ONLINE REINFORCEMENT LEARNING FOR MULTIMEDIA BUFFER CONTROL
Nicholas Mastronarde and Mihaela van der Schaar
Department of Electrical Engineering, UCLA
{nhmastro, mihaela}@ee.ucla.edu

ABSTRACT
We formulate the multimedia buffer control problem as a Markov decision process. Because the application’s rate-distortion-complexity behavior is unknown a priori, the optimal buffer control policy must be learned online. To this end, we adopt a low complexity reinforcement learning algorithm called Q-learning to learn the optimal control policy at run-time. We propose an accelerated Q-learning algorithm that exploits partial knowledge about the system’s dynamics in order to dramatically improve the performance. In our experiments, we show that the proposed application-aware reinforcement learning algorithm performs significantly better than existing application-independent reinforcement learning algorithms.

Index Terms— Multimedia buffer control, encoder complexity control, dynamic voltage scaling, Markov decision processes, reinforcement learning.

1. INTRODUCTION
Efficiently designing and implementing delay-sensitive multimedia applications on resource-constrained, heterogeneous devices and systems is challenging due to the real-time constraints, high workload complexity, and the time-varying environmental dynamics (e.g., video source characteristics, workload characteristics, number of running applications, memory/cache behavior, etc.). A promising technique to reduce the computational burden on multimedia enabled devices is to introduce a buffer that smoothes the multimedia workload, by slightly increasing the tolerable application delay [6]. This is similar to techniques for smoothing network traffic with a rate buffer [5].

In [5], the author’s show that dynamic programming (DP) can be used to achieve optimal buffered rate-control policies. However, the deployed DP techniques can be very complex, and they also require complete a priori knowledge about the stochastic buffer dynamics (e.g., arrival and departure rates). Unfortunately, this information is not known in real-time video processing and streaming scenarios because the rate-distortion-complexity behavior of multimedia applications depends on the time-varying multimedia source characteristics, which are not known a priori. Consequently, optimal buffer control policies must be learned online, at run-time.

One option for learning the policy online is to directly estimate the dynamics using statistical fitting [1] or maximum likelihood estimation [2], and then recompute the policy using DP. However, this solution does not address the problem of complexity. A second, low complexity, option based on supervised learning is proposed in [2]. Offline, they quantize every unknown probability parameter into NS samples (more samples yield more accurate results). For each quantized probability bin, they compute a corresponding policy. Then, online, they directly estimate the transition probability function and, based on this, interpolate the offline computed policies to determine the current policy. Unfortunately, this technique’s memory requirements do not scale well to complex problems with many unknown parameters.

In this paper, we propose to use long-term optimization techniques from the framework of Markov decision processes (MDPs) to formulate and solve the multimedia buffer control problem. Although we illustrate the proposed framework in the context of encoder complexity control, it can also be applied for rate-buffer control. We adopt a low complexity reinforcement learning (RL) technique called Q-learning to learn the optimal policy online [3]. After observing that Q-learning adapts too slowly (i.e., has high time-complexity), we propose a RL technique that exploits partial a priori knowledge about the system’s dynamics in order to accelerate the rate of learning and improve overall learning performance. Unlike existing RL techniques [3], which can only learn about previously visited state-action pairs, the proposed algorithm exploits our partial knowledge about the system’s dynamics in order to learn about multiple state-action pairs even before they have been visited.

The remainder of this paper is organized as follows. In Section 2, we present the system model. In Section 3, we formulate the buffer control problem as an MDP. In Section 4, we summarize a well-known RL algorithm called Q-learning and we propose an accelerated Q-learning algorithm. In Section 5, we present our experimental results and we conclude the paper in Section 6.

2. SYSTEM SPECIFICATION
We assume that the application can make rate-distortion-complexity tradeoffs by adapting its configuration and that the hardware is DVS-enabled (dynamic voltage scaling) such that it can make energy-delay tradeoffs by dynamically adapting the processor frequency. The system is illustrated in Figure 1.

We model the video source as a sequence of video data units (for example, video macroblocks, groups of macroblocks, or pictures), which arrive at a constant rate \( \eta \) (data units per second) into the application’s pre-encoding buffer. We assume that each data unit is encoded in each time slot. We interchangeably refer to the time slot during which the \( n \)th data unit is encoded as the \( n \)th “time slot” or “stage.”

![System diagram](image)

**Figure 1. System diagram.**

2.1 Dynamic voltage scaling model
Each data unit is processed at the current operating frequency, which is a member of the state set \( \mathcal{S}_f = \{ f : i = 1, \ldots, N_f \} \).

The operating frequency can be adapted from one time slot to the next by asserting a command in the action set \( \mathcal{A}_f = \{ u : u \in \mathcal{S}_f \} \). Similar to [2], we assume that there is delay associated with the operating frequency state transition. Specifically, given the current action \( u \), we assume that there is a
one stage transition delay such that

\[
p_f (f' | u) = \begin{cases} 
1, & \text{if } f' = u \\
0, & \text{otherwise,} 
\end{cases}
\]

(1)

where \( f' \) is the operating frequency in the next time slot. Note that, regardless of the operating frequency command, the current data unit is processed at the current operating frequency \( f \). We define the cost \( J_{HW} (f) = P (f) \) (watts) as the power dissipated at operating frequency \( f \in S_f \), where \( P (\cdot) \) is the system’s power-frequency function.

### 2.2 Application model

We classify each data unit as being one of \( N_z \) types in the state set \( S_z = \{ z_i : i = 1, \ldots, N_z \} \). In this paper, we assume that \( N_z = 3 \) because video streams are typically compressed into group of pictures structures containing intra-predicted (I), inter-predicted (P), and bi-directionally predicted (B) data units (e.g. MPEG-2, MPEG-4, and H.264/AVC); however, the set of data unit types can be further refined based on, for example, each frame’s activity level [1] [4]. It has been shown [4] that transitions among data unit types in an adaptive group of pictures structure can be modeled as a stationary Markov process with transition probabilities \( p_z (z' | z) \) where the probabilities depend on the ratio of I, P, and B data units, the source characteristics, and the condition used to decide when to code an I frame [4].

Each data unit can be coded using any one of \( N_h \) source coding parameter configurations (quantization parameter, motion vector search range, choice of motion estimation algorithm, etc.) in the action set \( A_h = \{ h_i : i = 1, \ldots, N_h \} \). Given the data unit type \( z \in S_z \), each configuration \( h \in A_h \) achieves different operating points in the rate-distortion-complexity space [1]. We let \( c(z, h) \) represent encoding complexity (cycles).

We penalize the application configuration by employing the Lagrangian cost measure \( J_{APP} (z, h) \) used in the H.264/AVC reference encoder for making rate-distortion optimal mode decisions.

### 2.3 Buffer Model

The application’s pre-encoding buffer has the state set \( S_q = \{ q : i = 0, \ldots, N_q \} \), where \( N_q \) is the maximum number of data units that can be stored in the buffer. The pre-encoding buffer’s state at stage \( n + 1 \) can be expressed recursively based on its state at stage \( n \) : i.e.,

\[
q_{n+1} = \min \left\{ \left[ q_n + \left[ t^n (z^n, h^n, f^n) \cdot \eta - 1 \right] ^+ \right] , N_q \right\}
\]

(2)

where \( q_0 = q_{\text{init}} \) and

\[
t^n (z^n, h^n, f^n) = c^n (z^n, h^n) / f^n \text{ (seconds)}
\]

(3)

is the \( n \)th data unit’s processing delay, which depends on its complexity \( c^n (z^n, h^n) \) and the processor’s operating frequency \( f^n \); \( \lfloor x \rfloor \) is the integer part of \( x \); and \( \lfloor x \rfloor ^+ = \max \{ x, 0 \} \).

Note that \( c(z, h) \) is an instance of the random variable \( C(z, h) \sim p_C (c | z, h) \). In (2), the \(-1\) indicates that the \( n \)th data unit departs the pre-encoding buffer after the encoding delay \( t (f^n, z^n, h^n) \) (seconds). Meanwhile, the number of data units that arrive in the pre-encoding buffer during the \( n \)th time slot,

\[
\tilde{t} (z^n, h^n, f^n) = \eta (C(z^n, h^n)) / f^n,
\]

is an instance of the random variable \( \tilde{C}(z^n, h^n, f^n) = \eta C(z^n, h^n) / f^n \) with distribution

\[
p_{\tilde{t}} (\tilde{t} | f, z, h) = (f / \eta) \cdot p_C (c / f | c, z, h)
\]

(4)

Based on (2) and (4), the pre-encoding buffer’s state transition can be modeled as a controllable Markov process with transition probabilities

\[
p_q (q^{n+1} | q^n, f^n, z^n, h^n)
\]

\[
= \begin{cases} 
\begin{align*}
p_{\tilde{t}} (\tilde{t} < 2 | f^n, z^n, h^n), & q^{n+1} = 0 \\
p_{\tilde{t}} (\tilde{t} \geq N_q - q^{n+1} + 1 | f^n, z^n, h^n), & q^{n+1} = N_q \\
p_{\tilde{t}} (\Omega \leq \tilde{t} < \Omega + 1 | f^n, z^n, h^n), & \text{otherwise,}
\end{align*}
\end{cases}
\]

(5)

where \( \Omega = q^{n+1} - q^n + 1 \) and

\[
p_q (q^{n+1} | q^n, f^n, z^n, h^n)
\]

is the probability that the queue’s occupancy transitions from \( q^n \) to \( q^{n+1} \) given the operating frequency \( f^n \), the data unit type \( z^n \), and the application configuration \( h^n \). Note that, from (2), \( q^{n+1} = q^n - 1 \), with equality when there are no data unit arrivals, i.e. \( \tilde{t} = 0 \).

We define a gain to non-linearly reward the system for maintaining queuing delays less than the maximum tolerable delay, thereby protecting against overflows that may result from sudden bursts in encoding delay. Formally, we define the gain as

\[
g([f, q, z], h) = 1 - \frac{(\tilde{t} (z, h, f) - 1)}{N_q^2}.
\]

(6)

The gain is near its maximum when the buffer is empty and is minimized when the buffer is full.

### 3. FORMULATION AS AN MDP

An MDP is a tuple \( \langle S, A, p, R, \gamma \rangle \), where \( S \) is a set of states, \( A \) is a set of actions, \( p (s' | s, a) \) is the probability of transitioning from state \( s \) to state \( s' \) after performing action \( a \), \( R (s, a) \) is the expected immediate reward for performing action \( a \) in state \( s \), and \( \gamma \in [0, 1) \) is a discount factor.

The action \( a = (h, u) \) includes the application configuration \( h \) and operating frequency command \( u \), and is a member of the action set \( A = A_h \times A_u \). The state \( s = (f, q, z) \), which includes the operating frequency \( f \), the buffer state \( q \), and the data type \( z \), is a member of the state set \( S = S_f \times S_q \times S_z \). Since \( f \), \( q \), and \( z \) are Markovian, the transition of state \( s \) is Markovian with transition probability

\[
p (s' | s, a) = p_f (f' | u) p_q (q' | q, f, z, h) p_z (z' | z).
\]

(7)

Lastly, the reward function is defined as

\[
R (s, a) = g (s, a) - \lambda J_{APP} (z, h) - \mu J_{HW} (f),
\]

(8)

where \( \lambda \) and \( \mu \) are positive weights.

Unlike existing buffer control solutions, which focus on optimizing the myopic (i.e. immediate) utility, the goal in the proposed framework is to find the optimal actions at each stage \( n \in \mathbb{N} \) that maximize the discounted sum of future rewards, i.e.

\[
\sum_{n=0}^{\infty} \gamma^n R (s^n, a^n | s^0),
\]

where the parameter \( \gamma \) (0 \leq \gamma < 1) is the “discount factor,” which defines the relative importance of present and future rewards, and \( s^0 \) is the initial
state. The optimal stationary policy $\pi^* : S \mapsto A$ maximizes the discounted sum of future rewards from each initial state $s^0 \in S$.

Throughout this paper, we will find it convenient to work with the optimal action-value function $Q^* : S \times A \mapsto \mathbb{R}$ [3], which satisfies the following Bellman optimal equation:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} p(s' | s,a) V^*(s'),$$

where $V^*(s) = \max_{a \in A} Q^*(s,a)$, $\forall s \in S$, is the optimal state-value function [3]. In words, $Q^*(s,a)$ is the expected sum of discounted rewards achievable by taking action $a$ in state $s$ and then following the optimal policy $\pi^* : S \mapsto A$ thereafter, where $\pi^*(s) = \arg \max_a Q^*(s,a)$, $\forall s \in S$.

### 4. LEARNING THE OPTIMAL POLICY

As we mentioned before, the multimedia system’s dynamics are unknown and non-stationary because the rate-distortion-complexity behavior of the application depends on the unknown video source characteristics. Therefore, the optimal action-value function $Q^*$ and the optimal policy $\pi^*$ must be learned online.

#### 4.1 Conventional Q-learning

Central to the conventional Q-learning algorithm is a simple update step performed at the end of each time slot based on the experience tuple (ET) $(s^n, a^n, r^n, s^{n+1})$:

$$\delta^n = [(r^n + \gamma \max_{a' \in A} Q^n(s^{n+1}, a')) - Q^n(s^n, a^n)],$$

$$Q^{n+1}(s^n, a^n) = Q^n(s^n, a^n) + \alpha^n \delta^n,$$

where $s^n$, $a^n$, and $r^n = g^n - \lambda J_{APP} - \mu J_{HW}$ are the state, performed action, and corresponding reward in time slot $n$, respectively; $s^{n+1}$ is the resulting state in time slot $n+1$; $a'$ is the greedy action (defined below) in state $s^{n+1}$; $\delta^n$ is the so-called temporal-difference (TD) error [3]; and, $\alpha^n \in [0,1]$ is a time-varying learning rate parameter. We note that the action-value function can be initialized arbitrarily at time $n = 0$.

During the learning process, to judiciously trade off exploration and exploitation [3], we use the $\epsilon$-greedy action selection method: i.e., with probability $1 - \epsilon$, take the greedy action that maximizes the action-value function, i.e. $a^* = \arg \max_a \{Q(s,a)\}$; and, with probability $\epsilon$, take an action randomly and uniformly over the action set.

#### 4.2 Accelerated learning using virtual experience

The large number of buffer states significantly limits the system’s learning speed because the Q-learning update step defined in (11) updates the action-value function for only one state-action pair in each time slot. Several existing variants of Q-learning adapt the action-value function for multiple state-action pairs in each time slot. These include model-free temporal-difference-\lambda updates, and model-based algorithms such as Dyna and prioritized sweeping [3]; however, these existing solutions are not system specific and assume no a priori knowledge of the problem’s structure. Consequently, they can only update previously visited state-action pairs, leaving the learning algorithm to act blindly (or with very little information) the first few times that it visits each state, thereby resulting in suboptimal learning performance.

In this subsection, we propose a new Q-learning variant that exploits the form of the transition probability and reward functions (defined in Section 3) in order to update multiple statistically equivalent state-action pairs in each time slot, including those that have never been visited. (We say that a state-action pair $(\bar{s},\bar{a})$ is statistically equivalent to the pair $(s,a)$ if $p(s' | s,a) = p(s' | \bar{s},\bar{a})$ and $R(s,a)$ can be determined from $R(\bar{s},\bar{a})$.) In our specific setting, we exploit the fact that the data unit arrival distribution $p_T(\bar{t} | f,z,h)$ (defined in (4)) is conditionally independent of the current buffer state $q$ (given $f$, $z$, and $h$). This allows us to extrapolate the experience obtained in each time slot to other buffer states and action pairs. Thus, in stationary (non-stationary) environments, the proposed algorithm improves convergence time (adaptation speed) at the expense of increased computational complexity.

Let $\boldsymbol{\sigma} = (s,a,r,s')$ represent the current ET. Given the buffer state $q$ and the number of data unit arrivals $\bar{t} = \bar{t}(z,h,f)$, the next buffer state $q'$ can be easily determined from (2). By exploiting our partial knowledge about the system’s dynamics, we can use the statistical information provided by $[\bar{E}^n]$ to generate virtual experience tuples (virtual ETs) that are statistically equivalent to the actual ET. This statistical equivalence allows us to perform the Q-learning update step for the virtual ETs using information provided by the actual ET.

We let $\bar{\sigma}^n = (\bar{s},\bar{a},\bar{r},\bar{s}') \in \Sigma(\sigma^n)$ represent one virtual ET in the set of virtual ETs $\Sigma(\sigma^n)$. In order to be statistically equivalent to the actual ET, the virtual ETs in $\Sigma(\sigma^n)$ must satisfy the following two conditions: First, the data unit arrival distribution $p_T(\bar{t} | f,z,h)$ (defined in (4)) must be the same for the virtual ETs as it is for the actual ET. In other words, the virtual and actual operating frequencies ($\bar{f}$ and $f$), the virtual and actual type ($\bar{z}$ and $z$), and the virtual and actual configuration ($\bar{h}$ and $h$) must be the same. This also implies that the virtual costs are the same as the actual costs, i.e. $\bar{J}_{APP} = J_{APP}$ and $\bar{J}_{HW} = J_{HW}$. Second, the next virtual buffer state $\bar{q}' \in S_{\bar{q}}$ must be related to the current virtual buffer state $\bar{q} \in S_{\bar{q}}$ through the buffer evolution equation (see (2)), where $|\bar{t}^n| = |t^n \cdot \eta|$ is the number of data unit arrivals under the actual ET.

Any virtual ET that satisfies the above two conditions can have its reward determined using information embedded in the actual ET. Specifically, from the first condition, the virtual ETs as it is for the actual ET. In other words, the virtual and actual operating frequencies ($\bar{f}$ and $f$), the virtual and actual type ($\bar{z}$ and $z$), and the virtual and actual configuration ($\bar{h}$ and $h$) must be the same. This also implies that the virtual costs are the same as the actual costs, i.e. $\bar{J}_{APP} = J_{APP}$ and $\bar{J}_{HW} = J_{HW}$. Second, the next virtual buffer state $\bar{q}' \in S_{\bar{q}}$ must be related to the current virtual buffer state $\bar{q} \in S_{\bar{q}}$ through the buffer evolution equation (see (2)), where $|\bar{t}^n| = |t^n \cdot \eta|$ is the number of data unit arrivals under the actual ET.

Performing the Q-learning update step for every virtual ET in $\Sigma(\sigma^n)$ incurs a computational overhead of approximately $O(\Sigma(\sigma^n) \times A)$ (where $|\Sigma(\sigma^n)| \leq N_q$).

### 5. EXPERIMENTS

In our simulations, we use actual video encoder trace data, which we obtained by profiling the H.264 JM Reference Encoder (version 13.2) on a Dell Pentium IV computer. Our traces comprise measurements of the encoded bit-rate (bits/MB), reconstructed distortion (MSE), and encoding complexity (cycles) for each video MB of the Foreman and Stefan sequences (30 Hz, CIF resolution, quantization parameter 24) under three different encoding configurations with different sub-pel motion estimation accuracies.
As in [1], we assume that the power frequency function is of the form $P(f) = \kappa f^\theta$, where $\kappa \in \mathbb{R}_+$ and $\theta \in [1, 3]$ (we let $\kappa = 1.5 \times 10^{-27}$ and $\theta = 3$). We simulate the system with the measured (non-stationary) trace data and artificial (stationary) trace data generated from the measured data distributions.

We compare the performance of the proposed virtual ET based learning algorithm to an existing RL algorithm called temporal-difference-$\lambda$ learning (i.e. $TD(\lambda)$) [3], which has comparable complexity. In our implementation, $TD(\lambda)$ applies the update step defined in (11) to the $\Psi$ most frequently visited states in the recent past. These recently visited states are updated using the TD error obtained for the current state weighted by the discounted frequency with which they have been recently visited [3].

Figure 2 illustrates the cumulative average reward achieved over $N = 64,000$ time slots when we allow $\Psi \in \{0, 1, 15\}$ virtual ET or $TD(\lambda)$ updates in each time slot (the case when $\Psi = 0$ is equivalent to the conventional centralized Q-learning algorithm in which only the actual ET is updated); and,

Table I illustrates corresponding detailed simulation results.

Interestingly, increasing the number of $TD(\lambda)$ updates in each time slot does not guarantee higher average rewards. However, increasing the number of virtual ET updates in each time slot dramatically improves the average reward. The reason for this stark difference in performance is because the virtual ET based learning algorithm can learn about state-action pairs without visiting them, while the $TD(\lambda)$ learning algorithm can only learn about previously visited state-action pairs. The ability to learn about unvisited states is very important in the buffer control setting because the buffer initially fills from early exploration, the algorithm must learn to efficiently drain the buffer without consuming too much power or unnecessarily reducing the application’s quality. This is simple when using virtual ETs because the information obtained from experience in a “near full” buffer state (e.g. $q = Q - 1$) can be extrapolated to a “near, near full” buffer state (e.g. $q = Q - 2$) and so on. When using $TD(\lambda)$ updates, however, the algorithm learns very slowly about “near, near full” buffer states because they are visited only infrequently from the “near full” buffer state. For this reason, $TD(\lambda)$ (even for a large number of updates) has trouble emptying the buffer, and performs no better than the conventional centralized Q-learning algorithm, which updates only one state-action pair in each time slot.

From the data in Table I, we observe that the percent difference between the reward obtained in the stationary case and the non-stationary case is less than 6%, which suggests that Q-learning with virtual experience can effectively track changes in encoding complexity across a video sequence.

**REFERENCES**


