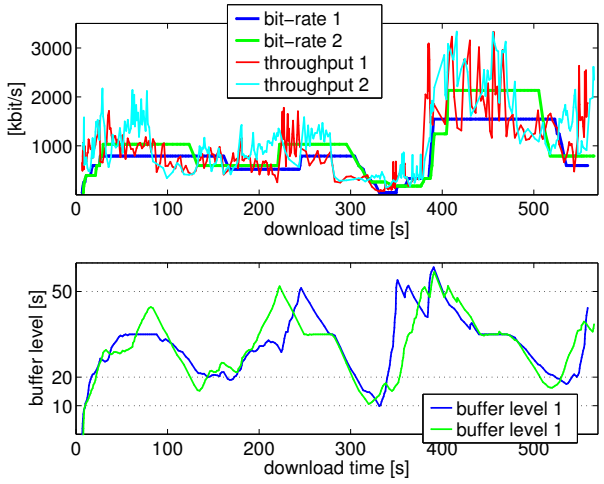

Fig. 6. Concurrent clients, busy domestic WiFi, restricted bandwidth

VI. Conclusion

In this paper, we presented a new algorithm for dynamic adaptation of video quality to the available network throughput. The implemented algorithm aims at avoiding interruptions of playback, maximizing video quality, minimizing the number of video quality shifts and minimizing the delay between user's request and the start of the playback. The algorithm was implemented as a software library and evaluated using a prototype of an HTTP streaming client compliant with the recently adopted standard MPEG DASH. The evaluation was performed both in artificial and real-world scenarios and demonstrated stable and fair behavior even under extremely challenging network conditions.

Acknowledgment

This work has been partially supported by the FP7 COAST project (FP7-ICT-248036), funded by the European Community.

References

[1] "GStreamer multimedia framework." [Online]. Available: http://gstreamer.freedesktop.org/
[2] "MPEG DASH specification (ISO/IEC DIS 23009-1.2)," 2011.
[3] C. Liu, I. Bouazizi, and M. Gabbouj, "Rate adaptation for adaptive HTTP streaming," in *Proc. of the second annual ACM conference on multimedia systems (MMSys)*, Feb. 2011.
[4] "Network simulator ns-2." [Online]. Available: http://isi.edu/nsnam/ns/
[5] D. Jarnikov and T. Ozcelebi, "Client intelligence for adaptive streaming solutions," in *Proc. of the IEEE International Conference on Multimedia and Expo (ICME)*, Jul. 2010.
[6] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP," in *Proc. of the second annual ACM conference on multimedia systems (MMSys)*, Feb. 2011.
[7] M. Zink, O. Künzel, J. Schmitt, and R. Steinmetz, "Subjective impression of variations in layer encoded videos," in *Proc. of the 11th international conference on Quality of service (IWQoS)*, Jun. 2003.
[8] "STMicroelectronics press release on the developed MPEG DASH client prototype," 2011. [Online]. Available: http://www.st.com/internet/com/press_release/t3217.jsp
[9] "DummyNet network emulation tool." [Online]. Available: http://info.iet.unipi.it/ luigi/dummynet/
[10] "Open source DASH content generation tool: DASHEncoder." [Online]. Available: https://github.com/slederer/DASHEncoder