Reinforcement learning for dynamic multimedia adaptation

Vincent Charvillat, Romulus Grigoraş *

IRIT – ENSEEIHT
Department of Computer Science and Applied Mathematics
2 rue Camichel
31071 Toulouse, France

Abstract

In this paper we present an integration of several user and resource-related factors for the design of dynamic adaptation techniques. Our first contribution is an original reinforcement-learning approach to develop better adaptation agents. Integrated with the content, these agents improve gradually, by taking into account both user's behaviour and the usage context. Our second contribution is to apply this generic approach to solve an ubiquitous streaming problem. Mobile users experience large latencies while accessing streaming media. We propose to adapt the streaming by prefetching and to model this decision problem by using a Markov Decision Process. We discuss this formal framework and make explicit reference to its relationship with reinforcement learning. We support the benefits of our approach by presenting results from simulations and experiments.

Key words: multimedia adaptation, reinforcement learning, Markov Decision Process, ubiquitous streaming, prefetching policies

1 Introduction

The ubiquitous access to multimedia services is facing an increasing amount of heterogeneity in devices, networks, content and users' preferences. Enabling

* *E-mail address:* grig@enseeiht.fr

transparent use of multimedia content (anytime, anywhere and anyway) therefore requires the intermediate help of adaptation techniques. These techniques usually adapt the multimedia content or the delivery mechanisms according to terminals' capabilities, networks' conditions and user' preferences. Far from previous Internet or TV paradigms, the emerging use cases are further complicated by the dynamic nature of ubiquitous communication and interactive services. Hence the aforementioned adaptation techniques must be dynamically designed to deal with the variability of each usage context or environment.

The description or modelling of such a "usage context" is a major issue of the current research in adaptation within several research communities. Sometimes limited to content transcoding in the past, the resource-based adaptation now integrates the user profiles and preferences in their framework (e.g. the Usage Environment Description of MPEG-21 DIA [1]). In addition, the user-modeling based personalization provides adaptation not only to user-models but also to the properties of available devices and network conditions.

All these adaptation mechanisms integrate widened usage contexts which are characterized by great unpredictability in the ubiquitous multimedia field. Two unpredictability sources can be illustrated. By nature, a nomadic user of a multimedia service with a limited terminal and varying bandwidth will experience an unpredictable execution environment. This unpredictability doubles if the service becomes interactive with random access to online content. Both types of uncertainty, those coming from the user as well as those from the available resources require dynamic adaptation techniques but it is no longer clear that they must be treated independently with separate mechanisms: "resizing an image for a user who will not scroll down to see it" is not relevant. Confronting all these numerous uncertainty sources on one hand, and the multitude of possible adaptation mechanisms, on the other hand, we are considering the question of how to dynamically choose among or compose these mechanisms.

We try to answer this question in an original manner by proposing a closed-loop recipe :

- observe the user's behaviour rather than presume it ;
- define performance criteria (possibly linked to available resources) ;
- learn adaptation strategies that maximize these criteria ;
- enrich multimedia content with adaptation agents that are updated as and when the content is used (then we get back to the first point).

In this sense, we argue the case for an intelligent integration of several user-based or resource-based factors to design dynamic adaptation techniques. The key idea of our approach is to map this closed-loop proposal in the Reinforcement Learning framework. Reinforcement learning is a very successful compu-

tational intelligence method investigated or simply used in several fields (AI, robotics, networking etc.).

In the following sections of this paper we first review the related research (section 2). Then we introduce our first contribution in the third section : an original approach to dynamic adaptation by reinforcement learning. Three illustrative adaptation problems are briefly discussed. In section 4 we make the link between reinforcement learning and Markov Decision Processes. Among the three problems considered in section 3, the adaptive streaming problem is detailed in section 5. We show in this paper that this problem can be modeled in a very elegant way by using a Markov Decision Process. We then bring a second contribution by deriving optimal and adaptive prefetching policies (section 6). Some experiments validate these ideas before concluding and listing possible avenues for further work.

2 Related work

Multimedia adaptation is a hot topic today. Naturally the literature provides us with a wealth of research papers. In this section we review the three key concepts related to our work: resource-based adaptation, user-aware adaptation and integration of user-based and resource-based factors.

2.1 Multimedia adaptation to limited resources

We need multimedia adaptation techniques to face the heterogeneity of networks and terminals [2]. The interconnected networks provide various performances in available bandwidths (from a few kbps to several mbps), packet loss rates (from almost lossless to error prone environment), delay and jitter etc. In addition, terminals capabilities range from high-performance workstations to mobile devices. The latter are often limited by their processing power, memory, display and audio capabilities. Solutions to these problems may combine content adaptation and adaptive delivery (streaming).

Content adaptation. Several authors [3,4] distinguish between adapting single media and adapting structured multimedia presentations. Single media adaptation can be further refined in transcoding [2,5](e.g. reduction of video size, sampling rate or color depth, codec substitution), in transmoding [6](e.g. modalities conversion such as speech to text or video to image) or in summarization [7].

Lemlouma and Layaida [3] review different techniques and tools for trans-

forming a complex multimedia composition or presentation. Such techniques combine adaptation of single media objects and adaptation of the presentation structure (e.g. spatial layout, temporal relationships between the media objects, hyperlinks structure).

Adaptive streaming. Many adaptive delivery techniques have considered the problem of time-varying bandwidth, loss rates and delays. We extend Girod's analysis [8] and distinguish between four main approaches: source-coding rate adaptations (scalable content), variable transmission rate techniques (optimisation of packet scheduling), adaptive media playout (AMP) at the receiver-side (playout rate adjustement), and proxy-caching or prefetching methods. We detail this last point since we will present our optimal prefetching policies later in this paper.

Accessing continuous media over the Internet implies unacceptable startup delays (access latencies). To overcome such limitations, various works propose to use proxy-caching to adaptively deliver streaming media. Since full media-object caching is not feasible because of its high costs, current methods investigate prefix caching [9,10], segments caching [11], or layers caching [12].

Additionally, since the objects' popularity and the users' access pattern dynamically change, it is natural to go beyond passive fetching (fetching of uncached segments only happens on a cache miss). Prefetching consists in predicting future access thus trying to assure a continuous delivery. Chen et al [11] recently proposed a review of the prefetching literature and also an active prefetching method combined with segment-based caching. Rejaie et al [13] studied the prefetching of layered videos. Fitzek and Reisslein [14] use a heuristic prefetching policy which allocates more transmission capacity to wireless client with small prefetched reserves. Yu et al [15] propose a prefetching scheme that is aware of network conditions. Other authors [16,17] address prefetching solutions inside video servers (optimisation of video objects' requests). All these approaches have in common the use of heuristically-designed prefetching policies. By contrast, our work aims to learn optimal prefetching policies.

2.2 User-aware adaptation

Another dimension of multimedia adaptation deals with the integration of human factors (users' preferences and satisfaction). Two main approaches can be identified: adaptive techniques that maximize user-perceived QoS and personalization approaches in adaptive hypermedia systems.

User-perceived QoS. Ruiz et al [18] propose a user-aware adaptive application in which the adaptation mechanism is driven by the user-perceived QoS.

Given some network conditions, they want to find the best settings (video frame rate, video size etc.) for satisfying the resource-based constraints (e.g. bandwidth) while maximizing the quality of perception (QoP). By optimizing protocols' reconfiguration, Ghinea and Malougas [19] also aim to maintain a high-level QoP as low-level QoS varies.

Several authors have evaluated the influence of adaptation mechanisms on the perceived quality. Apteker [20] investigated the impact of varying frame rates, Wijesekera [21] tracked the effect of packet losses with respect to several media sources, Gulliver [22] addressed the impact of video adaptation on informational transfer and user satisfaction.

Adaptive hypermedia systems. In the same trend, the"user modeling" community studies adaptive hypermedia systems [23,24]. These systems generally provide various types of navigation support. For example, they can be used to avoid improper navigation and therefore improper learning with an educational content, or to personnalize a shopping or educational session. These approaches are commonly based on learning techniques that allow to infer the intentions of the user by taking into account recorded past interactions of the same user or similar users [25]. These approaches are often applied to web sites or to interactive presentations. Unfortunately, random access to multimedia content [26] and consideration of resource constraints [27] have received little consideration.

2.3 Context-aware adaptation

By integrating both the resource-based constraints and the users' preferences we get the so-called "context-aware adaptation techniques".

Metadata-driven approaches. In context-aware adaptation, any "one-size fits all" is conceptually and technologically unrealistic. Hence researchers try to define common description toolkits [28]. These descriptors enable both new proposals for adaptation and interoperability between adaptation components or services [1]. In MPEG-21 Digital Item Adaptation framework, the context descriptors include user preferences, terminal capabilities, network characteristics, author recommendations, natural environment characteristics, semantic properties etc. Within the W3C framework, CC/PP (Composite Capabilities/Preferences Profiles) can also be a basis for a context model and for a context management system.

Decision-taking issues. Describing the context is necessary but not sufficient for deciding how to perform adaptation. The decision taking entity is a key component for context-aware adaptation [1]. As an example, consider a video streaming context. There are various settings that an adaptive system

can change in order to consume less than the available bandwidth : it can choose among several transcoding strategies (by combining video size or sampling rate reduction with codec substitution) or prefer a video-to-slideshow transmoding. Basic adaptation mechanisms just pick up a possible solution or apply simple priority rules to decide among these possibilities [4]. Ruiz et al [29] investigate the optimization of the user-perceived QoS by machine learning : they use genetic algorithms and rule induction algorithms to build up a context-aware adaptation mechanism. In their paper these authors also propose a review of machine learning approaches to solve specific issues in wireless networks. They point out several papers that use reinforcement learning techniques to optimize the power consumption of wireless interfaces or to minimize network congestion. We can note that decision-taking for multimedia adaptation is an emerging topic. Our approach using reinforcement learning fits within this research area particularly well because it also naturally addresses dynamic adaptation.

Static vs. dynamic adaptation. In context-aware adaptation, most of the context features can change dynamically : both the available resources and the user requirements may change over time. Layaida and Hagimont [30] review some adaptive applications considering their degree of "dynamicity". They first introduce "statically configured" adaptation mechanisms, that is adaptation algorithms that are statically designed or predefined. These algorithms can be instanciated with some parameters which can be fixed (no reconfiguration) or dynamically modified (reconfiguration). As an example of such "static adaptation with threshold-based reconfiguration", we can imagine a rate adaptive technique launched when RTCP feedbacks go above an alarm threshold. The "dynamic configuration" of adaptation mechanisms is also proposed (with or without reconfiguration) so that the adaptation algorithm modifies itself. For instance, simple algorithm's modification can be performed through dynamic downloading and composition of software codecs or filters. Obviously, it is natural to statically negociate (once forever and always in the same way) the preferred language of a user or the image resizing factor on a given mobile terminal: static adaptaptation mechanisms match static contexts well. However, by increasing the interactivity of the service and the context variability, we are quickly confronted with additional uncertainty sources that could justify the fully dynamic adjustments of parameters and algorithms themselves. Therefore our work argues for an intelligent integration of several user-based or resource-based factors for designing fully dynamic adaptation techniques.

3 A learning based approach to dynamic adaptation

Our first contribution is an original learning-based approach to dynamic adaptation. We address context-based adaptation (section 2.3) by integrating hu-

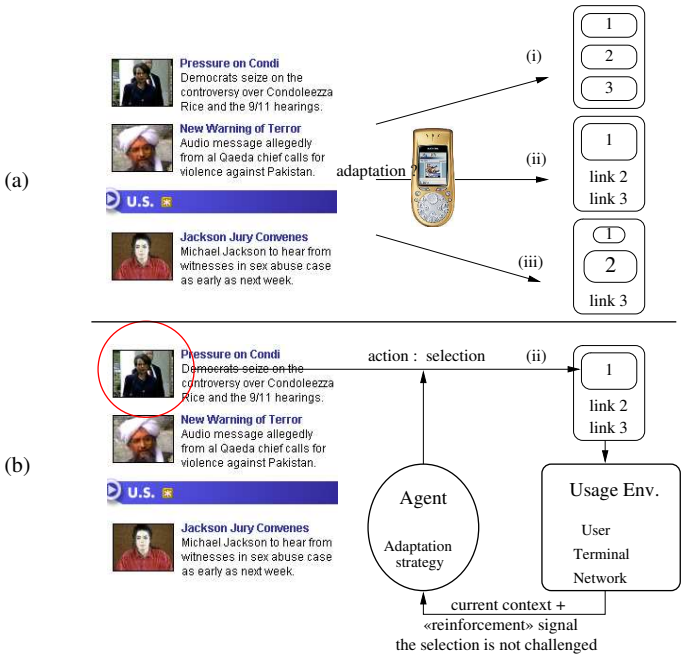

Figure 1. Adapting the access to news-on-demand on a mobile terminal

man factors and available resources. We specifically handle highly dynamic contexts where available resources are limited and time-varying and where users' interactions are also uncertain. We focus on dynamic decision techniques under these both types of uncertainties.

3.1 An introductory example

As an example, let us imagine a multimedia scenario (access to news-on-demand, figure 1(a)). At some point in time, three images are displayed and the user can click on one of them to express his choice. We can think of two strategies for adapting the display on a small screen. We can either proportionally reduce the size of the three images according to the terminal's profile (case i of the figure 1(a)), or we display just one image, the other two are represented as clickable annotations (case ii). Two questions arise: which strategy to choose and if (ii) is selected, which image to choose ? The one preferred by the current user ? The one that corresponds to the most likely/probable choice ? Why not ! And why not consider a third strategy which displays the three images according to the inferred preference of the user (case iii)? The last two cases get us closer to the research on recommendation systems while managing limited resources.

Figure 2 allows us to introduce our solution for answering the key question: what is the best adaptation strategy ? We propose to use an adaptation agent that takes the current context as an input signal. In our introductory example

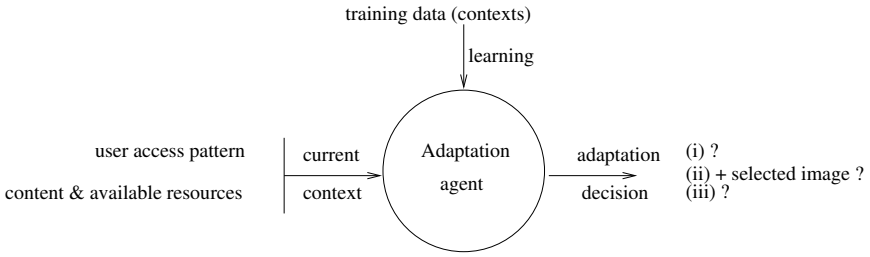

Figure 2. Adaptation agent decision by machine learning

the input includes the current user access pattern and also information about the content and the available resources (display capabilities). The adaptation agent should output a decision by prediction. This prediction is based on a training data set composed of previously observed resource limitations and user access patterns.

We still need to clarify two points: how the dynamicity of the context can be handled but also what is the performance criterion for evaluating a decision ? Firstly, the dynamic context requires the use of a closed-loop decision process since the decision at some point in time may influence the future events. For example, the choice of an image in the strategy (ii) may have an impact on future user requests. Secondly, a user-aware performance criterion can drive the decision. Figure 1(b) shows a closed-loop decision process in which the agent receives as input both the current context and a performance reward. For our example, the agent receives a reward if the image selection is not challenged by the user. This criterion is close to the ones commonly used in user-modeling solutions, but other criteria, such as quality of perception could also provide relevant rewards (section 2.2).

It is now straightforward to argue for the use of reinforcement learning methods for handling dynamic adaptation.

3.2 Reinforcement learning

In the context of reinforcement learning (RL)[31], let us consider an adaptation agent and its environment. This environment integrates the user, his mobile terminal, the network(s), the content (documents, media etc.). The adaptation agent can perceive (at least partially) the characteristics of the environment. These perceived characteristics give the state of the agent in its environment.

The agent can influence the environment : it can transform media, documents, create priorities among media (cf. 3.1), take actions on a user's behalf, relocate content, make bandwidth reservations, make a combination of all these actions or do nothing. Among various possible actions it can take in a given state, an agent has to find the best one. The agent's behaviour is called "decisional

policy", that is simply a function that returns an action for each state.

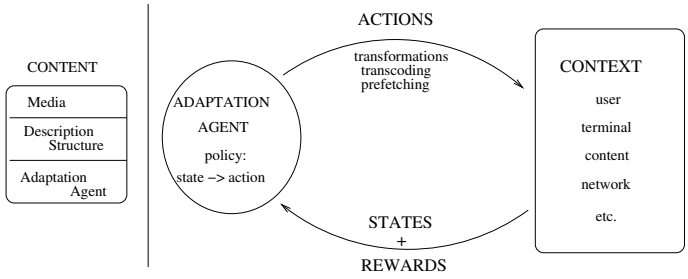

Figure 3. Adapting content using an adaptation agent in its environment

The agent learns from its interaction with the environment : as a reaction to its decisions, the environment returns a new state and a reward (figure 3). The aim is to operate a "trial and error" type of learning, located half-way between supervised and unsupervised learning. The agent tries to reinforce the decisions that resulted in a good accumulation of reward and, conversely, tries to avoid unfruitful decisions. This implies an iterative update of its "decisional policy" that should ideally tend to an optimal policy... The main difficulties of this general approach are the description of the environment and the identification of "rewards", that is performance criteria. It is tricky to find states that are expressive enough while avoiding a combinatorial explosion. The current research in AI concentrate on these points.

3.3 Embedding the agent

Our reinforcement-learning framework should also provide some solutions to locate the adaptation agent. According to the literature [2], there are three different strategies for locating the adaptation mechanism on the end-to-end path from the content server to the client: at the source, at the destination, and in between, somewhere in the network, with some special locations having special properties, e.g. at the boundary between the wireless and the wireline parts of a network (figure 4).

Since the content use (usage history) is essential for designing and updating our adaptation mechanisms, it is straightforward to consider them as meta-data. Therefore we propose to enrich the content by integrating the adaptation agent (left-hand side of the figure 3). When the user requests for some content, data and meta-data (including the agent) are transferred. Specific implementation constraints drive the adaptation agent to its optimal location, on the path from the server to the client. It then applies the current adaptation strategy according to "past experiences". The strategy is subsequently refined with respect to the current user. Therefore the agent's code becomes mobile [32].

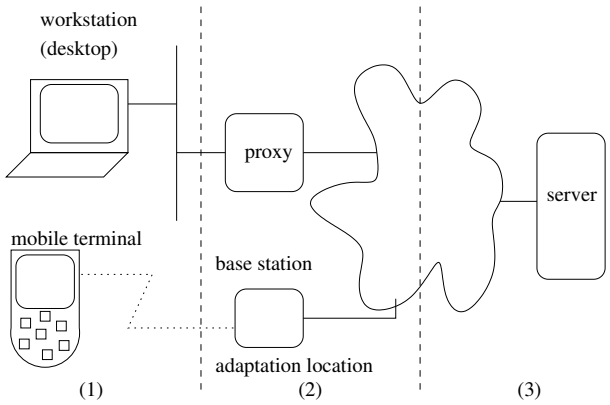

Figure 4. Possible locations of adaptation mechanisms: (1) at the receiver , (2) at a proxy, (3) at the source.

	maintenance	NoD	prefetching
learned actions	self-contained content selection	choice by default selection	playout buffer prefetching
minimization	number of connexions	challenge of recommendations	sum of latencies

Figure 5. Adaptation by reinforcement learning : 3 examples (minimizing such criteria also means maximizing related rewards)

3.4 Examples

In this section we are going to introduce three case studies (figure 5) that show the feasability and the generality of our proposal for solving dynamic adaptation problems. We give only hints for the first two case studies, whereas for the third one we provide a detailed explanation (adapted from Grigoras' PhD thesis [33]).

3.4.1 Nomadic operator in aeronautical maintenance

Maintenance operators usually access technical documentation or maintenance procedures from a lightweight terminal (PDA or tablet PC). The operator can be very far from the documentation servers or connected to them via low-quality networks. An interesting adaptation mechanism could aim at assuring maximal autonomy of the operator by minimizing the number of connexion requests to the server. Therefore there is a need to select and store on the mobile terminal the self-sufficient content necessary for a given maintenance procedure according to the skill level of the operator. For certain aeronautical checks we can imagine, for example, a choice between 2D diagrams and parts

of 3D models. According to his level of expertise, the experienced operator may not exhaustively browse the documentation. He may only need a task's synopsis, along with update information since the last access. On the contrary, a novice operator would need complete information. Therefore our approach of adaptation consists in observing the users' behaviours for learning how to select the content and thus minimize the number of connexion requests (see figure 5).

3.4.2 Browsing News on Demand

The navigation inside NoD (News on Demand) sites on a mobile is tricky : very often the user needs to scroll down the screen (containing images) many times, looking for the news he is interested in (figure 1(a)). In order to adapt the access, a recommendation system [25] can use RL techniques and propose to the user *by default* choices or choices sorted according to the preferences inferred from previously expressed choices (this is called "sequence mining"). We have already given some hints (see section 3.1) for exploiting these preferences concurrently with managing the limited resources of the mobile terminal. The learning of *by default* choices can be simply reinforced by minimizing the number of times it is challenged by the choice the user makes (see figures 1 and 5).

3.4.3 Reducing latencies in hypermedia navigation

Hypermedia access from a mobile platform brings out the problem of latency reduction. In fact, following a link to a heavy content (a streaming audio/video sequence, for example), requires to download a playout buffer before the stream can play. This buffering process results in important delays, usually as long as a few seconds. The inconvenience caused to the user is all the more so in that, during the navigation (a sequence of clicks), the user experiences the sum of these individual delays. The application could therefore make an effort to adapt the access by setting up a prefetching system that would reduce these delays. The streaming becomes adaptive. In this case the agent has to decide what are the best prefetching actions for minimizing the latencies (see figure 5).

4 Markov Decision Processes and Reinforcement Learning

In the previous section, we have shown that reinforcement learning is a relevant framework to tackle dynamic adaptation problems.

In this section, we introduce the formalization of reinforcement learning with Markov Decision Processes (MDP).

4.1 The Markov property

In the reinforcement learning (RL) framework the agent makes its decision by applying a policy. A policy is a function that takes a signal from the environment as input and outputs a decided action. Figure 3 shows such a function of the environment's state in the RL closed-loop.

Since the decision are taken sequentially, we would like an input signal that summerizes past events compactly and retains only relevant information for decision-taking. Usually this requires more than instantaneous measurements but never more than the complete history of visited states.

According to R.S. Sutton, "a state signal that succeeds in retaining all relevant information have the Markov property" (see also the formal definition in [31] p.63).

If an environment has this property, it has a one-step dynamic : the next state and the next reward can be predicted only from the current state and the current action.

A reinforcement learning process that satisfies the Markov Property is called MDP (Markov Decision Process).

4.2 MDP definition

A MDP is a stochastic controlled process that assigns rewards to transitions between states [34]. It is defined as a quintuple $(S; A; T; p_t; r_t)$ where S is the state space, A is the action space, T is the discrete temporal axis of instants when actions are taken, $p_t()$ are the probability distributions of the transitions between states and $r_t()$ is a function of reward on the transitions. We rediscover in a formal way the ingredients necessary to understand the figure 3 : at each instant T, the agent observe his state $\sigma \in S$, apply on the system the action $a \in A$ that brings the system (randomly, according to $p(\sigma'|\sigma, a)$) to a new state σ', and receives a reward $r_t(\sigma, a)$.

As we have already mentioned it, we are looking for the best policy, that is a function π that associates an action $a \in A$ to each state $\sigma \in S$.

The MDP theoretical framework assigns to each policy π a *value function* V_π

that associates to each state $\sigma \in S$ a global reward $V_\pi(\sigma)$, obtained by applying π beginning with σ. Such a value function allows us to compare policies.

A policy π outperforms another policy π' if

$$\forall \sigma \in S, \quad V_\pi(\sigma) \geq V_{\pi'}(\sigma)$$

The expected sum of rewards is weighted by a parameter γ in order to limit the influence of infinitely distant rewards (especially in the case when S is infinite) :

$$\forall \sigma \in S, \quad V_\pi(\sigma) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = \sigma\right]$$

In brief, for each state, this value function gives the expected sum of future rewards that can be obtained if the policy π is applied from this state on.

This value function allows to formalize the research of the optimal policy π^* : it is the one associated to the best value function $V^* = V_{\pi^*}$.

Bellman's optimality equations ([35]) characterize the optimal value function V^* and an optimal policy π^* that can be obtained from it. In the case of the γ-weighted criterion, they can be written :

$$V^*(\sigma) = \max_{a \in A} \left(r(\sigma, a) + \gamma \sum_{\sigma' \in S} p(\sigma'|\sigma, a) V^*(\sigma') \right)$$

$$\forall \sigma \in S, \quad \pi^*(\sigma) = \operatorname*{argmax}_{a \in A} \left(r(\sigma, a) + \gamma \sum_{\sigma' \in S} p(\sigma'|\sigma, a) V^*(\sigma') \right)$$

4.3 Resolution and reinforcement learning

To solve these equations, we can notice that the L operator that verifies $V_{n+1} = LV_n$ in :

$$\forall \sigma, \quad V_{n+1}(\sigma) = \max_a \left(r(\sigma, a) + \gamma \sum_{\sigma'} p(\sigma'|\sigma, a) V_n(\sigma') \right)$$

is a contraction. We could therefore solve Bellman's equations in $V^*(\sigma)$ by using an iterative fixed-point method that computes a sequence $(V_n)_n$. We randomly choose V_0 and then iteratively apply the L operator. Every iteration

makes an improvement on the current policy, associated to V_n. If the rewards are bounded, the sequence converges to V^*, from which π^* can be computed [34].

The reinforcement learning approach aims at finding an optimal policy through iterative estimations of the optimal value function. Today, reinforcement learning is one of the main methods able to solve sequential decision problems for which the transitions' probabilities are not known (lack of "model") or for MDPs with a very large state space. There are a few algorithms for reinforcement learning, such as *Q-learning*, *R-learning*, *Sarsa* etc. [31]. One way to get around the lack of model is to use an indirect method and estimate the transitions' probabilities from simulations. The main difficulty is to decide the moment when these probabilities are sufficiently reliable to classically solve the problem. On the contrary, a direct method only tries to estimate the value function and not bother with the probabilities.

In our work we will choose the latter method and use the most famous (but also the simplest) algorithm : *Q-learning* [36]. The *Q-learning* algorithm is a reinforcement learning method that is able to solve Bellman's equations for the γ-weighted criterion. It uses simulations to iteratively estimate the value function V^*, based on the observations of instantaneous transitions and their associated reward. For this purpose Watkins [34] introduced a function Q, that carries a significance similar to that of V but makes it easier to extract the associated policy: there is no need to have the transitions' probabilities of the Markov model any more.

To a given policy π and its value function V_π we associate the new function, called "Q-value" :

$$\forall \sigma \in S, a \in A, \quad Q_\pi(\sigma, a) = r(\sigma, a) + \gamma \sum_{\sigma'} p(\sigma'|\sigma, a) V_\pi(\sigma')$$

Now it is easy to see that, in spite of the lack of transitions' probabilities, we can easily "trace back" to the optimal policy :

$$\forall \sigma \in S, \quad V^*(\sigma) = \max_a Q^*(\sigma, a) \quad \pi^*(\sigma) = \operatorname*{argmax}_a Q^*(\sigma, a)$$

The principle of the *Q-learning* algorithm (figure 6) is the following: after each observed transition $(\sigma_n, a_n, \sigma_{n+1}, r_n)$ the current value function Q_n for the couple (σ_n, a_n) is updated, where σ_n represents the current state, a_n the chosen and applied action, σ_{n+1} the resulted state and r_n the immediate reward.

In this algorithm, N_{tot} is an initial parameter that represents the number of iterations. The *learning rate* $\alpha_n(\sigma, a)$ is particular to each pair state-action,

Initialize Q_0

for $n = 0$ **to** $N_{\text{tot}} - 1$ **do**

$\sigma_n =$`chooseState`

$a_n =$`chooseAction`

$(\sigma'_n, r_n) =$`simulate`(σ_n, a_n)

`/*` update Q_{n+1} `*/`

$Q_{n+1} \leftarrow Q_n$

$d_n = r_n + (\gamma \max_b Q_n(\sigma'_n, b)) - Q_n(\sigma_n, a_n)$

$Q_{n+1}(\sigma_n, a_n) \leftarrow Q_n(\sigma_n, a_n) + \alpha_n(\sigma_n, a_n) d_n$

end for

return $Q_{N_{\text{tot}}}$

Figure 6. The *Q-learning* algorithm.

and decreases toward 0 at each iteration. The function `simulate` returns a new state and its associated reward according to the dynamics of the system. The choice of the current state and of the action to execute is made by the functions `chooseState` and `chooseAction`. The function `initialize` is used most of the time to initialize the values Q_0 to 0.

The convergence of this algorithm has been thoroughly studied and is now well established. Assuming that S and A are finite, that each pair (σ, a) is visited a infinite number of times, that $\sum_n \alpha_n(\sigma, a) = \infty$, that $\sum_n \alpha_n(\sigma, a)^2 < \infty$, that $\gamma < 1$ or if $\gamma = 1$ and finally that for each policy there is an absorbent state of null reward, then the function Q_n converges almost surely to Q^*. Let us recall that the almost-sure convergence means that $\forall \sigma, a$, the sequence $Q_n(\sigma, a)$ converges to $Q^*(\sigma, a)$ with a probability equal to 1.

Practically, the sequence $\alpha_n(\sigma, a)$ is often defined as following ($n_{\sigma,a}$ represents the number of times the state σ was visited and the decision a was made) :

$$\alpha_n(\sigma, a) = \frac{1}{n_{\sigma,a}}$$

5 Adapting the streaming by prefetching

In this section we are going to apply our approach to dynamic adaptation by reinforcement learning. We confront our framework with a concrete problem.

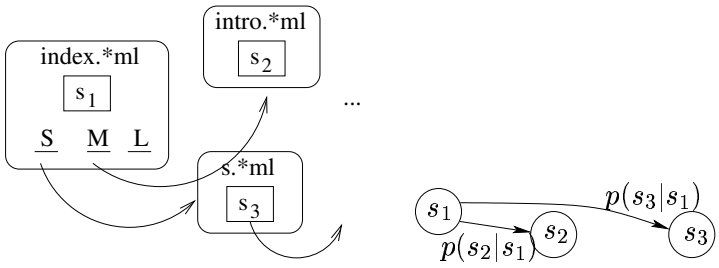

Figure 7. A navigation graph example.

Among the use cases discussed in section 3.4, we develop the last example that deals with reducing the latencies in hypermedia navigation (see section 3.4.3). First we define the behaviour of our agent (the prefetching policy) and then consider it as a component of an adaptive streaming platform.

5.1 Prefetching policies

In order to introduce prefetching policies, let us consider an elementary hypermedia document composed of three pages, each containing a heavy, latency-inducing video segment s_i. The temporal scenario associated to the hypermedia navigation is presented in the figure 7.

While the user plays the media object (namely the video segment s_1), he can either follow a link to the second media object or access the third media object.

In order to take into account the users' behaviours in this simple case, we can associate to the links $s_1 \rightarrow s_2$ and $s_1 \rightarrow s_3$ the probabilities $p(s_2|s_1)$ and $p(s_3|s_1)$ respectively, to go to s_2 when he is in s_1 and to go to s_3 from s_1, respectively. Obviously, here we have $p(s_2|s_1) + p(s_3|s_1) = 1$.

While playing a media object, certain conditions may justify the use of prefetching : if the user pauses the media or if the available bandwidth is underused. In such conditions, in order to try to reduce future latencies, a prefetching agent may decide to prefetch another media object in background. These decisions that are made in sequence during a navigation make up a so-called prefetching policy.

Simple heuristic policies can be considered [37]. For example, a policy that we call "best-first" aims at prefetching the immediately following media object that has the highest access probability : in our example, it is the prefetching of s_2 while playing s_1 if $p(s_2|s_1) > p(s_3|s_1)$.

Another heuristic, static policy called "proportional", may be used : it means the prefetching of the two media objects s_2, s_3 using bitrates that are proportional with their transitions' probabilities.

In general, if the user really wants to play objects that have already been prefetched and the prefetching agent has had enough resources (time and bandwidth), the startup delay will be at its minimum. Most of the time, prefetching the media prefix (playout buffer) only is enough to reach this minimum [9].

5.2 Drawbacks of static policies

A question arises naturally : how to choose dynamically the best policy, that is the one that reduces the latencies at their minimum ?

In section 2 we showed that the literature often gives insufficient solutions such as static or heuristic prefetching policies. For example, at the authoring stage of the hypermedia document, SMIL's *prefetch* tag can be used to declare invariant prefetching orders that should be carried out in parallel with the streaming of other objects (expressed by the $< par >$ tag).

The heuristics solutions that we have given in the previous section are not satisfactory either. On one hand, they do not take into account the possibility of prefetching objects that are not immediately accessible from the current state. This restriction prevents us from finding an optimal decision : we may need to prefetch very early a heavy object that is located at a certain depth in the navigation graph. On the other hand, these heuristics do not consider the variability of the environment. Even in our elementary scenario we need to handle the unpredictable user behaviour and the random performance of the network. Naturally, the quality(performance) of a prefetching decision depends on the time (unknown) that the user spends playing s_1 before accessing the following objects, and on the available bandwidth. Even worse, this performance can influence on the following decisions : if a media object has completed prefetching, we may think of prefetching another object, else the best decision is to carry on prefetching the former...

These various arguments clearly indicate the need for learning the best adaptation strategies (prefetching in our case) rather then statically choose them.

5.3 An agent-based prefetching architecture

A streaming adaptation system by prefetching (figure 8) can thus use an assistant located on a proxy (the access point of the mobile terminal, for example). This is generally the most natural location in wireless environments [14,15]. On one hand, the access point is on the dividing line between the WAN and the mobile network. On the other hand, the proxy can monitor the optimal

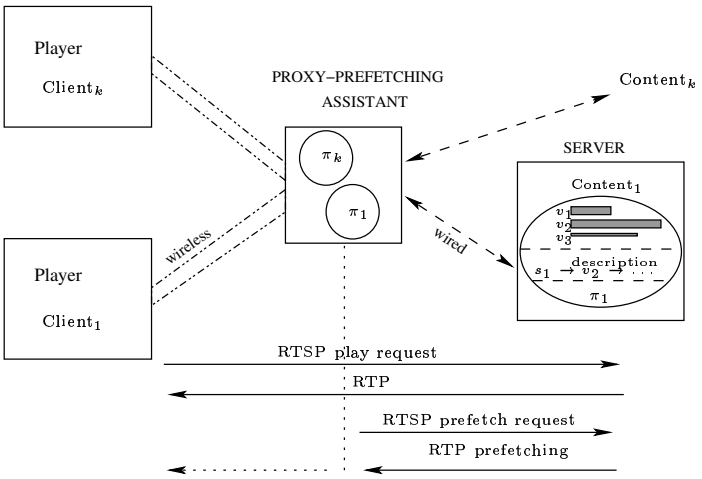

Figure 8. Adaptive streaming and prefetching

bandwidth allocation between ongoing wireless connections that effectively share the downlink.

The hypermedia content is enriched with a prefetching policy (noted π_1 for the content 1 of the figure 8). The assistant manages the streaming using the policies π_1, π_k, transferred when the content is accessed (this is an adaptation approach that uses mobile adaptation agent).

Two types of streams are simultaneously managed for both control (RTSP commands) and media delivery (RTP packetized streams): the currently requested streams and the prefetching streams. The scenario is as following :

- the client requests for the content (RTSP commands) ,
- the media data, the document description and the policy are sent back by the server ,
- the proxy intercepts the policy ,
- the proxy applies it by issuing RTSP commands for background streaming of prefetching streams ,
- these streams can be buffered at the proxy but also at the client, since the terminals have increasing capabilities (computational power, storage and bandwidth) ,
- subsequent prefetching commands are synchronized with the user's clicks along the navigation path.
- finally, feedback is used for updating the policy according to the reinforcement learning paradigm.

To summarize, our prefetching agent integrates well within our reinforcement learning paradigm. It takes decisions "in sequence": these decisions are evaluated, reinforced or challenged at each click that makes the transition from one content to another. Moreover, the decisional environment is uncertain. There are two major sources of uncertainty: the user and the network. Finally, the

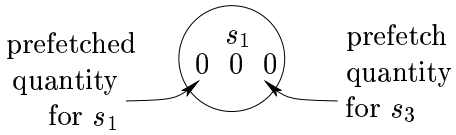

Figure 9. The initial buffer state

agent receives cumulative rewards simply derived from cumulative latencies.

6 Adaptive and optimal prefetching policies

In this section we show that our reinforcement learning solution to prefetching can be modeled as a Markov Decision Process. Since a reinforcement learning task that satisfies the Markov property becomes a MDP (see section 4.1), we first propose a relevant decisional state for our problem, then we derive optimal prefetching policies from our MDP. We validate these policies by presenting experimental results.

6.1 Well tailored Markov states

We now introduce a decisional state called *buffer state* (BS) that will be the basis of our MDP model [33] and that has the Markov property. Such a state is associated to a media object called s_k and to the buffering levels for each media object. The levels depends on the previously taken prefetching decisions. We formally express a buffer state as $\sigma = (s_k, r_1, ..., r_n)$, where n is the number of media objects and r_i is the filling level of the playout buffer b_i (in KB) of s_i. The playout buffer contains the prefix of the stream [1].

With these notations, the behaviour of the system is the following: in our example (figure 7), while the video segment s_1 is played, we are in a state that integrates the fact that s_1 is played and that nothing had been prefetched (the three buffers are empty).

Figure 9 is a graphical representation of our initial buffer state $\sigma_0 = (s_1, 0, 0, 0)$. Before presenting the next buffer states, let us introduce both actions and transitions.

While entering σ_0, the client makes the following actions :

- It downloads the necessary playout buffer in order to be able to start playing s_1 (a quantity called b_1). If bw is the average bandwidth, this will imply a

[1] This model can be quite directly extended to integrate streams playing simultaneously. In this case, s_k stands for their union and b_k sums their prefixes

latency of b_1/bw.

- It also decides if, during the presentation of the current segment s_1, it will prefetch other video segments. In our example, the possible actions are :$a_2 =$ prefetch s_2 or $a_3 =$ prefetch s_3 .

The maximum duration of s_1 is d_1 and we consider that the user clicks on a link at $d_1/2$. He chooses either s_2 or s_3. Therefore we obtain the transition graph of figure 10, where $p(\sigma'|\sigma, a)$ is the probability to go to σ' when we are in σ, knowing that we have chosen the action a while entering in σ.

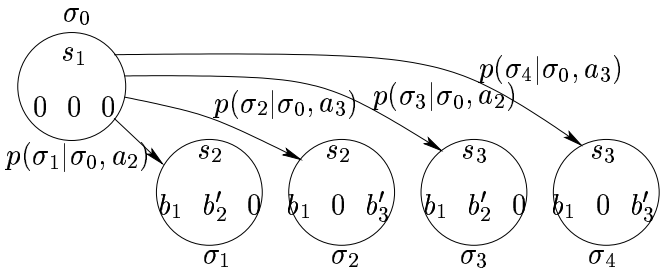

Figure 10. Some transitions showing the effects of user and prefetching actions

Assuming that :

- br_i is the bitrate of s_i,
- $(\text{bw} - \text{br}_i)$ is the amount of bandwidth available for prefetching

the formula for calculating b'_i is simple : $b'_i = \min\{b_i, (\text{bw} - \text{br}_i) \times d_1/2\}$. The 4 possibilities are expressed by the transitions : we have prefetched wrongly or rightly s_2 and s_3 respectively, and s_1 has played (its playout buffer b_1 is full).

At this point, our model is able to capture in a buffer state the effect of all previous actions and of all the previous transitions. We can now introduce the two sources of uncertainty previously cited : the random variations of the transition moments and the variable bandwidth. The main idea is to multiply the number of target states such that the random conditions are taken into account. These conditions influence heavily on the performance of the prefetching actions, that is the buffers' filling levels. In order to do this, the filling level of each playout buffer is coded with an integer between 0 and B.

If we consider the link $s_1 \rightarrow s_2$ and the action a_3 (prefetch s_3 while s_1

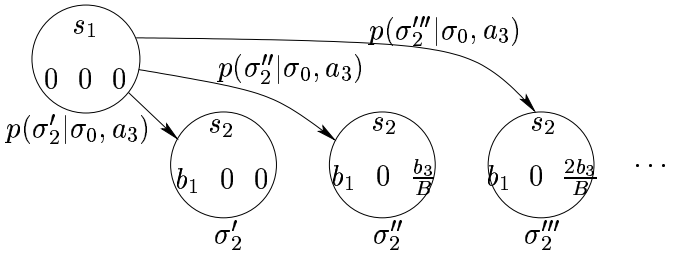

Figure 11. Variable prefetching effectiveness

is playing), then we can go to $B+1$ different states for which a fraction $0, 1/B, 2/B, \ldots, b/B, \ldots 1$ of b_3 is prefetched (figure 11). There is an unknown probability that is associated to each transition between σ_0 and these new buffer states $(\sigma_2', \sigma_2'', \sigma_2''')$ which refine σ_2.

This model indicates (in an approximate manner, since it uses a quantization on $B+1$ levels), the quantity of data that has been prefetched.

6.2 A formal MDP

The prefetching of the latency-related components of a hypermedia content can be viewed as a *sequential decision problem under uncertainty* in which we look for an optimal policy :

- the decisions have to be taken sequentially ; the decision refers to the choice of a video segment to be prefetched. We only deal here with simple actions, where there is only one segment prefetched at any moment. We could deal with more sophisticated policies in the same way : proportional policies (prefetch *simultaneously* s_i and s_j using bandwidths proportional to their probabilities), sequential policies (prefetch s_i and *then* s_j if time left).
- the consequences of a decision are not completely known, because of the uncertainty linked to the users' behaviour and to the network conditions : a good prefetching decision can turn out to be ineffective if the user does not provide enough time for the prefetching to operate (it clicks rapidly) or if the network does not provide the necessary bandwidth.

For our problem :

- S is the set of buffer states ;
- A contains the possible prefetching actions ;
- T can be represented as $\mathbb{N}$: the decision moments are the clicks on the navigation path ;
- the transitions' probabilities have been defined in the introductory example : they concern the transitions between two buffer states under hazards due to network and users ;
- the reward $r(\sigma, a)$ is defined as the reduction of latency compared to the policy that does no prefetching at all. For a state $\sigma = (s_k, r_1, ..., r_n)$, the reward is proportional with the filling level r_k.

By solving the MDP with a reinforcement-learning algorithm such as Q-learning, we can obtain optimal prefetching policies. Applying Q-learning is relatively simple. In order to iterate with Q-learning (figure 6), we need to generate both possible navigation paths and realistic network conditions. These data are usually obtained by simulation and/or by collection of logs. Therefore

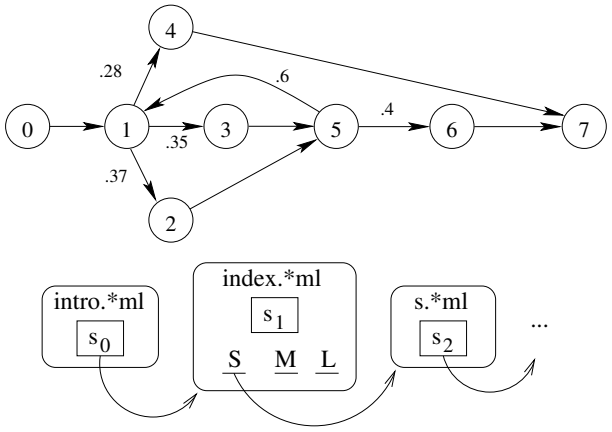

Figure 12. Another navigation graph example.

the *simulate* function takes as input and returns possible/realistic buffer states only (σ_n, σ'_n). In our implementation, we build many buffer-state chains from an initial buffer state and the *simulate* function. Prefetching actions are chosen randomly (*chooseAction* function). By contrast, the *chooseState* function does not return a random state since it makes chaining possible. Among the important number of theoretical buffer states, this protocol ensures that the algorithms only visits a subset that corresponds to realistic conditions. This guided iterative process converges quite rapidly and allows us to compute the optimal action for each buffer state.

This optimal property is surprisingly strong : regardless of wether the previous actions were right or wrong (that is, the prefetching was effective or not), we always obtain the best action, according to the current buffer state. In that sense, the optimality of our solution compares favourably with other heuristic prefetching policies, mentioned in section 2.

6.3 Experiments

In order to validate our MDP model we conducted experiments in two steps. Firstly, we estimate theoretical latencies' reduction by using simulation. Secondly, we compare these latencies with real latencies, as measured by our streaming platform. In both cases numerous navigations are simulated based on given access logs.

It is relatively easy to obtain theoretical values for latencies measured with or without prefetching. In order to illustrate this, let us take the example of a navigation graph such as the one of the figure 12. The transitions' probabilities summarize here some elementary information on the users' behaviour and are extracted from access logs. Such a probabilistic graph suffice to generate thousands of realistic navigations by simulation. We have 8 video segments that

segment	filename	duration (s)	fps	bitrate (KBps)	prefix (s) (nb. of images, size (KB))
0	`m8ah.mov`	1	4	8	1 (4, 8)
1	`m4ah.mov`	4	4	4	1 (4, 4)
2	`m12ah.mov`	1	4	12	1 (4, 12)
3	`m12ah.mov`	1	4	12	1 (4, 12)
4	`m12ah.mov`	4	4	12	4 (16, 48)
5	`m12ah.mov`	6	4	12	4 (16, 48)
6	`m12ah.mov`	1	4	12	1 (4, 12)
7	`m12ah.mov`	1	4	12	1 (4, 12)

Table 1
Characteristics of the segments of figure 12

are the media objects responsible for important latencies. Their characteristics are given in the table 1. One can notice that the durations are kept artificially low in order to limit the (real) navigation duration in (real) experiments.

We choose a set of highly variable experimental conditions :

- the bandwidth available between the content server and the playing client varies between 96 kbps and 128 kbps (following a uniform distribution) ;
- the path length varies between 4 and 10 clicks : the user is presented with a few choices while visiting the segments 1 and 5. He can come back from the segment 5 to the segment 1. Therefore there are cycles ;
- click times vary around the following average values :$\overline{t_{\text{clic1}\to 2}} = \overline{t_{\text{clic1}\to 4}} = 3$ s et $\overline{t_{\text{clic5}\to 1}} = 5$ s .

The overall learning process is done in two passes : off-line production of optimal policies that can be further refined on-line.

By off-line simulation of tens of thousands navigations we are able to have *Q-learning* converge rapidly. Naturally we use a sparse matrix for storing the few thousands non-null Q-values, since not every buffer state is reachable. The convergence and the number of reachable states are hardly influenced by the buffer granularity B, used for the quantization of the buffer [33].

Later the policies produced off-line can be used and then updated to maintain their optimality during the navigations of real users.

Now we have an optimal policy and can therefore compare it with two other policies : one that is heuristically chosen (*best-first*, see section 5.1) and one that does no prefetching. These theoretical comparisons are given in the left-

side of the table 2 (simulation column). For each policy/row, we give the average latency and its dispersion. For simulation, the average figures are computed over a set of 10000 navigations, returning results which are statistically significant. We can notice that the optimal and best-first policies reduce the expected average latency by more that 55% and 35%, respectively.

The experimental framework that is most suitable for validating our model is represented by the figure 13. A client and a server are connected via a link whose characteristics match those used during the learning step (bandwidth varying between 96 and 112 kbps and with an average of 112 kbps). The hypermedia content (HV) is stored on the server. This content includes the video segments s_i, the description of the hypermedia structure and the prefetching policy π. The policy manager is located on the client (streaming video player). Through a set of buffers, the client also handles the buffering for the prefetched streams.

The client-server interactions are represented at the bottom of figure 13. In this example, two video segments s_1 and s_2 with the same bitrate (96 kbps) are visited in sequence. The policy indicates the prefetching of s_2 while playing s_1. The residual bandwidth, of approximately the difference between 112 kbps and 96 kbps, implies a prefetching request of approximately 16 kbps. The expected latency reduction is obtained at the next click $s_1 \rightarrow s_2$. This event lets the client generate further requests to the server. At the end of the navigation, the client transfers back to the server the information needed for updating the policy. Due to technical constraints, we set up a more complex implementation of this streaming architecture. We built the player using LiveMedia (an open-source API for RTSP/RTP streaming) and used the Darwin open-source streaming server. Our experimental platform is called HyVideoSim.

This platform allowed us to compare the predicted theoretical latencies and the real latencies (measured with HyVideoSim).

Table 2 compares theoretical (simulation column) and measured latencies (experiments). For simulation, the average figures were computed over a set of 10000 navigations. Unfortunately, we cannot perform real experiments in such large numbers, since each navigation lasts its real duration (tens of seconds). This is the reason for which the experimental results are computed only over 300 navigations.

From simulation to real experiments, we keep a good performance improvement while using optimal prefetching policies, even if theoretical performance is not fully achieved. This is mainly due to experimentally observed situations that were not sufficiently explored while Q-learning. To overcome this limitation we should improve our simulation process and bring it closer to real conditions (e.g. use a better bandwidth distribution).

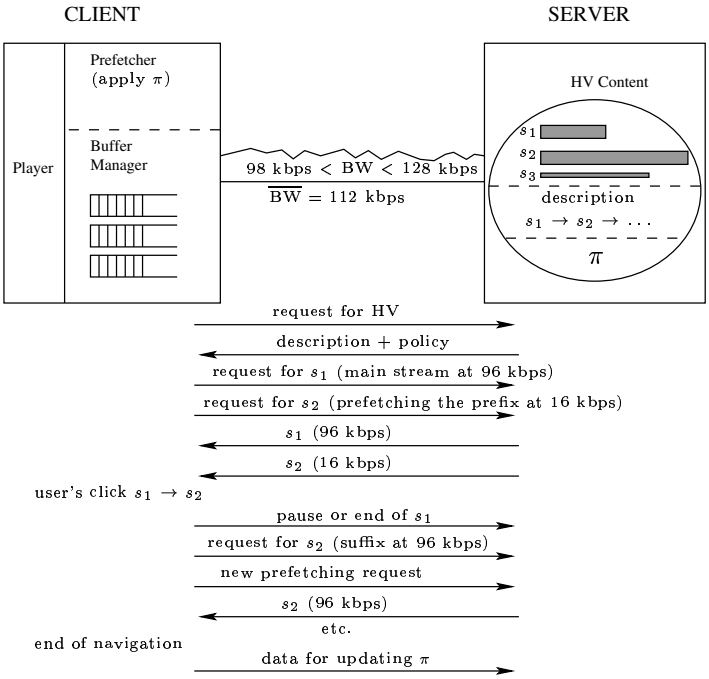

Figure 13. Operation of our client-server experimental platform (HyVideoSim)

Figure 14 reports the same good results. The histograms can be used to compare the distribution of the theoretical and HyVideoSim-measured latencies. The results are coherent and allow us to clearly see the gradual and significant improvements in latency reductions throughout the three policies. The multimodal distribution (obvious if no prefetching) is still visible with prefetching, but is translated and spread out due to users (click times) and network (bandwidth) variations.

If we use a magnifying glass to have a look at some paths (table 3), we can also estimate the cost of prefetching (the decomposition of used bandwidth). As previously stated, these average results correspond to 10000 simulated navigations. The average cumulated latencies for the 2755 simulated navigations for the path $0-1-4-7$ are approximately 7 seconds without prefetching, 5 seconds for the *best-first* policy (a reduction of 28%) and 2 seconds for the optimal policy (a reduction of 70% in this very favorable case) ! On the contrary, latency reductions for another path $(0-1-2-5-6-7)$ are less important for both best-first and optimal policies. Obviously, the performance depends largely on the path since navigation paths are more or less able to benefit from prefetching.

policy	simulation (theoretical model)				experiments (HyVideoSim)			
	min	max	ave	σ	min	max	ave	σ
no pref.	7	13	8.158	1.526	7.058	13.315	8.383	1.611
π_{bf}	3.666	8.997	5.219	1.166	4.689	10.814	5.969	1.185
π^*	2	5.637	3.614	0.990	2.103	7.386	4.680	1.635

Table 2
Comparison of experiments and simulations on the graph of the figure 12.

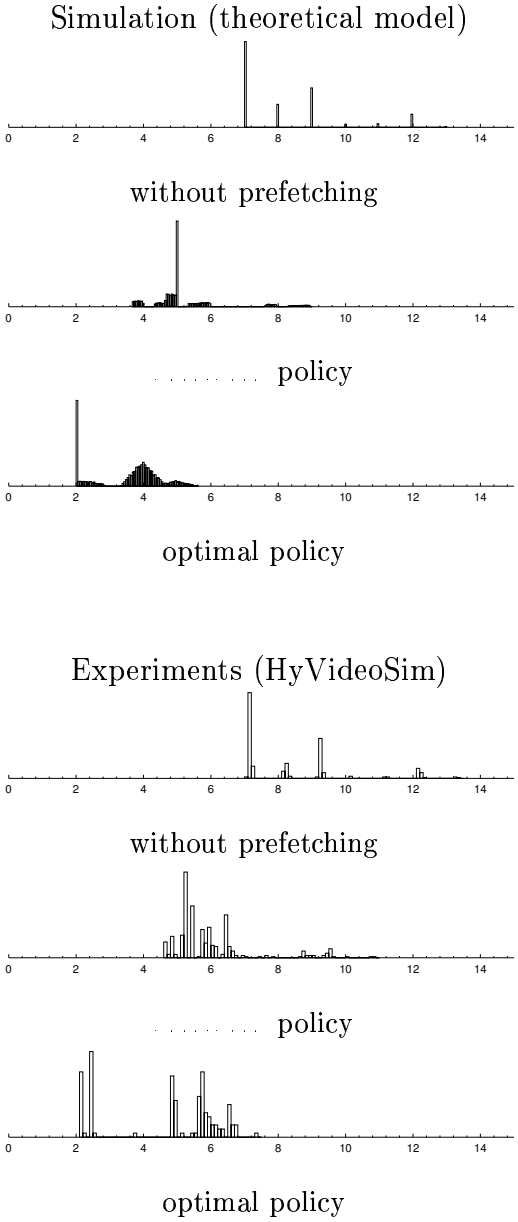

Figure 14. Comparison of the distributions of theoretical and experimental latencies (latencies in seconds on abscissa)

path (nb of occurrences)	policy	playing time (s)	latency (s)	reduction ratio	bandwidth (Bps)		
					used	incl.pref.	avail.
0-1-4-7	no pref.	9.591	7.000	0 %	5186	0	14358
(2755)	π_{bf}		5.000	28.57 %	6739	1966	14358
	π^*		2.156	69.20 %	9068	6814	14364
0-1-2-5-6-7	no pref.	12.141	9.000	0 %	6246	0	14346
(1066)	π_{bf}		4.829	46.34 %	7782	2538	14347
	π^*		3.980	55.78 %	10247	5884	14353

Table 3
Statistics for two paths of the graph of the figure 12

7 Conclusion

This paper presented an original approach for learning and dynamically updating multimedia adaptation strategies. The originality of this approach lies in:

- the observation of users' behaviours in order to better handle limited resources,
- the inclusion of the adaptation agent in the content,
- the suitability of reinforcement learning for designing dynamic adaptation strategies.

Our framework allowed us to use measurable yet stochastic performance criteria and thus handle naturally various sources of uncertainty and variability. We proposed the adaptation of ubiquitous streaming by means of prefetching as a case study. We have also shown how a Markov Decision Process, that we were able to solve using a reinforcement learning algorithm, allows us to identify optimal prefetching policies. The results from experiments validate our approach.

The implications of this work are numerous. In the field of hypermedia navigation, the observation of users' behaviours can be used to build MDP models for automatically mining content. By identifying navigation sequences used frequently it is possible to distinguish among the important parts of a multimedia document as well as to classify usage profiles (this is a typical approach of the user modeling community). There could also be other applications for our adaptation platform. In ubiquitous multimedia, it could be interesting to design optimal stream transformation agents (located on proxies) or optimal caching agents. Furthermore, many adaptation problems of data access or in-

formation retrieval could benefit from the integration of a learning agent that improves with usage.

Finally, the good observed performance of our optimal prefetching policies encourages us to consider the other two case studies mentioned in this paper. The solution to these cases will certainly require more thorough semantic hints.

Vincent Charvillat received his MEng from ENSEEIHT and PhD in computer science from INPT (National Polytechnics Institute of Toulouse) in 1997. He is currently an associate professor in the Department of Computer Science and Applied Mathematics at INPT-ENSEEIHT. His research interests include image and video processing, hypermedia content management (interactive video, learning techniques for content adaptation). He is the leader of the SIGMA group (Special Interest Group on Multimedia Applications) at IRIT.

Romulus Grigoras received his engineering degree in computer science from the Polytechnics Institute of Bucharest. He obtained his PhD from INPT in 2003. In 2004 he joined ENSEEIHT as an associate professor in computer science. Dr. Grigoras is specialised in adaptive and distributed multimedia. His research interests include adaptive streaming for cooperative applications, streaming of large 3D environments and decision techniques for multimedia adaptation.

References

[1] C. Timmerer, H. Hellwagner, Interoperable adaptive multimedia communication, IEEE MultiMedia 12 (1) (2005) 74–79.

[2] M. Margaritidis, G. C. Polyzos, Adaptation techniques for ubiquitous internet multimedia, Journal on Wireless Communications and Mobile Computing 1 (2) (2001) 141–163.

[3] T. Lemlouma, N. Layaida, Media resources adaptation for limited devices, in: Proceedings of the Seventh International Conference on Electronic Publishing (ICCC/IFIP), 2003, pp. 209–218.

[4] M. K. Asadi, Multimedia content adaptation with mpeg-21, Ph.D. thesis, ENST Paris (June 2005).

[5] T. Lemlouma, N. Layaida, Encoding multimedia presentation for user preferences and limited environments, in: Proceedings of IEEE International Conference on Multimedia & Expo (ICME), 2003, pp. 165–168.

[6] M. K. Asadi, J.-C. Dufourd, Multimedia adaptation by transmoding in mpeg-21, in: Proceedings of the International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS'04), 2004.

[7] A. Divakaran, K. A. Peker, R. Radhakrishnan, Z. Xiong, R. Cabasson, Video Summarization Using MPEG-7 Motion Activity and Audio Descriptors in Video Mining, Kluwer Academic Publishers, 2003.

[8] B. Girod, M. Kalman, Y. J. Liang, R. Zhang, Advances in channel-adaptive video streaming, in: Proceedings of IEEE International Conference on Image Processing (ICIP), 2002, pp. 1–8.

[9] S. Sen, J. Rexford, D. F. Towsley, Proxy prefix caching for multimedia streams, in: Proceedings of INFOCOM (3), 1999, pp. 1310–1319.

[10] J. I. Khan, Q. Tao, Partial prefetch for faster surfing in composite hypermedia, in: Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS), 2001.

[11] S. Chen, H. Wang, X. Zhang, B. Shen, S. Wee, Segment-based proxy caching for internet streaming media delivery, IEEE Multimedia 12 (3) (2005) 59–67.

[12] T. Kim, M. H. Ammar, A comparison of layering and stream replication video multicast schemes, in: NOSSDAV '01: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video, ACM Press, New York, NY, USA, 2001, pp. 63–72.

[13] R. Rejaie, H. Yu, M. Handley, D. Estrin, Multimedia proxy caching mechanism for quality adaptive streaming applications in the internet, in: Proceedings of INFOCOM (2), 2000, pp. 980–989.

[14] F. H. P. Fitzek, M. Reisslein, A prefetching protocol for continuous media streaming in wireless environments, IEEE Journal on Selected Areas in Communications 19 (10) (2001) 2015–2028.

[15] F. Yu, Q. Zhang, W. Zhu, Y. Zhang, Qos-adaptive proxy-caching for multimedia streaming over the internet, IEEE Transactions on Circuits and Systems for Video Technology 13 (3) (2003) 257–269.

[16] L. Brunie, H. Kosch, A. Mostefaoui, Semantic based prefetching in news-on-demand video servers, Multimedia Tools and Applications Journal, Kluwer Publishing 18 (2) (2002) 159–179.

[17] H. Kosch, A. Moustefaoui, L. Bszrmnyi, L. Brunie, Heuristics for optimizing multi-clip queries in video databases, Multimedia Tools and Applications. Kluwer Press 22 (3) (2004) 253–262.

[18] P. M. Ruiz, J. Botia, A. Gomez-Skarmeta, Seamless multimedia communications in heterogeneous mobile access networks, in: Proceedings of Terena Networking Conference, 2004, pp. 36–44.

[19] G. Ghinea, G. Magoulas, Quality of service for perceptual considerations: an integrated perspective, in: Proceedings of IEEE International Conference on Multimedia & Expo (ICME), 2001, p. 146.

[20] R. T. Apteker, J. A. Fisher, V. S. Kisimov, H. Neishlos, Video acceptability and frame rate, IEEE MultiMedia 2 (3) (1995) 32–40.

[21] D. Wijesekera, J. Srivastava, A. Nerode, M. Foresti, Experimental evaluation of loss perception in continuous media, Multimedia Systems 7 (6) (1999) 486–499.

[22] S. R. Gulliver, T. Serif, G. Ghinea, Pervasive and standalone computing: the perceptual effects of variable multimedia quality, International Journal of Human-Computer Studies 60 (5-6) (2004) 640–665.

[23] P. Brusilovsky, Adaptive hypermedia, User Modeling and User-Adapted Interaction, Kluwer Academic Publishers 11 (1-2) (2001) .

[24] P. Brusilovsky, M. T. Maybury, From adaptive hypermedia to the adaptive web, Communications of the ACM 45 (5) (2002) 30–33.

[25] I. Zukerman, D. Albrecht, Predictive statistical models for user modeling, User Modeling and User-Adapted Interaction, Kluwer Academic Publishers 11 (1-2) (2001) 5–18.

[26] T. Syeda-Mahmood, Learning and tracking browsing behavior of users using hidden markov models, in: Proceedings of IBM Make It Easy Conference, 2001.

[27] P. D. Bra, N. Stash, Multimedia adaptation using AHA!, in: ED-MEDIA 2004 Conference, Lugano, Switzerland, 2004.

[28] H. Kosch, L. Boszormenyi, M. Doller, M. Libsie, P. Schojer, The life cycle of multimedia metadata, IEEE Multimedia 12 (1) (2005) 80–86.

[29] P. M. Ruiz, J. Botia, A. Gomez-Skarmeta, Providing qos through machine learning driven adaptive multimedia applications, IEEE Transactions on Systems, Man and Cybernetics 34 (3) (2004) 1398–1411.

[30] O. Layaida, S. B. Atallah, D. Hagimont, A framework for the dynamic configuration and reconfiguration of network-based media adaptation, Journal of Internet Technology, Special Issue on Real time media delivery over the Internet 5 (4) (2004) .

[31] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.

[32] T. Kunz, Mobile code middleware for mobile multimedia information access, ACM SIGMOBILE Mobile Computing and Communications Review archive 6 (4) (2002) 68–70.

[33] R. Grigoraş, Hypermedia stream management: optimized prefetching policies and causal ordering (in French), Ph.D. thesis, INP Toulouse, France (Dec. 2003).

[34] M. Puterman, Markov decision processes: discrete stochastic dynamic programming, 1st Edition, Wiley-Interscience, 1994.

[35] R. E. Bellman, Dynamic programming, Princeton University Press, 1957.

[36] D. Bertsekas, J. Tsitsikis, Neurodynamic programming, Athena Scientific, Belmond, Massachusetts, 1996.

[37] R. Grigoraş, V. Charvillat, M. Douze, Optimizing hypervideo navigation using a Markov decision process approach, in: Proceedings of ACM Multimedia, 2002.