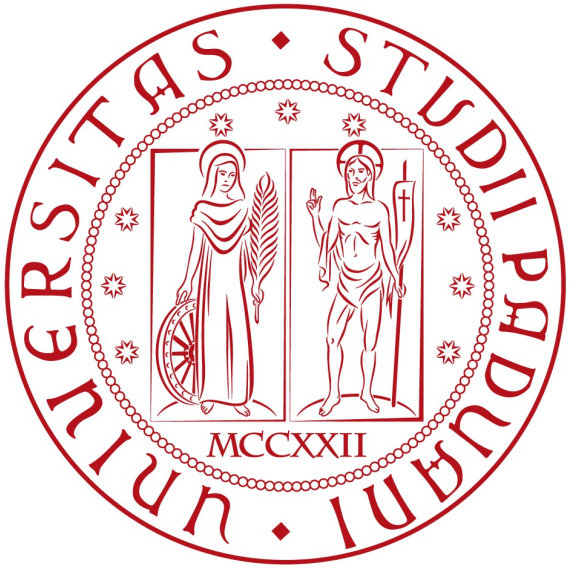
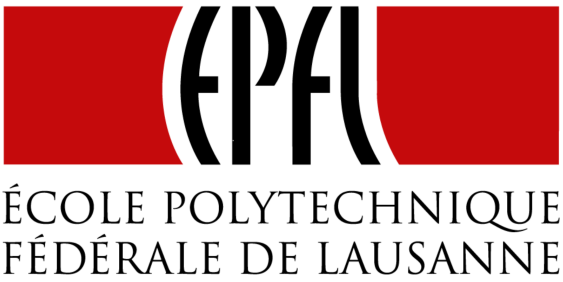
University of Padova
Academic year 2014-2015 (793rd)
Department of Information Engineering (DEI)
Master's thesis in Telecommunications Engineering





Reinforcement learning algorithms for DASH video streaming

Author: Federico Chiariotti
Advisors: Pascal Frossard*, Andrea Zanella°
Co-advisors: Stefano D'Aronco*, Laura Toni*

° University of Padova
* Ecole Polythecnique Fédérale de Lausanne

To my family, who supported me and helped me up to the last hurdle
To Francisco, Kristi and Robert, for all the good times in Lausanne

It should be noted that no ethically-trained software engineer would ever consent to write a DestroyBaghdad procedure. Basic professional ethics would instead require him to write a DestroyCity procedure, to which Baghdad could be given as a parameter.

N. Borenstein

Abstract

Dynamic Adaptive Streaming over HTTP (DASH) is a video streaming standard developed in 2011; the servers have several copies of every video at different bitrates, leaving the clients complete freedom to choose the bitrate of each segment and adapt to the available bandwidth.

The research on client-side strategies to optimize user Quality of Experience (QoE) is ongoing; one of the most promising approaches is based on Reinforcement Learning (RL). RL controllers do not have a pre-set model of the situation, but learn the optimal policy by trial and error.

This thesis presents two RL-based algorithms: Offline and Online. The Offline algorithm relies on a training phase to gain information about its environment and refine its policy, while the Online algorithm has a slimmer model and focuses on learning as quickly as possible, allowing immediate deployment and short convergence times without a training phase.

Contents

Chapter 1

Introduction

In the last 10 years, the data rates of both mobile and fixed networks have improved dramatically; this paved the way for the rise of video traffic, which has rapidly become the most important traffic source on the Internet. By 2012, video traffic represented more than 50% of total consumer traffic [1], and mobile video has a predicted growth rate of 70% until 2018. The total monthly video traffic should reach an impressive 100 EB in three years' time, more than all the traffic on the Internet right now, as Fig. 1.1 shows.

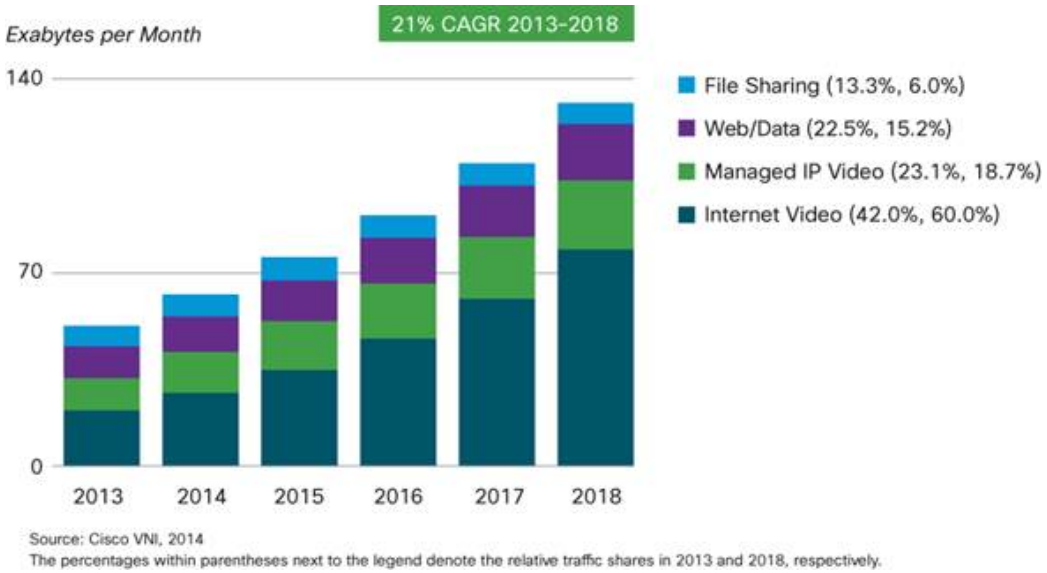


Fig. 1.1: Predicted Internet traffic by category (source: [1])

This extremely fast increase in demand poses a challenge to network operators and researchers; while the infrastructure continues to improve its capacity, the software optimization to provide high-quality video streaming without congesting it has established itself as an important research field.

While video streaming research in the late 1990s and early 2000s mostly focused on the User Datagram Protocol (UDP) along with a control stream [2], due to the overhead issues of the Transmission Control Protocol (TCP), it has shifted to TCP-based streaming in the late 2000s. The reason for this shift is mostly compatibility: the existing infrastructure of servers and network switches is based on TCP traffic, as most of the Internet relies on the HyperText Transfer Protocol (HTTP) which runs over it. The overhead cost of HyperText Transfer Protocol (HTTP) streaming was an acceptable price for efficient caching, no firewall problems and easier deployment of Content Delivery Networks (CDNs). Another advantage is that TCP effectively eliminates any error artifacts, as the transmission is reliable; freezes are the major quality issue in HTTP streaming, as they are due to delays that are hard to control in HTTP/TCP.

HTTP streaming systems use adaptive bitrate streaming to optimize user Quality of Experience (QoE); Apple, Adobe, Microsoft and other software companies have developed streaming systems based on it. In 2011, the Moving Picture Experts Group (MPEG) presented the new Dynamic Adaptive Streaming over HTTP (DASH) standard [3], the first international standard HTTP-based streaming solution.

Adaptive video streaming is a technique to dynamically change the video bitrate, adapting to network conditions. An adaptive client can measure the available bandwidth and choose one of several video representations at different compression levels, keeping the best possible QoE while avoiding rebuffering events, i.e., emptying the playout buffer completely and freezing the video until a new segment is available.

All current commercial implementations of DASH clients use heuristic algorithms, often extremely simple, to perform bitrate adaptation; these heuristics have been proven to be far from optimal, and the complexity of the problem makes it hard to find the optimum algorithm.

Reinforcement Learning (RL) is an efficient solution for this kind of problem: rather than relying on a fixed algorithm, learning agents can try different actions and gradually learn the best strategy for each situation. This is achieved by trial and error, with a reward function to provide reinforcement for efficient behavior. RL has been applied in a number of different applications, and if the problem is well-formulated it can become extremely efficient after a short training period. However, the research on the applications of RL for DASH systems has only just begun; the few published works about it have only been tested in limited and tightly controlled situations [4].

This work focuses on a novel RL solution, proposing two algorithms and testing them with extensive simulations in several different scnarios. Their efficiency and adaptability is shown to be superior to the existing examples in the literature [5].

In order to solve the bitrate adaptation problem with RL techniques, I developed a Markov Decision Process (MDP) that models its most important aspects; the Markov model focused on QoE aspects, and the prevention of rebuffering events was explicitly stated as a constraint in the problem formulation.

The two RL algorithms that work on the MDP are called Offline and Online. The Offline algorithm uses Q-learning [6], a standard RL algorithm, on a slightly expanded Markov model; it requires an extensive offline pre-training phase, but it can achieve a very high efficiency. The Online algorithm can learn to optimize the bitrate online, as the name suggests, and does not need a pre-training phase; it can achieve this through the use of parallelization techniques, which tweak the Q-learning algorithm to generalize its experience and update several states at once.

While the Online algorithm's performance is slightly worse, its extreme flexibility and reactiveness to changes in the model make it perfect for applications in fast-varying scenarios; the Offline algorithm is the other extreme in the trade-off, having better performance when the scenario fits its pre-training but having a harder time reacting to sudden changes.

1.1 Thesis outline

The rest of this thesis is organized as follows: Chapter 2 presents the general framework of the work, explaining the main ideas behind the DASH standard as well as the theoretical underpinnings of the RL algorithms. It also gives an overview of the state of the art on video bitrate optimization. Chapter 3 models the streaming problem as an MDP and describes the two proposed algorithms, outlining all the relevant modeling and design choices. Chapter 4 presents the simulation results for both algorithms, while Chapter 5 contains the conclusions and some possible avenues for future research on the topic.

Chapter 2

Framework

This chapter provides an overview of the systems and techniques underlying the whole work. It first presents the DASH standard and the video bitrate adaptation mechanism it supports, with a review of the most important video QoE issues.

As the DASH system architecture gives clients total control over the video bitrate, the client-side adaptation algorithm have to make foresighted choice to provide a high QoE. The dual objective of a video bitrate adaptation system is to provide the highest possible quality without emptying the buffer.

This is an ideal application for a reinforcement learner, as it can be formalized as an MDP and the optimal action policy can be learned by experience (either with an offline training or on the fly) even if the future evolution of the channel rate is unknown. The learner will optimize the expectation of the reward function, but since it can re-evaluate its choices for every segment it will never stray far from the correct path.

Although the advantages of learning systems in DASH video bitrate optimization are clear, the examples of RL-based algorithms for dynamic video quality adaptation in the literature are few and limited in scope; before describing these attempts, which share the general perspective of this work, we will introduce the basic theoretical concepts of RL, specifying the equations behind the most common RL algorithms.

2.1 DASH video streaming

As HTTP adaptive streaming was rapidly growing in the late 2000s, with several commercial implementations, the MPEG started to work on creating an open international standard for the technique. DASH was published in 2011, and it has rapidly become pervasive: the newest version of the HyperText Markup Language (HTML), HTML5, supports JavaScript-based DASH players as a way to embed video in webpages without using external plugins with the Media Source Extensions standard. Fig. 2.1 shows an example of how a DASH adaptive system might work.

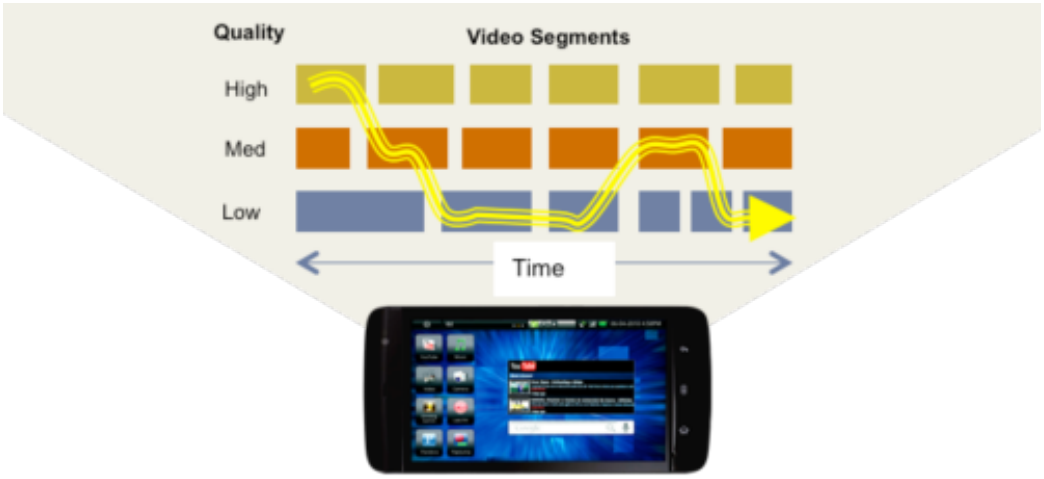


Fig. 2.1: An example of DASH video bitrate adaptation (source: Qualcomm DASH project)

One of the most important features of the DASH standard is the Media Presentation Description (MPD), a document containing all the metadata that the client needs in order to stream the video in Extended Markup Language (XML). As the video is divided into short segments to make quality adaptation possible, the MPD contains the location and playout information of each segment [7]. When a client requests a segment, it can download it with a simple HTTP GET request, as all the necessary information is embedded in the MPD. The DASH standard is also codec agnostic; it works with several of the main codecs, leaving decodification entirely to the client.

2.1.1 The DASH data model

In the DASH data model, a video is split into different adaptation sets, containing different aspects such as the video stream, the audio stream and subtitles. Each adaptation set has a series of representations, whose number is constant throughout the video (or at least a video period, such as the first act of a movie). Clients can choose among different representations in the same adaptation set, as they effectively represent the same content; this allows the user to choose the compression level for each video segment and to adapt the requested video bitrate to the available channel rate [8].

The different segments are identified by their Uniform Resource Locators (URLs), and may be either separate files or within the same file (using byte offsets to separate them); using separate files allows for more efficient caching, so it is often preferred. The MPD entry about each segment also contains its video bitrate, resolution and duration, as well as optional fields such as a number of QoE metrics. Each segment must begin with a Stream Access Point (SAP), allowing the client to decode and play it without any previous information. This is necessary to avoid errors during quality switches, as the previous segment may be part of a different representation and have different characteristics.

The length of segments must be decided by the server when preparing the representation, and it is the result of an application-dependent tradeoff between reaction speed to changing network conditions and overhead. Short segments allow the client to adapt to the channel rate faster, but they increase the HTTP overhead.

The type of compression schemes used to adapt the video bitrate also has an effect on the resulting QoE: the most common are reducing the video frame rate or resolution and increasing the Quantization Parameter (QP) to get a coarser compression. The QoE strongly depends on the encoder, as well as on the content itself; [9] shows that H.264 [10] outperforms other encoders and that high QoE can be obtained by tuning the QP. Combinations of various adaptation techniques are also possible to optimize the tradeoff between QoE and video bitrate.

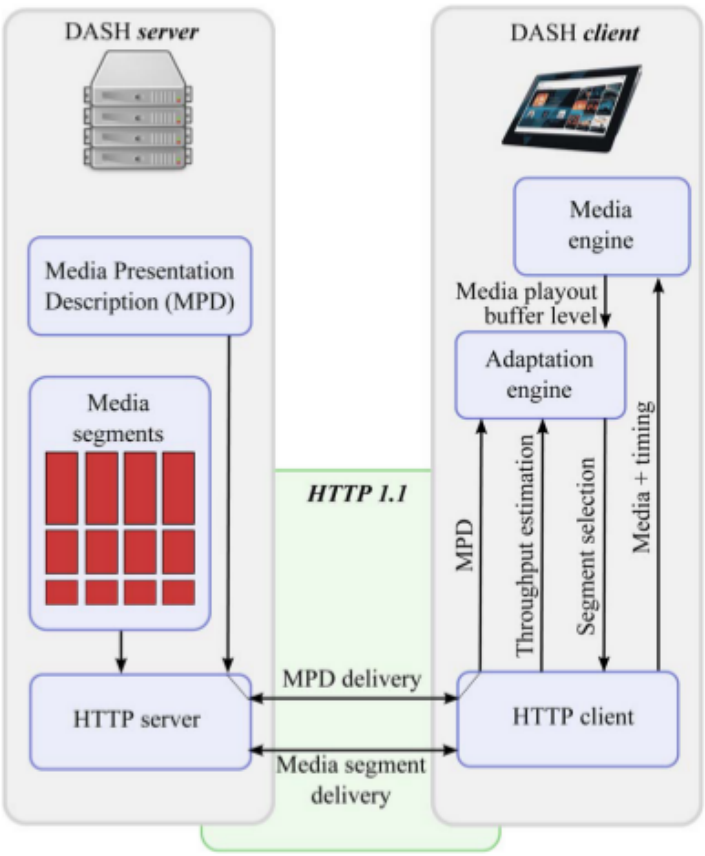


Fig. 2.2: Structure of a typical DASH system (source: [11])

As DASH servers are basically standard HTTP servers, the video bitrate adaptation algorithm is entirely client-side (see Fig. 2.2); a client only needs to send the appropriate HTTP requests to implement any policy. This means that DASH clients are free to choose the quality for each segment from the available adaptations, performing video bitrate adaptation to avoid exceeding the available channel rate, which may otherwise lead to the most important source of QoE drops: rebuffering events. Whenever the video buffer empties, the client has to wait for the next segment to finish downloading, stopping the video playback and annoying the user [12].

Most of the commercially available systems implement simple heuristics that try to stabilize the buffer levels: intuitively, this avoids most rebuffering events by keeping a high buffer level, while increasing the quality every time

the channel rate is high enough to fill the buffer over the desired level. These heuristics do not take QoE effects into account, and several studies have found them to be inefficient in terms of QoE optimization [13].

A possible performance enhancement to the DASH paradigm is Pipelined DASH: a pipelined client can send more than one HTTP 1.1 request at the same time [14], downloading either more than one segment or several parts of the same segment (using byte offset). Pipelining can be beneficial in mobile networks with high packet loss and latency [15].

The DASH standard also supports live streaming (although the server needs to support it), as the MPD can be fragmented and downloaded piecewise as the new content appears on the web server; obviously, live streaming presents additional problems, as both the information and the segments available to the client are limited by the real-time constraint.

2.1.2 QoE issues

One of the characteristics that make video transmission challenging is the complex structure of videos: the effects of errors, compression and delay depend on the video compression algorithm as well as the video content itself. Several attempts at correlating user QoE and lower-layer Quality of Service (QoS) parameters have been made [16], and most optimization algorithms consider explicit QoE metrics as well as QoS.

The TCP retransmission mechanism overcomes most of the main sources of QoE issues in UDP streaming: as the application-layer transmission unit is a video segment whose length is measured in seconds, normal Internet latency does not affect the user experience, and out-of-order packets are sorted out by the transport layer. Furthermore, while in UDP-based streaming a lost packet simply propagates to the application layer, in TCP it manifests as lower channel rate as the packet retransmission adds strain to the channel.

The QoS parameter that impacts the most on DASH streaming QoE is the channel rate, which imposes a strict limit on the video bitrate. As DASH is an Application layer standard, the channel rate that DASH systems experience already accounts for TCP packet retransmission, which happens

on the Transport layer. Its relation with QoE is approximately logarithmic, i.e., a channel rate increase has a stronger effect on QoE if the channel rate is low, while when the channel rate is already high the benefits are minimal.

Along with the limit given by the channel rate, streaming clients must take into account the dynamic nature of most channels: a variation in channel rate may cause the available video buffer to empty out before the next segment has been fully downloaded, triggering a rebuffering event. Rebuffering events always damage QoE, but their impact depends on their frequency and duration [12], and a series of several rebuffering events in a short time seems to be the most undesirable pattern even if the freezing time is relatively short [17]. The initial buffering delay also plays a role in video QoE, but [18] shows that the impact on Mean Opinion Score (MOS) of delays up to 16 seconds is marginal, and 90% of users prefer it to rebuffering.

Another problem is caused by the time variance of the video quality: significant video bitrate adjustments by the DASH client may result in noticeable quality switches, which have been proven to be detrimental to user QoE [19]. Choosing between maintaining a constant quality level and following the channel rate closely to prevent rebuffering events and get the highest possible quality is the most important trade-off in DASH video bitrate adaptation.

2.1.3 QoE metrics

The first and most immediate metric for QoE is MOS: it is simply the average of a series of users' evaluations, going from 1 to 5, in controlled experiments. For obvious reasons, MOS is not a practical metric for online applications or when the dataset is large; its subjective nature makes an automatic calculation impossible. MOS is often used as a benchmark to compare the accuracy of objective metrics; the more strongly a metric correlates with MOS, the closer it is to representing the actual user QoE.

One of the most used metrics in the scientific literature is Peak Signal to Noise Ratio (PSNR); it is a very simple metric, and it requires a reference image to measure similarity. It is usually expressed in dB, and it represents

the ratio between the highest possible value of a pixel and the Mean Square Error (MSE) between the reference image and the received image; one of the issues with PSNR is that it does not take the dynamic characteristics of the video into account, but only operates on a frame-by-frame level.

$$\mathrm{PSNR}_{dB} = 20\log_{10}(\mathrm{MAX}_i) - 10\log_{10}(\mathrm{MSE}) \tag{2.1}$$

Another important metric is the Structural Similarity Index (SSIM) [20]; it calculates errors locally, providing a better model for the human perception of images than PSNR, but it is also more complex. Like PSNR, SSIM is a full reference metric, as it measures similarity to a reference image on a frame-by-frame level. Mathematically, the SSIM of a frame X with respect to a reference frame Y is given by

$$\mathrm{SSIM} = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{2.2}$$

where x and y are usually 8×8 pixel windows. SSIM values have a dynamic range that goes from -1 to 1; an SSIM value of 1 is reached only if the two images or videos are exactly identical, while an SSIM value of -1 means they are perfect negatives. In Eq. (2.2), μ_j stands for the average of j, σ_j stands for the variance of j and σ_{ij} stands for the covariance of i and j. The two constants c_1 and c_2 depend on the dynamic range of the pixel values and are needed to stabilize the division, and

More complex metrics have been developed in the past few years, often using neural networks to approximate the human evaluation of images and videos, such as the Pseudo-Subjective Quality Assessment (PSQA) metric [21]. In [22], Singh *et al.* adapted the PSQA metric to DASH with H.264 encoding and trained a neural network on a series of videos, obtaining values close to the empirically measured MOS. These learning-based metrics are quite accurate but extremely complex, making them less common in the literature than PSNR and SSIM, which provide good accuracy with a limited computational cost.

2.2 Reinforcement learning

RL is a machine learning paradigm inspired by behaviorist psychology [23]. The basic mechanism behind RL is trial and error: the agent does not have a complete model of the environment or the effects of its actions, but it gradually learns how to maximize the reward by trying different actions and making mistakes. Learners memorize the consequences of their actions and avoid them if they yielded low rewards in the past or if they led to a low-rewarding state. Fig. 2.3 summarizes the main concepts of RL, with a clear distinction between the agent and its environment.

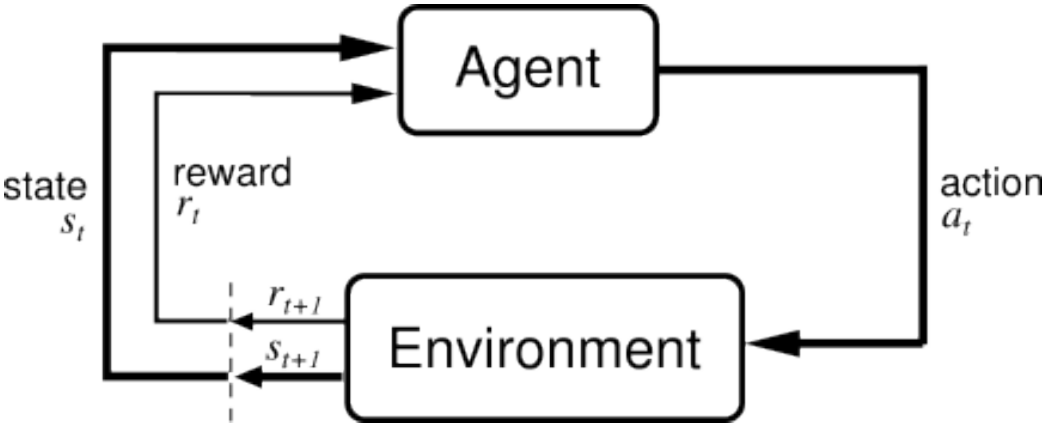


Fig. 2.3: The reinforcement learning basic concept (source: [23])

RL problems are formulated as MDPs [24], which can be defined by the 5-tuple $(S, A_s, P_a(s_t, s_{t+1}), R_a(s_t, s_{t+1}), \gamma)$.

- S is a finite set of states
- A_s is a finite set of actions available to the agent in state $s \in S$
- $P_a(s_t, s_{t+1})$ is the transition probability from state s_t to state s_{t+1} when the agent chooses action $a \in A_{s_t}$
- $R_a(s_t, s_{t+1})$ is the immediate reward that the agent gets when it chooses action $a \in A_{s_t}$ and the state changes from s_t to s_{t+1}

- γ is an exponential discount factor on future rewards. In order to avoid divergence, it needs to be in the $[0, 1]$ interval. The long-term reward is given by $\sum_{\tau=t}^{\infty} r_\tau \gamma^{\tau-t}$

Basically, the learner's environment is a Markov chain whose transition probabilities depend both on the learner's action and on its external environment [25].

A policy $\pi(s)$ is a function that links every state to an action; it usually depends on the expected long-term reward of each actions that the learning agent estimated from previous experiments. The objective of the agent is to find the optimal policy $\pi^*(s)$ that maximizes the long-term reward in each state. The long-term reward is an exponentially weighted sum of the future rewards: the learner's reward depends on the future evolution of the MDP, so a myopic choice that maximizes instantaneous reward but puts the learner in a low-yielding state may not be the best course of action.

When the time window for the long-term reward is finite, and the MDP is relatively small, standard dynamic programming techniques can find the optimal policy. RL algorithms do not provide an exact solution, but they do not need complete knowledge of the MDP to converge and can work for very large MDPs, with state spaces that would be unmanageable for dynamic programming.

Unless the learner acts on a predefined learning set, one of its most important parameters is the exploration rate: if it always tries to take the optimal action given its limited information, it may never learn the whole system and get stuck on a sub-optimal policy. On the other hand, taking random actions increases the learner's knowledge of the system, even if it is far from the optimal policy. The balance between exploration and exploitation is one of the central problems in RL.

2.2.1 Exploration strategies

Different policies can have different balances of exploration versus exploitation. One extreme is the greedy policy: a greedy agent always tries to maximize its reward to the best of its current knowledge, without any concession

to exploration. The greedy policy is optimal if the agent has perfect knowledge of the environment, but it can lead the RL agent to getting stuck on sub-optimal actions, as it will never try actions it thinks are sub-optimal to verify its information. Because of this, the greedy policy is not very suited to learning agents, who must be able to converge to the optimal policy with little or no initial information. It may however be a good choice after the training phase is complete and the learner does not need further exploration.

One of the most common RL policies is ε-greediness [26]: an ε-greedy client behaves greedily with probability $1-\varepsilon$ and chooses a non-greedy action at random with probability ε. This strategy guarantees a certain degree of exploration, which can be changed by adjusting the ε parameter. However, this strategy does not distinguish between the non-greedy actions, and so it does not discard very damaging actions.

Softmax [27] is another policy that deals explicitly with the exploration problem; it works by converting each action's expected reward to a probability and choosing the action to take according to the resulting distribution, which is given by the following formula:

$$P(a_j) = \frac{e^{\frac{Q_t(a_j)}{\tau}}}{\sum_{i=1}^{|A_s|} e^{\frac{Q_t(a_i)}{\tau}}} \tag{2.3}$$

In Eq. (2.3), the parameter τ, usually referred to as "exploration temperature" or simply "temperature", can be adjusted to control the exploration rate; $Q_t(a)$ represents the estimate of the expected reward if the learner took action a. Actions with high expected rewards $Q_t(a)$ have a higher probability to be chosen, so the exploration is directed towards the most promising directions.

Both Softmax and the ε-greedy policy can use time-dependent parameters, reducing the exploration rate as the agent gains more knowledge about the MDP and converges towards the optimum actions for all states.

Learning algorithms have a behavior policy and an estimate policy. The estimate policy is used for updating expected rewards for each state-action

pair, while the behavior policy is used for choosing the next action. Algorithms with the same behavior and estimate policies are called on-policy algorithms, while the others are called off-policy. Off-policy algorithms usually have a greedy estimation policy, while their behavior policy allows for more exploration.

2.2.2 Q-learning

Q-learning is a method for off-policy control developed by Watkins in 1989 [6]. It is a Temporal Difference (TD) algorithm, as it can deal with delayed rewards. The algorithm keeps a table of the expected long-term rewards (or Q-values) for all the possible state-action pairs, gradually improving its estimates by adjusting them to its experience. The Q-learning update formula guarantees that the Q-values will eventually converge to the real long-term expected reward [28], and it is given by:

$$Q_t(s_t, a_t) = Q_{t-1}(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q(s_t, a_t)] \qquad (2.4)$$

The long-term rewards are approximated using the next state's Q-values, with a greedy estimate policy; this ensures convergence, although rewards delayed by more than a few steps might take a long time to propagate to the choice that caused them. The learning rate α is a parameter that adjusts the update speed; it is usually decreased at the end of the training phase.

The algorithm needs some initial Q-values to start, as it does not specify an initial state; the common solution is to set all Q's to a high value, encouraging exploration until some reliable values are estimated.

The on-policy equivalent of Q-learning is the State-Action-Reward-State-Action (SARSA) algorithm [29], in which the max operator in Eq. (2.4) is replaced by the behavior policy.

Although the difference between Q-learning and SARSA seems to be minimal, it can make a significant difference in terms of results: SARSA generally tends to be more conservative, as it needs to take into account the sub-optimal exploration actions and avoids actions that take the agent to a

risky state (i.e., if one of the sub-optimal actions in state s_{t+1} has damaging consequences, SARSA tends to avoid actions that take the learner to that state, while Q-learning just assumes the next action will be the optimal one).

2.2.3 Eligibility traces

TD algorithms can use different strategies to get an estimate of the long-term reward when updating Q-values; as Q-learning and SARSA do not look beyond a single step in their state updates, relying on the Q-value of the next state to get an estimate of the long-term reward, they are called TD(0).

TD(λ) algorithms [30] can converge faster by using eligibility traces: instead of updating the Q-values of the previous state at each step, they wait for a number of steps and then update all the visited states. This is a way to increase the accuracy of the estimate of the long-term reward, using actual rewards for a number of steps before relying on an estimate. The weight of the estimate can even become negligible, depending on the number of steps and on the exponential discount term λ.

Using eligibility traces makes state updates more efficient, thus speeding up the convergence of the Q-values by reducing estimation errors; most TD(0) algorithms can be modified to add eligibility traces.

Q(λ) is a combination of standard Q-learning and the TD(λ) approach, adding eligibility traces to Watkins' original algorithm. There are several versions of Q(λ); one of the most used is Watkins' implementation, which only looks ahead until the next non-greedy action (see Fig. 2.4). If an exploratory action is taken n steps after step t, the backup formula for the state-action pair (s_t, a_t) is

$$R_{t+n}(s_t, a_t) = \sum_{i=1}^{n-1} \lambda^i r_{t+i} + \lambda^n \max_a Q_t(s_{t+n}, a) \tag{2.5}$$

$$Q_{t+n}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(R_{t+n}(s_t, a_t) - Q_t(s_t, a_t)) \tag{2.6}$$

Eq. (2.6) is almost identical to Eq. (2.4), but the future reward is given by the actual discounted reward instead of the next state's Q-value. The

eligibility trace is derived in Eq. (2.5).

Watkins' Q(λ) is one of the simplest implementations of Q(λ), and more complex implementations might converge faster as they do not stop updating at every exploratory action. However, its convergence is still guaranteed.

2.2.4 Continuous states

All the RL algorithms presented in the previous sections work on MDPs, learning the optimal action for each of the states of the underlying Markov chain. Each of the states is discrete and clearly separated from the others; however, any continuous variable (such as network capacity) needs to be quantized in order to be represented as a Markov chain.

Quantizing a variable, i.e., mapping it to a finite number of discrete values, always results in a loss of information, whose magnitude depends on the quantization step, i.e., on the size of the intervals mapped on the same value.

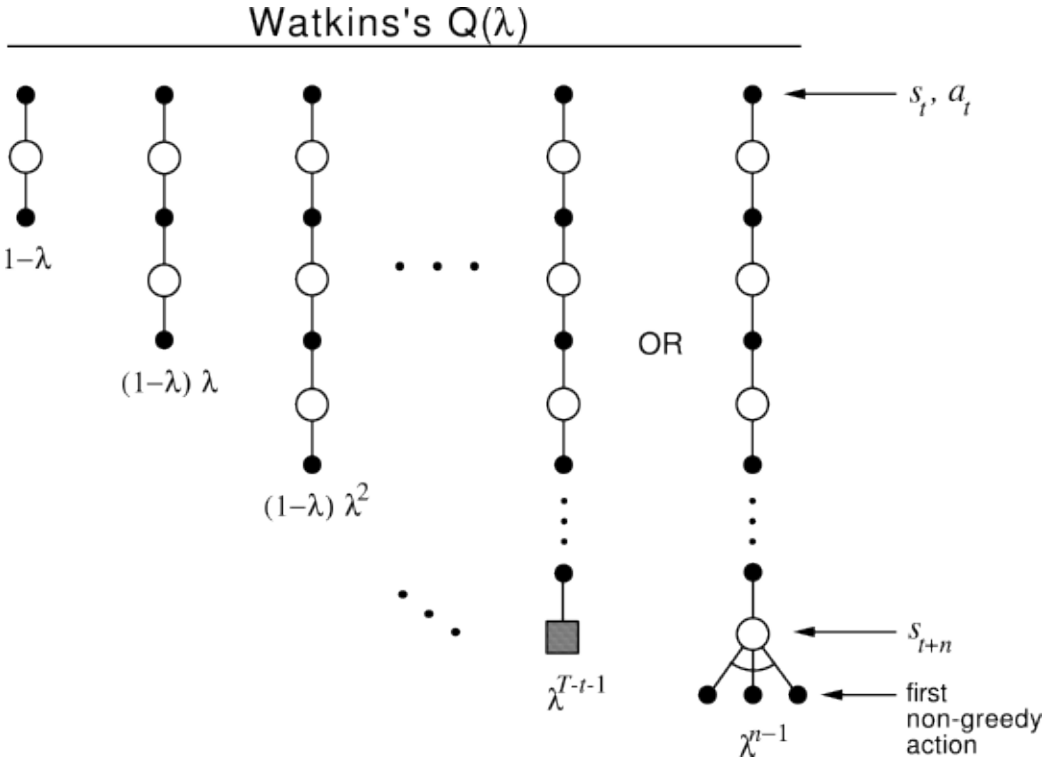


Fig. 2.4: Eligibility trace generation in Watkins' Q(λ) (source: [23])

If the quantization step is small enough, the dynamic range of the variable within a given state will be very small, and thus have a small impact on the learning agent.

If the quantization step is too coarse, border effects may arise: depending on the choice of the quantization intervals, the states may not capture the behavior of the system and small transitions close to the border between two states may be perceived as huge changes by the learner, which only knows that the state of the system has changed.

On the other hand, a fine quantization increases the complexity of the system, which grows with the number of states. If the number of states is too large, there may be problems with the training or even with the in-memory storage of the Q-value table. In order for the learner to converge, the training process should involve visiting most of the states several times so that the learner can try out different actions and discover their consequences. This is especially true when the learner has no experience at the beginning of the training (i.e., all of its Q-values are set to the same value). This poses limits on the number of states, as the computational cost of the training can become overwhelming.

The trade-off between capturing the full complexity of the state and achieving an acceptable computational performance is one of the main problems of Markovian models; one method of overcoming it is clustering states [31], another is using softer borders. Soft state borders mean that whenever the learner is close to the border between two states it can use a linear combination of both states' Q-values, avoiding hard transitions and the associated border effects.

Generalizations to states based on fuzzy rules [32] and more rigorous studies of state approximation [33] [34] can be found in the literature.

2.3 Optimization of streaming strategies

After introducing the DASH standard and the theoretical basis of RL, this section gives an overview on the optimization of video bitrate adaptation strategies, with a particular focus on RL-based strategies.

The current commercial streaming clients use buffer-based heuristics to perform video bitrate adaptation. Ever since HTTP streaming became popular and commercial solutions based on it started appearing, academic research has focused on providing better algorithms for video bitrate adaptation.

An early example is a centralized dynamic programming-based resource allocation algorithms for cellular networks by Thakolsri *et al.* [35]. They aimed at providing resource fairness between DASH clients connected to the same base station, but the complexity of the dynamic programming puts a heavy computational load on the base station itself.

A more systematic approach has been proposed by Jiang *et al.* with the FESTIVE framework [36]: along with other video streaming optimization techniques, they proposed a multi-user algorithm that aims at fairness and efficiency. They also addressed an important problem of streaming over TCP: if the client waits between segment downloads, the channel rate estimation may not be accurate and suffer from vicious cycles with competing clients, which results in wasted resources and increased unfairness. The authors included a random component in the waiting times to avoid this problem.

Another scheme is proposed by Li *et al.* in [37]; the authors use dynamic programming with a reward function tuned to decrease needless quality switches that negatively impact user QoE. The algorithm is also tested in a multi-user environment, and it manages to improve quality, but it tends to increase the buffer needlessly, which may be suboptimal in variable network conditions and is generally considered wasteful.

The dynamic programming-based algorithms are all extremely computationally expensive, and they might not be ideal for mobile devices with battery and computational power limitations.

Xing *et al.* propose a dynamic programming-based algorithm [38] for multiple access networks (specifically, WiFi+3G) that optimizes video quality

while privileging the WiFi link over the expensive 3G link. This algorithm also penalizes frequent quality fluctuations.

A comprehensive review of QoE issues and adaptation algorithms in DASH is presented by Seufert *et al.* in [11].

2.3.1 Reinforcement learning and DASH

Video bitrate adaptation is a textbook use case for RL: there is a clearly defined reward (the QoE, measured with one of the established metrics) and an environment (both the video content and the channel, which are outside the client's direct control) which can be modeled as an MDP. Due to the newness of DASH, the current attempts at designing RL-based video streaming clients are all very limited in scope.

One of the first applications of Q-learning to video streaming is the centralized algorithm proposed by Fei *et al.* [39]. The authors considered a cellular system with intelligent base stations and applied Q-learning to optimize Call Admission Control (CAC) and video bitrate adaptation in order to try to reduce handoff dropping probabilities in 3G networks. This approach is similar to that adopted in [35], but its centralized nature makes it unsuitable for distributed systems such as DASH clients.

A number of works on Q-learning in DASH systems have been published by Claeys, Petrangeli *et al.* since 2013. In [5], the authors designed a system based on the Q-learning algorithm that uses a quality-based reward function, with penalties for "buffer panic" states. The algorithm rewarded filling the buffer as well as keeping a high quality level, testing both the Softmax and VDBE-Softmax exploration policies and different discount factors. Their simulations showed a significant improvement over the Microsoft Smooth Streaming (MSS) heuristic, but the scenario is limited to a very simple channel with a constant channel rate and slow-varying cross-traffic.

In [40], the same research group modified the Q-learning algorithm they used in [5] to speed up updates of low-frequency states, improving perfromance in a scenario with a sinusoidally varying channel rate without effects on the static scenario.

The same authors investigated a way to achieve fairness in multi-user scenarios using RL algorithms [41]; by including a global fairness signal into the reward function, each client behaves less greedily towards network resources. The global signal needs to be transmitted, and the authors propose inserting network elements called fairness proxies at the bottleneck links in order to propagate it. Although this approach is very expensive in terms of network infrastructure, achieving fairness in a complex network scenario with cross traffic and multiple links is a difficult problem and no efficient solutions are currently available.

Finally, in [4] the same research group proposed a learning adaptation to the existing heuristic: instead of using a learning agent to control the dynamic video bitrate adaptation, they used a variant of the MSS heuristic with the same basic algorithm but different parameters and optimized the parameters by reinforcement learning. The learning-based algorithm reacted better than MSS to changes in the channel rate and was able to reduce playout delay in live streaming situations by keeping the buffer shorter, while avoiding buffer panic events.

Although these works show promise in the application of RL techniques to HTTP video streaming, the limitation of these first systems are evident: the simulations only use very simple cases, and the complexity of the states makes a practical implementation daunting.

Other RL-based DASH clients focus on different aspects of the problem; a work by Changuel *et al.* [42] focused on time-varying channels. The authors used a different Markov model and considered the case of partial or even no channel rate information in a varying channel in their simulations. They used the reference algorithm for Scalable Video Coding (SVC) as a baseline and showed that the learning-based client performed slightly better in all the situations they considered.

Another important aspect of the DASH optimization problem is the estimation of the network condition and its impact on RL algorithms; this issue has been addressed by Marinca *et al.* in [43]. In their work, the authors created a Partially Observable MDP (POMDP) model of the streaming problem, considering the fact that some of the network characteristics may

be unkown to the client, and solved the optimization problem offline in a series of examples. After creating a simplified MDP that represented part of the complete POMDP problem and solving it, they compared the results with the optimal solution and found that RL algorithms can perform well in this scenario even when only part of the data is available.

The approach in this work radically simplifies the MDP model, aiming at a simple but descriptive formulation with a limited number of states that increases convergence speed and adaptability without negatively affecting system performance.

Chapter 3

RL-based DASH optimization

RL-based solutions in the literature often use very complex Markov chains with millions of states, created *ad hoc* for a particular scenario; the optimization problem and MDP formulation in this work avoid this by making the underlying model as simple and powerful as possible, increasing the convergence speed of the Q values and the adaptability of the algorithms that work on the model.

The function of the learning agent is to choose an action a_t, i.e., choose which of the available representations to use for the next video segment download. The segment is the basic download unit for the video streaming client, as it is not possible to switch representations in mid-segment; its duration does not influence the formulation of the optimization problem.

The quality q_t of a segment can be measured with any of the standard quality metrics, and is a function of the chosen representation and the complexity D_t of the segment: $q_t = g(a_t, D_t)$. Complexity is a parameter that reflects the relation between video bitrate and QoE; as the effect of video compression on QoE is strongly dependent on the video content, considering the complexity of the video is fundamental. The other parameters that the learner uses are the buffer level B_t and the estimated channel rate h_t. Both are easily available to a video streaming client, and the quality of a segment can be calculated as $g(a_t, D_t)$ or estimated online. The buffer B_t is measured in seconds instead of bits, as the agent's objective is to avoid rebuffering

events during playout (which is measured in seconds) and not to manage the buffer memory.

The download time D_t of a segment is a function of the chosen representation a_t and the channel rate h_t: $D_t = f(a_t, h_t)$. Estimating h_t for downloaded segmengs is possible by simply inverting the function, as both the chosen representation and its download time are known.

The notation used in the problem formulation is the following:

- q_{t-1}: the quality level at which the last segment has been downloaded at time t
- h_{t-1}: the average channel rate during the last segment download at time t
- D_t: the complexity parameter of the current segment that the user needs to download at time t
- B_t: the status of the buffer before downloading the segment at time t
- s_t: the MDP state, corresponding to the 4-tuple $\{q_{t-1}, h_{t-1}, D_t, B_t\}$
- A_{s_t}: set of possible actions in state s_t
- a_t: action (i.e., video bitrate level) at time t
- $R(a_t)$: size of the enxoded segment a_t
- $\mathbf{A}$: possible sequence of future actions from time t onwards
- P_t: playout duration of the next downloaded segment at time t
- γ: exponential discount parameter
- $V(s_t)$: state value function

The symbols β, γ, ρ, σ and B_M represent the parameters of the reward function.

3.1 Optimization problem

The client has to select a representation a_t for segments with $t = \tau, \ldots, \infty$, where τ is the current time instant. In a deterministic case in which the future dynamics of both the channel and the video are known for any instant, the client should select the best set of actions $\mathbf{A}^\star$ among all possible action vectors $\mathbf{A} = [a_\tau, a_{\tau+1}, \ldots, a_\infty]$, so that

$$\begin{aligned} \mathbf{A}^\star : \underset{\mathbf{A}}{\operatorname{argmax}} & \left\{ \sum_{t=\tau}^{\infty} \gamma^t \left[g(a_t, D_t) + \beta |g(a_t, D_t) - q_{t-1}| \right] \right\} \\ \text{s.t. } & B_t - f(a_t, h_t) \geq 0, \quad \forall t \end{aligned} \tag{3.1}$$

where γ^t is an exponential discount factor. The reward function is essentially the current segment quality, with a penalty for variations (a similar model for QoE has been proposed and validated in [19]).

Note that the constraint on the buffer imposes that the current action a_t does not lead to a rebuffering event. In particular, starting from a buffer B_t and taking the action a_t when a channel h_t is experienced leads to a future buffer state B_{t+1} expressed as

$$B_{t+1} = B_t + P_t - f(a_t, h_t) \tag{3.2}$$

where we sum P_t, and we subtract the downloading time $f(a_t, h_t) = R(a_t)/h_t$, where $R(a_t)$ is the size of the encoded segment a_t (in bits) and h_t is the estimated channel rate (in b/s). The new segment is added to the buffer only when it has been downloaded completely, so it does not appear in the buffer constraint equation.

Implementing the buffer constraint with stochastic channels is a challenge: as the value of $f(a_t, h_t)$ is unknown, any action may trigger a rebuffering event. Rather than having a hard constraint on the buffer, a simpler solution is imposing a soft constraint through a penalty function $r_u[B, f]$. This leads

to the following optimization problem:

$$\mathbf{A}^{\star} : \underset{\mathbf{A}}{\operatorname{argmax}} \left\{ \sum_{t=\tau}^{\infty} \gamma^t \left[g(a_t, D_t) + \beta |g(a_t, D_t) - q_{t-1}| \right] - r_u \left[B_t, f(a_t, h_t) \right] \right) \tag{3.3}$$

The function $r_u[B_t, f(a_t, h_t)]$ gets the form

$$r_u \left[B_t, f(a_t, h_t) \right] = \rho(\max[0, f(a_t, h_t) - B_t]) + \sigma(\max[B_M - B_{t+1}, 0])^2 \tag{3.4}$$

where B_M is a "safe" buffer level above which there is no penalty, and ρ and σ are relative weights given to the two buffer management terms.

We can then call the quality component of the reward $r_k(a_t, D_t, q_{t-1})$ and define the total reward $r(a_t, s_t, s_{t+1})$:

$$r_k(a_t, D_t, q_{t-1}) = g(a_t, D_t) + \beta |g(a_t, D_t) - q_{t-1}| \tag{3.5}$$

$$r(a_t, s_t, s_{t+1}) = r_k(a_t, D_t, q_{t-1}) + r_u \left[B_t, f(a_t, h_t) \right] \tag{3.6}$$

3.1.1 Markov decision problem

We can model the system as an MDP to solve the optimization problem in Eq. (3.3). We characterize the state at time t as $s_t : \{q_{t-1}, h_{t-1}, D_t, B_t\}$.

As the possible actions a_t correspond to the available representations for the next segment, q_{t-1}, h_{t-1} and B_t are necessary to calculate the reward; the value of D_t comes into play only if the video's complexity changes over time, while it is constant for a constant complexity video.

We denote by A_{s_t} the set of possible actions that can be taken from s_t. The total reward for an action a_t taken from the state s_t leading to a state s_{t+1} is given by the sum of the quality component r_k and the buffer component r_u, which is stochastic: as the channel rate can only be estimated after the segment download, the variable h_t is unknown when the decision is

made (it is, in fact, part of the next state s_{t+1}).

$$r(a_t, \underbrace{D_t, q_{t-1}, B_t}_{s_t}, \underbrace{h_t}_{s_{t+1}}) = r_k(a_t, D_t, q_{t-1}) - r_u\left[B_t, f(a_t, h_t)\right]) \tag{3.7}$$

Given the current state and the taken action, the transition to state $s_{t+1} : \{q_t, h_t, D_{t+1}, B_{t+1}\}$ is Markovian. In particular, q_t depends on the current action and on the complexity level D_t, the buffer B_{t+1} is given by Eq. (3.2). We assume that the complexity of the video D_{t+1} and the future channel are both random components of the system, and that the two processes are mutually independent and Markovian. Finally, note that the future status of the buffer, B_{t+1}, depends on the channel h_t; this means that we can only evaluate the penalty function $r_u\left[B_t, f(a_t, h_t)\right]$ if we know the future state s_{t+1}. For this reason, we separated the reward function in two components, as done in [44]: The reward component $r_k(a_t, s_t)$ is *known* given the current state and the action, while $r_u\left[B_t, f(a_t, h_t)\right]$ is *unknown* until the end of the transition to s_{t+1}.

If we introduce the non-deterministic component into Eq. (3.1), we obtain the following optimization problem

$$\mathbf{A}^\star : \underset{\mathbf{A}}{\operatorname{argmax}} \left\{ \sum_{t=\tau}^{\infty} \gamma^t r(a_t, D_t, q_{t-1}, B_t, h_t) \right\} \tag{3.8}$$

that can be solved with the Bellman's equation introducing the state value function $V(s_t)$ as follows

$$V^\star \underbrace{(q_{t-1}, h_{t-1}, D_t, B_t)}_{s_t} = \max_{a_t \in \mathcal{A}(s_t)} \Big\{ r_k(a_t, D_t, q_{t-1}) + \sum_{h_t, D_{t+1}} p(h_t|h_{t-1}) p(Q_{t+1}|D_t) \left[-r_u\left[B_t, f(a_t, h_t)\right]) + \gamma V\left(q_t, h_t, D_{t+1}, B_{t+1}\right) \right] \Big\} \tag{3.9}$$

where $p(h_t|h_{t-1})$ is the one-step transition probability of the process that

describes the channel rate. Equivalently, Eq. (3.9) can be expressed as:

$$V^{\star}(s_t) = \max_{a_t \in \mathcal{A}(s_t)} \left\{ r_k(a_t, s_t) + \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) \left[-r_u(a_t, s_t, s_{t+1}) + \gamma V^{\star}(s_{t+1}) \right] \right\} \tag{3.10}$$

where $V^{\star}(s_t)$ is the state value function of $V(s_t)$ under the optimal policy $\pi^{\star}$.

3.1.2 Soft borders

The channel rate is the variable that influences the learner's behavior the most; as it is also continuous, the border effects between adjacent states can be very noticeable and lead to unstable behavior. For this reason, the learning agent uses a soft border (see Section 2.2.4) with a simple linear rule. On the border between states h_i and h_{i+1}, the learner uses a linear combination of the two states' Q-values, with the following rule:

$$h = \alpha h_i + (1 - \alpha) h_{i+1} \tag{3.11}$$

$$\alpha = \begin{cases} 0.5 + 2\frac{b_{i+1} - h}{b_{i+1} - b_i} & h < \frac{b_i + b_{i+1}}{4} \\ 1 & h \in \left[\frac{b_i + b_{i+1}}{4}, \frac{3b_i + b_{i+1}}{4} \right] \\ 1.5 - 2\frac{b_{i+1} - h}{b_{i+1} - b_i} & h > \frac{3b_i + b_{i+1}}{4} \\ 0.5 & h = b_{i+1} \end{cases} \tag{3.12}$$

where b_{i+1} is the border between states h_i and h_{i+1}. Eq. (3.12) can be tuned to increase or decrease the softness of the border, changing the linear combination and even including more than two states.

3.2 The Offline and Online algorithms

After modeling video bitrate adaptation as an optimization problem and defining the associated MDP, it is possible to define RL-based algorithms to solve the optimization problems. The two algorithms described in this section represent the main contribution of this thesis: they are both rooted in standard Q-learning, but they focus on two slightly different situations.

The **Offline** algorithm relies on an extensive training set; after the training, its Q-values are frozen and the algorithm works as a purely exploitative table lookup: the client chooses the action with the best Q-value every time. This method is computationally lighter than continuing to update the values and exploring the state space, but if the statistics of the channel and video complexity change in time the algorithm has no way to correct its actions and adapt to the new situation. We decided to add a more detailed characterization of h_{t-1} to the state s_t of the Offline algorithm in order to make it more adaptable: if its state definition is more precise, it will have a clearer picture of the state and the impact of changes in the model may be softened.

As the name suggests, the **Online** algorithm does not have any offline pre-training phase: it can be deployed immediately, and it learns the appropriate policy online. In order to achieve a good performance on an acceptable timescale, the algorithm was designed to speed up the updates by exploiting symmetries and redundancies in the MDP. This also makes the Online algorithm reactive to changes in the statistics of the channel and in the video dynamics; as long as the Markov assumption is valid, this algorithm can react optimally to any kind of channel after a short transition phase.

Before introducing the two algorithms and their specific characteristics, the following sections will present two features that are shared by both: the video complexity model, which is necessary to implement the function $g(a_t, D_t)$ linking video bitrate and quality, and the buffer control mechanism.

3.2.1 Video complexity model

The quality metric used by the learning agent is SSIM [45], which has been described in Section 2.1.3. The curves in the MDP state are based on a QoE model proposed in [46]. The expected SSIM value is approximated as a fourth-degree polynomial of the logarithm of the relative rate. If R_1 is the full-quality bitrate and R_i is the bitrate of adaptation i, we can set:

$$\rho_i = \log\left(\frac{R_i}{R_1}\right) \tag{3.13}$$

$$q_i \simeq 1 + d_{(1,v)}\rho_i + d_{(2,v)}\rho_i^2 + d_{(3,v)}\rho_i^3 + d_{(4,v)}\rho_i^4 \tag{3.14}$$

The vector d_v represents the complexity of a video scene with four real values.

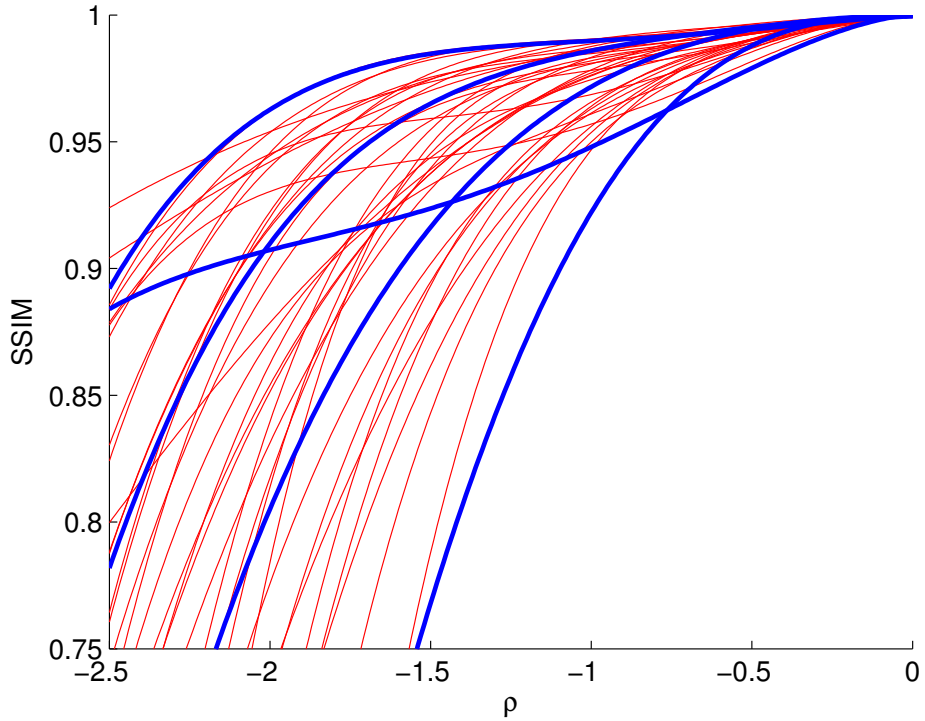


Fig. 3.1: The 5 reference SSIM curves for D_t (in blue) over the whole dataset

Using a reference set of curves can summarize the complex relation be-

tween QoE and QoS in a single parameter; if a representative set is chosen, the error in the approximation can be limited.

The reference set was elaborated from the EvalVid CIF video trace reference database (*http://www2.tkn.tu-berlin.de/research/evalvid/cif.html*) according to the model, and specifically from the *Claire*, *News*, *Bridge (far)*, *Harbor* and *Husky* videos. The 5 reference curves were chosen so as to be representative of the whole dataset. Four of the curves have the same basic shape, but different complexity, while the last one exhibits a different behavior and intersects the others.

The value of D_t is extrapolated from the next segment's available representations by calculating the quality values for each available video bitrate and choosing the curve that results in the smallest MSE.

$$D_t = \min_D \sum_a |g(a, D) - q(a)|^2 \tag{3.15}$$

3.2.2 Buffer constraints

Both algorithms use the same mechanism for implementing the soft buffer constraint: namely, the penalty function r_u, given in Eq. (3.4). However, streaming systems also avoid buffer overflow: having a limited buffer is more efficient, avoiding wastes of battery and traffic if the user does not watch the whole video, and reducing the memory requirements of the video player.

In order to make the learning algorithms aware of this principle, a waiting mechanism was implemented: whenever the buffer exceeds a threshold level $B_{\max}$ (which could be set to the safe value B_M multiplied by two), the client pauses the download until the buffer has returned to its previous value B_{t-1}. As segments are 2 seconds long, the waiting time is calculated as

$$t_w = 2 - f(a_t, h_t) \tag{3.16}$$

3.2.3 The Offline algorithm

The Offline algorithm uses standard Q-learning, which has been proven to converge to the optimal policy. Its advantage over dynamic programming is its low complexity: while dynamic programming can only work over limited state spaces due to the heavy computational cost of policy iteration, the Q(λ) algorithm can learn the expected long-term reward of each action-state pair through trial and error. The algorithm uses a Softmax behavior policy.

The main phase of the Offline algorithm's deployment happens, as the name suggests, offline: the learning agent starts its training phase with no knowledge of its environment, and all its Q-values are set to $\frac{1}{\lambda}$. As the maximum achievable long term reward is exactly $\frac{1}{\lambda}$, this initial setting encourages the algorithm to explore every action before trying to optimize its policy. The initial value of the exploration temperature is high, as is the learning rate; both can be gradually lowered as the algorithm refines its estimate of the long-term reward for each state-action pair, allowing its Q-values to slowly converge. The pseudocode for the training phase is given in Algorithm 3.1. Note that $t_{\max}$ is the number of segments, and t_u keeps track of the last update.

Algorithm 3.1 Offline algorithm: training phase

```
t_u ← 1
for t = 1 to t_max do
    s_t ← findState(q_{t−1}, B_t, h_{t−1}, D_t, Δh_{t−1})
    a_t ← softmax(Q(s_t))
    r_t ← reward(s_t, a_t)
    if Q(s_t, a_t) < max_a Q(s_t, a) then
        qUpdate(Q, t_u, s, a, r)
        t_u ← t
```

The Q(λ) algorithm only updates its Q-values when an exploratory action is taken, i.e., when the chosen action's Q-value is not the maximum for the current state. The pseudocode for the qUpdate function is given in Algorithm 3.2.

The findState function is extremely simple: it takes the relevant parameters of the state and maps them to a state.

Algorithm 3.2 Offline algorithm: Q(λ) update function

function QUPDATE(Q, t_u, s, a, r)
  $r \leftarrow \max_a Q(s_{t+1}, a)$
  for $\tau = t$ to t_u **do**
    $r \leftarrow \lambda r + r_\tau$
    $Q(s_\tau, a_\tau) \leftarrow Q(s_\tau, a_\tau)(1 - \alpha) + \alpha r$

After the Q-values have converged and the pre-training phase is over, the Offline algorithm can be deployed: its learning rate and exploration temperature are set to 0, so its policy becomes purely greedy. The Offline algorithm is very computationally light in this phase: as there are no learning updates, its operation only involves a simple table lookup. Its pseudocode is given in Algorithm 3.3.

Algorithm 3.3 Offline algorithm: deployment phase

for $t = 1$ to $t_{\max}$ **do**
  $s_t \leftarrow$ FINDSTATE($q_{t-1}, B_t, h_{t-1}, D_t, \Delta h_{t-1}$)
  $a_t \leftarrow \max_a Q(s_t, a)$

The simplicity of the Offline algorithm is also its biggest limit: as learning is no longer performed once it has been deployed, the algorithm is close to the optimal policy when the environment is the same as during the training, but major changes in the environment make the algorithm perform suboptimally.

In order to limit its weakness to changes, the Offline algorithm uses an extension of the state s_t as it is described in Section 3.1.1, introducing an additional parameter Δh_{t-1} that represents the difference between the channel states h_{t-2} and h_{t-1}. This additional parameter captures part of the channel dynamics, giving a more accurate picture of the situation and compensating for the lack of flexibility of the Offline algorithm. As the parameter is part of the state, it does not affect the algorithm directly; its state space is simply larger, with a larger Q-value table.

3.2.4 Post-decision states

Before presenting the Online algorithm, it is worth indtroducing the main mathematical concept behind it: Post-Decision States (PDSs).

A PDS is an intermediate state between two states of the MDP that already considers the consequences that follow deterministically from an action a_t: right after making the decision, the learning agent already knows the quality reward r_k, while the buffer component r_u is unknown until the transition to state s_{t+1}.

It has been shown that introducing PDSs can significantly speed up the learning process [44]. As Fig. 3.2 shows, it is possible to separate the state transition in a deterministic component and a random one: knowing s_t and a_t, the learning agent can find the PDS $\tilde{s}_t$ deterministically, as it only depends on the agent's choices. The transition from the PDS $\tilde{s}_t$ to the state s_{t+1} is random, and entirely independent of the previous state s_t.

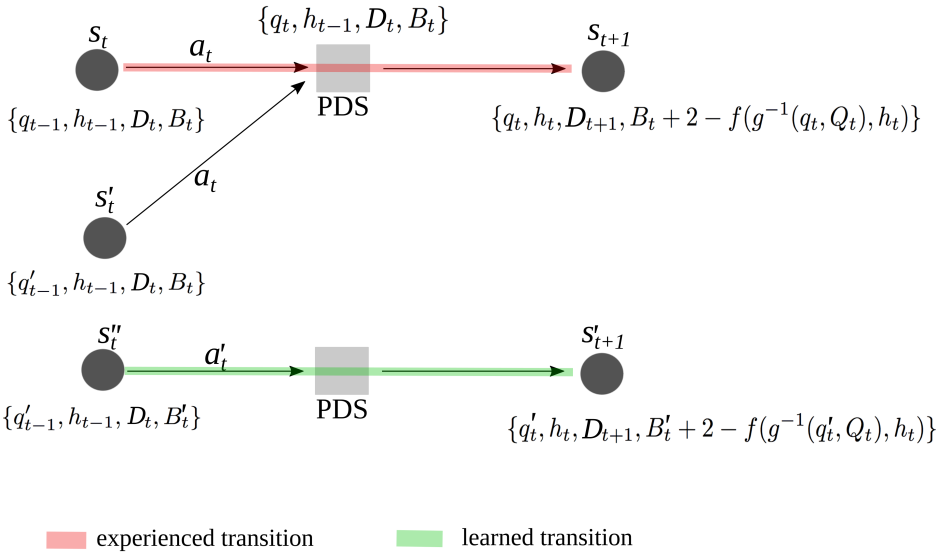


Fig. 3.2: Post decision process and parallel updates

We can then define the value function of the post decision state as

$$\tilde{V}^{\star}(\tilde{s}_t) = \sum_{s_{t+1}} p_u(s_{t+1}|\tilde{s}_t) \left[-r_u(\tilde{s}_t) + \gamma V^{\star}(s_{t+1})\right] \tag{3.17}$$

where

$$V^{\star}(s_t) = \max_{a_t \in \mathcal{A}(s_t)} \left\{ r_k(a_t, s_t) + \sum_{\tilde{s}_t} p_k(\tilde{s}_t | \tilde{s}_t, a_t) \tilde{V}^{\star}(\tilde{s}_t) \right\} \tag{3.18}$$

In Eq. (3.17), $p_u(s_{t+1}|\tilde{s}_t)$ is an unknown transition probability; the action a_t is already accounted for in $\tilde{s}_t$. At the same time, $p_k(\tilde{s}_t|\tilde{s}_t, a_t)$ in Eq. (3.18) is a deterministic probability, equal to 1 only for one allowed transition.

The reward is split in two parts: the quality component r (see Eq. 3.1) can be calculated before the random transition, while the buffer control component r_u only depends on the PDS $\tilde{s}_t$, and not on s_t. This validates the PDS formulation, as the value function of the PDSs is a valid estimate of long-term reward.

It is interesting to note that states with the same quality q_{t-1} lead to the same PDS, as that component only appears in the formula of r and does not affect the random transition.

The update equation in the learning process becomes:

$$\tilde{V}_{t+1}(\tilde{s}_t) \leftarrow (1 - \alpha^t)\tilde{V}_t(\tilde{s}_t) + \alpha^t \left[-r_u(\tilde{s}_t) + \gamma V_t\left(s_{t+1}\right) \right] \tag{3.19}$$

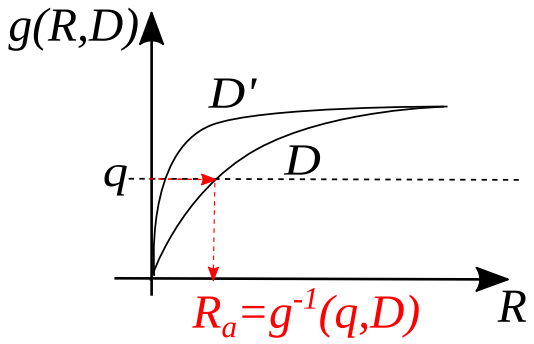


Fig. 3.3: The quality function and its inversion

The random transition probability from $\tilde{s}_t$ to s_{t+1} does not depend on the action a_t. This means that when a transition from s_t to s_{t+1} is experienced, we can update all post decision states with the common transition

$(D_t, h_{t-1}) \rightarrow (D_{t+1}, h_t)$. Fig. 3.2 shows how the system can extrapolate several possible transitions after experiencing only one, learning from the virtual transitions (highlighted in green in Fig. 3.2).

This means that the following updating process is computed for $\forall q_t, \forall B_t$,

$$\begin{aligned}\tilde{V}^{t+1}(q_t, \hat{h}_{t-1}, \hat{Q}_t, B_t) \leftarrow (1-\alpha^t)\tilde{V}^t(q_t, \hat{h}_{t-1}, \hat{Q}_t, B_t) + \alpha^t \\ \Big[- r_u(q_t, \hat{h}_{t-1}, \hat{Q}_t, B_t) + \gamma V^t(q_t, \hat{h}_t, \hat{Q}_{t+1}, B_{t+1}))\Big]\end{aligned} \tag{3.20}$$

3.2.5 The Online algorithm

The Online algorithm learns the Q-values online: instead of having a pre-training phase, it starts trying to optimize the video bitrate with no knowledge of the environment and gradually refines its estimates. In order to do this, it must converge faster than standard Q-learning: the algorithm uses the PDS model to exploit symmetries and redundancies in the MDP to apply its experience to as many states as possible. The pseudocode for the Online algorithm is given in Algorithm 3.4.

Algorithm 3.4 Online algorithm

for $t = 1$ to $t_{\max}$ **do**
  $s_t \leftarrow$ findState$(q_{t-1}, B_t, h_{t-1}, D_t)$
  for $a = 1$ to $a_{\max}$ **do**
    $r_k(a) \leftarrow$ qualityReward(s_t, a)
    $\tilde{s}_t \leftarrow$ findPDS(s_t, a)
    $Q(s_t, a) \leftarrow r_k(a) + \tilde{Q}(\tilde{s}_t, a)$
  $a_t \leftarrow$ softmax$(Q(s_t) + r_k)$
  $r_{k,t} \leftarrow r_k(a_t)$
  $r_{u,t} \leftarrow$ bufferReward$(\tilde{s}_t, a_t)$
  qUpdate$(\tilde{Q}, s, \tilde{s}, a, r)$

The decision-making part of the algorithm is similar to the Offline training phase (Algorithm 3.1), while the rest of the algorithm is centered on the use of PDSs. As the algorithm does not store the Q-values for the states s_t but only for the PDSs $\tilde{s}_t$, the algorithm reconstructs the Q-values for every potential action a by summing its quality reward r and its PDS's Q-value $\tilde{Q}(\tilde{s}_t, a)$. Both r and $\tilde{s}_t$ can be found deterministically from s_t and a.

The Online algorithm's update function updates the Q-values of the Post-Decision State, as well as generalizing the experience to other PDSs with the same h_t and D_t. For easier comprehension, the pseudocode was split: the singleUpdate function updates a single PDS and is given in Algorithm 3.5, while the qUpdate function performs the parallel update and is given in Algorithm 3.6.

Algorithm 3.5 Online algorithm: $\tilde{Q}$ update function

function singleUpdate($\tilde{Q}, s_t, \tilde{s}_{t-1}, a, r_k$)
  for $a = 1$ to $a_{\max}$ **do**
    $r_k(a) \leftarrow$ qualityReward(s_t, a)
    $\tilde{s}_t \leftarrow$ findPDS(s_t, a)
    $Q(s_t, a) \leftarrow r_k(a) + \tilde{Q}(\tilde{s}_t, a)$
  $r \leftarrow r_t + \max_a Q(s_t, a)$
  $\tilde{Q}(\tilde{s}_{t-1}, a_{t-1}) \leftarrow \tilde{Q}(\tilde{s}_{t-1}, a_{t-1})(1-\alpha) + \alpha r$

Algorithm 3.6 Online algorithm:parallel update function

function qUpdate($Q, t_u, s, \tilde{s}, a, r$)
  singleUpdate$(Q, t_u, s_t, \tilde{s}_{t-1}, a, r)$
  for all q'_{t-1}, B'_{t-1} **do**
    $s'_{t-1} \leftarrow$ findState$(q'_{t-1}, B'_{t-1}, h_{t-2}, D_{t-1})$
    $r'_k \leftarrow$ qualityReward(s'_{t-1}, a_{t-1})
    $\tilde{s}'_{t-1} \leftarrow$ findPDS(s'_{t-1}, a_{t-1})
    $s'_t \leftarrow$ findState$(q_t, B'_t, h_{t-1}, D_t)$
    singleUpdate$(\tilde{Q}, s'_t, \tilde{s}'_{t-1}, a, r'_k)$
  $r \leftarrow r_t + \max_a Q(s_t, a)$
  $\tilde{Q}(\tilde{s}_{t-1}, a_{t-1}) \leftarrow \tilde{Q}(\tilde{s}_{t-1}, a_{t-1})(1-\alpha) + \alpha r$

The parallel update relies on the fact that the buffer B_t can be determined for every B_{t-1} if the update function knows the download time of the last segment.

It should be noted that the Online algorithm is TD(0), as it does not use eligibility traces; the extreme parallelization of the updates makes frequent, though imprecise, updates more efficient than slower but more precise updates. Along with a non-zero learning rate, this makes the Online algorithm's

choices less stable, sacrificing a little reward in the trade-off with adaptability. The main strength of the Online algorithm is its adaptability: while the Offline algorithm is limited by its training set, the Online algorithm reacts to changes in the environment and it can learn entirely new models after deployment.

Chapter 4

Simulation and results

Both the Offline and Online algorithms were tested in several scenarios via Matlab simulation.

The performance baseline is given by the Benchmark algorithm, a simple heuristic that is defined in Algorithm 4.1 (in which a is the chosen action, represented by a number from 1 to $a_{\max}$ ordered by decreasing bitrate).

In relatively stable channel rate condition, this is a simple but efficient strategy to get a high average quality while avoiding rebuffering events.

Algorithm 4.1 Benchmark algorithm

```
if segment = 1 then
    a ← a_max
else
    a ← 1
    while a ≥ h_t & a < a_max do
        a ← a + 1
```

The algorithm uses a different procedure for the first segment: as channel rate is unknown, the benchmark tries to build up the buffer and speed up the loading time of the video by choosing the lowest possible rate.

4.1 Simulation settings

The basic parameters of the optimization problem are set as in Table 4.1 and used for all the simulations.

Parameter	α	β	ρ	σ	B_M	$B_{\max}$	γ
Value	1	2	50	0.001	12 s	20 s	0.9

Table 4.1: Parameters of the optimization problem

In order to implement the MDP, we first needed to define the range of values of the 4-tuple that defines the state $s_t : \{q_{t-1}, h_{t-1}, D_t, B_t\}$. Except for D_t, the variables were all continuous, so state borders were set to discretize the state space. A special first state is defined for the first segment, as there are no previous segments and no channel rate information is available. It may also be interesting to note that the buffer at decision time can never go below 2 s (except for the first segment), as all decisions the learning agent makes happen right after the previous segment has been downloaded. The state border values are listed in Table 4.2.

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
Δh_{t-1}	-1	-0.5	0	0.5					

Table 4.2: State borders

The available rates in Mb/s are fixed in the set {0.3, 0.5, 1, 2, 3, 4, 6, 8, 10}. The video clips (episodes) last 800 s, divided into 400 segments of 2 seconds each.

The main metric we used to measure the performance of the algorithms was the mean quality reward: we averaged r_k over 10 episodes and compared

the two algorithms' results with the benchmark's. The values of r_k are on a scale from 0 to 1, but even small variations can have a perceptible effect on QoE.

4.2 Simulation scenarios

The simulation scenario is extremely simple: a client streams a DASH video by downloading the segments, while playing the available video. The channel is assumed to be single-hop and its channel rate is constant during the download of a segment. The videos have a complexity curve chosen from the 5 reference curves defined in Section 3.2.1, and the complexity is constant for a single segment.

We first tested the validity of the algorithms in the simple case of videos with constant complexity and a constant channel rate. The **Static** scenario used the "Harbor" video curve and three different channel rates: first with $h = 3\ Mb/s$ (equal to one of the available video bitrates), then $h = 3.5\ Mb/s$ (halfway between two of the available video bitrates) and finally $h = 3.9\ Mb/s$ (close but slightly lower than one of the available bitrates).m

The next scenario is called **Variable scenes**: while the channel rate was kept constant ($h = 3.5\ Mb/s$), the video complexity varied, adding a layer of complexity. The video had scenes with a constant complexity and an exponential duration (with an average of 1, 5, or 10 segments). At the end of a scene, the complexity of the video was chosen randomly among the other four reference curves.

The **Dynamic** scenario used a video with a constant complexity with a Markovian channel: after each segment, the channel's underlying Markov chain could change its channel rate. By changing the transition matrix, the various scenarios covered a variety of cases, with both fast-varying and low-varying channels. The transition matrices we used had either jumps to adjacent states or allowed two-state jumps. We also tested a completely random channel, in which the channel rate h_t was independent of the previous value h_{t-1}, corresponding to a uniform Markov transition matrix.

Finally, the **Complete** scenario had a Markov channel and scene changes:

this represented the most difficult test for the learning algorithms, as all the parameters in the environment were dynamic and the learning agent had to adapt to their changes without having a pre-existing model.

4.3 Offline algorithm: results

As the Offline algorithm needs extensive training before being deployed, its convergence times are not significant in its evaluation; its main limitation is its training regime, but it should be more effective than the Online algorithm in static situations. In order to confirm this, the algorithm was tested in the scenarios described in the previous section.

The results in the following sections were obtained by extensively training the learner on the scenarios (200000 episodes with decreasing exploration temperature), then setting the learning rate α and the exploration temperature τ to zero.

4.3.1 Static scenario

The tests in the **Static** scenario confirm the validity of the algorithm in simple situations: when $h = 3\ Mb/s$, the algorithm builds up the buffer in the first few segments, then chooses the same actions as the benchmark. Fig. 4.1 shows the quality reward for one episode, while the buffer dynamics are plotted in Fig. 4.2.

A more interesting case is the convergence of the learner when the channel rate is between two available rates: the two cases considered here are the channel rates $h = 3.5\ Mb/s$ and , $h = 3.9\ Mb/s$. In the first case, shown in Fig. 4.3, the learner found that the penalty for changing the quality is not worth switching back and forth and settled on the lower rate. The second case is different: the learner found that switching to the higher video bitrate and using the buffer to keep a higher quality for a while is beneficial (see Fig. 4.4). The learner behaves in a cyclical manner, switching the quality when it reaches a certain buffer level, as Fig. 4.5 shows.

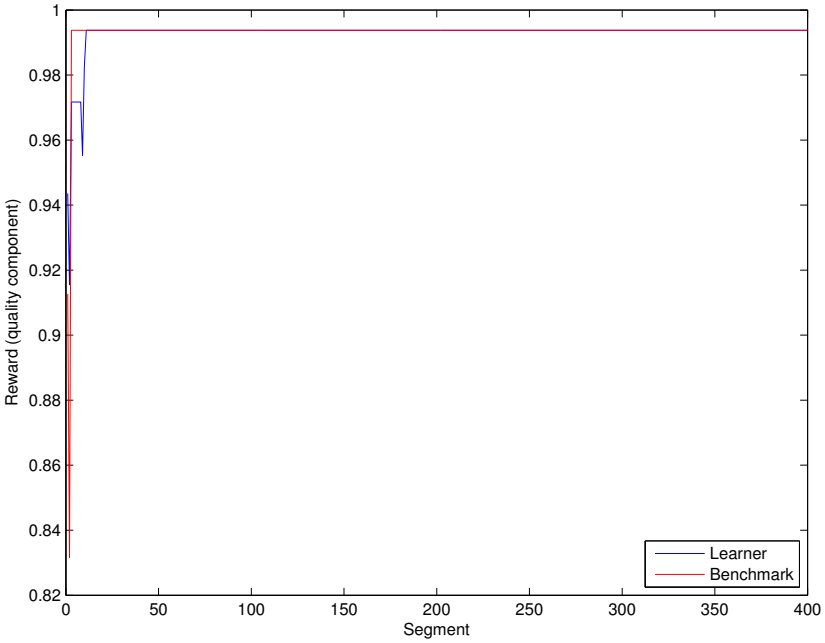


Fig. 4.1: Reward (quality component) for one episode, $h = 3\ Mb/s$

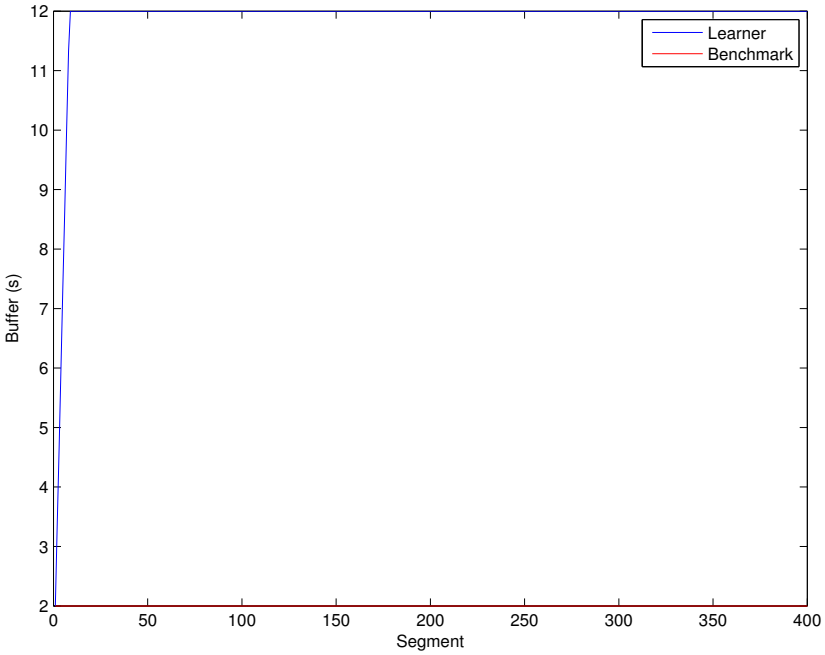


Fig. 4.2: Buffer plot for one episode, $h = 3\ Mb/s$

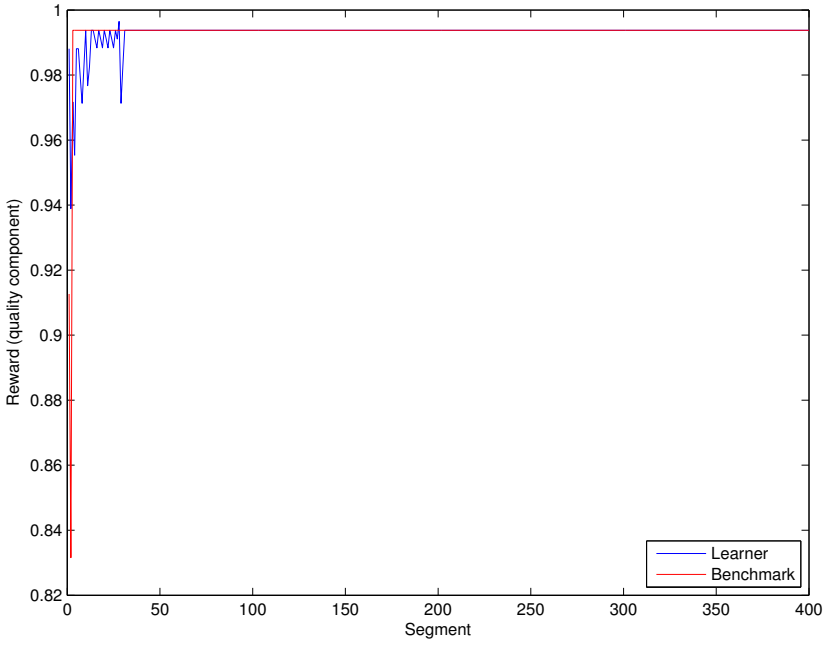


Fig. 4.3: Reward (quality component) for one episode, $h = 3.5\ Mb/s$

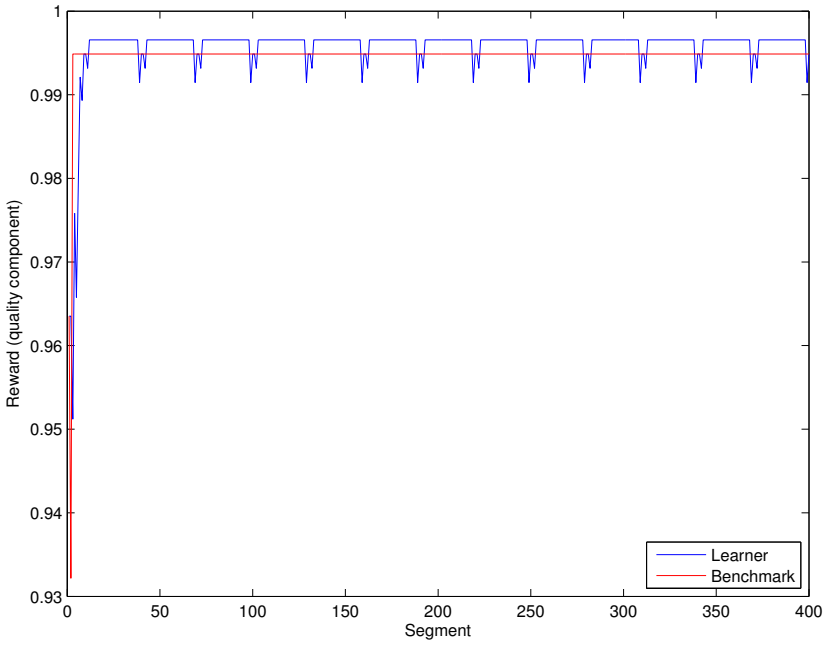


Fig. 4.4: Reward (quality component) for one episode, $h = 3.9\ Mb/s$

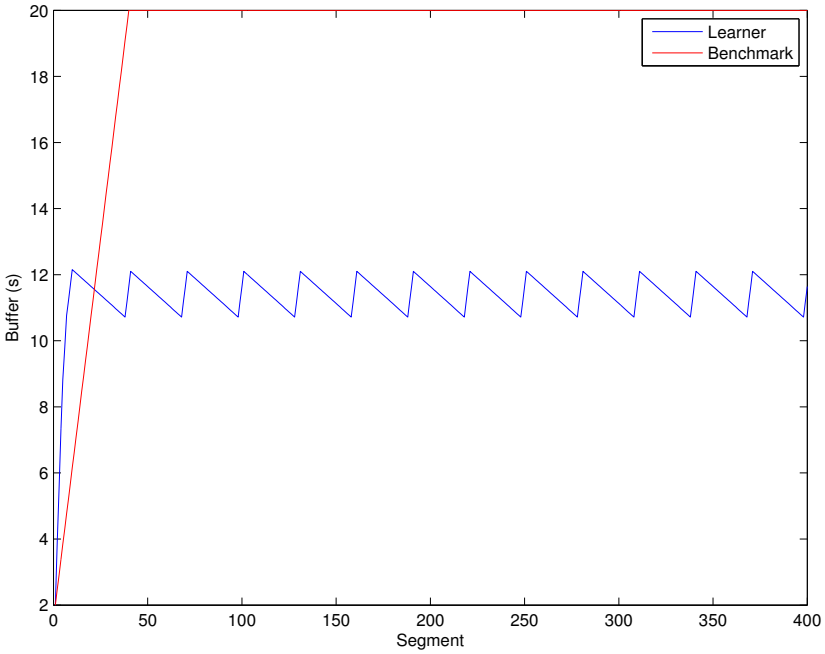


Fig. 4.5: Buffer plot for one episode, $h = 3.9\ Mb/s$

4.3.2 Variable scenes scenario

The next results have been obtained in the **Variable scenes** scenario, with a constant channel rate of $h = 3.5\ Mb/s$ and a variable video. The video for each episode was randomly generated from the reference dataser using the model described in Section 3.2.1; each scene had an exponentially distributed duration.

The mean of the scene length distribution was set to 10, 5 and 1 in the three considered cases, corresponding to 20, 10 and 2 seconds; we expected the learner to compensate for more complex scenes by increasing the video bitrate, using the accumulated buffer to get through the difficulty.

Nothing of the sort happened when the mean scene duration was 10 segments: as Fig. 4.6 (plotting r_k over the downloaded segments) shows, the Offline algorithm shows no improvement over the benchmark, and its mean reward is exactly the same as the benchmark's. This effect is probably due to the length of the scenes, as the accumulated buffer is not enough and the

learner finds itself in the middle of a complex scene with a depleted buffer if it tries to compensate.

The learner shows a slight improvement over the benchmark with a mean scene duration of 5 segments: as Fig. 4.7 shows, the learner avoids the worst quality drops, but its gain over the benchmark is still very small: the quality reward r_k is higher by 0.002, which is almost negligible even if we consider that it is a mean value over several episodes and that in most cases the benchmark still makes the optimal choice.

The difference between the Offline algorithm and the benchmark becomes noticeable with a mean scene duration of only 1 segment: in this dynamic situation, the learner's use of the buffer makes the quality level much smoother, as Fig. 4.8 and Fig. 4.9 show. The average quality reward is higher than the benchmark's by 0.006, enough to make a significant difference in the user QoE. Fig. 4.10 shows the Offline algorithm's buffer management, with cyclic phases of buffer build-up and complexity compensation.

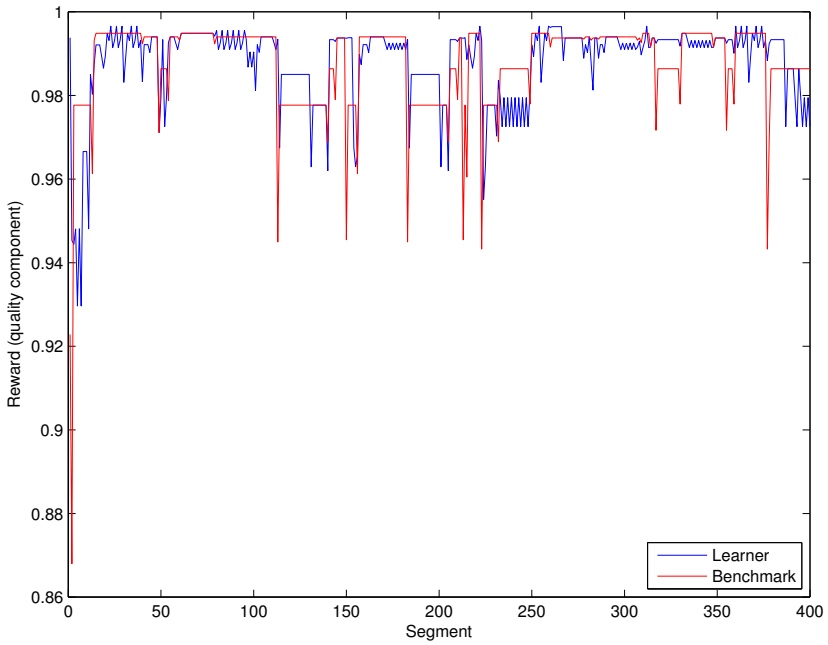


Fig. 4.6: Reward (quality component) with a dynamic video (mean scene duration 10 segments)

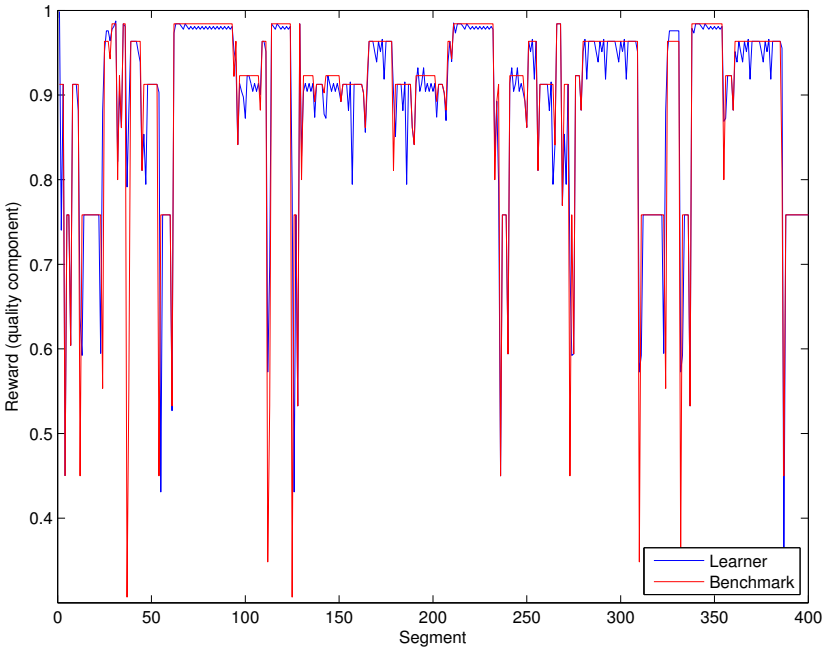


Fig. 4.7: Reward (quality component) with a dynamic video (mean scene duration 5 segments)

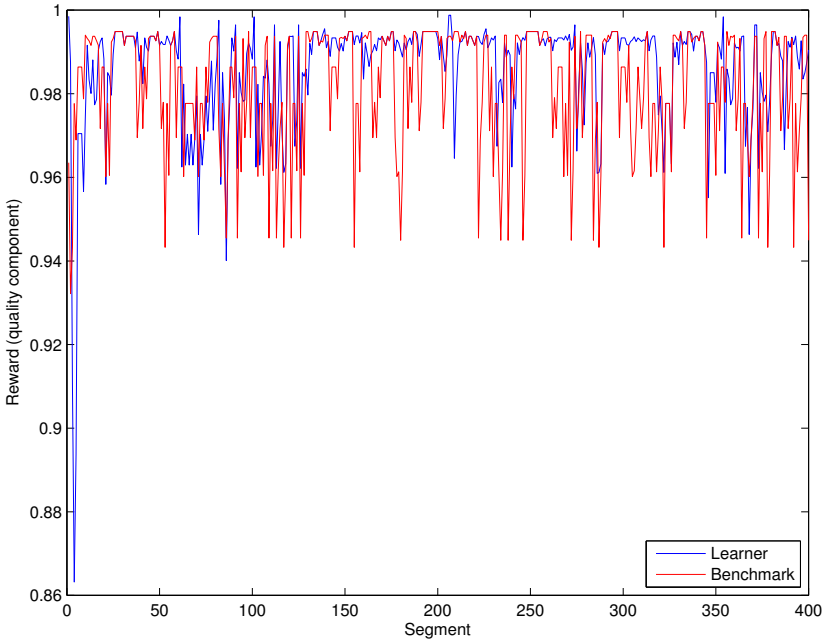


Fig. 4.8: Reward (quality component) with a dynamic video (mean scene duration 1 segment)

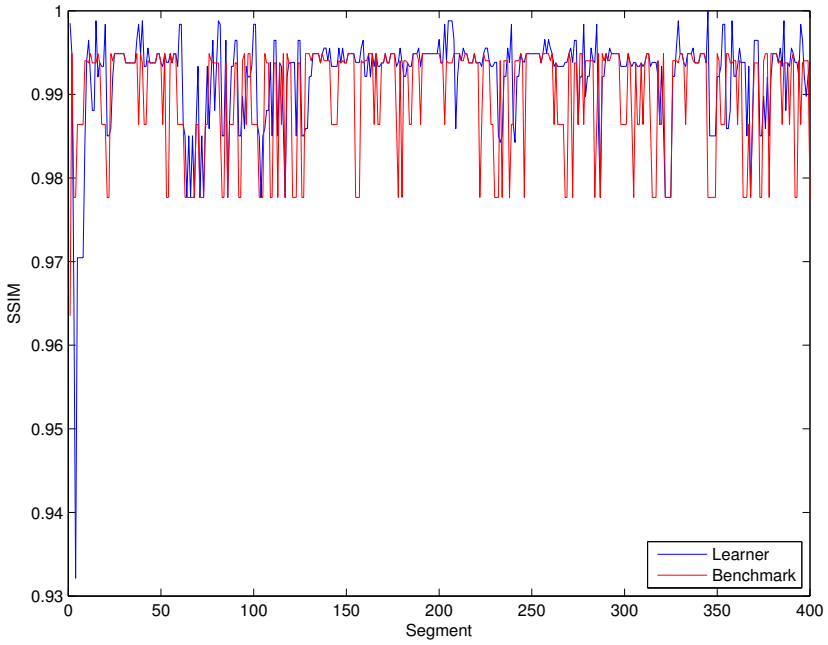


Fig. 4.9: SSIM plot with a dynamic video (mean scene duration 1 segment)

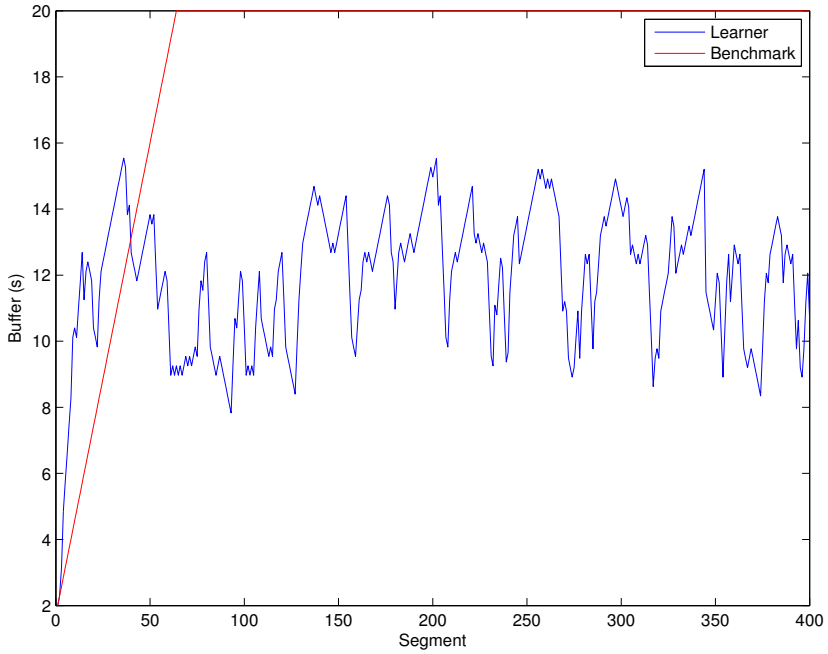


Fig. 4.10: Buffer plot with a dynamic video (mean scene duration 1 segment)

4.3.3 Dynamic scenario

The **Dynamic** scenario had a static video with constant complexity and a Markovian channel rate. The first transition matrix only considers a positive transition probability p between adjacent states, so that the channel is a random walk. The second Markov channel had transition probability $\frac{2p}{3}$ to each of the adjacent states, and $\frac{p}{3}$ to the second-next states. Transitions from the border states to themselves have a higher probability than for other states, as they only have one adjacent state instead of two.

With $p = 0.25$, the Offline algorithm only has a small gain over the benchmark; the average reward r_k was higher than the benchmark's by 0.002 with both transition matrices (see Fig. 4.11). The Offline algorithm tends to keep a more stable SSIM than the benchmark, as Fig. 4.12 shows. However, it sometimes needs to drop the quality to avoid rebuffering events, and this has a negative impact on the quality, as Fig. 4.13 shows.

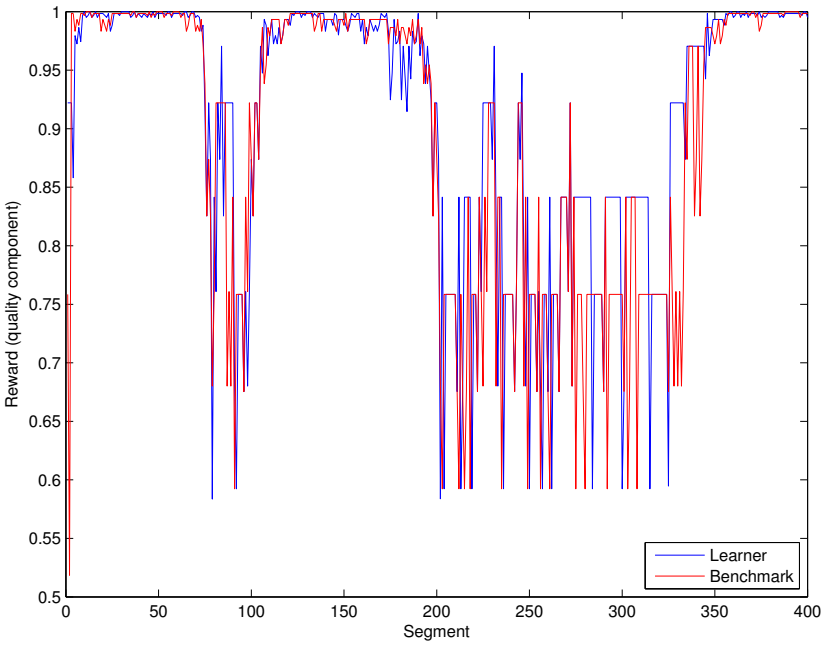


Fig. 4.11: Reward (quality component) with $p = 0.25$ (transitions to adjacent states)

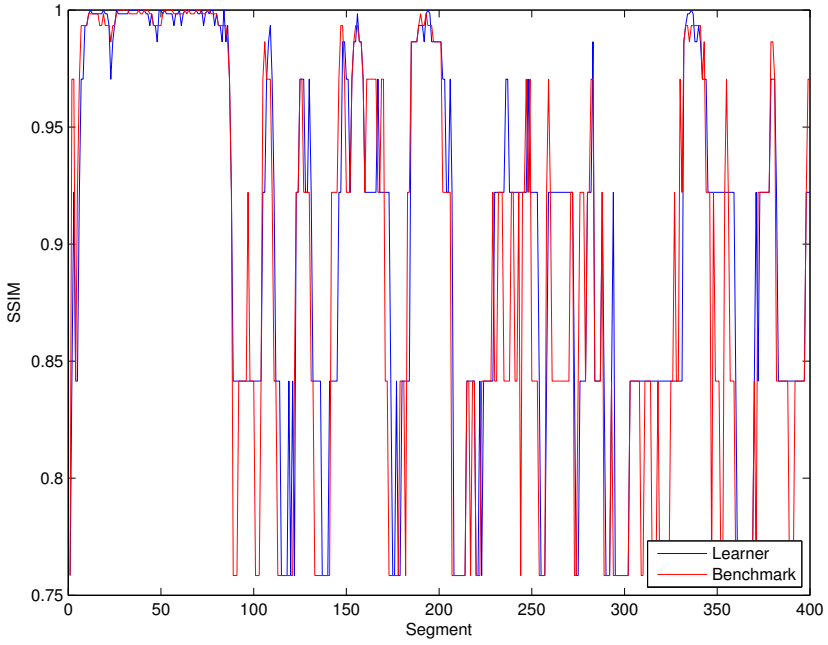


Fig. 4.12: SSIM plot with $p = 0.25$ (with two-state jumps)

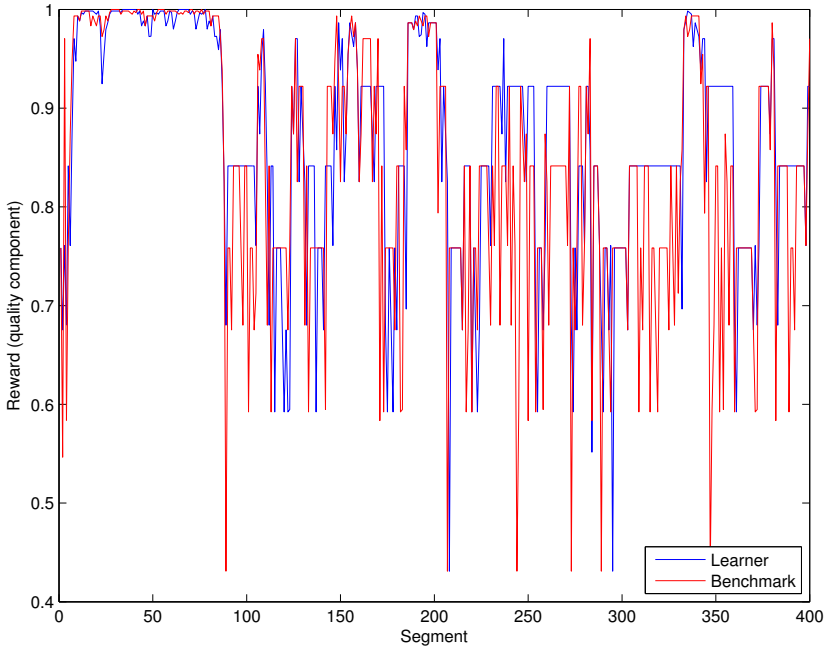


Fig. 4.13: Reward (quality component) with $p = 0.25$ (with two-state jumps)

With $p = 0.5$ and a fast-varying channel rate, the Offline algorithm's efficient use of the buffer proves very effective: the average reward r_k with the first transition matrix is higher than the benchmark's by 0.005, as Fig. 4.14 shows. The SSIM plot in Fig. 4.15 shows that the Offline algorithm manages to avoid most of the quality drops due to changes in the channel rate.

The Offline algorithm performs even better with the second transition matrix, as Fig. 4.16 shows; its efficient buffer management, shown in Fig. 4.17, prevents quality drops and keeps a more constant SSIM. The SSIM plot in Fig. 4.18 shows that the worst quality drops are almost always avoided by the Offline algorithm, which also avoids switching back and forth between video bitrates too frequently. The algorithm's mean r_k is higher than the benchmark's by 0.013.

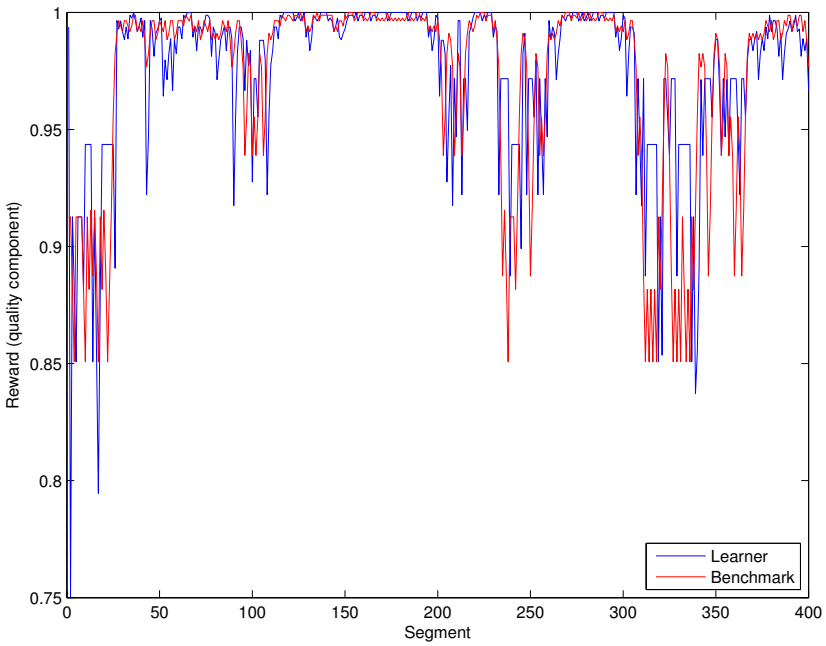


Fig. 4.14: Reward (quality component) with $p = 0.5$ (transitions to adjacent states)

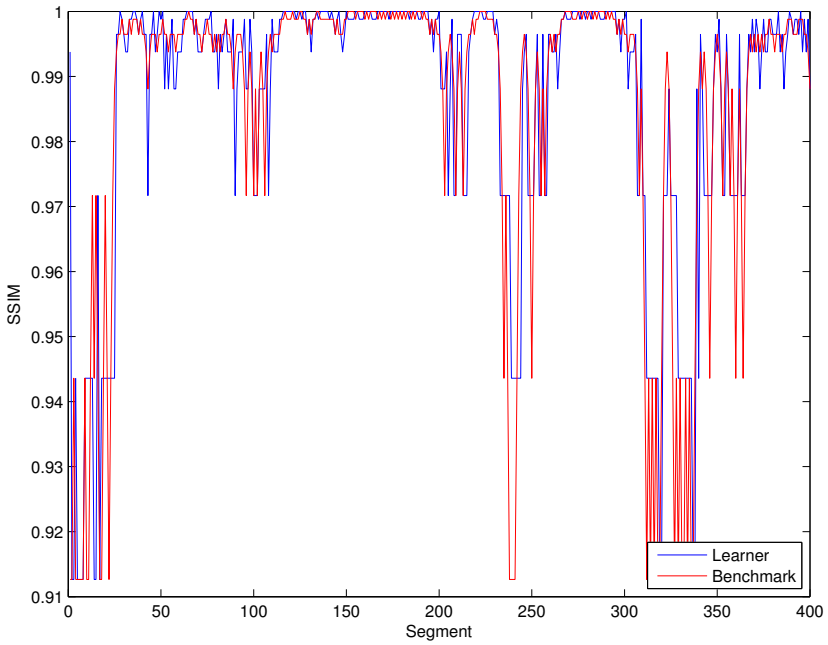


Fig. 4.15: SSIM plot with $p = 0.5$ (transitions to adjacent states)

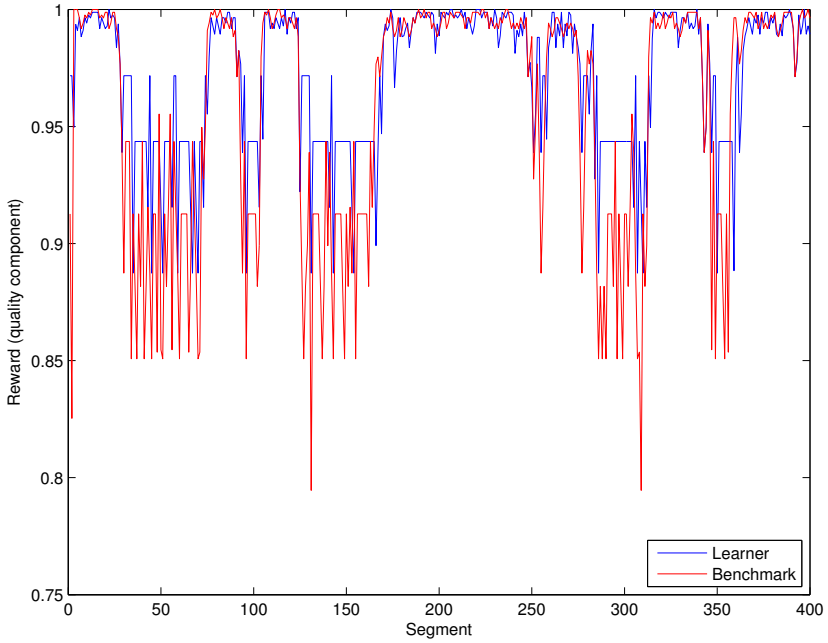


Fig. 4.16: Reward (quality component) with $p = 0.5$ (with two-state jumps)

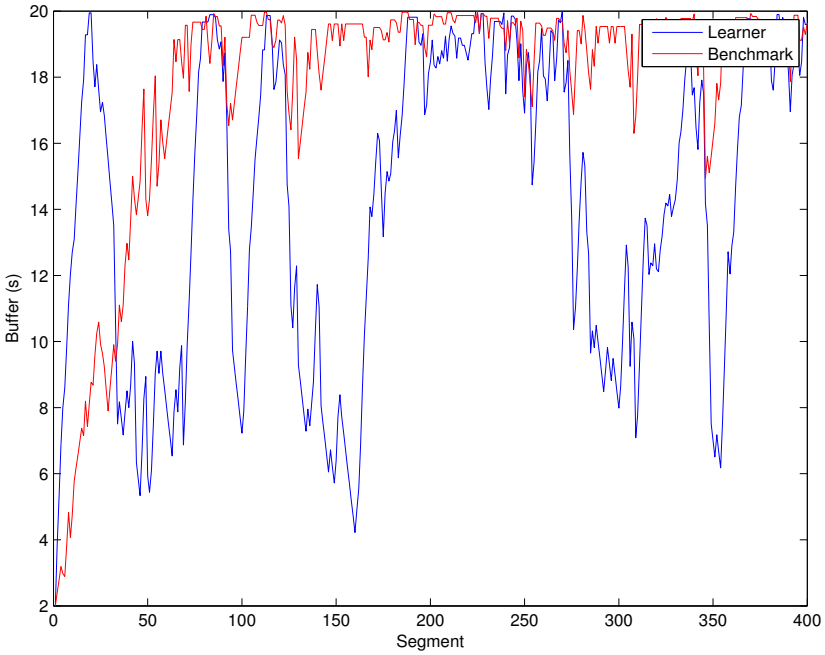


Fig. 4.17: Buffer plot with $p = 0.5$ (with two-state jumps)

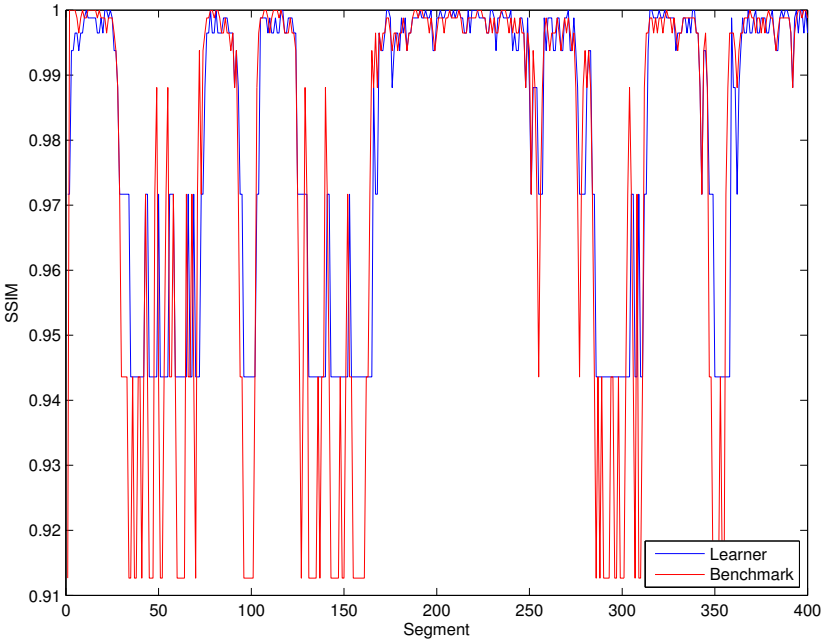


Fig. 4.18: SSIM plot with $p = 0.5$ (with two-state jumps)

The random uncorrelated channel shows the power of the Offline algorithm in challenging situations: while the benchmark is hopelessly swamped and often triggers rebuffering events due to the absence of a buffer management strategy, the Offline algorithm ignores the channel variations and keeps the quality more or less constant, choosing video bitrates between 1 Mb/s and 2 Mb/s, as Fig. 4.19 shows. Its mean reward r_k is higher than the benchmark's by an impressive 0.047, which would typically mark the difference between a satisfactory and an annoying user experience.

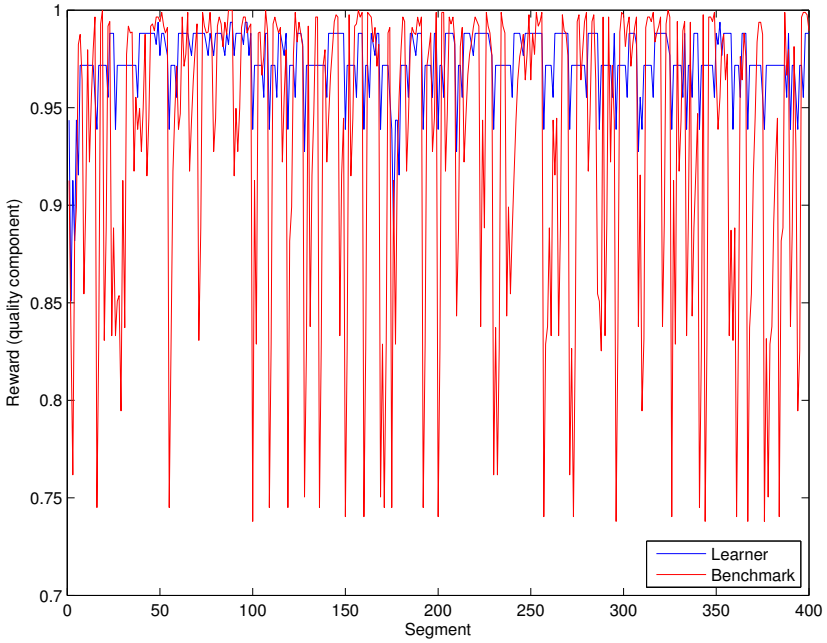


Fig. 4.19: Reward (quality component) with a uniform transition matrix

4.3.4 Complete scenario

The **Complete** scenario, the video had variable scenes, with a mean duration of 5 segments, and the channel was Markovian, using the second transition matrix. This is the most realistic case, and the Offline algorithm clearly outperforms the benchmark, as its reward r_k is higher by 0.008 on average. Fig. 4.20 shows this clearly.

This is even clearer with $p = 0.5$: the Offline algorithm's mean r_k is

higher than the benchmark by 0.013, and Fig. 4.21 shows that it has fewer and shallower quality drops and keeps a higher overall quality.

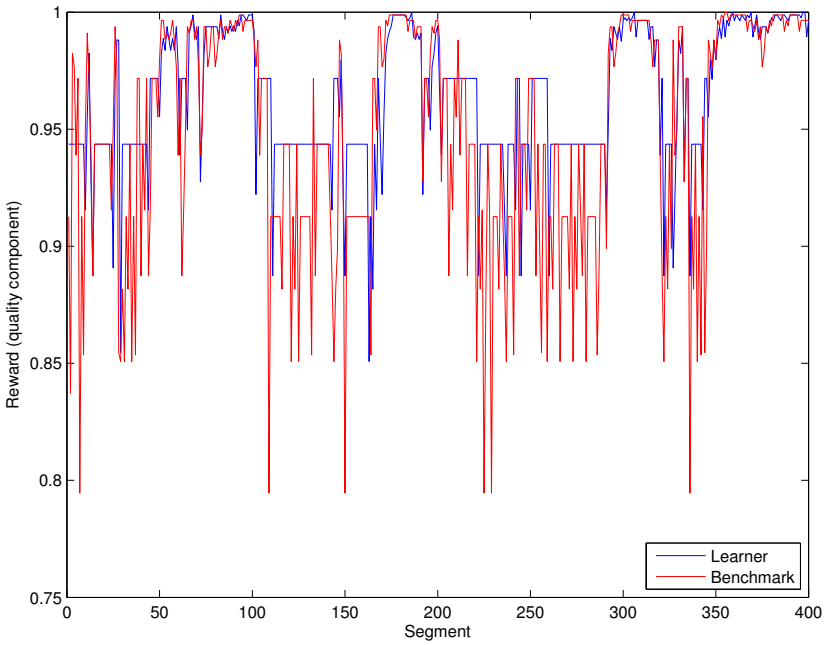


Fig. 4.20: Reward (quality component) with $p = 0.25$ (complete scenario)

Finally, an adaptation test was performed in the **Complete** scenario: the mean scene duration was set to 1 segment, and the learner was trained with a mean scene duration of 5 segments. As expected, the Offline algorithm does not react well to changes in the environment, as it has no way to update its Q-values to the new scenario, and even if it had they would still need a considerable time to converge.

As Fig. 4.22 shows, the learner does not adapt to the new situation and makes some sub-optimal choices, lowering the average reward to just 0.006 above the benchmark's.

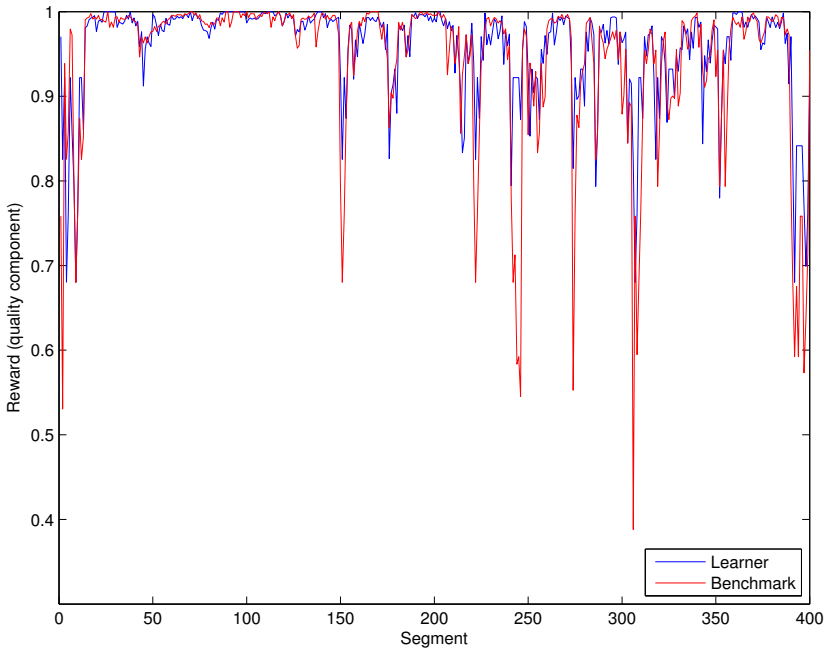


Fig. 4.21: Reward (quality component) with $p = 0.5$ (complete scenario)

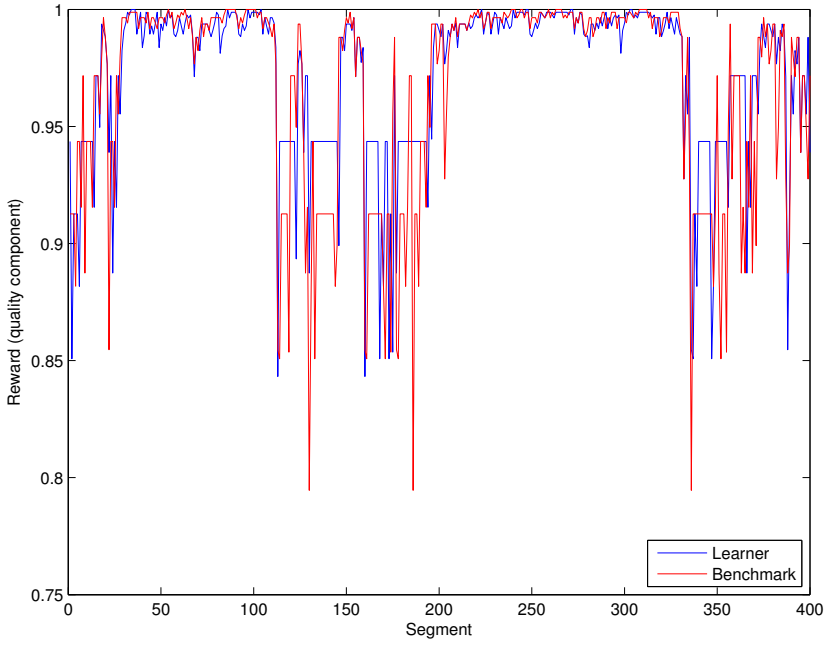


Fig. 4.22: Reward (quality component) for the first episode after the switch

4.4 Online algorithm: results

The first and foremost quality of the Online algorithm is its quick convergence: while the Offline learner needs an extended training sequence, the Online learner should be able to reach convergence relatively quickly, even from a "cold start" with no information on the Q-values. The initial Q-values were set to 10 to encourage exploration, as 10 is the highest achievable long-term reward, and the Online algorithm was run for 10 consecutive episodes of 400 segments. The learning rate α was 0.2 in all the convergence simulations, except where otherwise stated.

4.4.1 Static scenario

The algorithm's first run was in the **Static** scenario. Fig. 4.23 shows the Online algorithm converging extremely quickly to the optimal policy: after the third episode, the algorithm's actions stabilize, except for an initial buffer build-up for each new episode (visible in the downward spikes in the plot every 400 segments). The algorithm maintains a a buffer level that allows it to avoid a high low-buffer punishment and the risk of rebuffering (see Fig. 4.24).

Setting a low exploration temperature from the start instead of decreasing it gradually increases the learner's convergence speed; its actions stabilize after only one episode, as shown in Fig. 4.25.

With a channel rate $h = 3.5\ Mb/s$, the learner takes slightly longer to converge, but ultimately finds that the penalty for changing the quality was not worth switching back and forth and settles on the lower rate (see Fig. 4.26 and Fig. 4.27). Its behavior after reaching convergence with $h = 3.9\ Mb/s$ is interesting: it parallels the Offline algorithm's, switching back and forth between the higher and lower video bitrate, but its cycles are far less stable due to the continuous updates of the Q-values, as shown in Fig. 4.28. The buffer plot parallels this instability, as Fig. 4.29 shows.

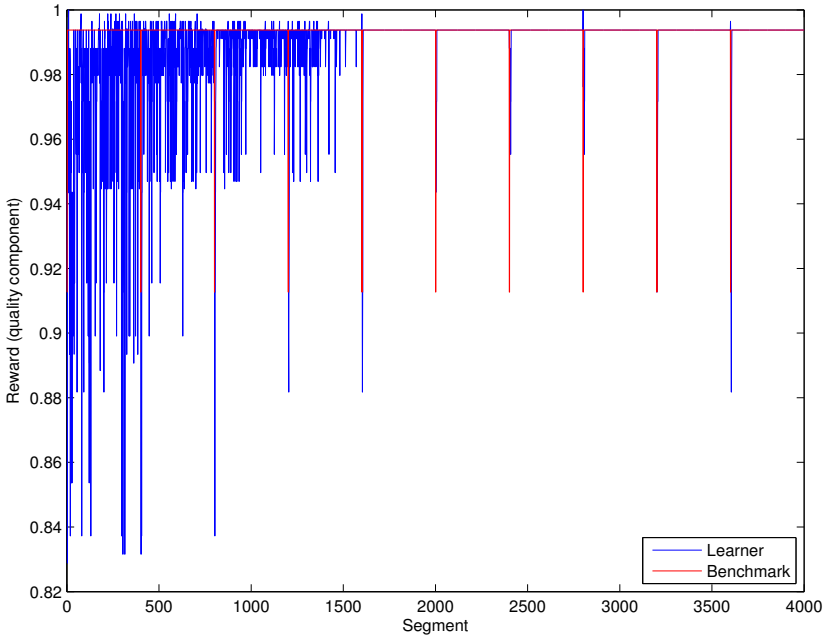


Fig. 4.23: Reward (quality component) for 10 consecutive episodes, $h = 3\ Mb/s$

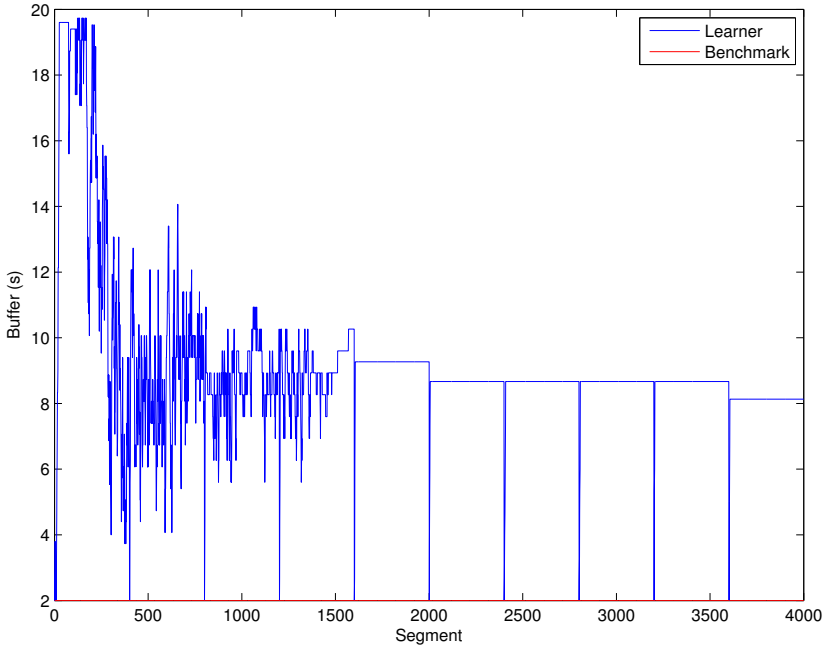


Fig. 4.24: Buffer level plot for 10 consecutive episodes, $h = 3\ Mb/s$

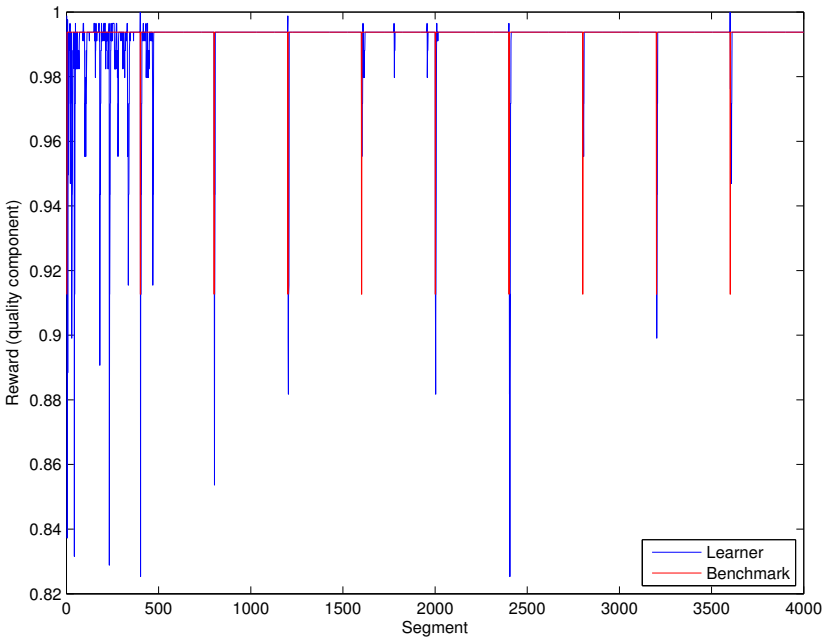


Fig. 4.25: Reward (quality component) for 10 consecutive episodes (low exploration temperature), $h = 3\ Mb/s$

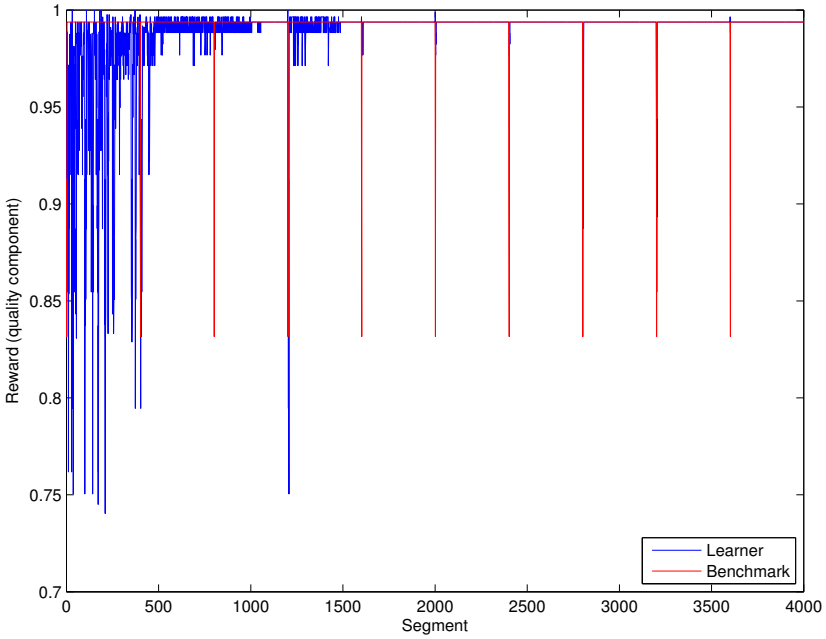


Fig. 4.26: Reward (quality component) for 10 consecutive episodes, $h = 3.5\ Mb/s$

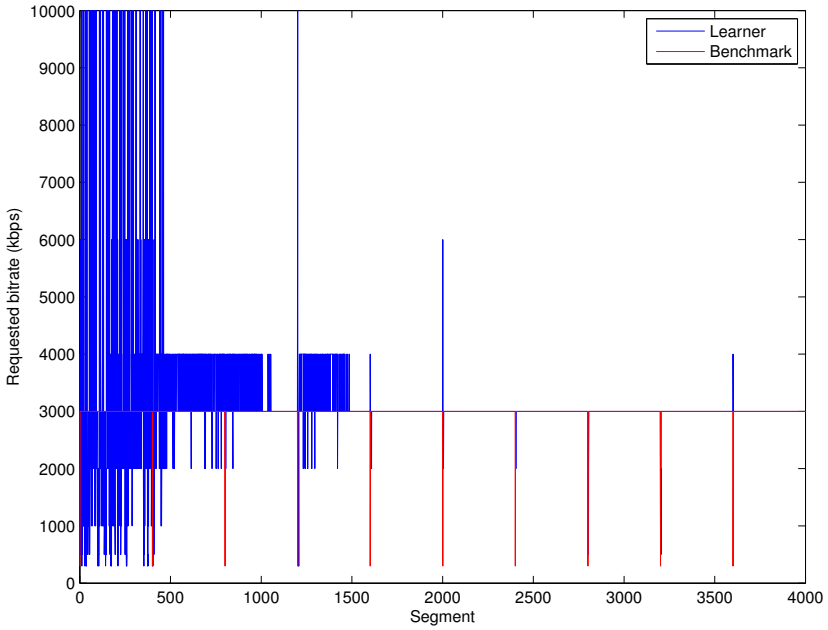


Fig. 4.27: Requested rate for 10 consecutive episodes, $h = 3.5\ Mb/s$

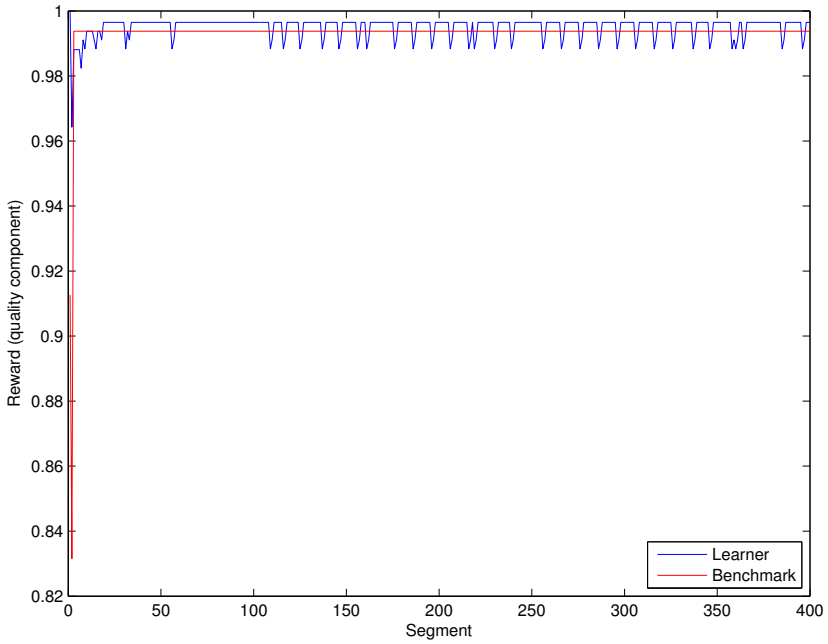


Fig. 4.28: Reward (quality component) for one episode, $h = 3.9\ Mb/s$

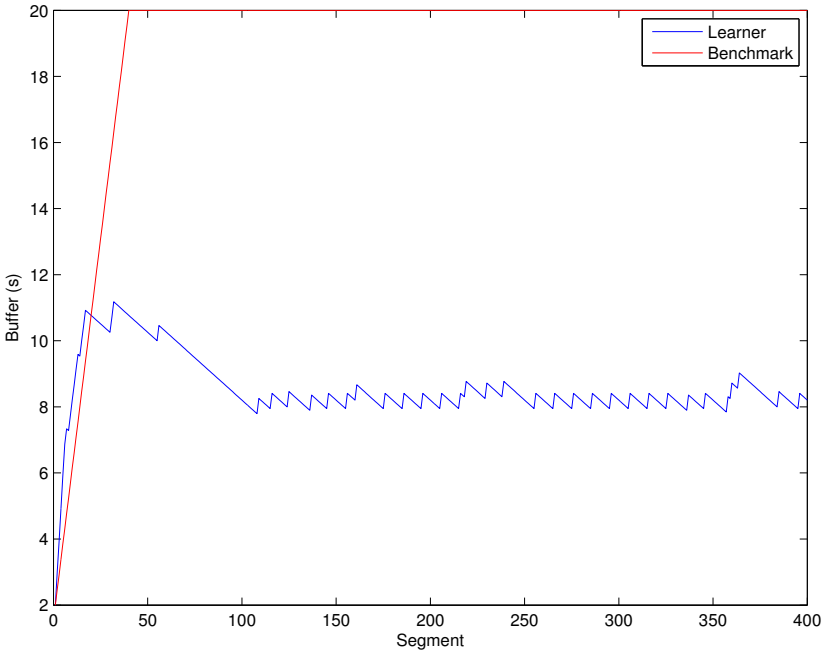


Fig. 4.29: Buffer plot for one episode, $h = 3.9\ Mb/s$

4.4.2 Variable scenes scenario

In the **Variable scenes** scenario, the Online algorithm was slightly outperformed by Offline: there were no differences with a mean scene duration of 10 segments, as the Online algorithm also had the same performance as the benchmark, and Fig. 4.30 shows that its improvement over the benchmark with a mean scene duration of 5 segments is close to the Offline algorithm's. The mean r_k was higher than the benchmark's by only 0.002, just like the Offline algorithm.

The difference between the Offline algorithm and the benchmark is noticeable with a mean scene duration of only 1 segment: in this dynamic situation, the Offline algorithm's stability makes it more efficient than the Online algorithm. Fig. 4.31 shows how the learner uses its buffer to mitigate SSIM variations when the scene is extremely dynamic, but the mean r_k is higher than the benchmark by 0.005, 0.001 less than the Offline algorithm.

Fig. 4.32 shows the algorithm's buffer management in the last case.

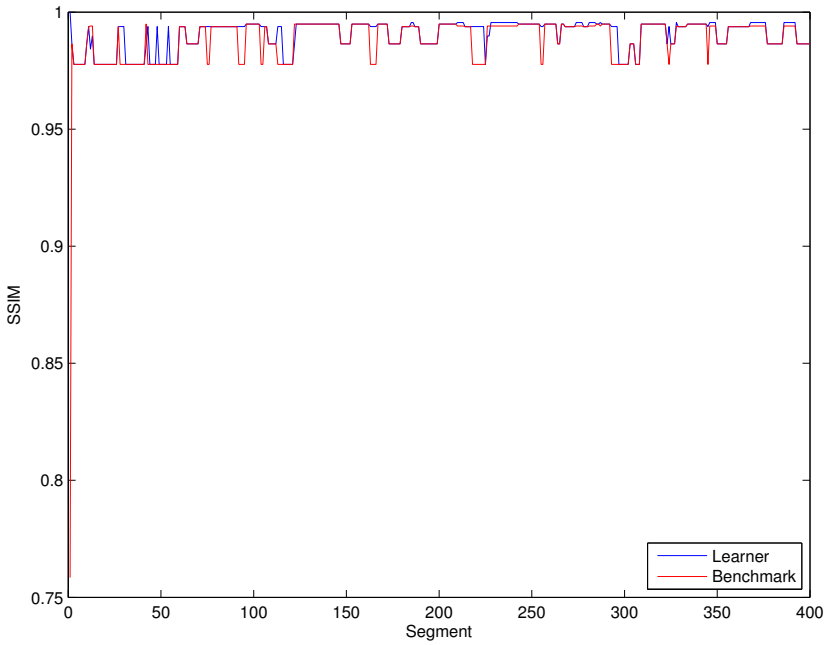


Fig. 4.30: SSIM of the last episode with a dynamic video (mean scene duration 5 segments)

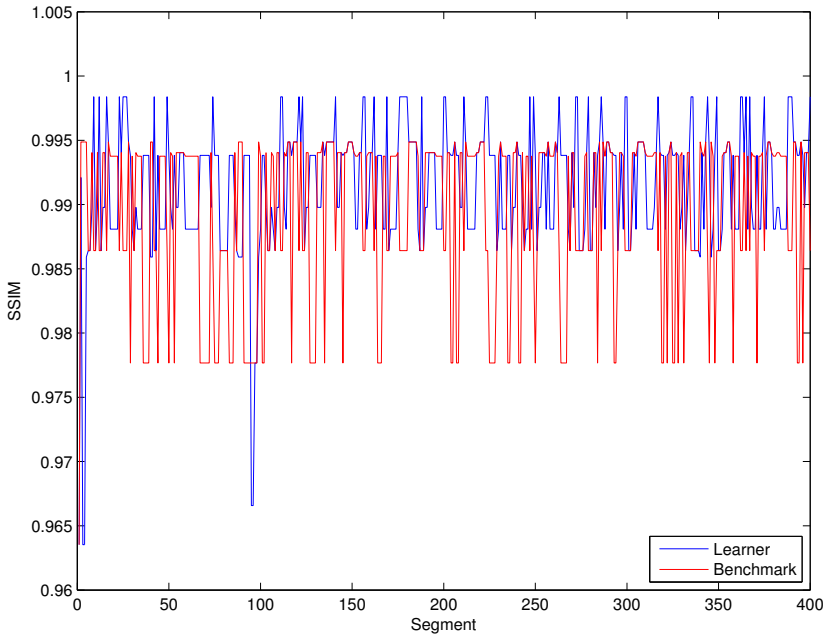


Fig. 4.31: SSIM of the last episode with a dynamic video (mean scene duration 1 segment)

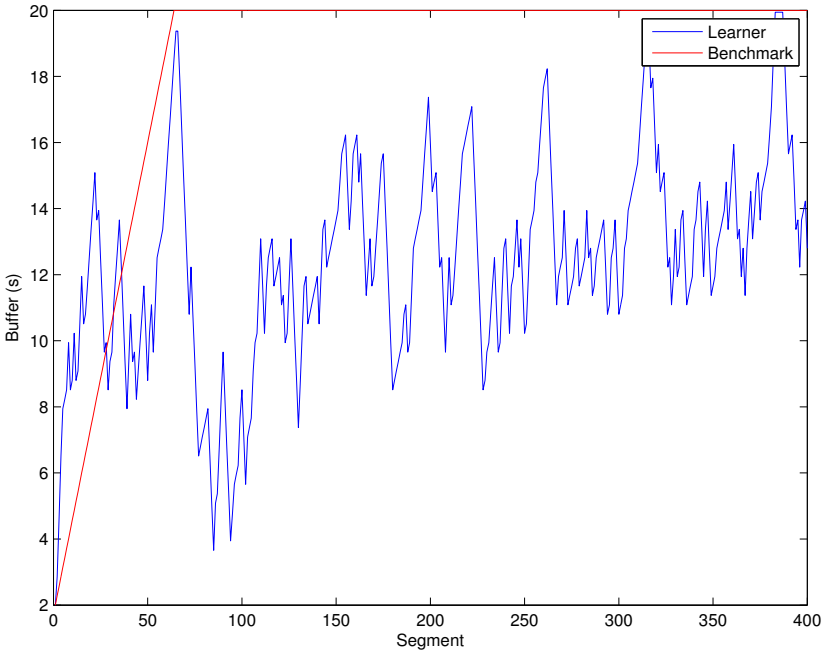


Fig. 4.32: SSIM of the last episode with a dynamic video (mean scene duration 1 segment)

4.4.3 Dynamic scenario

In the **Dynamic** scenario, the difference between the Offline and Online algorithms was clearer.

With $p = 0.25$, the Online algorithm's mean r_k was higher than the benchmark by 0.002 with both transition matrices. Fig. 4.33 and Fig. 4.34 show how the learner was able to effectively use the buffer to maintain a higher SSIM throughout the video with a Markov channel using the first transition matrix, avoiding most of the damaging quality drops when the channel channel rate was low. A plot of the SSIM values over time for a Markov channel using the second transition matrix is shown in Fig. 4.35.

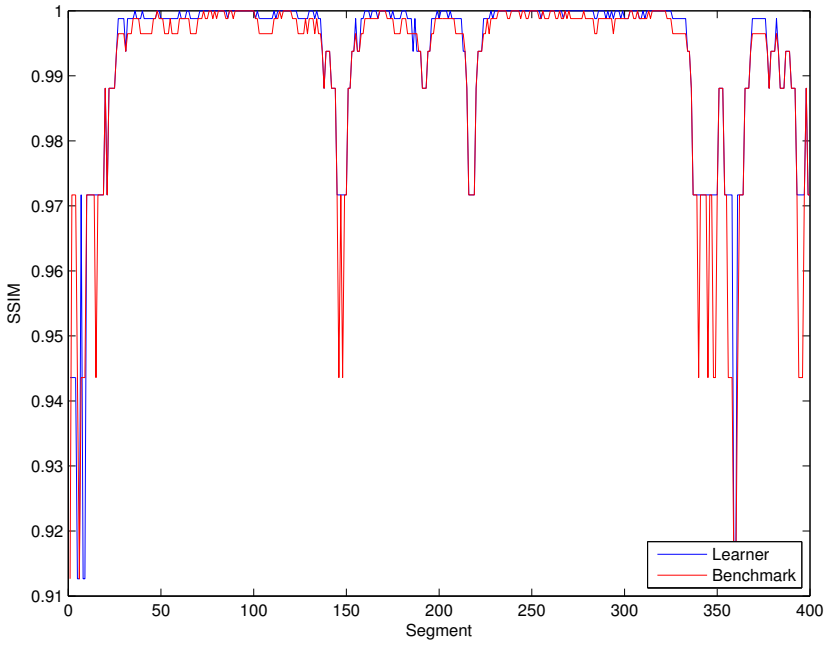


Fig. 4.33: SSIM of the last episode with a Markov channel ($p = 0.25$)

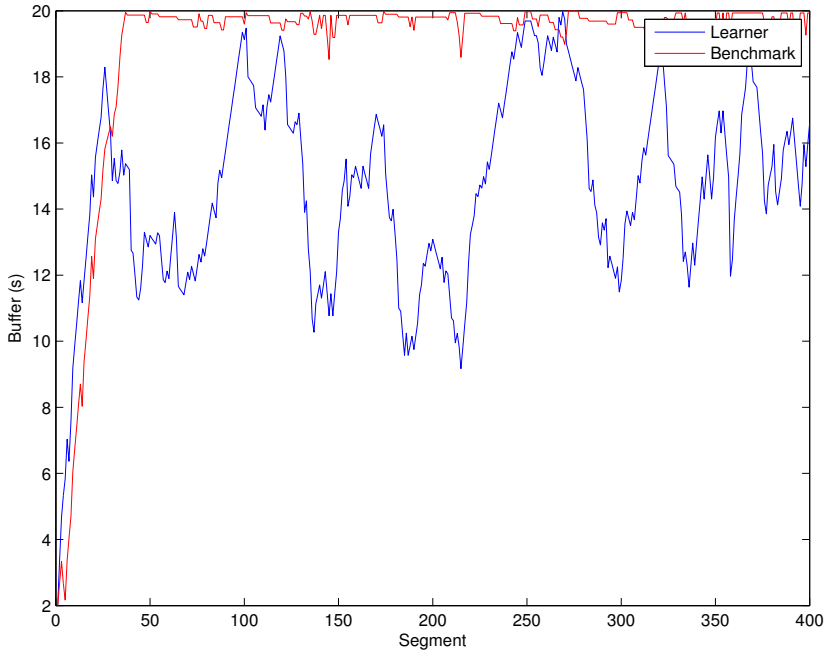


Fig. 4.34: Buffer of the last episode with a Markov channel ($p = 0.25$)

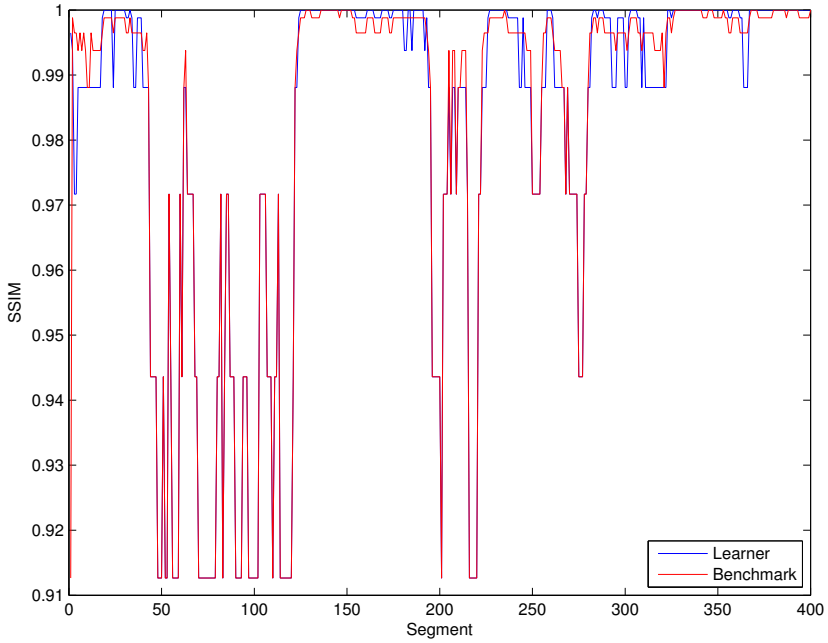


Fig. 4.35: SSIM of the last episode with two-state jumps ($p = 0.25$)

With $p = 0.5$, Online is clearly outperformed by the Offline algorithm: while the Online algorithm's r_k with the second matrix is higher than the benchmark's by 0.008, 0.005 less than the Offline algorithm, its performance with the first transition matrix is essentially the same as the benchmark's. In that situation, the Offline algorithm's mean r_k is higher by 0.005.

Fig. 4.36 and Fig. 4.37 show how the learner's intelligent exploitation of the buffer gives it an advantage over the benchmark with a Markov channel using the second transition matrix. However, a comparison between Fig. 4.18 and Fig. 4.36 clearly shows that the Offline algorithm performs better in this scenario.

The case with the uncorrelated channel shows an interesting side effect of the difference in the MDP formulations: while the Offline algorithm's chosen video bitrate is not constant, the Online algorithm settles on a constant rate, getting the same results in terms of r_k but keeping the quality constant at all times by always choosing the 1 Mbs video bitrate.

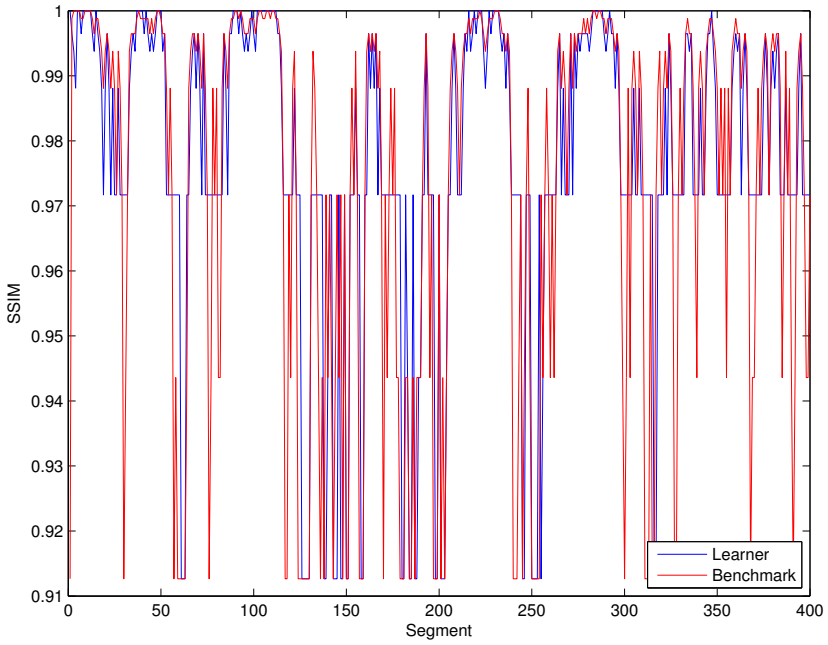


Fig. 4.36: SSIM of the last episode with a Markov channel ($p = 0.5$)

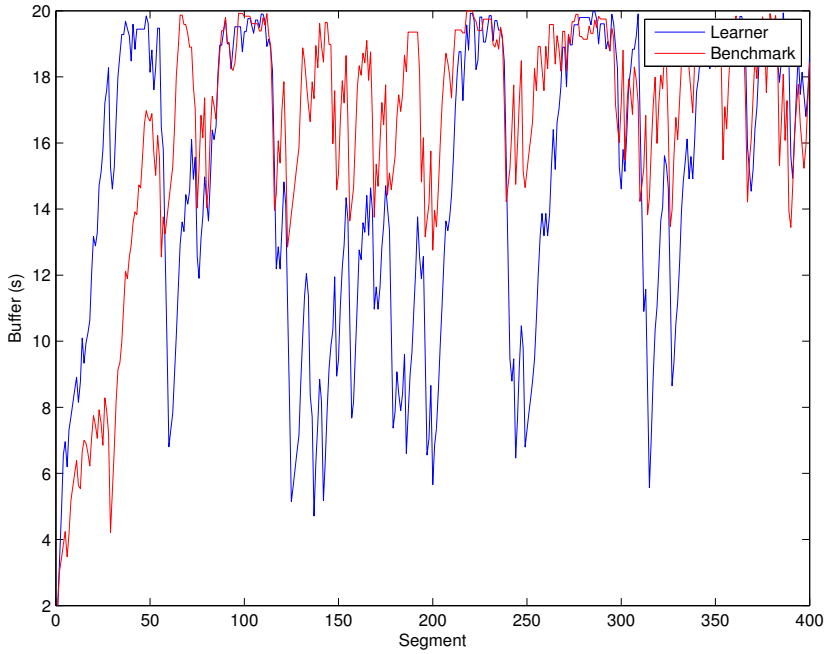


Fig. 4.37: Buffer of the last episode with a Markov channel ($p = 0.5$)

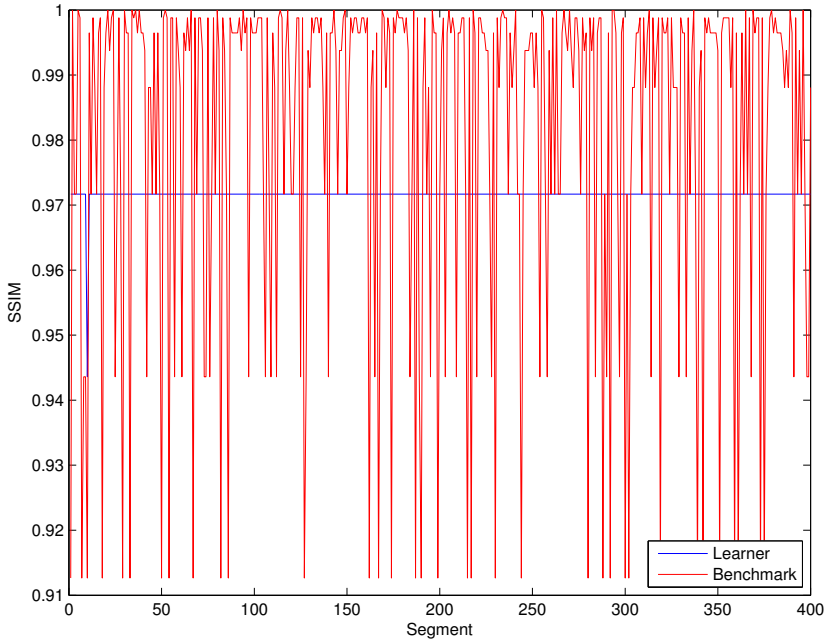


Fig. 4.38: SSIM of the last episode with a uniform Markov transition matrix

4.4.4 Complete scenario

The next simulations concerned the **Complete** scenario, with a Markov channel and a dynamic video: the average scene duration was set to 5 segments, and the Markov transition matrix was set to allow two-state transitions as before.

The Online algorithm's performance shows no significant differences with the Offline algorithm's with $p = 0.25$; both algorithms managed to combine its awareness of the variability of the channel and of the video complexity, achieving a higher gain than in the static simulations. A plot of r_k is shown in Fig. 4.39.

With $p = 0.5$, the algorithm's mean reward r_k is 0.01 higher than the benchmark's, 0.003 less than the Offline algorithm. Fig. 4.40 shows a plot of the SSIM for one episode after convergence.

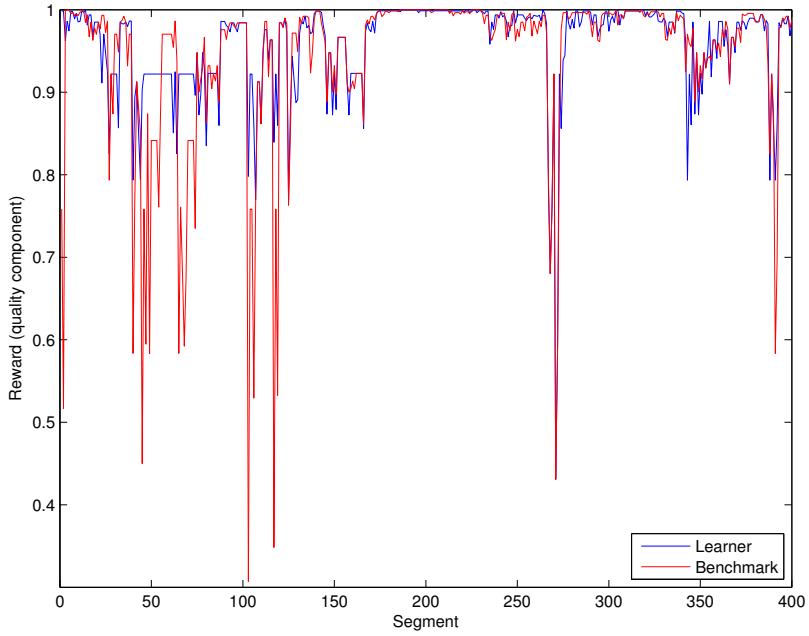


Fig. 4.39: Reward (quality component) with $p = 0.25$ and a dynamic video

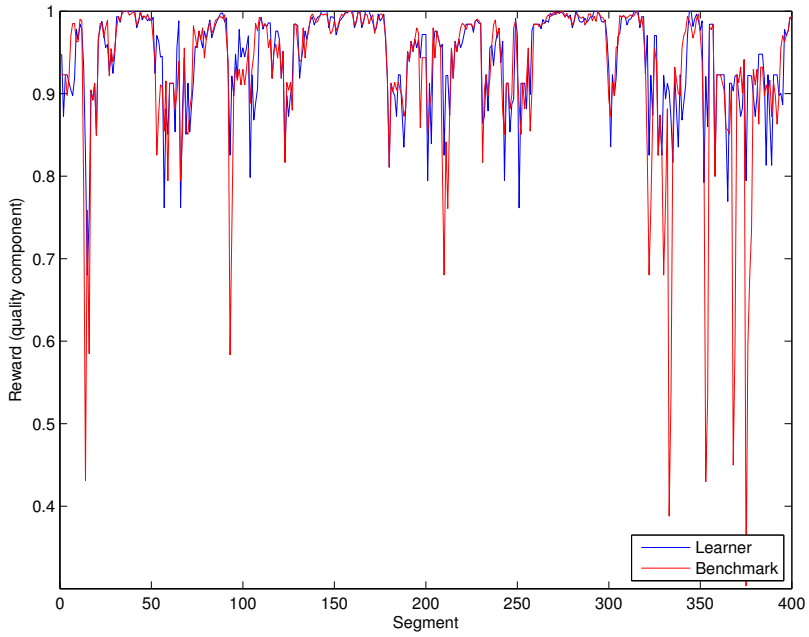


Fig. 4.40: Reward (quality component) with $p = 0.5$ and a dynamic video

Finally, the adaptation test proved that reactiveness to changes in the environment is the Online algorithm's main strength: while the Offline algorithm's performance degraded after a sudden change in the environment, the Online algorithm's mean r_k is higher than the benchmark's by 0.015 over the 10 episodes after the change. Fig. 4.41 shows the first episode after the shift; the learner clearly avoids the worst quality drops, resulting in a better performance even if its Q-values are still adapting to the new situation.

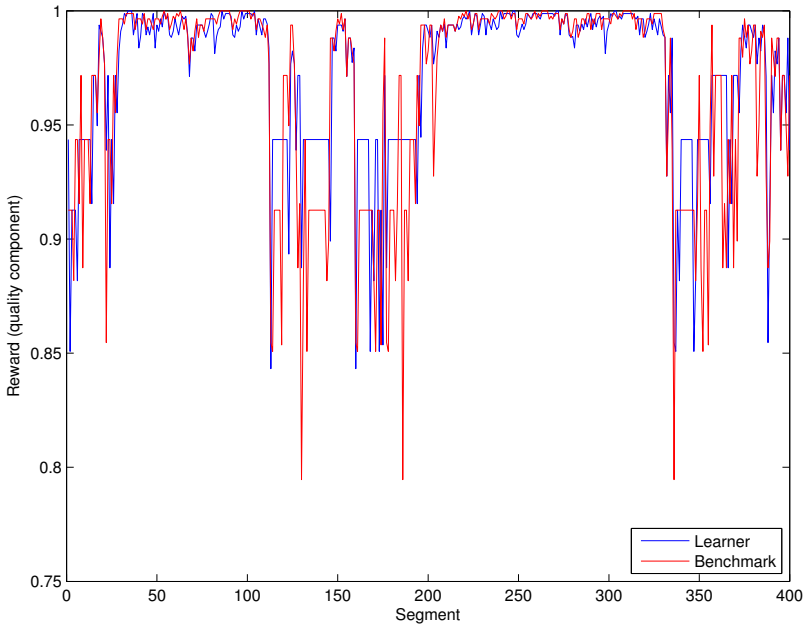


Fig. 4.41: Reward (quality component) after changing the video

4.5 Comparison

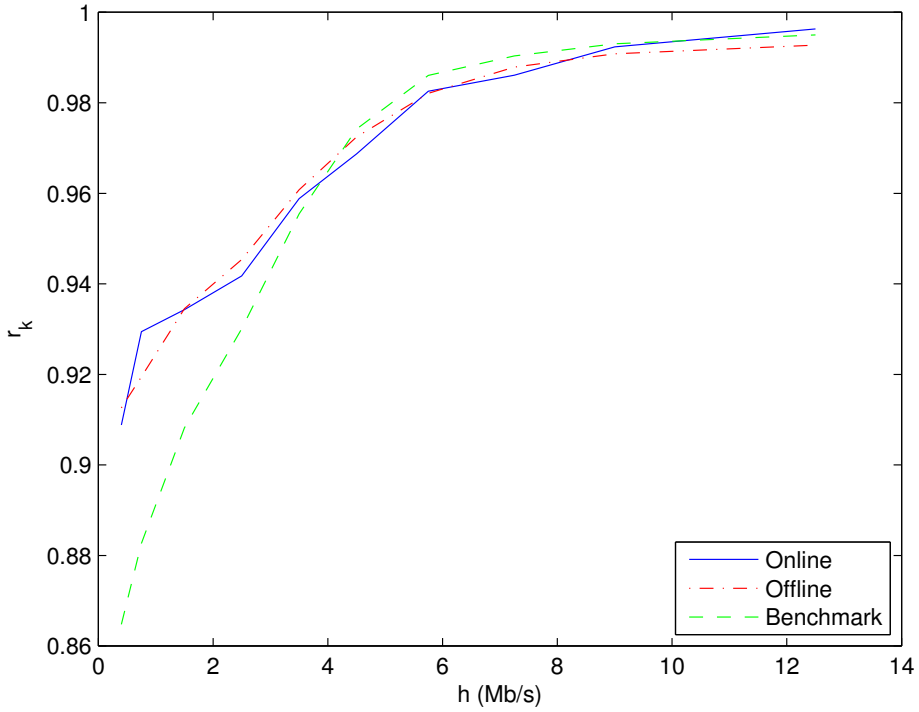


Fig. 4.42: Average reward (quality component) over channel rate

As Fig. 4.42 shows, the two learning algorithms have a significant advantage at low channel rates when the channel is Markovian. The comparison was performed in the complete scenario, with a Markov channel with two-state jumps and $p = 0.5$ and an average scene duration of 5 segments; the results in the figure were obtained by averaging the reward (considering only the quality component r_k) over 100 episodes after convergence, sorting the segments by channel rate.

The slight advantage (0.004 at its highest) of the benchmark at high capacities is probably due to the learners' reluctance to fully exploit the channel, knowing its inherent instability, and it is more than compensated by the gains when the channel channel rate is low, which amount to almost 0.05 in terms of the mean reward r_k.

The Offline algorithm proved to have an edge over the Online algorithm in complex situations, as its more detailed Markov model allows it to react to the channel changes with a higher precision. The lack of the need for continued learning also makes its actions more regular in some situations (e.g. when $h = 3.9\ Mb/s$, see Fig. 4.4 and Fig. 4.28), as updating the Q-values online occasionally upsets the perceived optimal action in some states.

Chapter 5

Conclusions and future work

The results of the simulation proved that the two RL bitrate adaptation algorithms are effective in a variety of environments, and their performance is satisfactory even in challenging scenarios. Part of the credit goes to the MDP model, which has shown remarkable descriptive power and flexibility.

The MDP formulation has another advantage over the existing examples in the literature: the Markov chains underlying the MDP that the bitrate adaptation algorithms solved were relatively small, reducing both the convergence times of the Q-values and the memory requirements on the client device. The combination of high performance, flexibility and simplicity represent a significant step forward over existing RL solutions, which were both highly complex and with a limited scope.

The simplicity of the Online and Offline algorithms constrasts even more starkly with the complex stateful algorithms in the literature: once the training phase is over, the Offline algorithm involves a simple look-up operation, and the model can be made more complex with no computational cost. The training phase will necessarily be longer, but it can be pre-loaded on the device. An even simpler implementation might not save all the Q-values, but just the best action for each state, considering the algorithm does not have to perform any exploration. The Online algorithm's higher complexity is justified by the fact that it is not based on a fixed model, but progressively learns the model and adapts it to environmental changes. However, even the

Online parallel updates are not computationally expensive.

On the other hand, the Online algorithm converges quickly enough to allow it to quickly react to changes in the model. 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 model. As long as the channel is approximately Markovian from one segment to the next, the algorithm can adapt to varying channel and video conditions, and, although it is not as efficient as the Offline algorithm, its adaptability makes it more suited to applications that operate in unpredictable environments, such as mobile video streaming.

Possible future extensions on the work on the two learning algorithms include a more comprehensive simulation study, using the full TCP stack. Although the simulation results prove their efficiency and resilience to different network situations, a test with the full TCP stack would provide additional information to further refine the system parameters.

In parallel, further work can be devoted to the improvement of the algorithms: a possibility is to limit errors and speed up the convergence of the system by giving the learning agent additional information on the video model. An interesting enhancement to the current system can be a fluid Markov model: the learner may use pattern matching and other machine learning techniques to change its own model (e.g., shift the state borders).

Investigation into soft state transitions for other parameters, such as the buffer, or even use of function approximation techniques would make the state space a better fit for the continuous variables involved, making the system perform better without the need to increase the number of states or the complexity level of the learning algorithm itself.

A final 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. The use of game theoretical models [47] and emergent behavior theory [48], which have already been successfully applied in other communications problems [49], represents a novel perspective on an exciting challenge.

Bibliography

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

[2] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time applications," *RFC 3550*, vol. 7, 2003.

[3] MPEG, "Dynamic Adaptive Streaming over HTTP (DASH) – Part 1: Media Presentation Description and segment formats."

[4] J. van der Hooft, S. Petrangeli, M. Claeys, J. Famaey, and F. De Turck, "A learning-based algorithm for improved bandwidth-awareness of adaptive streaming clients,"

[5] M. Claeys, S. Latré, J. Famaey, T. Wu, W. Van Leekwijck, and F. 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)*, pp. 30–37, 2013.

[6] C. J. Watkins, *Learning from delayed rewards*. PhD thesis, University of Cambridge, England, 1989.

[7] T. Stockhammer and M. G. Luby, "DASH in mobile networks and services," in *Visual Communications and Image Processing (VCIP)*, pp. 1–6, IEEE, 2012.

[8] T. C. Thang, Q.-D. Ho, J. W. Kang, and A. T. Pham, "Adaptive streaming of audiovisual content using MPEG DASH," *IEEE Transactions on Consumer Electronics*, vol. 58, no. 1, pp. 78–85, 2012.

[9] S. Jumisko-Pyykkö and J. Häkkinen, "Evaluation of subjective video quality of mobile devices," in *Proceedings of the 13th annual ACM international conference on Multimedia*, pp. 535–538, ACM, 2005.

[10] L. Aimar, L. Merritt, E. Petit, M. Chen, J. Clay, M. Rullgrd, C. Heine, and A. Izvorski, "X264 - a free H264/AVC encoder," *Online (last accessed on: 04/01/07): http://www. videolan. org/developers/x264. html*, 2005.

[11] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hobfeld, and P. Tran-Gia, "A survey on quality of experience of http adaptive streaming," *Communications Surveys & Tutorials, IEEE*, vol. 17, no. 1, pp. 469–492, 2014.

[12] R. K. Mok, E. W. Chan, and R. K. Chang, "Measuring the quality of experience of HTTP video streaming," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 485–492, IEEE, 2011.

[13] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP," in *Proceedings of the second annual ACM conference on Multimedia systems*, pp. 157–168, ACM, 2011.

[14] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, and C. Lilley, "Network performance effects of HTTP/1.1, CSS1, and PNG," in *ACM SIGCOMM Computer Communication Review*, vol. 27, pp. 155–166, ACM, 1997.

[15] C. Müller, S. Lederer, and C. Timmerer, "An evaluation of dynamic adaptive streaming over HTTP in vehicular environments," in *Proceedings of the 4th Workshop on Mobile Video*, pp. 37–42, ACM, 2012.

[16] M. Alreshoodi and J. Woods, "Survey on QoE\QoS correlation models for multimedia services," *International Journal of Distributed and Parallel Systems*, vol. 4, no. 3, p. 53, 2013.

[17] Y. Qi and M. Dai, "The effect of frame freezing and frame skipping on video quality," in *International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP'06)*, pp. 423–426, IEEE, 2006.

[18] T. Hossfeld, S. Egger, R. Schatz, M. Fiedler, K. Masuch, and C. Lorentzen, "Initial delay vs. interruptions: between the devil and the deep blue sea," in *Fourth International Workshop on Quality of Multimedia Experience (QoMEX)*, pp. 1–6, IEEE, 2012.

[19] J. De Vriendt, D. De Vleeschauwer, and D. Robinson, "Model for estimating QoE of video delivered using HTTP adaptive streaming," in *IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pp. 1288–1293, IEEE, 2013.

[20] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[21] S. Mohamed and G. Rubino, "A study of real-time packet video quality using random neural networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 12, pp. 1071–1083, 2002.

[22] K. D. Singh, Y. Hadjadj-Aoul, and G. Rubino, "Quality of experience estimation for adaptive HTTP/TCP video streaming using H.264/AVC," in *Consumer Communications and Networking Conference (CCNC)*, pp. 127–131, IEEE, 2012.

[23] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*. MIT Press, 1998.

[24] R. Bellman, "A Markovian decision process," tech. rep., DTIC Document, 1957.

[25] D. L. Poole and A. K. Mackworth, *Artificial Intelligence: foundations of computational agents*. Cambridge University Press, 2010.

[26] S. B. Thrun, "Efficient exploration in reinforcement learning," tech. rep., 1992.

[27] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing*, pp. 227–236, Springer, 1990.

[28] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[29] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," 1994.

[30] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[31] S. P. Singh, T. Jaakkola, and M. I. Jordan, "Reinforcement learning with soft state aggregation," *Advances in neural information processing systems*, pp. 361–368, 1995.

[32] P. Y. Glorennec and L. Jouffe, "Fuzzy Q-learning," in *Proceedings of the Sixth IEEE International Conference on Fuzzy Systems*, vol. 2, pp. 659–662, IEEE, 1997.

[33] J. C. Santamaría, R. S. Sutton, and A. Ram, "Experiments with reinforcement learning in problems with continuous state and action spaces," *Adaptive behavior*, vol. 6, no. 2, pp. 163–217, 1997.

[34] L. Li, T. J. Walsh, and M. L. Littman, "Towards a unified theory of state abstraction for MDPs," in *ISAIM*, 2006.

[35] S. Thakolsri, W. Kellerer, and E. Steinbach, "QoE-based rate adaptation scheme selection for resource-constrained wireless video transmission," in *Proceedings of the international conference on Multimedia*, pp. 783–786, ACM, 2010.

[36] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with FESTIVE," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pp. 97–108, ACM, 2012.

[37] Z. Li, A. C. Begen, J. Gahm, Y. Shan, B. Osler, and D. Oran, "Streaming video over HTTP with consistent quality," in *Proceedings of the 5th ACM Multimedia Systems Conference*, pp. 248–258, ACM, 2014.

[38] M. Xing, S. Xiang, and L. Cai, "Rate adaptation strategy for video streaming over multiple wireless access networks," in *Global Communications Conference (GLOBECOM)*, pp. 5745–5750, IEEE, 2012.

[39] Y. Fei, V. W. Wong, and V. Leung, "Efficient QoS provisioning for adaptive multimedia in mobile communication networks by reinforcement learning," *Mobile Networks and Applications*, vol. 11, no. 1, pp. 101–110, 2006.

[40] M. Claeys, S. Latré, J. Famaey, T. Wu, W. Van Leekwijck, and F. De Turck, "Design and optimisation of a (FA) Q-learning-based HTTP adaptive streaming client," *Connection Science*, vol. 26, no. 1, pp. 25–43, 2014.

[41] S. Petrangeli, M. Claeys, S. Latré, J. Famaey, and F. De Turck, "A multi-agent Q-learning-based framework for achieving fairness in HTTP adaptive streaming," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pp. 1–9, IEEE, 2014.

[42] N. Changuel, B. Sayadi, and M. Kieffer, "Online learning for QoE-based video streaming to mobile receivers," in *Globecom Workshops (GC Wkshps)*, pp. 1319–1324, IEEE, 2012.

[43] D. Marinca, D. Barth, and D. De Vleeschauwer, "A Q-learning solution for adaptive video streaming," in *2013 International Conference on Selected Topics in Mobile and Wireless Networking (MoWNeT)*, pp. 120–126, IEEE, 2013.

[44] N. Mastronarde and M. van der Schaar, "Fast reinforcement learning for energy-efficient wireless communication," *IEEE Transactions on Signal Processing*, vol. 59, no. 12, pp. 6262–6266, 2011.

[45] S. Wang, A. Rehman, Z. Wang, S. Ma, and W. Gao, "SSIM-motivated rate-distortion optimization for video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 4, pp. 516–529, 2012.

[46] M. Zanforlin, D. Munaretto, A. Zanella, and M. Zorzi, "SSIM-based video admission control and resource allocation algorithms," in *12th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, pp. 656–661, IEEE, 2014.

[47] S. Parakh and A. K. Jagannatham, "Game theory based dynamic bit-rate adaptation for H.264 scalable video transmission in 4G wireless systems," in *International Conference on Signal Processing and Communications (SPCOM)*, pp. 1–5, IEEE, 2012.

[48] M. J. Prietula and K. M. Carley, "Computational organization theory: Autonomous agents and emergent behavior," *Journal of Organizational Computing and Electronic Commerce*, vol. 4, no. 1, pp. 41–83, 1994.

[49] I. Kassabalidis, M. El-Sharkawi, I. Marks, R.J., P. Arabshahi, and A. Gray, "Swarm intelligence for routing in communication networks," in *Global Telecommunications Conference (GLOBECOM)*, vol. 6, pp. 3613–3617 vol.6, IEEE, 2001.

Acronyms

CAC Call Admission Control.

CDN Content Delivery Network.

DASH Dynamic Adaptive Streaming over HTTP.

HTML HyperText Markup Language.

HTTP HyperText Transfer Protocol.

MDP Markov Decision Process.

MOS Mean Opinion Score.

MPD Media Presentation Description.

MPEG Moving Picture Experts Group.

MSE Mean Square Error.

MSS Microsoft Smooth Streaming.

POMDP Partially Observable MDP.

PSNR Peak Signal to Noise Ratio.

PSQA Pseudo-Subjective Quality Assessment.

QoE Quality of Experience.

QoS Quality of Service.

RL Reinforcement Learning.

SARSA State-Action-Reward-State-Action.

SSIM Structural Similarity Index.

SVC Scalable Video Coding.

TCP Transmission Control Protocol.

TD Temporal Difference.

UDP User Datagram Protocol.

URL Uniform Resource Locator.

List of Algorithms

Acknowledgments

I would like to thank prof. Frossard for his help and ready advice, as well as for the time and patience it took to guide me through the complicated first steps.

Another huge thank you goes to Stefano and Laura, who talked with me for dozens of hours and wrote me hundreds of emails to help make everything work.

I would not have had the opportunity to even begin this thesis if it weren't for prof. Zanella and prof. Zorzi, who I thank for all the support before and during my stay in Lausanne.

Finally, a big hug to all the friends who had to bear my frustrated messages when things went wrong, even if they were scattered all over Europe: Leo, Irene, Elena, Daina, Giorgia, Chiara, Giovanni and Luca.