Online Learning Adaptation Strategy for DASH Clients

Federico Chiariotti
Department of Information Engineering (DEI)
University of Padua, Italy
chiariot@dei.unipd.it

Stefano D'Aronco, Laura Toni, and Pascal Frossard
LTS4, Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland
{stefano.daronco, laura.toni, pascal.frossard}@epfl.ch

ABSTRACT

In this work, we propose an online adaptation logic for Dynamic Adaptive Streaming over HTTP (DASH) clients, where each client selects the representation that maximize the long term expected reward. The latter is defined as a combination of the decoded quality, the quality fluctuations and the rebuffering events experienced by the user during the playback. To solve this problem, we cast a Markov Decision Process (MDP) optimization for the selection of the optimal representations. System dynamics required in the MDP model are a priori unknown and are therefore learned through a Reinforcement Learning (RL) technique. The developed learning process exploits a parallel learning technique that improves the learning rate and limits sub-optimal choices, leading to a fast and yet accurate learning process that quickly converges to high and stable rewards. Therefore, the efficiency of our controller is not sacrificed for fast convergence. Simulation results show that our algorithm achieves a higher QoE than existing RL algorithms in the literature as well as heuristic solutions, as it is able to increase average QoE and reduce quality fluctuations.

Keywords

DASH; bitrate adaptation; reinforcement learning

1. INTRODUCTION

In the last few years, video streaming has rapidly become the dominant source of traffic over the Internet [1]. To face this growing demand of media traffic, new efficient video streaming techniques have been developed, such as the popular Dynamic Adaptive Streaming over HTTP (DASH), which works over the already deployed HTTP/TCP protocol for video transmission. The key concept behind DASH is that each video is encoded at different bitrates and resolutions, which results in several *representations* that are stored on servers. These representations are divided into small segments or chunks (typically $2s$ long) which are encoded independently. These segments are then made available to clients, which have the freedom to select representations according to resource availability and to switch streams dynamically at the chunk level. This so called *adaptation logic* is carried out independently for each client, in a distributed way, with the ultimate goal to select the representation that achieves the best Quality of Experience (QoE) for the given resource availability. However, DASH adaptation logics are usually greedy, suffering from large quality variations and playback buffer underflow at the receivers, which drastically affect the quality experienced by users. Most of the current research on adaptive streaming aims at overcoming these issues.

In most adaptation logics, clients request the best segment (i.e., the one with the highest quality) that prevents rebuffering. This, however, comes at the price of frequent quality switches that highly affect the QoE [2], since the adaptation logic mainly follows network fluctuations. Several optimizations have been proposed with the final goal of limiting bitrate variations between successive segments. However, preserving a constant bitrate of the selected representations does not necessarily translate into a constant quality level [3] if the video complexity changes over time. This means that video complexity and bandwidth fluctuations need to be jointly taken into consideration in the adaptation logic. Moreover, while the playback buffer can absorb some of the bitrate variations, rebuffering events are still frequent in highly dynamic network environments. The only viable solution to preserve a good buffer level while maximizing QoE is to implement foresighted optimization in the adaptation logic [4]; however, most of the existing optimization techniques require a priori information about the dynamics of the system, which are usually not available in practice. An efficient online adaptation logic should therefore be able to learn the best representation selection strategy that maximizes the long term reward.

In this work, we propose an online optimization framework for representation selection at the DASH client. We assume that the channel bandwidth experienced by the users as well as the video characteristics vary over time following a Markovian model, which is not known a priori by the clients. We formulate the representations selection problem as a Markov Decision Process (MDP). We further define a novel objective function that takes into account not only the experienced quality, but also the quality variations and the rebuffering states. We then design an online DASH controller, which learns online the temporal evolution of the system. Our algorithm is able to select the best representation, which is defined as the one that maximizes the long term reward,

i.e., the one that maximizes the quality experienced over time and yet limits both the quality variations and rebuffering events. We carefully define the Markov states in the MDP formulation such that they are as descriptive as possible, while maintaining a low number of total states. We also exploit the particular structure of the problem to parallelize the learning process and speed it up by minimizing the number of non-optimal selections. Finally, we carry out Matlab simulations to compare the proposed framework with more conventional DASH controllers. Simulation results show the gain of the proposed adaptation logic with respect to adaptation logics that do not implement any learning strategy. The proposed controller is able to provide a more stable quality level by reducing the quality fluctuations experienced over time, and to quickly react to variations in the network or in the video characteristics. We then compare our model with the other Reinforcement Learning (RL)-based DASH controller available in the literature [5]. We show the performance gain of our new algorithm due to an improved MDP model and better learning techniques. The proposed algorithm has the main advantage of performing a *fast* learning that allows the system to quickly converge to optimal selection strategies. This leads to an improvement both in terms of quality and of quality variations experienced by clients.

Previous works [5, 6] usually consider a trade-off between convergence speed and strategy optimality. They have to choose between a simplified (and less efficient) state characterization and more complex (and slower) controllers. To the best of our knowledge, our work is the first to overcome this tradeoff by proposing an online learning strategy for DASH clients able to *i*) use sophisticated reward metrics that maximize the QoE and still limit quality fluctuations and rebuffering events; *ii*) consider an MDP based solution, which is complete (in its state definition) and yet fast in learning novel system conditions. The improved learning rate and the reduced complexity make the proposed algorithm suitable for implementation at DASH clients' side.

The remainder of this paper is organized as follows: Section 2 presents the current state of the art on DASH adaptation logics, while Section 3 defines the system model, and Section 4 characterizes the video representation selection optimization problem. We describe the proposed solving method in Section 5, and provide the simulation parameters and results are in section 6. Finally, we conclude the work in Section 7.

2. RELATED WORKS

We now describe the most relevant works published in the literature on DASH client-side adaptation strategies. In particular, we provide the works focusing on reducing the quality variations at the client side. We refer the reader to [7] for a comprehensive review of QoE issues and adaptation algorithms for DASH clients.

Adaptation logics can be categorized in *i*) heuristic-based methods and *ii*) optimization-based methods.

In the first category, one of the most relevant works that include quality variations in the QoE definition is the Probe and Adapt (PANDA) algorithm [8]. It is based on heuristic rules that try to maximize quality while preventing the selection of high bitrates that might not be sustainable by the network. This effectively reduces the bitrate variations in stable network conditions. However, the algorithm is less effective when the video complexity changes quickly. Moreover, the algorithm does not always fully utilize the available capacity. As shown in [9], a prediction of the system evolution is fundamental to improve the system performance. The authors propose a method which applies the Microsoft Smooth Streaming (MSS) heuristic rules, but learns the parameters of the controllers over time. This algorithm favors stability over efficiency. However, the heuristic is not foresighted enough to exploit the bandwidth aggressively without experiencing frequent rebuffering events, leading to a tradeoff between stability and efficiency.

To improve efficiency whil preserving stability, an optimization problem is needed instead of heuristics. In [4], the authors propose a receding horizon optimization for the representation selection aimed at maximizing their quality while reducing quality variations. The proposed adaptation logic leads to good and stable quality values during the video streaming session. However, the algorithm requires a priori information on the future status of the system, in terms of video complexity and network resources. In practice, this future information cannot be known a priori and needs to be learned by the system. Other works use MDP models in similar ways [10, 11], but still assuming some a priori knowledge about system statistics.

Another interesting work [12] uses MDPs to keep a high and stable QoE in an online learning adaptation logic: the client gathers bandwidth statistics and improves its system model, which is then used to solve the MDP with dynamic programming. As highlihghted by the authors, the main limitation of the proposed scheme is its computational complexity. Solutions are proposed in [12] to reduce the complexity issue but at the price of having an unacceptable number of rebuffering events or reducing the problem to offline optimizations.

To the best of our knowledge, only the works in [5, 6] address *online* reinforcement learning DASH control strategies based on optimization problems rather than heuristics. The authors designed a Q-learning based algorithm [6] that learns the optimal representation selection strategy. The online algorithm uses a quality-based reward function that highly penalizes rebuffering events. The algorithm substantially outperforms heuristic-based strategies, but the model parameters are tuned to work for slow-varying channels, and the extension to more dynamic conditions is not straightforward. While this algorithm is pretty efficient, it is unfortunately very slow in adapting to new network conditions or different video dynamics. To overcome this limitation, the authors propose a faster learning algorithm called FA(Q) [5]. The authors substantially simplify the MDP model, improving the learning rate of the process. However, this simplification leads to a suboptimal streaming performance.

The learning algorithms in [6, 5] tradeoff between learning speed and model accuracy: while the earlier algorithm uses a cumbersome model that needs extensive training and has no flexibility to adapt to new environments, the FA(Q) algorithm uses an MDP model that is too simple to capture the environment efficiently, so that the overall QoE degrades. The goal of our work is to overcome the current limitations by using efficient learning strategies to have both a complete model with high descriptive power and fast convergence to the optimal policy.

3. SYSTEM MODEL

As depicted in Fig. 1, we consider a DASH client down-

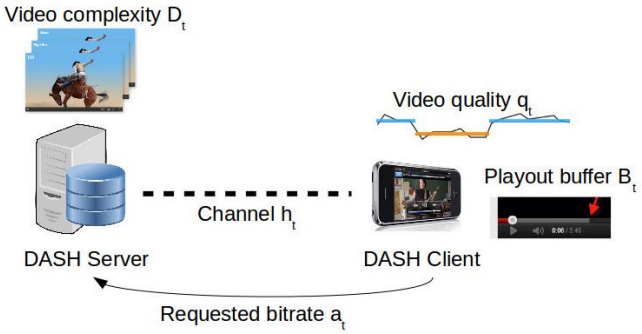

Figure 1: Summary of the problem scenario

loading successive segments of a given video sequence from a server. For each segment, the client optimally selects the representation to download, namely, the best coding rate and resolution for the video segments. We denote by $a_t \in \mathcal{A}$ the selected representation for the t-th segment L_t of the video of interest, with $\mathcal{A}$ being the set of representation segments available at the server. The quality of the segment L_t in its representation a_t is

$$q_t = g(a_t, D_t) \tag{1}$$

with g being the rate quality curve that depends on the complexity D_t of the video segment L_t. The latter reflects the complexity of the content in the scene. For example, a blue-sky sequence has low complexity, while a very dynamic sequence with new appearing objects has a higher complexity. In general, the evolution of the video complexity cannot be known a priori. However, in the following we assume that once the segment L_t is downloaded, the complexity D_{t+1} of the next segment L_{t+1} is known. This complexity can be available in the Media Presentation Description (MPD) file, which contains information about the available representations and can be updated over time, especially in live-streaming scenarios. Otherwise, the complexity can be accurately predicted since the complexity of consecutive scenes is genenally highly correlated.

We define the client QoE given the selection a_t for the segment L_t as follows:

$$\begin{aligned} r^{\mathrm{q}}(a_t, D_t, q_{t-1}) &= q_t - \beta|q_t - q_{t-1}| \\ &= g(a_t, D_t) - \beta|g(a_t, D_t) - q_{t-1}| \end{aligned} \tag{2}$$

which is the weighted combination of the current segment quality and quality variations (a similar model for QoE has been proposed and validated in [13]). The parameter β is a positive weight factor that balances the importance of the quality variations in the QoE evaluation.

The download time f_t of a representation a_t for the segment L_t depends on the encoding rate of the chosen representation a_t as well as on the available channel capacity. We denote h_t the average bandwidth experienced while downloading representation a_t, and the download time as $f_t = f(a_t, h_t) = \frac{S(a_t)}{h_t}$, where $S(a_t)$ is the size in Bytes of the selected segment a_t. Note that h_t is not known by the client before downloading the chunk, but it can be evaluated from the encoding bitrate of a_t when the downloading time f_t is known. We further consider the variations of the channel bandwidth to be Markovian, with a $p(h_t|h_{t-1})$ transition probability between consecutive channel states. The Markovian channel assumption is reasonable since it covers a wide range of scenarios and it has already been successfully adopted in other learning works [6, 5]. The model still extends to non-Markovian channels, where the learned policy will be suboptimal.

Finally, we denote by B_t the amount of video time that is stored in the playout buffer, namely, the buffer level. The value of B_t is computed just before sending the request for segment a_t and it is expressed in seconds. Denoting by T_P the playing video time of every segment (e.g., $2s$), we can define the evolution of the buffer level as:

$$B_{t+1} = B_t + T_P - f(a_t, h_t), \tag{3}$$

which means that the buffer level B_t decreases by $f(a_t, h_t)$ when downloading the representation a_t of the segment L_t. When the download is complete, the new segment is added to the buffer, which increases by the segment duration T_P. Ideally, $f(a_t, h_t)$ should be lower than B_t to avoid rebuffering. That leads to the condition $B_t \geq f(a_t, h_t)$ or, equivalently, $B_{t+1} \geq T_P$.

In the following, we show how DASH clients can learn the best selection strategy that improves the long-term QoE while avoiding rebuffering.

4. OPTIMIZATION PROBLEM

In this section, we formulate the optimization problem that needs to be performed at the client side for the optimal representation selection.

A DASH client requests the representation a_τ for the segment L_τ by solving online a foresighted optimization aimed at maintaining a high and stable QoE level while respecting the buffer constraint. Ideally, this means finding the optimal set of actions $\mathbf{A}^\star$, which maximizes the long term QoE of the user among all possible action vectors $\mathbf{A} = [a_\tau, a_{\tau+1}, \ldots, a_\infty]$. More formally,

$$\begin{aligned} &\max_{\mathbf{A}} \left\{ \sum_{t=\tau}^{\infty} \gamma^{t-\tau} \left[g(a_t, D_t) - \beta|g(a_t, D_t) - q_{t-1}| \right] \right\} \\ &\text{s.t. } B_t - f(a_t, h_t) \geq 0, \quad \forall t. \end{aligned} \tag{4}$$

where $\gamma \in [0, 1)$ is an exponential discount factor on future rewards, and the constraint imposed on the playback buffer guarantees that the current action a_t does not lead to a rebuffering event.

While the optimization problem of Eq. (4) is exact, it requires the knowledge of $f(a_t, h_t)$. In practice, the function $f(a_t, h_t)$ depends on the channel conditions that will be experienced during the next segment download. Therefore, the function $f(a_t, h_t)$ is known by the client only after the action a_t is taken. The hard constraint imposed in Eq. (4) cannot be imposed in practice. Thus, we consider a soft constraint that avoids actions with a large rebuffering probability. By adding the soft constraints, we get the following optimization problem:

$$\max_{\mathbf{A}} \left\{ \sum_{t=\tau}^{\infty} \gamma^{t-\tau} \left[r^{\mathrm{q}}(a_t, D_t, q_{t-1}) - r^{\mathrm{b}}(B_t, f(a_t, h_t)) \right] \right\} \tag{5}$$

In particular, we introduce a penalty function r_t^b to discourage the learning controller from violating the constraint.

We define the penalty function r_t^b as follows

$$r^{\mathrm{b}}\left(B_t, f(a_t, h_t)\right) = \rho(\max[0, f(a_t, h_t) - B_t]) + \sigma(\max[B_M - B_{t+1}, 0])^2 \quad (6)$$

where the first term highly penalizes the system for rebuffering events, while the second one is an incentive for the system to maintain the buffer level around B_M, which is a "safe" buffer level above which there is no penalty. The second term further reduces the risk of rebuffering events by penalizing actions that lead to risky low-buffer states. Finally, ρ and σ are relative weights given to the two buffer management term to adjust the tradeoff between QoE and rebuffering events.

In the next section, we formulate and solve the problem as a MDP [14] optimization, where the evolution of the system state is learned in real time.

5. ONLINE SOLVING METHOD

We consider the DASH client as a RL agent who learns the best action (i.e., the best representation to download) based on the experienced environment, which is characterized by the network status, the video complexity, and the previous requests of the client. In particular, the state of the environment is identified by a Markov state, for which we optimize the best action.

As we explained in the previous section, we define the state as $s_t : \{q_{t-1}, h_{t-1}, D_t, B_t\}$. The system is characterized by the quality of the previously downloaded segment, q_{t-1}, the estimated bandwidth experienced during the previous download, h_{t-1}, the complexity of the segment that needs to be downloaded, D_t, and B_t, which is the status of the buffer at the instant when the decision is made.

Before going into the details of the MDP-based solving method, we define the transitions to future states and the reward associated to each action. When the client is in the state s_t and takes action a_t, it transitions to a future state s_{t+1}. Our process is Markovian since future states $s_{t+1} : \{q_t, h_t, D_{t+1}, B_{t+1}\}$ depend only on the current state s_t and action a_t. In particular, q_t depends on the current action a_t and on the video complexity level D_t, as defined in Eq. (1). From Eq. (3), we see that the buffer level B_{t+1} depends on the buffer status B_t (given in s_t), on the action a_t, as well as on the channel h_t (given in s_{t+1}). Finally, the available bandwidth h_t and the video complexity D_t are the two only random variables of the system, and they are mutually independent. Both variables are Markovian, as explained in section 4; we can then conclude that the overall behavior of the system is Markovian.

Finally, we denote by $r(s_t, a_t, s_{t+1})$ the reward of taking action a_t in s_t when s_{t+1} is the next experienced state. From Eq. (2) and Eq. (6), we can define the action reward as

$$r(\underbrace{D_t, q_{t-1}, B_t}_{s_t}, a_t, \underbrace{h_t}_{s_{t+1}}) = r^{\mathrm{q}}(a_t, D_t, q_{t-1}) - r^{\mathrm{b}}(B_t, f(a_t, h_t)) = r^{\mathrm{q}}(s_t, a_t) - r^{\mathrm{b}}(s_t, a_t, s_{t+1}) \quad (7)$$

where $r^{\mathrm{q}}(a_t, s_t)$ is a deterministic function of the current state and the chosen action, while $r_t^{\mathrm{b}}\left(B_t, f(a_t, h_t)\right)$ is unknown until the future state s_{t+1} is revealed.

Equipped with the above notations, we can solve the optimization problem in Eq. (5), introducing the optimal state-value function $V^\star(s_t)$. The best state-value function $V^\star(s_t)$ represents the value (in terms of long term reward) of the state s_t under the best-action policy. More formally,

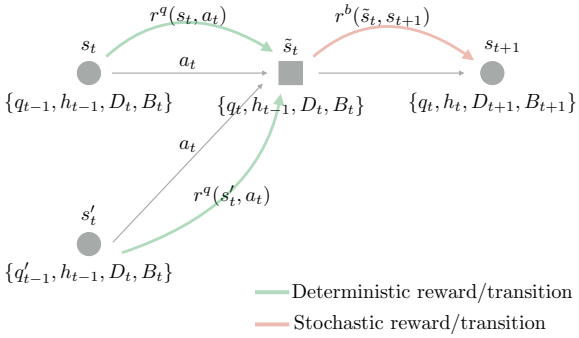

Figure 2: Post decision states representation.

$$V^\star(\underbrace{q_{t-1}, h_{t-1}, D_t, B_t}_{s_t}) = \max_{a_t \in \mathcal{A}(s_t)} \Big\{ r^q(a_t, D_t, q_{t-1}) + \sum_{h_t, D_{t+1}} p(h_t|h_{t-1}) p(D_{t+1}|D_t)[-r^b\left(B_t, f(a_t, h_t)\right)) + \gamma V^\star(\underbrace{q_t, h_t, D_{t+1}, B_{t+1}}_{s_{t+1}})] \Big\} \quad (8)$$

where $p(h_t|h_{t-1})$ and $p(D_{t+1}|D_t)$ are the one-step transition probabilities of the processes that describe the channel rate and the video complexity, respectively. As they are the only random variables, the quality reward r^{q} is a deterministic function of the state and action. Using a more compressed notation, Eq. (8) can be equivalently expressed as

$$V^\star(s_t) = \max_{a_t \in \mathcal{A}} \left\{ r^{\mathrm{q}}(a_t, s_t) + \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) \Big[-r^{\mathrm{b}}(a_t, s_t, s_{t+1}) + \gamma V^\star\left(s_{t+1}\right) \Big] \right\} \quad (9)$$

The optimization problem can be solved by backward induction using Bellman's optimality equations [15, 16], given the channel and video complexity statistics, that need to be learned online. For the client to quickly converge to the optimal decision strategy, this online learning process needs to be fast and accurate, and standard algorithms such as vanilla Q-learning [15] converge too slowly. To reduce the complexity of the learning agent, we introduce a new set of states $\tilde{s}_t$, which are called Post-Decision States (PDSs) [17]. The key concept is to use an intermediate state $\tilde{s}_t$ to divide the transition from s_t to s_{t+1} in two steps, as depicted in Fig. 2. The transition from s_t to $\tilde{s}_t$ is deterministic and depends on the current state and the action taken, while the transition from $\tilde{s}_t$ to s_{t+1} depends on the random variables of the system and not on the action. In our case, the PDS $\tilde{s}_t$ is defined by the quadruple $\{q_t, h_{t-1}, D_t, B_t\}$. After choosing the action a_t, the value of q_t is a deterministic function of the complexity D_t. We refer to [18] for details on PDS theory.

We can then define the value function of the PDS as

$$\tilde{V}^{\star}(\tilde{s}_t) = \sum_{s_{t+1}} p_{\mathrm{u}}(s_{t+1}|\tilde{s}_t) \left[-r^{b}(s_{t+1}, \tilde{s}_t) + \gamma V^{\star}(s_{t+1}) \right] \tag{10}$$

where

$$V^{\star}(s_t) = \max_{a_t \in \mathcal{A}} \left\{ r^{\mathrm{q}}(a_t, s_t) + \sum_{\tilde{s}_t} p_{\mathrm{k}}(\tilde{s}_t|s_t, a_t) \tilde{V}^{\star}(\tilde{s}_t) \right\}. \tag{11}$$

In Eq. (10), $p_{\mathrm{u}}(s_{t+1}|\tilde{s}_t)$ is an unknown transition probability that depends on the future channel state and video complexity. At the same time, $p_{\mathrm{k}}(\tilde{s}_t|\tilde{s}_t, a_t)$ in Eq. (11) is known. It is a deterministic probability equal to 1 only for one allowed transition $s_t \rightarrow \tilde{s}_t$. The optimal action a_t is the one that solves solves Eq. (11) for the given PDS value function $\tilde{V}^{\star}(\tilde{s}_t)$.

The value of $\tilde{V}^{\star}(\tilde{s}_t)$ can be evaluated by using Temporal-Difference (TD) learning as described in [19]. In order to learn the PDS value function, we update its value at every step t according to the following update equation:

$$\tilde{V}^{t+1}(\tilde{s}_t) \leftarrow (1-\alpha)\tilde{V}^{t}(\tilde{s}_t) + \alpha \left[-r^{\mathrm{b}}(s_{t+1}, \tilde{s}_t) + \gamma V^{t}(s_{t+1}) \right], \tag{12}$$

where $\alpha \in [0, 1]$ is the learning rate of the agent. The above equation guarantees that, for appropriate values of the learning rate, the PDS state values converges to the optimal ones as t goes to infinity.

In the following section, we show how the DASH client selects the best next action given Eq. (12) and describe the implementation of the online learning optimization framework.

5.1 Implementation and pseudocode

In learning strategies, it is well known that the learning agent needs to find the right trade-off between exploration (learning the reward of new actions) and exploitation (selecting the best action among the known ones) [15, 19]. In our algorithm, we consider the common Softmax policy selection strategy [19], in which the agent selects a given action a_j with a probability $P(a_j)$ that is a function of the state value function; namely,

$$P(a_j) = \frac{e^{\frac{U_t(a_j)}{\tau}}}{\sum_{i=1}^{|A_s|} e^{\frac{U_t(a_i)}{\tau}}} \tag{13}$$

where $U_t(a_j)$ is

$$U_t(a_j) = r^{q}(a_j, s_t) + \sum_{\tilde{s}_t} p_k(\tilde{s}_t|s_t, a_j) \tilde{V}^{t}(\tilde{s}_t). \tag{14}$$

The Softmax policy is used to choose the action to take in in state s_t. However, the update of the PDS value function in Eq. (12) is performed using the optimal action that maximizes Eq. (11). This is known as the off-policy learning method, in which the policy used to select the actions is different from the policy used to evaluate the state values. Off-policy learning algorithms are generally less conservative, and converge to the optimal action more quickly [15].

Finally, to speed up the convergence of the proposed controller, in addition to the theoretical efforts in defining states in the most compact way and introducing PDSs, we also implemented a parallel update technique [17]. It allows us to update multiple state value functions in parallel for each experienced transition. In particular, the stochastic transition $p_{\mathrm{u}}(s_{t+1}|\tilde{s}_t)$ depends only on the channel h_t and on the complexity D_t. When a transition of these two quantities is observed we can parallelize the state value update, for all the PDSs that share the transition $h_{t-1} \rightarrow h_t$ and $D_t \rightarrow D_{t+1}$, i.e., $\{q'_t, h_{t-1}, D_t, B'_t\} \rightarrow \{q'_t, h_t, D_{t+1}, B'_{t+1}\}$ for all q'_t and B'_t.

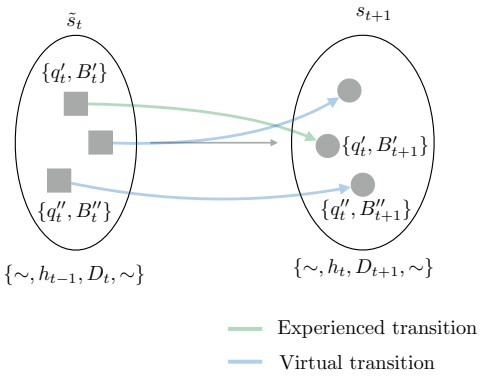

Figure 3: Virtual and experienced transitions

Algorithm 5.1 Online algorithm

```
Initialize Ṽ = 1/γ
Initialize s_t to the starting state
repeat
    for all a_j ∈ A do
        r_j^q ← QUALITYREWARD(s_t, a_j)
        s̃_j ← FINDPDS(s_t, a_j)
        U_t(a_j) ← r_j^q + Ṽ(s̃_j)
    a_t ← SOFTMAX(U_t(·))
    s̃_t ← FINDPDS(s_t, a_t)
    Observe s_{t+1}
    VUPDATE(Ṽ, s̃_t, s_{t+1})
    s_t ← s_{t+1}
until end of simulation
```

In Algorithm 5.1 we provide the pseudocode for the proposed online algorithm. In the first step, we initialize the value of the PDS value function, setting all the state values to the inverse of the exponential discount parameter γ, and the initial state to the current state of the system. We then repeat the following steps until the end of the simulation.

Given the state s_t, we first need to choose the action to take. We compute the quality reward for all possible as defined in (2) by using the QUALITYREWARD function, then find its associated PDS $\tilde{s}_t$ using the function FINDPDS, which computes the PDS for which $p_{\mathrm{k}}(\tilde{s}_t|s_t, a_t) = 1$ to (s_t, a). Given these two values, we can compute the variables $U_t(\cdot)$ needed by the Softmax policy, implemented in the SOFTMAX function, to select the action a_t. This set of steps corresponds to Eq. (13) and Eq. (14).

After the action is selected, we find the real PDS $\tilde{s}_t$ and observe the effects of the action, namely, the state s_{t+1}. Finally, we can update the PDS value function $\tilde{V}$; this operation is carried out by the functions VUPDATE and SINGLE-

Algorithm 5.2 parallel $\tilde{V}$ update function

function vUpdate($\tilde{V}, \tilde{s}_t, s_{t+1}$)
  Extract $\{h_{t-1}, D_t\}$ from $\tilde{s}_t$
  Extract $\{h_t, D_{t+1}\}$ from s_{t+1}
  for all (q', B') **do**
    $\tilde{s}'_t : \{q', h_{t-1}, D_t, B'\}$
    $s'_{t+1} : \{q', h_t, D_{t+1}, B' + T_P - f(g^{-1}(q', D_t), h_t)\}$
    $r_t^{b\prime} \leftarrow$ bufferReward($\tilde{s}'_t, s'_{t+1}$)
    singleUpdate($\tilde{V}, \tilde{s}'_t, s'_{t+1}, r_t^{b\prime}$)

Algorithm 5.3 single $\tilde{V}$ update function

function singleUpdate($\tilde{V}, \tilde{s}_t, s_{t+1}, r_t^b$)
  for all $a_j \in \mathcal{A}$ **do**
    $r_j^q \leftarrow$ qualityReward(s_{t+1}, a_j)
    $\tilde{s}_j \leftarrow$ findPDS(s_t, a_j)
  $r \leftarrow \max_j[r_j^q + \gamma \tilde{V}(\tilde{s}_j)] - r_t^b$
  $\tilde{V}(\tilde{s}_t) \leftarrow \tilde{V}(\tilde{s}_t)(1-\alpha) + \alpha r$

Update, whose pseudocode is reported in Algorithms 5.2 and 5.3 respectively.

The vUpdate function parallelizes the updates as described above, using the real capacity and video complexity to make inferences about other states with different buffer and quality values. The fictitious state s'_{t+1} can be inferred using the defined states $\tilde{s}'_t$ and the known transition of h_t and D_t. Note that the function $g^{-1}(q', D_t)$ returns the action a_t. This function represents the inverse of the function $g(\cdot, D_t)$ when considering D_t fixed. The bufferReward function corresponds to Eq. (6). Once the fictitious transition $\tilde{s}'_t \rightarrow s'_{t+1}$ and the buffer reward $r_t^{b\prime}$ are known, we can execute the updates of the PDS value function using the function singleUpdate. The singleUpdate function corresponds to the PDS value update function defined in Eq. (12): it updates the state value for the given PDS using the buffer component of the reward and the estimated value of the next state s_{t+1}, assuming the optimal policy is used. In order to update the PDS value function of state $\tilde{s}_t$ we need to know the state value function of the state s_{t+1}. Therefore, we need to solve Eq. (11) and select the action with the largest long term reward. Here we can easily recognize that our algorithm is using an off-policy technique: when we select the action to take we use the Softmax policy, whereas when we update the PDS value function we use the max policy.

6. RESULTS

To evaluate the system performance of the proposed algorithm at the client side in a DASH-like scenario, we carry out some Matlab simulations. In the following, we first describe the simulation settings and how we implemented our simulator and the other considered benchmark controllers, then provide simulation results under different DASH-like scenarios and show the performance improvements that can be achieved using our method with respect to a baseline conventional solution and a state of the art RL-based DASH controller [5].

6.1 Simulation settings

In this subsection, we first characterize the states, the channel and the video complexity models in DASH-like scenarios. We then define the different parameters adopted by our controller. Finally, we introduce the benchmark algorithms that are used as comparison.

We express the quality function $q = g(a, D)$ as a function of the the Structural Similarity Index (SSIM) [20], which has been chosen for its fidelity of the metric with the subjective quality. Typical values of SSIM range from 0.8 to 1; SSIM values below 0.9 are usually considered poor bad video quality levels [20].

As in [21], we approximate the SSIM curves to a fourth-degree polynomial of the logarithm of the normalized rate as follows

$$\begin{aligned} q &= g(a, \mathbf{d}) \\ &\simeq 1 + \mathbf{d}(1)\rho_a + \mathbf{d}(2)\rho_a^2 + \mathbf{d}(3)\rho_a^3 + \mathbf{d}(4)\rho_a^4 \end{aligned} \tag{15}$$

where $\rho_a = \log(R_a/R_1)$, with R_a being the encoding rate for representation a and R_1 being the maximum experienced bitrate. The vector $\mathbf{d}$ is a synthetic representation of the complexity of a video scene.

For a continuos domain of $\mathbf{d}$, we could consider an infinite number of SSIM curves, covering the full set of video sequences that can be stored in the system. In practice, we need to sample the video complexity to preserve a finite (and limited) number of states in the MDP formulation. Therefore, we extract a small set of reference curves which can cover different behaviors of typical rate-quality curves. The index of the reference curve is the complexity D we use in our algorithm. The reference set was elaborated from the EvalVid CIF video trace reference database[1]. Fig. 4 shows all the curves of the dataset, emphasizing the 5 reference curves that we select as representative of the whole dataset. In the MDP formulation, the parameter D_t indicates the best fit (among the 5 sample curves) for the complexity of segment t.

The $\{q, h, B\}$ values also need to be quantized. The state border values are listed in Table 1. Finally, we define the set of available bitrates (expressed in Mb/s) that represents the set of possible actions as $\mathcal{A} : \{0.3, 0.5, 1, 2, 3, 4, 6, 8, 10\}$.

[1] http://www2.tkn.tu-berlin.de/research/evalvid/cif.html

$q_{t-1}[SSIM]$	0.84 0.87 0.9 0.92 0.94 0.96 0.98 0.99 0.995
$B_t[s]$	3 4 5 6 8 10 12 15 18
$h_{t-1}[Mb/s]$	0.5 1 2 3 4 5 6 8 10

Table 1: State thresholds.

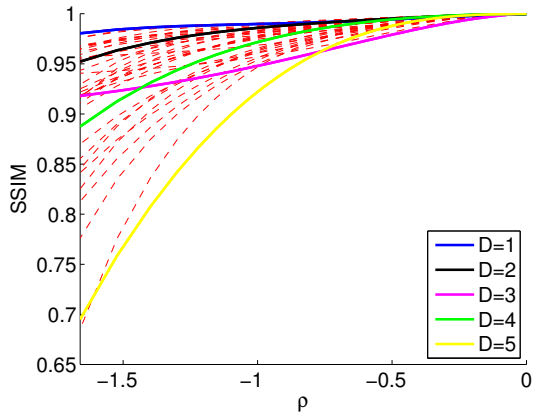

Figure 4: The rate-distortion curves for the videos in the reference database.

The reward function parameters are defined in Table 2. All the parameters have already been introduced in the previous sections, but we briefly recall their meaning below. The parameter β, used in (2), establishes how much the quality switches affect user QoE. We set $\beta = 2$ to get a good tradeoff between the maximization of the average quality and the minimization of the quality switches. We need the parameters ρ, σ, B_M and $B_{\max}$ to adjust the buffer penalty in (6). We set $B_{\max} = 20$ s, since this is a common choice among DASH controllers. We then empirically set $B_M = 12$ s, $\rho = 50$, and $\sigma = 0.001$ to achieve a sufficiently low probability of rebuffering events and yet allow the learner to efficiently use the buffer in order to decrease quality variations. Finally, the value of γ defines how greedy the learner is. The lower the value of γ, the more the learner tries to maximize the reward for the very next step, even at the cost of penalizing the future ones. A value of 0.9 allows the learner to achieve a good trade-off between the immediate and the future rewards.

The channel capacity follows a Markovian model as explained in the previous sections. Although the effects of the interaction between TCP and the adaptation logic are not considered in this capacity model, a statistical analysis on the NYU DASH dataset (from the Crawdad[2] trace database) has shown that a Markov-Gauss process closely fits the capacity fluctuations in a real DASH streaming session, and modeling capacity variations as red noise [22] is not unrealistic. Markov capacity models are widely used in the literature, both on DASH adaptation [23][24] and TCP performance [25] studies. We built a transition matrix for all possible channel states h_t and h_{t+1}. The matrix we use allows transitions to adjacent states with probability $\frac{2p}{3}$ and two-state jumps with probability $\frac{p}{3}$. By varying the value of p we can simulate more or less dynamic channels; in our simulations, we used $p = 0$, $p = 0.25$ and $p = 0.5$.

In order to mimic a video's changing complexity, we consider it to be composed of a series of scenes. Video complexity is constant over a scene, but changes from one scene to the next. Each new scene is selected randomly among the reference ones using a uniform distribution. The duration of the scenes follows an exponential random distribution (with an average of 1, 5, or 10 segments); this model has been experimentally validated by Rose [26].

[2]http://www.crawdad.org/nyupoly/video/20140509/

Parameter	β	ρ	σ	B_M	$B_{\max}$	γ
Value	2	50	0.001	12 s	20 s	0.9

Table 2: Parameters used in our simulations.

Algorithm 6.1 Rate-based algorithm

```
if t = 1 then
    a ← a_max
else
    a ← 1
    while a ≥ h_t and a < a_max do
        a ← a + 1
```

In our simulations, we compare the proposed controller with two benchmarks: a heuristic algorithm that greedily optimizes the representation selection, and a learning strategy that learns the best representation selection online. The heuristic one is a *rate-based* controller that selects the bitrate of the segment to download based on the measured channel bandwidth h_{t-1}. Although they are still widely used, methods based on rate-based heuristics represent only one category among the currently adopted systems. However, most commercial adaptation logics, whether rate-based or buffer-based, are greedy algorithms that do not consider the channel and video complexity dynamics in a foresighted manner. The *rate-based* controller is defined in Algorithm 6.1 (in which $a \in \{1, a_{\max}\}$ is the chosen action, and actions are ordered by bitrate). For the first segment, as the channel rate is unknown, the rate-based algorithm tries to build up the buffer and speed up the loading time of the video by choosing the lowest possible rate. Then, the following segment requests are adapted to the experienced bandwidth.

The other benchmark algorithm is FA(Q) [5], an algorithm based on RL similar to our work but with a simpler structure (in terms of state definition). The state definition in [5] is simpler than the one we propose, as it only considers B_t and h_{t-1}, neglecting the previous quality level q_{t-1} or the video complexity D_t. This reduces the overall system performance (sub-optimality of the algorithm) but it speeds up the learning processing. In the following comparison, we show that our algorithm converges as quickly as FA(Q) thanks to the use of PDSs and parallel learning, while obtaining a better QoE thanks to the more complete MDP model.

Both benchmark algorithms and the proposed one have been simulated over the same scenario, characterized by one DASH client downloading and playing the segments of one video. In all the simulations we use the term "episode" to define a stream of a video composed by 400 segments of 2 seconds each, i.e., the length of the video is 800 s per episode.

The performance of the algorithms is measured in terms of SSIM; while the dynamic range of this quality metric goes from -1 to 1, its mapping to actual QoE is non-linear. Values below 0.8 represent poor quality video sequences, and acceptable SSIM values are in the range $[0.8 - 1]$, where even a small variation in terms of SSIM score, e.g. 0.01, can have a large effect on QoE [27].

It is worth noting that, although the Matlab simulations do not offer a complete realistic DASH streaming scenario, they clearly show the benefits of the learning algorithm when compared to a conventional algorithm in controlled condi-

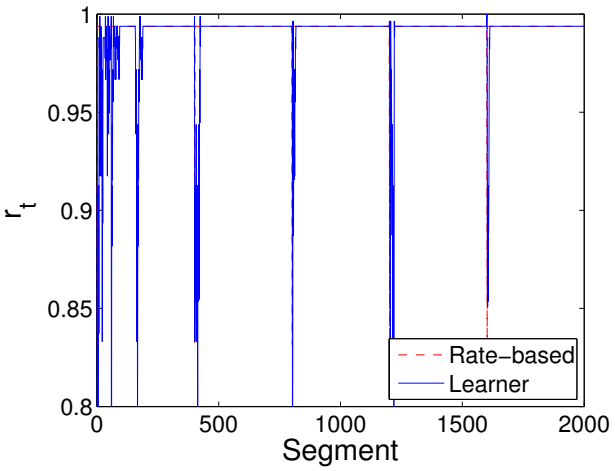

Figure 5: Reward experienced at the client side for the first 2000 segments (i.e., 5 episodes) of a static scenario, with constant channel ($h_t = 3\ Mbps, \forall t$) and constant video complexity $D = 4$. Both the proposed controller and the rate-based adaptation logic are implemented.

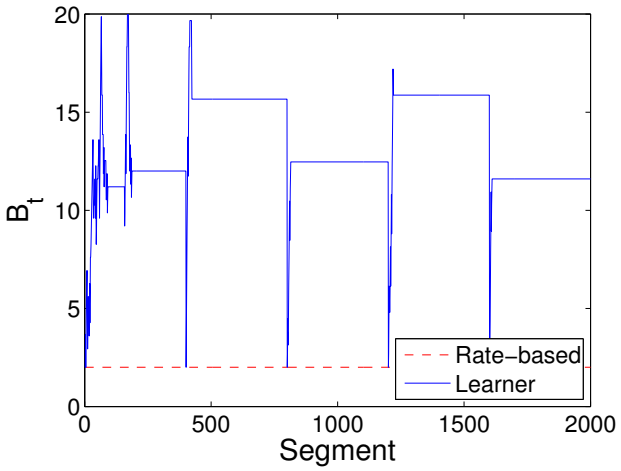

Figure 6: Buffer level B_t for the first 2000 segments (i.e., 5 episodes) of a static scenario, with constant channel ($h_t = 3\ Mbps, \forall t$), and constant video complexity $D = 4$.

tions, which is the main purpose of this section. Furthermore, their aim is also to show the ability of the proposed algorithm to work in different conditions, static and dynamic circumstances, and to show its fast adaptivity when conditions change.

6.2 Simulation Results

As a first result, we analyze the convergence of the proposed algorithm in a static scenario where both channel and video complexity are constant over time. In particular, we consider a constant bandwidth $h_t = 3\ Mb/s, \forall t$, and a constant value of D ($D_t = 4, \forall t$; all episodes have the same reference curve). To show the convergence of the controller, we provide the reward $r_t = r(D_t, q_{t-1}, B_t, a_t, h_t)$ experienced for different segments. Results are provided for both the proposed controller and the rate-based adaptation logic. Fig. 5 shows that the algorithm converges in less than 3 episodes (i.e., 1200 segments). Note that the start of each episode is clearly noticeable, as the reward has a downward spike every 400 segments, which is the episode duration. Since the first segment of each video is downloaded with an empty buffer and no information on the channel state, both the rate-based algorithm and the proposed algorithm (labeled as "learner") take a conservative approach and choose low bitrates, causing the downward spike in the reward. Fig. 6 shows the associated buffer behavior for each requested segment in the same constant scenario for both the proposed algorithm and the rate-based adaptation logic. The rate-based algorithm never builds up the buffer, as it always chooses the rate that matches the capacity exactly. The learner builds up the buffer in the first few segments, avoiding the risk of damaging rebuffering events, then it stabilizes. The rate-based algorithm never triggers rebuffering events, and neither does the learner after the first 200 segments.

We now consider a constant scenario with $h_t = 3.9\ Mbps, \forall t$. In this case, there is no action that matches the available bitrate exactly, since the closest video bitrates are 3 $Mbps$ and 4 $Mbps$; the most convenient policy is to alternate between the two closest available rates by selecting the lower one to build up the buffer, then switching to the higher one and maintaining it until the buffer supports it. The channel capacity is high enough to make this strategy optimal, since the switches are rare and the quality gain offsets the switching penalties. Fig. 7 shows the reward for each downloaded segment for both the rate-based and the learning controller. The learner can find the best trade-off between average quality and switching penalties while avoiding rebuffering events. On the contrary, the rate-based algorithm pays for being greedy and selecting a conservative best action. Fig. 8 shows the buffer status for the same episodes depicted in Fig. 7. The buffer management of the proposed RL algorithm maintains an acceptable level, avoiding the risk of rebuffering events while trying to maximize user QoE. The rate-based controller just fills the buffer blindly, without exploiting the available bandwidth fully to increase QoE.

The convergence of the learner is slower in more complex situations, but we found its speed to be similar to FA(Q)'s in dynamic cases: both learning algorithms reach convergence after a few hundred episodes at most, and none of the two learning algorithms was significantly faster than the other in any of the considered scenarios.

After giving the first intuitions on the controller behavior in a static scenario, we now consider dynamic ones. In the following simulations, we construct a channel transition matrix with $p = 0.25$ and one with $p = 0.5$. We also consider a dynamic video with an average scene duration of 5 segments; the proposed algorithm was compared to the rate-based and the FA(Q) algorithms. To maintain focus on the efficiency of the algorithm we provide results after 1000 training episodes in order to make sure that both learning algorithms have learned the model correctly, rather than another convergence scenario. We run 100 different episodes and for each episode we select the best actions according to the three algorithms, then evaluate the average SSIM and the SSIM standard deviation. From this set of 100 values we extract the minimum, the first quartile, the median, the

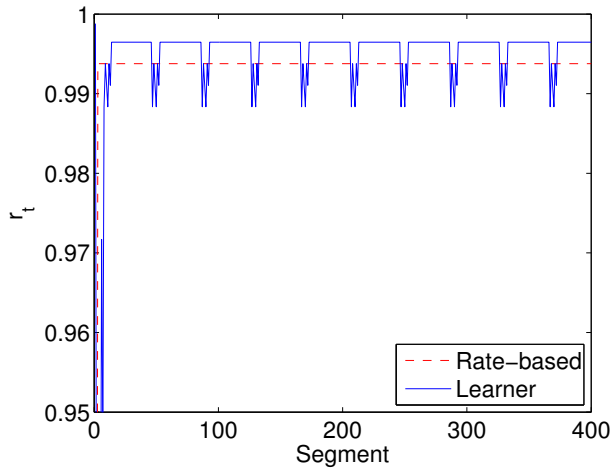

Figure 7: Reward experienced at the client side for one episode (after convergence) in a static scenario, with constant channel ($h_t = 3.9\ Mbps, \forall t$) and constant video complexity $D = 4$. Both the proposed controller and the rate-based adaptation logic are implemented.

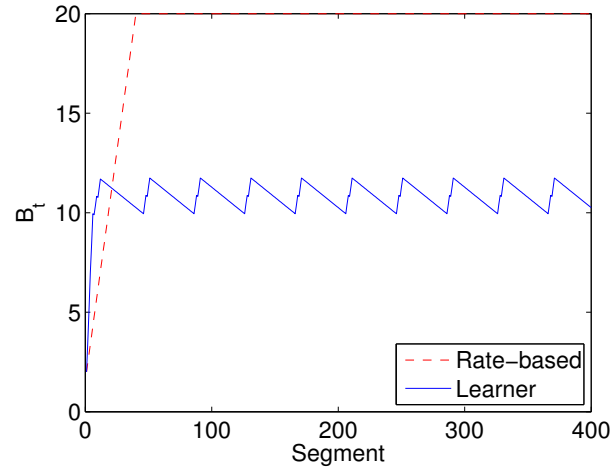

Figure 8: Buffer plot for one episode (after convergence) in a static scenario, with constant channel ($h_t = 3.9\ Mbps, \forall t$) and constant video complexity $D = 4$. Both the proposed controller and the rate-based adaptation logic are implemented.

third quartile and the maximum sample.

The results are presented as boxplots [28]: each box contains the values from the 25th to the 75th percentile; the extreme outliers are indicated. The line inside the box indicates the median, while the dot indicates the average value. Fig. 9 and Fig. 10 show the SSIM mean and variation respectively. The proposed algorithm has a higher average SSIM in most episodes for both values of p; the overall average SSIM for 100 episodes is 0.9786 for the proposed algorithm and 0.9758 for the rate-based algorithm. The FA(Q) algorithm has the worst performance in this scenario, with an average SSIM of 0.9726; this is probably due to its excessive

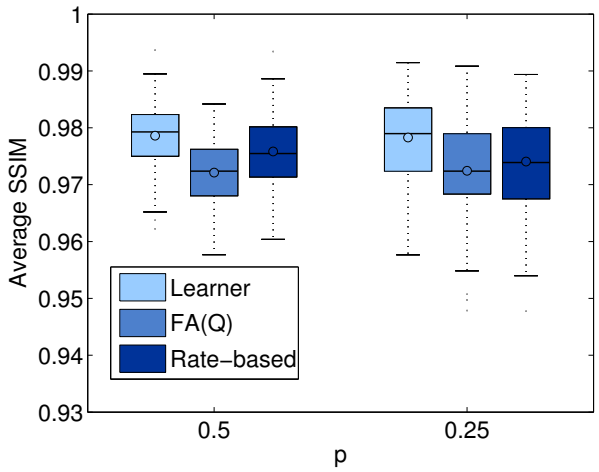

Figure 9: Average SSIM boxplot for 100 episodes with an average scene of 5 segments and different values of p

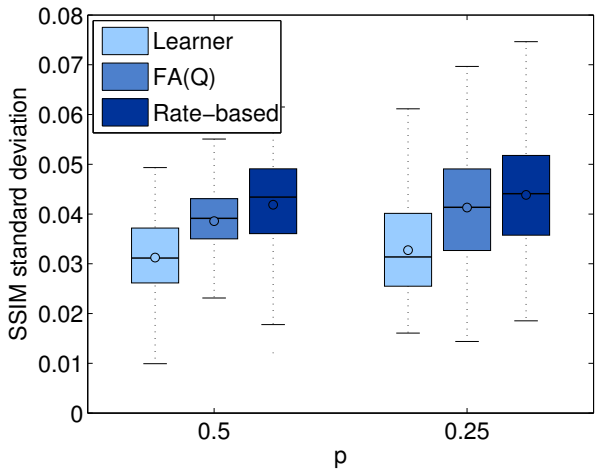

Figure 10: SSIM standard deviation boxplot for 100 episodes with an average scene of 5 segments and different values of p

prudence and unawareness of the video model. The results for $p = 0.25$ confirm this behavior; it is interesting to note that the results are not substantially different from the ones we described above, although the channel is less dynamic. This might be due to the fact that $p = 0.25$ still makes the channel very unstable, and any benefit from stability is insubstantial.

The proposed algorithm is also able to keep a steadier quality throughout each episode, as Fig. 10 shows; the average standard deviation is 0.0313 for $p = 0.5$, while the rate-based algorithm's is 0.0419. The FA(Q) algorithm manages to keep the quality steadier than the rate-based algorithm, with an average standard deviation of 0.0391, this value is however still higher than the value achieved by our controller.

In order to show the adaptive behavior of the proposed algorithm, we now consider another scenario, whose param-

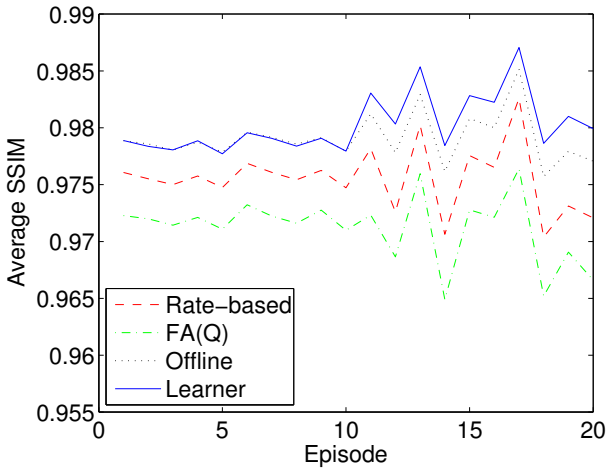

Figure 11: Adaptation: av. scene duration 5 segments, p goes from 0.5 to 0 after 10 episodes. Average SSIM

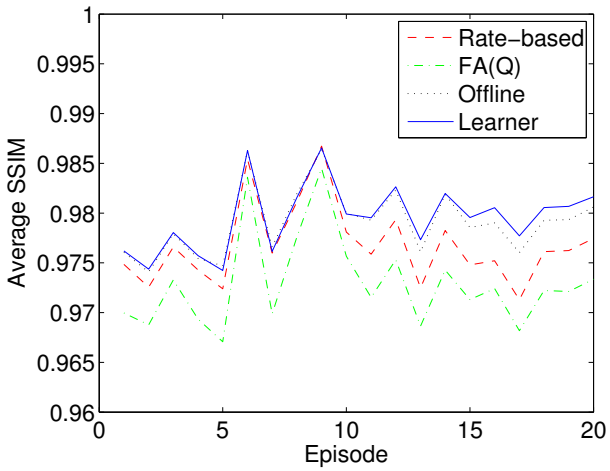

Figure 12: Adaptation: $p = 0.5$, from a static video to av. scene duration 1 segment. Average SSIM

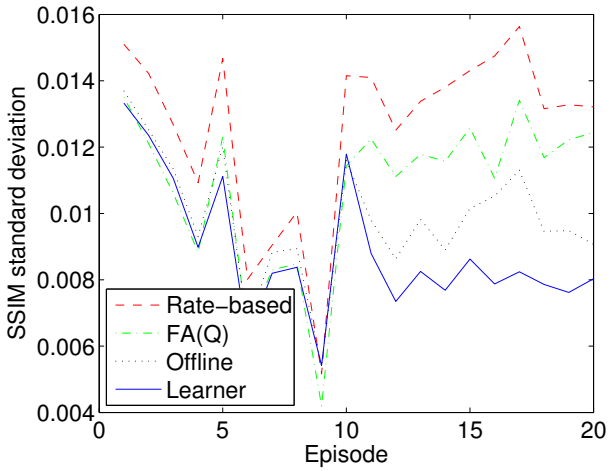

Figure 13: Adaptation: $p = 0.5$, from a static video to av. scene duration 1 segment. SSIM standard deviation

eters are constant in the first 10 episodes and then sharply change. The video statistics stay the same, while the channel goes from extremely dynamic ($p = 0.5$) to completely static ($p = 0$) after the 10th episode.

The simulations are averaged over 100 realizations to get more accurate statistics. In addition to the rate-based algorithm and the learner, we also show an "Offline" version of the proposed algorithm; after getting the same number of training episodes, the Offline Q-values are frozen and do not adapt to changes in the environment. The Offline version is meant to represent an offline learning algorithm like the one proposed in [6].

Fig. 11 shows the average SSIM for each episode; the online learning algorithm quickly adapts to the change after the 10th episode, increasing its advantage over the rate-based algorithm, while the Offline version's performance degrades after the change in the environment. The FA(Q) algorithm has serious difficulties in this scenario, since the controller does not consider the video dynamics in the state definition.

Fig. 12 and Fig. 13 shows the average SSIM and the SSIM standard deviation for another adaptation scenario; in this case, the channel model is constant (with $p = 0.5$) throughout the 20 episodes, while the video complexity model changes. The first 10 episodes have a static video, while the last 10 have a very dynamic video with an average scene duration of only 1 segment; the chosen value is purposely extreme to emphasize the effect of the adaptation. As FA(Q) cannot perceive the change in the video complexity, its performance is not affected by the change, but its choices are sub-optimal relative to the other two controllers.

Finally, we repeated the simulations for various transition matrices with two common features:

- the matrices were symmetrical and only allowed one-state transition (i.e., the channel could be described as a random walk with symmetric probabilities)
- the steady-state distribution was gaussian with $\sigma = 4$ centred on different values of capacity.

These two rules were chosen as a way to have a realistic channel model with the desired average capacity, without changing the state definition and providing comparable results.

The simulations have been repeated for 100 episodes. The results confirm that the proposed algorithm can achieve a higher SSIM for any average bitrate (see Fig. 14), as well as a lower standard deviation (Fig. 15).

The efficient buffer management allows to increase both the absolute quality and the quality stability, while avoiding annoying rebuffering events. The frequency of rebuffering events in all simulations was never higher than 10^{-3}, in particular it was never higher than the rebuffering frequency of the rate-based algorithm. Considering that the rate-based algorithm is very conservative, this result speaks for the validity of the learning approach. The FA(Q) algorithm proved very efficient at avoiding rebuffering events, experiencing at most one in every 10^{-4} segments, but proved to be inferior to the proposed algorithm in terms of QoE: Fig. 14 and Fig. 15

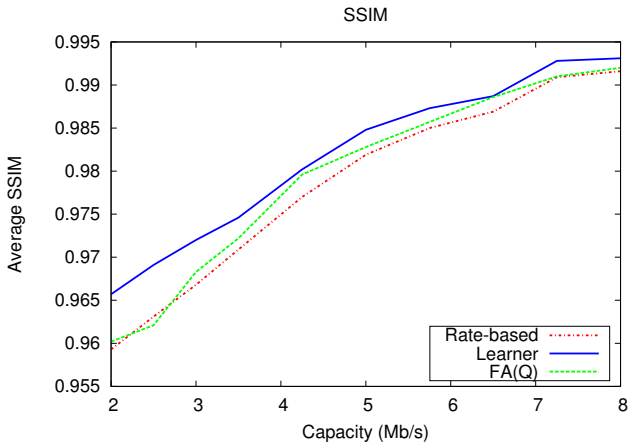

Figure 14: Average SSIM as a function of the average available bitrate

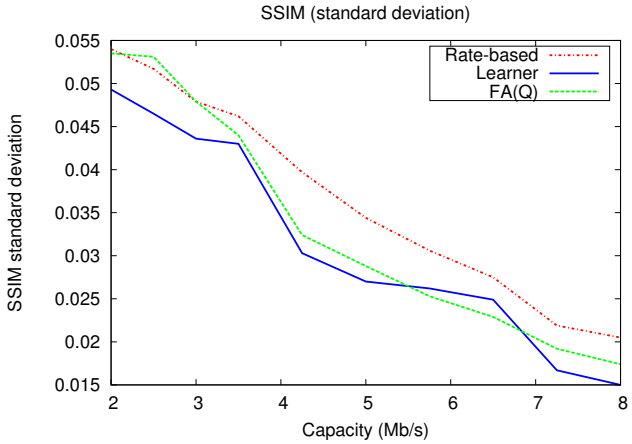

Figure 15: SSIM standard deviation as a function of the average available bitrate

show the performance degradation of FA(Q) for lower average capacity values, reaching performance results similar to the one achieved by the rate-based algorithm. Its quality is approximately as steady as the proposed algorithm's for higher capacities, but the proposed algorithm maintains a higher SSIM with its more aggressive buffer management strategy.

7. CONCLUSIONS

In this work we propose and design a RL-based online DASH controller. The objective of the controller is to maximize the QoE provided to the user aas a combination of both vide quality and quality variations, with the latter being perceived as an annoyance by the user. At the same time, the controller also tries to avoid rebuffering events, which severely degrade the user's experience. The problem is formulated as an MDP where the optimal actions are the ones maximizing the long-term reward. To learn the system dynamics that define the long-term reward, we consider a learning process. To speed up the learning process, we formulate the learning algorithm using PDSs, which decrease the amount of variables that need to be learned. Secondly, we exploit the particular structure of our system to parallelize the learning by using virtual transitions.

We implement our controller in Matlab and test it under different scenarios, comparing its performance with other baseline algorithms. The results of the simulations prove that the proposed RL bitrate adaptation algorithm is effective in a variety of environments, and its performance is satisfactory even in challenging scenarios. The proposed algorithm performs better than the most advanced RL-based example in the literature and a rate-based heuristic. The algorithm converges fast enough to allow it to quickly react to changes in the environment. This means that the system can be deployed without a significant pre-training effort and, more importantly, without any detailed prior knowledge of the channel and video model.

As for future work, we aim at implementing the algorithm in a more realistic scenario, using a real testbed or a packet level simulation which uses a realistic TCP transmission. Another interesting avenue of research is the interaction of several learning systems that share a single network bottleneck: in this work, the system has been developed from the perspective of a single client, but a network-wide approach would be very useful.

Acknowledgments

This work has been partly supported by the Swiss National Science Foundation under grant CHISTERA FNS 20CH21_151569.

8. REFERENCES

[1] Cisco. Cisco visual networking index: forecast and methodology, 2013–2018. *Cisco Public Information*, 2014.

[2] Ricky KP Mok, Edmond WW Chan, and Rocky KC Chang. Measuring the quality of experience of HTTP video streaming. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 485–492, 2011.

[3] Thomas Zinner, Oliver Hohlfeld, Osama Abboud, and Tobias Hoßfeld. Impact of frame rate and resolution on objective QoE metrics. In *Quality of Multimedia Experience (QoMEX), Second International IEEE Workshop on*, pages 29–34, 2010.

[4] Zhi Li, Ali C Begen, Joshua Gahm, Yufeng Shan, Bruce Osler, and David Oran. Streaming video over HTTP with consistent quality. In *5th ACM Multimedia Systems Conference*, pages 248–258, 2014.

[5] Maxim Claeys, Steven Latré, Jeroen Famaey, Tingyao Wu, Werner Van Leekwijck, and Filip De Turck. Design and optimisation of a (FA) Q-learning-based HTTP adaptive streaming client. *Connection Science*, 26(1):25–43, 2014.

[6] Maxim Claeys, Steven Latré, Jeroen Famaey, Tingyao Wu, Werner Van Leekwijck, and Filip De Turck. Design of a Q-learning-based client quality selection algorithm for HTTP adaptive video streaming. In *Adaptive and Learning Agents Workshop, part of AAMAS2013 (ALA-2013)*, pages 30–37, 2013.

[7] Michael Seufert, Sebastian Egger, Martin Slanina, Thomas Zinner, Tobias Hobfeld, and Phuoc Tran-Gia. A survey on quality of experience of HTTP adaptive

streaming. *Communications Surveys & Tutorials, IEEE*, 17(1):469–492, 2014.

[8] Shenghong Hu, Lingfen Sun, Chao Gui, Emmanuel Jammeh, and Is-Haka Mkwawa. Content-aware adaptation scheme for QoE optimized DASH applications. In *Global Communications Conference (GLOBECOM), IEEE*, pages 1336–1341, 2014.

[9] Jeroen van der Hooft, Stefano Petrangeli, Maxim Claeys, Jeroen Famaey, and Filip De Turck. A learning-based algorithm for improved bandwidth-awareness of adaptive streaming clients. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 131–138, 2015.

[10] Jongwook Lee and Saewoong Bahk. On the MDP-based cost minimization for video-on-demand services in a heterogeneous wireless network with multihomed terminals. *Mobile Computing, IEEE Transactions on*, 12(9):1737–1749, 2013.

[11] Stefania Colonnese, Francesca Cuomo, Tommaso Melodia, and Raffaele Guida. Cloud-assisted buffer management for HTTP-based mobile video streaming. In *Performance evaluation of wireless ad hoc, sensor, & ubiquitous networks, 10th ACM symposium on*, pages 1–8, 2013.

[12] Ayub Bokani, Mehdi Hassan, and Salil Kanhere. HTTP-based adaptive streaming for mobile clients using Markov Decision Processes. In *20th IEEE International Packet Video Workshop (PV)*, pages 1–8, 2013.

[13] Johan De Vriendt, Danny De Vleeschauwer, and David Robinson. Model for estimating QoE of video delivered using HTTP adaptive streaming. In *Integrated Network Management (IM 2013), IFIP/IEEE International Symposium on*, pages 1288–1293, 2013.

[14] Richard Bellman. A Markovian decision process. Technical report, DTIC Document, 1957.

[15] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*. MIT Press, 1998.

[16] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, et al. *Introduction to algorithms*, volume 2. MIT press, 2001.

[17] N. Mastronarde and M. van der Schaar. Fast reinforcement learning for energy-efficient wireless communication. *Signal Processing, IEEE Transactions on*, 59(12):6262–6266, 2011.

[18] Andrzej Ruszczynski. Commentary-Post-Decision States and separable approximations are powerful tools of approximate dynamic programming. *INFORMS Journal on Computing*, 22(1):20–22, 2010.

[19] Andrew G Barto, Richard S Sutton, and Chris JCH Watkins. *Learning and sequential decision making*. University of Massachusetts, 1989.

[20] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4):600–612, 2004.

[21] Marco Zanforlin, Daniele Munaretto, Andrea Zanella, and Michele Zorzi. SSIM-based video admission control and resource allocation algorithms. In *Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), 12th International IEEE Symposium on*, pages 656–661, 2014.

[22] Donald L Gilman, Fred J Fuglister, and J Murray Mitchell Jr. On the power spectrum of "red noise". *Journal of the Atmospheric Sciences*, 20(2):182–184, 1963.

[23] Dongeun Suh, Gwangwoo Park, Haneul Ko, and Sangheon Pack. Mobility-aware DASH for cost-optimal mobile multimedia streaming services. In *Computer Communications Workshops (INFOCOM WKSHPS), 2015 IEEE Conference on*, pages 57–58, 2015.

[24] Min Xing, Siyuan Xiang, and Lin Cai. A real-time adaptive algorithm for video streaming over multiple wireless access networks. *Selected Areas in Communications, IEEE Journal on*, 32(4):795–805, 2014.

[25] Eitan Altman, Konstantin Avrachenkov, Chadi Barakat, and Parijat Dube. Performance analysis of AIMD mechanisms over a multi-state Markovian path. *Computer Networks*, 47(3):307–326, 2005.

[26] Oliver Rose. Statistical properties of MPEG video traffic and their impact on traffic modeling in ATM systems. In *Local Computer Networks, 20th IEEE Conference on*, pages 397–406, 1995.

[27] Lark Kwon Choi, Yiting Liao, and Alan C Bovik. Video QoE models for the compute continuum. *IEEE Multimedia Communications Technical Committee (MMTC) E-Letters*, 2013.

[28] David F Williamson, Robert A Parker, and Juliette S Kendrick. The box plot: a simple visual method to interpret data. *Annals of internal medicine*, 110(11):916–921, 1989.