Towards a General Framework for Cross-Layer Decision Making in Multimedia Systems

Nicholas Mastronarde*, Mihaela van der Schaar

***Abstract*—In recent years, cross-layer multimedia system design and optimization has garnered significant attention; however, there is no existing rigorous methodology for optimizing two or more system layers (e.g. the application, operating system, and hardware layers) jointly while maintaining a *separation* among the decision processes, designs, and implementations of each layer. Moreover, existing work often relies on *myopic* optimizations, which ignore the impact of decisions made at the current time on the system's future performance. In this paper, we propose a novel systematic framework for jointly optimizing the different system layers to improve the performance of one multimedia application. In particular, we model the system as a layered Markov Decision Process (MDP). The proposed layered MDP framework enables each layer to make *autonomous* and *foresighted* decisions, which optimize the system's long-term performance.**

***Index Terms*—Cross-layer multimedia system design, Foresighted decision making, Layered Markov decision process.**

I. INTRODUCTION

Cross-layer adaptation is an increasingly popular solution for optimizing the performance of complex real-time multimedia applications implemented on resource constrained systems [5]-[8] [12] [13]. This is because system performance can be significantly improved by jointly optimizing parameters, configurations, and algorithms across two or more system layers, rather

The authors are with the University of California Los Angeles, Dept. of Electrical Engineering (EE), 56-147 Engineering IV Building, 420 Westwood Plaza, Los Angeles, CA 90095; phone: 1-310-825-5843; fax: 1-310-206-4685; e-mail: nhmastro@ee.ucla.edu (N. Mastronarde); mihaela@ee.ucla.edu (M. van der Schaar).

*Corresponding author. Address and contact information above.

than optimizing them in isolation. The system layers that are most frequently included in the cross-layer optimization are the *application* (APP), *operating system* (OS), and *hardware* (HW) layers. For example, in [5] [6] [13], the APP layer's configuration (e.g. encoding parameters) and the HW layer's operating frequency are adapted; meanwhile, in [7] [8] [12], the resource allocation and scheduling strategies at the OS layer are also adapted.

The abovementioned cross-layer solutions share two important shortcomings. First, the formulations and the presented solutions are all highly dependent on a specific cross-layer problem, and therefore cannot be easily extended to other joint APP-OS-HW optimizations. This exposes the need for a rigorous and systematic methodology for performing cross-layer multimedia system optimization. Second, the abovementioned cross-layer solutions result in sub-optimal performance for dynamic multimedia tasks because they are *myopic*. In other words, cross-layer decisions are made reactively in order to optimize the *immediate* reward (utility) without considering the impact of these decisions on the future reward. For example, even the "oracle" solution in [6], which has exact knowledge about the consumed energy and required instruction counts under different APP and HW layer configurations, is not globally optimal. This is because, "the configuration selected for each frame affects all future frames" [6], however, the oracle only myopically chooses the cross-layer configuration to optimize its immediate reward. This motivates the use of *foresighted* (i.e. long-term) optimization techniques based on dynamic programming. With foresighted decisions, the layers no longer reactively adapt to their experienced dynamics (e.g. time-varying multimedia source characteristics at the APP layer or time-varying resource availability at the OS layer due to resource-sharing with other applications); instead, layers actively select actions to influence and provision for the system's future dynamics in order to achieve optimal performance over time, even if this

requires sacrificing immediate rewards.

In this paper, we take inspiration from work on dynamic power management at the HW layer [9] [14] and formulate the cross-layer optimization problem within the framework of discrete-time Markov decision processes (MDP) in order to optimize the system's long-term performance. We believe that this paper can be viewed as a *nontrivial* extension of the aforementioned work because we investigate previously unaddressed problems associated with cross-layer system optimization.

A trivial and ill-advised solution to the cross-layer system optimization using MDP is to glue the decision making processes of the layers together by performing a centralized optimization in which a single layer (e.g. the OS), a centralized optimizer, or middleware layer must know the states, actions, rewards, and dynamics at every layer. Such a centralized solution violates the layered system architecture, thereby complicating the system's design, increasing implementation costs, and decreasing interoperability of different applications, operating systems, and hardware architectures. Respecting the layered architecture is especially important in situations where system layers are designed by different companies, which (i) may not want for their layer to relinquish its decision making process to a centralized optimizer or middleware layer, or (ii) may not allow access to the underlying implementation of their layer. Under these constraints, centralized solutions such as those proposed in [5] [7] [8] [12] are infeasible because they require that the designer can augment the implementation of every layer.

In this paper, we propose an alternative solution to the cross-layer optimization in which we optimize the system's performance without a centralized optimizer. To do this, we identify the dependencies among the dynamics and decision processes at the various layers of the multimedia system (i.e. APP, OS, and HW) and then factor these dependencies in order to decompose the

centralized optimization into separate optimizations at each layer.

The remainder of this paper is organized as follows. In Section II, we define all of the parameters and concepts in our framework and present the objective of the foresighted cross-layer optimization problem. In Section III, we provide concrete examples of the abstract concepts introduced in Section II for an illustrative cross-layer video encoding problem similar to those explored in [5]-[8]. In Section IV, we describe a centralized cross-layer optimization framework for maximizing the objective function introduced in Section II, and we discuss the framework's limitations. In Section V, we propose a layered optimization framework for maximizing the same objective. In Section VI, we present our experimental results based on the example system described in Section III. Finally, we conclude in Section VII.

II. Cross-Layer Problem Statement

In this section, we present the considered system model and formulate the cross-layer system optimization problem.

A. *Layered System Model*

The considered system architecture comprises three layers: the *application* (APP), *operating system* (OS), and *hardware* (HW) layers. For generality, however, we assume that there are L layers participating in the cross-layer optimization. Each layer is indexed $l \in \{1,...,L\}$ with layer 1 corresponding to the lowest participating layer (e.g. HW layer) and layer L corresponding to the highest participating layer (e.g. APP layer). We note that for a layer to "participate" in the cross-layer optimization it must be able to adapt one or more of its parameters, configurations, or algorithms (e.g. the HW layer can adapt its processor frequency); alternatively, if a layer does not "participate", then it is omitted. In all of the illustrative examples in this paper, we assume that $L = 3$ and that the HW, OS, and APP layers correspond to layers 1, 2, and 3, respectively.

The layered optimization framework proposed in this paper deals with three types of time-varying dynamics, which impact multimedia system performance, but are not simultaneously considered in most existing research. In particular, our framework considers the multimedia application's time-varying (probabilistic) rate-distortion behavior, its time-varying resource requirements, and the system's time-varying resource availability due to contention with other applications, each with their own time-varying requirements. Cross-layer optimization frameworks that do not explicitly consider these dynamics are inherently suboptimal.

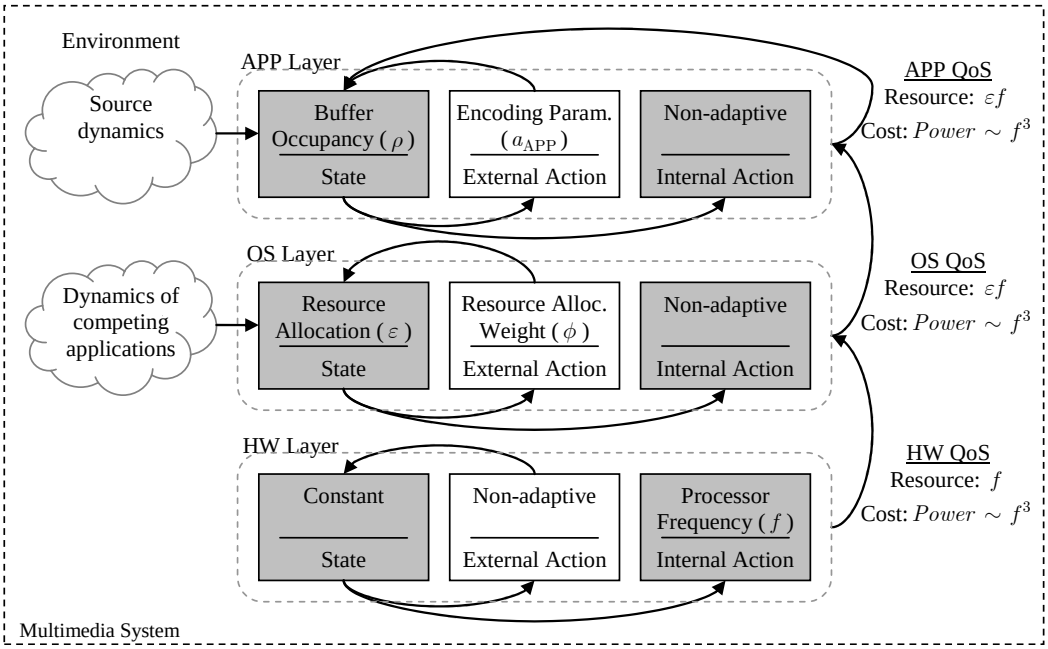

Fig. 1. Layered decision making process illustrating the intra- and inter-layer dependencies in the system. System layers adapt to different components of the dynamic environment by deploying different "actions" at each layer.

Fig. 1 illustrates how each system layer can make autonomous decisions in a layered manner based on their own local information about different components of the dynamic *environment* and based on limited information forwarded from other layers. In our setting, environment refers to anything that affects the system's performance but is not controllable by the system (e.g. the video source characteristics). We will frequently refer back to Fig. 1 in order to make the abstract concepts discussed throughout this section more concrete.

B. States

In this paper, the state encapsulates all of the information relevant to making processing decisions. Illustrative examples of layer states are provided in Section III and are included in Fig. 1. We assume that the states are Markovian such that, given the present state, the past and future states are independent[1]. Since each system layer interacts with a different component of the dynamic environment (see Fig. 1), we define a state $s_l \in \mathcal{S}_l$ for each layer l. We denote the state of the entire system by $\boldsymbol{s} \in \boldsymbol{\mathcal{S}}$, with $\boldsymbol{\mathcal{S}} = \times_{l=1}^{L} \mathcal{S}_l$ (where $\times_{l=1}^{L} \mathcal{S}_l = \mathcal{S}_1 \times \cdots \times \mathcal{S}_L$ is the L-ary Cartesian product). If the lth layer's state is constant, then we write $|\mathcal{S}_l| = 1$, where $|\boldsymbol{\mathcal{S}}|$ denotes the cardinality of set $\boldsymbol{\mathcal{S}}$.

C. Actions

The multimedia system takes different processing actions depending on the state of each layer. The actions at each layer can be classified into two types [3]:

1. *External actions* impact the state transition of the layer that takes the external action. (See Sections III.B and III.C for examples of external actions at the OS and APP layers, which are also illustrated in Fig. 1.) The external actions at layer l are denoted by $a_l \in \mathcal{A}_l$, where $\mathcal{A}_l$ is a finite set of the possible external actions at layer l. The aggregation of the external actions across all of the layers is denoted by $\boldsymbol{a} \in [a_1, \ldots, a_L] \in \boldsymbol{\mathcal{A}}$, where $\boldsymbol{\mathcal{A}} = \times_{l=1}^{L} \mathcal{A}_l$.
2. *Internal actions* determine the Quality of Service (QoS) provided to the upper layers and the state transition at layer L. (See Section II.F for our definition of QoS, and Section III.A and Fig. 1 for an example of internal actions at the HW layer.) The internal actions at layer l are denoted by $b_l \in \mathcal{B}_l$, where $\mathcal{B}_l$ is a finite set of the possible external actions at layer l. The aggregation of the internal actions across all of the layers is denoted by $\boldsymbol{b} \in [b_1, \ldots, b_L] \in \boldsymbol{\mathcal{B}}$, where $\boldsymbol{\mathcal{B}} = \times_{l=1}^{L} \mathcal{B}_l$.

The action at layer l is the aggregation of the external and internal actions, denoted by $\xi_l = \left[a_l, b_l\right] \in \mathcal{X}_l$, where $\mathcal{X}_l = \mathcal{A}_l \times \mathcal{B}_l$. Finally, the joint action of the system is denoted by $\boldsymbol{\xi} = [\xi_1, ..., \xi_L] \in \boldsymbol{\mathcal{X}} = \times_{l=1}^{L} \mathcal{X}_l$.

We note that some layers may have non-adaptive (i.e. fixed) external and internal actions. If the lth layer's external or internal action is non-adaptive, then $|\mathcal{A}_l| = 1$ or $|\mathcal{B}_l| = 1$, respectively. Illustrative examples of external and internal actions are provided in Section III and in Fig. 1.

D. State Transition Probabilities

We assume that the state transition in each layer is synchronized and operates at the same time scale, such that the transition can be discretized into stages during which the multimedia system has constant state and performs a single joint-action. The length of each stage does not need to be constant. For instance, in our H.264/AVC video encoding example, the length of each stage is the duration of time between encoding successive data-units (DUs) such as video macroblocks or frames. We use a superscript n to denote stage n (i.e. the stage in which we make the encoding decision for DU n). For notational simplicity, we omit the superscript n when no confusion will arise.

In general, because states are Markovian, the state transition of the system only depends on the current state $\boldsymbol{s}$, the current joint action, and the environmental dynamics. The corresponding transition probability is denoted by $p(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{\xi})$, where $\boldsymbol{s} = \boldsymbol{s}^n$ and $\boldsymbol{s}' = \boldsymbol{s}^{n+1}$ ($n \in \mathbb{N}$).

The state transition probability can be factored to reflect the dependencies in the layered system architecture. First, using Bayes' rule, the transition probability can be factored as

$$p(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{\xi}) = \prod_{l=1}^{L} p(s_l' \mid \boldsymbol{s}'_{1 \to l\text{-}1}, \boldsymbol{s}, \boldsymbol{\xi}), \tag{1}$$

where $\boldsymbol{s}'_{1 \to l} = [s_1', ..., s_l']$ and $\boldsymbol{s}'_{1 \to 0} = \mathbf{0}$ is an empty state. In this paper, we assume that the next state

[1] We investigate the validity of this assumption for a realistic multimedia system in Section VI.C.

of layer l, s_l', is statistically independent of the next state of the lower layers, $\boldsymbol{s}_{1\to l\text{-}1}'$, conditioned on the current state. Hence, we can rewrite (1) as

$$p(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{\xi}) = \prod_{l=1}^{L} p(s_l' \mid \boldsymbol{s}, \boldsymbol{\xi}) \tag{2}$$

Second, due to the layered architecture and the definitions of the internal and external actions as illustrated in Fig. 1, (2) can be factored further as follows:

$$p(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{\xi}) = p(s_L' \mid \boldsymbol{s}, a_L, \boldsymbol{b}) \prod_{l=1}^{L-1} p(s_l' \mid s_l, a_l). \tag{3}$$

We would like to make the following remarks about (3)**:**

- The term $\prod_{l=1}^{L-1} p(s_l' \mid s_l, a_l)$ in (3) is taken from the first $L-1$ layers in (2). Since the state-transition at layer $l \in \{1, \ldots, L-1\}$ only depends on its own external action a_l and state s_l, these replace the joint state $\boldsymbol{s}$ and joint mixed action $\boldsymbol{\xi}$ in (2). The state transitions at the HW and OS layers illustrated in Fig. 1 reflect this model.
- The term $p(s_L' \mid \boldsymbol{s}, a_L, \boldsymbol{b})$ in (3) is taken from layer L in (2). The state transition at layer L depends on the joint state $\boldsymbol{s}$, its own external action a_L, and the joint internal action $\boldsymbol{b}$. In other words, the state transition at the application layer depends on the states and internal actions of all of the layers, which serve the application at layer L. This is illustrated in Fig. 1, except that the states and internal actions at the lower layers are abstracted by the QoS values, which we discuss in Section II.F.
- If the lth layer's state is constant (i.e. $|\mathcal{S}_l| = 1$), then its state transition probability $p(s_l' \mid s_l, a_l) = 1$. On the other hand, if the lth layer's external action is non-adaptive (i.e. $|\mathcal{A}_l| = 1$), then its state transition probability $p(s_l' \mid s_l, a_l) = p(s_l' \mid s_l)$. This means that the state transition is governed solely by the *environment*.

Illustrative examples of state transition probabilities are provided in Section III.

E. Reward Function and Layer Costs

Performing the external and internal actions at layer l incurs the costs $\alpha_l(s_l, a_l)$ and $\beta_l(s_l, b_l)$, respectively. We define example cost functions for each layer in Section III (e.g. the power consumed at the HW layer and mean squared error at the APP layer).

We assume that the expected reward for taking joint action $\boldsymbol{\xi}$ in state $\boldsymbol{s}$ is a weighted sum of the costs at each layer plus an additional gain (we define an example gain function in Section III), i.e.

$$R(\boldsymbol{s}, \boldsymbol{\xi}) = g(\boldsymbol{s}, \boldsymbol{b}) - \sum_{l=1}^{L} \omega_l^a \alpha_l(s_l, a_l) - \sum_{l=1}^{L} \omega_l^b \beta_l(s_l, b_l) \tag{4}$$

where $g(\boldsymbol{s}, \boldsymbol{b})$ is the expected gain, and ω_l^a and ω_l^b weight the external and internal costs at layer l, respectively. We assume that the weights ω_l^a and ω_l^b are known and have been determined based on the desired tradeoff among the various layer costs. The reward in Eq. (4) can be separated into two parts: one is the internal reward, which depends on the internal actions, and the other is the external reward, which depends on the external actions. The internal reward is

$$R_{\text{in}}(\boldsymbol{s}, \boldsymbol{b}) = g(\boldsymbol{s}, \boldsymbol{b}) - \sum_{l=1}^{L} \omega_l^b \beta_l(s_l, b_l), \tag{5}$$

and the external reward is

$$R_{\text{ex}}(\boldsymbol{s}, \boldsymbol{a}) = -\sum_{l=1}^{L} \omega_l^a \alpha_l(s_l, a_l). \tag{6}$$

Hence, the total reward is $R(\boldsymbol{s}, \boldsymbol{\xi}) = R_{\text{in}}(\boldsymbol{s}, \boldsymbol{b}) + R_{\text{ex}}(\boldsymbol{s}, \boldsymbol{a})$.

F. Quality of Service

Definition: Quality of Service (QoS). The QoS at layer l is defined as a pair $Q_l = (\nu_l, \eta_l)$ comprised of (i) the amount of reserved resources for the application layer, denoted by ν_l, and (ii) the immediate cost for reserving those resources, denoted by η_l.

The QoS serves as an abstraction of the states and internal actions at the lower layers such that they do not have to directly reveal their parameters and available configurations to the upper

layers. In this paper, ν_l is the effective service rate in cycles per second reserved for the application layer (i.e. layer L) and $\eta_l = \sum_{l' \le l} \beta_{l'}(s_{l'}, b_{l'})$ is the cumulative internal action cost[2] incurred by layers $l' \in \{1,\ldots,l\}$. Importantly, the QoS at layer l can be recursively computed given the QoS at layer $l-1$:

$$Q_l = \begin{cases} (F_l^{\nu_l}(s_l, b_l, Q_{l-1}), F_l^{\eta_l}(s_l, b_l, Q_{l-1})), & l = 2,\ldots,L \\ (F_l^{\nu_l}(s_l, b_l), F_l^{\eta_l}(s_l, b_l)), & l = 1 \end{cases}$$

where $F_l^{\nu_l}$ and $F_l^{\eta_l}$ are functions, which map the state s_l, internal action b_l, and QoS Q_{l-1} to the service rate ν_l and cost η_l, respectively. Because the QoS can be recursively computed, at no point do the upper layers require specific information about the state sets and internal action sets at the lower layers. For notational simplicity, we will write the recursive QoS function compactly as $Q_l = \vec{F}_l(s_l, b_l, Q_{l-1})$. In Sections III.A, III.B, and III.C we define illustrative QoS pairs for the HW, OS, and APP layers, respectively. These are also illustrated in Fig. 1.

Given the QoS provided by the lower layers, we can rewrite the Lth layer's transition probability function $p(s_L' \mid \boldsymbol{s}, a_L, \boldsymbol{b})$ in (3) as

$$p(s_L' \mid \boldsymbol{s}, a_L, \boldsymbol{b}) = p(s_L' \mid s_L, a_L, Q_L), \tag{7}$$

and its internal reward function $R_{\text{in}}(\boldsymbol{s}, \boldsymbol{b})$ in (5) as

$$R_{\text{in}}(\boldsymbol{s}, \boldsymbol{b}) = R_{\text{in}}(s_L, Q_L). \tag{8}$$

In other words, there is a one-to-one correspondence between the Lth layer's QoS and the states $\boldsymbol{s}_{1\to L-1} = (s_1,\ldots,s_{L-1})$ and internal actions $\boldsymbol{b}_{1\to L} = (b_1,\ldots,b_L)$. Hence, the Lth layer's QoS provides all of the information required for the Lth layer to determine its transition probability function and internal reward. The relationships in (7) and (8) are required for the layered optimization solution proposed in Section V.

[2] We include the internal cost in the QoS because the APP layer's immediate reward and state transition depend on the internal actions at the lower layers. Hence, when the application selects the optimal QoS level, it must consider the costs incurred by these layers to ensure optimal

G. Foresighted decision making

Unlike traditional cross-layer optimization, which focuses on the myopic (i.e. immediate) utility, the goal in the proposed cross-layer framework is to find the optimal actions at each stage that maximize the *expected discounted sum of future rewards*, i.e.

$$E\left\{\sum_{n=n_0}^{\infty}(\gamma)^{n-n_0}R(\boldsymbol{s}^n,\boldsymbol{\xi}^n \mid \boldsymbol{s}^{n_0})\right\}, \tag{9}$$

where the parameter γ $(0 \le \gamma < 1)$ is the "discount factor," which defines the relative importance of present and future rewards, $R(\boldsymbol{s}^n,\boldsymbol{\xi}^n \mid \boldsymbol{s}^{n_0})$ is the reward at stage n conditioned on the state at stage n_0 being $\boldsymbol{s}^{n_0}$, and the expectation is taken over the states $\{\boldsymbol{s}^n : n = n_0+1, n_0+2, \ldots\}$. We refer to decisions that maximize (9) as *foresighted* cross-layer decisions because, by maximizing the cumulative discounted reward, the multimedia system is able to take into account the impact of the current actions on the *future* reward. In Section VI.B, we discuss the impact of the discount factor on system's performance. In Section IV, we describe how to maximize (9) using a centralized MDP. Then, in Section V, we present an alternative solution to the same problem based on a layered MDP.

III. ILLUSTRATIVE EXAMPLE

In this section, we provide concrete examples of states, internal and external actions, state transition probabilities, cost functions, and QoS pairs. Our examples are organized by layer, starting with the HW layer and working up to the APP layer. Throughout the illustrative examples in this paper, we assume that $L = 3$ and that the HW, OS, and APP layers correspond to layers 1, 2, and 3, respectively. To facilitate understanding of our examples, we will use the subscripts HW, OS, and APP, instead of only specifying the layers by their indices.

We note that if a layer's state is constant, then it has a non-adaptive external action because

---

performance across all layers. In contrast, the external cost is determined by each individual layer when it selects its external action, which does

there is nothing it can to do adapt its constant state. We also assume that there is no cost associated with a non-adaptive external action or a non-adaptive internal action.

A. *HW Layer Examples*

We assume that the HW layer's processor can be operated at different frequency-voltage pairs to make energy-delay tradeoffs [4].

HW state: In this example, the HW layer has a constant state (i.e. $|\mathcal{S}_{\mathrm{HW}}| = 1$).

HW actions: We let the HW layer's internal action, $b_{\mathrm{HW}} \in \mathcal{B}_{\mathrm{HW}}$, set the processor to one of X frequencies, i.e. $\mathcal{B}_{\mathrm{HW}} = \{f_1, \ldots, f_X\}$. We denote the frequency used for processing the n th DU as $f(n) = b_{\mathrm{HW}}^n$. This is an internal action because it determines the HW layer's QoS, which is defined below.

HW state transition: Since the HW layer has only one state, it has a deterministic state transition, i.e. $p(s'_{\mathrm{HW}} \mid s_{\mathrm{HW}}, a_{\mathrm{HW}}) = 1$.

HW costs: The HW layer's internal cost is the amount of power required for it to run at frequency $f = b_{\mathrm{HW}}$. We define the HW layer's internal cost as [5]:

$$\beta_{\mathrm{HW}}(s_{\mathrm{APP}}, a_{\mathrm{APP}}) = f^3 . \tag{10}$$

HW QoS: We define the quality of service (QoS) that the HW layer provides to the OS layer as the pair $Q_{\mathrm{HW}} = (\nu_{\mathrm{HW}}, \eta_{\mathrm{HW}})$, where $\nu_{\mathrm{HW}} = f$ is the CPU frequency and $\eta_{\mathrm{HW}} = \beta_{\mathrm{HW}}(s_{\mathrm{APP}}, a_{\mathrm{APP}})$ is the HW layer's internal cost. The HW layer's QoS is shown on the right hand side of Fig. 1.

B. *OS Layer Examples*

OS state: We denote the OS state by $s_{\mathrm{OS}} \in \mathcal{S}_{\mathrm{OS}}$. We let $s_{\mathrm{OS}}^n = \varepsilon(n)$, where $\varepsilon(n) \in (0,1]$ is the CPU time fraction that the OS reserves for encoding the n th DU (hence, $1 - \varepsilon(n)$ is reserved for other applications). We assume that the OS can be in any one of M states such that

---

not impact the immediate reward at the APP layer. Therefore, the external costs do not need to be included in the QoS.

$s_{\mathrm{OS}} \in \boldsymbol{\mathcal{S}}_{\mathrm{OS}} = \{\varepsilon_1,\ldots,\varepsilon_M\}$.

OS actions: In this example, the OS has a non-adaptive internal action (i.e. $|\mathcal{B}_{\mathrm{OS}}| = 1$) because we assume that the OS has no actions that impact the immediate reward (or, by extension, the immediate QoS) at the APP layer. The OS layer's external action at stage n, however, impacts the amount of resources reserved for the application in stage $n+1$. We assume that the weighted max-min fairness (WMM) [7] resource allocation strategy is used to divide processor time among the competing applications. Hence, the OS layer's external action is a declaration of the application's weight ϕ. We assume that there are W possible external actions: i.e., $a_{\mathrm{OS}} \in \mathcal{A}_{\mathrm{OS}} = \{\phi_1,\ldots,\phi_W\}$.

OS state transition: We model the OS state transition as a finite-state Markov chain with transition probabilities $p(s'_{\mathrm{OS}} \mid s_{\mathrm{OS}}, a_{\mathrm{OS}})$, with $s_{\mathrm{OS}}, s'_{\mathrm{OS}} \in \mathcal{S}_{\mathrm{OS}}$ and $a_{\mathrm{OS}} \in \mathcal{A}_{\mathrm{OS}}$. This is similar to the Markov model of the *service provider* in [9]. If we assume that the weights of other applications are unknown when the OS layer's external action is selected, then its state transition is non-deterministic; On the other hand, if the weights of other tasks are known when the OS layer's external action is selected, then its state transition is deterministic, because it can directly calculate its resource allocation for each weight.

OS costs: We define the external cost associated with the OS layer as the application's weight, hence

$$\alpha_{\mathrm{OS}}(s_{\mathrm{OS}}, b_{\mathrm{OS}}) = b_{\mathrm{OS}} = \phi .$$

This cost prevents the application from requesting excessive resources when it stands to gain very little additional reward from them.

OS QoS: We define the QoS that the OS layer provides to the APP layer as the pair $Q_{\mathrm{OS}} = (\nu_{\mathrm{OS}}, \eta_{\mathrm{OS}})$, where $\nu_{\mathrm{OS}} = \varepsilon f$, ε is the CPU time fraction allocated to the application, f is

the CPU frequency, and $\eta_{\mathrm{OS}} = \eta_{\mathrm{HW}}$ because there are no internal costs incurred at the OS layer. The OS layer's QoS is illustrated on the right hand side of Fig. 1.

C. APP Layer Examples

APP state: The APP layer's state at time index n, s^n_{APP}, is equivalent to the number of DUs $\rho(n) \in [0, \rho^{\max}]$, $\rho^{\max} \in \mathbb{N}$ in its post-encoding buffer. The post-encoding buffer is placed between the encoder and the network (if the encoded video is to be streamed), or between the encoder and a storage device (if the encoded video is to be stored for later use). The buffer allows us to mitigate hard real-time deadlines by introducing a maximum allowable latency, which is proportional to the maximum buffer size $\rho^{\max}$. In [10], a similar buffer is used at the decoder for decoding video frames with high peak complexity requirements in real-time.

Although one goal of our illustrative cross-layer optimization problem is to minimize costs (e.g. power consumption at the HW layer), a competing goal is to avoid *buffer underflow* (caused by DUs missing their delay deadlines) and to avoid *buffer overflow* (caused by DUs being encoded too quickly) [10] [11]. Hence, the goal of the cross-layer optimization is for all layers to cooperatively adapt in order to achieve the optimal balance between the costs incurred at each layer and the buffer occupancy. We note that an alternative buffer model (referred to as a *service queue* model) is presented in [9].

APP actions: We assume that the APP layer has a non-adaptive internal action (i.e. $|\mathcal{B}_{\mathrm{APP}}| = 1$) but that it has external actions $a_{\mathrm{APP}} \in \mathcal{A}_{\mathrm{APP}}$[3]. In a typical video encoder, for example, application configurations can include the choice of quantization parameter, the motion-vector search range, etc. The application's configuration affects the nth DU's encoding complexity $c(n, a_{\mathrm{APP}})$, which is an instance of the random variable C with distribution.

$$C \sim p_{a_{\mathrm{APP}}}(c) = p(c \mid a_{\mathrm{APP}}) \tag{11}$$

APP state transition: We define the actual post-encoding buffer occupancy recursively as

$$\begin{aligned} &\rho(n+1) = \min\{\max\{\rho(n) + 1 - \lfloor t(n, s_{\mathrm{OS}}, a_{\mathrm{APP}}, \boldsymbol{b}) \cdot v \rfloor, 0\}, \rho^{\max}\} \\ &\rho(0) = \rho^{\mathrm{initial}}, \end{aligned} \tag{12}$$

where

$$t(n, s_{\mathrm{OS}}, a_{\mathrm{APP}}, \boldsymbol{b}) = \frac{c(n, a_{\mathrm{APP}})}{\varepsilon(n) f(n)} \ (\mathrm{seconds}) \tag{13}$$

is the nth DU's processing delay, which depends on the processor frequency $f(n)$ and the fraction of time, $s_{\mathrm{OS}}^{n} = \varepsilon(n)$, allocated to the application at stage n. In (12), the 1 indicates that $\mathrm{DU}(n)$ is added into the post-encoding buffer after the processing delay $t(n, s_{\mathrm{OS}}, a_{\mathrm{APP}}, \boldsymbol{b})$; $\lfloor X \rfloor$ takes the integer part of X; v is the average number of DUs that must be processed per second (for example, if DUs are frames, then the service rate required for real-time encoding is typically $v = 30$ frames per second); and, ρ^{initial} is the initial post-encoding buffer occupancy, which we set to $\rho^{\mathrm{initial}} = \rho^{\max}/2$ so that we do not initially bias the buffer toward overflow or underflow.

As described before, the DU's encoding complexity is a random variable, which depends on the application's external action (i.e. encoding parameter selection). This causes uncertainty in the buffer state transition. This state transition also depends on the states and internal actions at the lower layers (since they affect the CPU frequency f and the CPU time fraction ε reserved for the application). Hence, the APP state transition probability is given by $p(s'_{\mathrm{APP}} \mid \boldsymbol{s}, a_{\mathrm{APP}}, \boldsymbol{b})$, which is congruent with the first term on the right hand side of (3).

Given the processor frequency f from the HW layer and the time fraction ε from the OS layer, the nth DU's processing delay t is an instance of the random variable $T = \dfrac{C}{\varepsilon f}$ (seconds) with distribution $T \sim p_{a_{\mathrm{APP}}}(t) = \varepsilon f \cdot p_{a_{\mathrm{APP}}}(\varepsilon f \cdot c)$. Finally, we let $\tilde{T} = v \cdot T$ with distribution

[3] We know from (3) that, unlike the other layers, the APP layer's state transition depends on its external *and* internal actions. For our

$\tilde{T} \sim p_{a_{\mathrm{APP}}}(\tilde{t}) = \frac{1}{v} \cdot p_{a_{\mathrm{APP}}}\left(\frac{t}{v}\right)$. Using $\tilde{T}$, the APP state transition probability can be written as

$$p(s'_{\mathrm{APP}}{=}\rho' | \boldsymbol{s}, a_{\mathrm{APP}}, \boldsymbol{b}) = \begin{cases} p_{a_{\mathrm{APP}}}\{\tilde{t} \geq \rho + 1\}, & \rho \in \{0,\ldots,\rho^{\max}\},\ \rho' = 0 \\ p_{a_{\mathrm{APP}}}\{0 \leq \tilde{t} < 2\}, & \rho = \rho' = \rho^{\max} \\ p_{a_{\mathrm{APP}}}\{\rho - \rho' + 1 \leq \tilde{t} < \rho - \rho' + 2\}, & \text{otherwise} \end{cases} \tag{14}$$

APP costs: We penalize the APP's external action by employing the Lagrangian cost measure used in the H.264/AVC reference encoder for making rate-distortion optimal mode decisions. Formally, we define this cost as

$$\alpha_{\mathrm{APP}}(s_{\mathrm{APP}}, a_{\mathrm{APP}}) = d(a_{\mathrm{APP}}) + \lambda_{\mathrm{rd}} r(a_{\mathrm{APP}}), \tag{15}$$

where $d(a_{\mathrm{APP}})$ and $r(a_{\mathrm{APP}})$ are the distortion (mean squared error) and compression rate (bits/DU), respectively, incurred by the application's external action, and $\lambda_{\mathrm{rd}} \in [0,\infty)$ is used to weight the relative importance of the distortion d and the rate r in the overall cost.

APP QoS: We define the QoS at the APP layer as the pair $Q_{\mathrm{APP}} = (\nu_{\mathrm{APP}}, \eta_{\mathrm{APP}})$, where $\nu_{\mathrm{APP}} = \nu_{\mathrm{OS}}$ and $\eta_{\mathrm{APP}} = \eta_{\mathrm{OS}}$ because there are no internal costs incurred at the APP layer. Hence, in this example, $Q_{\mathrm{APP}} = Q_{\mathrm{OS}}$. The APP layer's QoS is illustrated on the right hand side of Fig. 1.

D. The System Reward

Recall that the reward defined in (4) can be expressed as $R(\boldsymbol{s}, \boldsymbol{\xi}) = R_{\mathrm{in}}(\boldsymbol{s}, \boldsymbol{b}) + R_{\mathrm{ex}}(\boldsymbol{s}, \boldsymbol{a})$. Given the system state $\boldsymbol{s} = (s_{\mathrm{HW}}, s_{\mathrm{OS}}, s_{\mathrm{APP}})$ and the joint action $\boldsymbol{\xi} = (\boldsymbol{a}, \boldsymbol{b})$, where $\boldsymbol{a} = (a_{\mathrm{HW}}, a_{\mathrm{OS}}, a_{\mathrm{APP}})$ and $\boldsymbol{b} = (b_{\mathrm{HW}}, b_{\mathrm{OS}}, b_{\mathrm{APP}})$, the internal reward defined in (5) can be written as

$$R_{\mathrm{in}}(\boldsymbol{s}, \boldsymbol{b}) = g(\boldsymbol{s}, \boldsymbol{b}) - \omega^{b}_{\mathrm{HW}} \underbrace{\beta_{\mathrm{HW}}(s_{\mathrm{HW}}, a_{\mathrm{HW}})}_{f^3}. \tag{16}$$

In this paper, we define the expected gain $g(\boldsymbol{s}, \boldsymbol{b})$ as a penalty for buffer underflow and buffer overflow. The gain is a component of the reward that is designed to keep the buffer state away from overflow and underflow. Formally, we define the expected gain as

---

application, this makes the distinction between the two types of actions at the APP layer largely inconsequential.

$$g(\boldsymbol{s},\boldsymbol{b}) = \sum_{s'_{\mathrm{APP}} \in \mathcal{S}} g\left(s_{\mathrm{APP}}, \boldsymbol{b}, s'_{\mathrm{APP}}\right) p(s'_{\mathrm{APP}} | \boldsymbol{s}, a_{\mathrm{APP}}, \boldsymbol{b}), \tag{17}$$

where we let

$$g\left(s_{\mathrm{APP}}, \boldsymbol{b}, s'_{\mathrm{APP}}\right) = \begin{cases} \Omega, & \text{if } \Lambda \le s'_{\mathrm{APP}} \le \rho^{\max}\text{-}\Lambda \\ \dfrac{s'_{\mathrm{APP}}}{\Lambda}\Omega, & \text{if } s'_{\mathrm{APP}} < \Lambda \\ \dfrac{\rho^{\max} - s'_{\mathrm{APP}}}{\Lambda}\Omega, & \text{otherwise} \end{cases} \tag{18}$$

with $\Omega > 0$ and $0 \le \Lambda < \rho^{\max}/2$. This is a good metric to include in the reward because it is indicative of the quality degradation experienced by the application when encoded DUs are lost (due to buffer overflow) or when they miss their deadlines (due to buffer underflow).

Lastly, the external reward defined in (6) can be written as

$$R_{\mathrm{ex}}(\boldsymbol{s},\boldsymbol{a}) = -\omega^a_{\mathrm{APP}} \underbrace{\alpha_{\mathrm{APP}}(s_{\mathrm{APP}}, a_{\mathrm{APP}})}_{d+\lambda_{\mathrm{rd}} r} - \omega^a_{\mathrm{OS}} \underbrace{\alpha_{\mathrm{OS}}(s_{\mathrm{OS}}, a_{\mathrm{OS}})}_{\phi}. \tag{19}$$

IV. Centralized Cross-Layer Optimization

Some existing cross-layer optimization frameworks for systems [7] [8] [12]assume that there is a centralized coordinator to determine the actions taken by each layer. This coordinator usually resides at the OS layer or a middleware layer.

In this section, we describe how to maximize (9) using a centralized MDP. Then, in Section V, we present an alternative solution to the same problem based on a layered MDP.

A. Centralized Foresighted Decision Making Using a Markov Decision Process

We model the foresighted centralized cross-layer optimization problem as a Markov decision process (MDP) with the objective of maximizing the discounted sum of future rewards defined in (9). In this way, we are able to consider the impact of the current actions on the future rewards in a rigorous and systematic manner. An MDP is defined as follows:

Definition: Markov decision process (MDP). An MDP is a tuple $(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{X}}, p, R, \gamma)$ where $\boldsymbol{\mathcal{S}}$ is the joint state-space, $\boldsymbol{\mathcal{X}}$ is the joint action-space, p is a transition probability function

$p : \boldsymbol{\mathcal{S}} \times \boldsymbol{\mathcal{X}} \times \boldsymbol{\mathcal{S}} \mapsto [0,1]$, R is a reward function $R : \boldsymbol{\mathcal{S}} \times \boldsymbol{\mathcal{X}} \mapsto \mathbb{R}$, and γ is a discount factor.

In the context of our layered system architecture, $\boldsymbol{\mathcal{S}} = \times_{l=1}^{L} \mathcal{S}_l$, $\boldsymbol{\mathcal{X}} = \times_{l=1}^{L} \mathcal{X}_l$, the transition probability is given by (3), and the reward is given by (4). The solutions to the centralized MDP described below and the layered solution described in Section V are implemented offline, prior to the execution of the application. Additionally, the two solutions assume that $\boldsymbol{\mathcal{S}}$, $\boldsymbol{\mathcal{X}}$, p and R are all known a priori and that they do not change during the execution of the application. In Section VI.C, we discuss the impact of an inaccurate probability transition model p on the system's performance.

In this paper, the goal of the MDP is to find a Markov policy π, which maximizes the discounted sum of future rewards. A Markov policy $\pi \in \Pi$ is a mapping from a state to an action, i.e. $\pi : \boldsymbol{\mathcal{S}} \mapsto \boldsymbol{\mathcal{X}}$. Π is the Markov policy space. The policy can be decomposed into external and internal policies, i.e. $\pi^a : \boldsymbol{\mathcal{S}} \mapsto \boldsymbol{\mathcal{A}}$ and $\pi^b : \boldsymbol{\mathcal{S}} \mapsto \boldsymbol{\mathcal{B}}$, respectively; and, further decomposed into layered external and internal policies, i.e. $\pi_l^a : \boldsymbol{\mathcal{S}} \mapsto \mathcal{A}_l$ and $\pi_l^b : \boldsymbol{\mathcal{S}} \mapsto \mathcal{B}_l$, respectively. We assume that the Markov policy is stationary, hence the mapping of state to action does not depend on the stage index n.

A commonly used metric for evaluating a policy π is the state-value function V^π [2], where

$$V^\pi(\boldsymbol{s}) = \underbrace{R(\boldsymbol{s}, \pi(\boldsymbol{s}))}_{\text{current expected reward}} + \underbrace{\gamma \sum_{\boldsymbol{s}' \in \mathcal{S}} p\big(\boldsymbol{s}' \mid \boldsymbol{s}, \pi(\boldsymbol{s})\big) V^\pi(\boldsymbol{s}')}_{\text{expected future rewards}}. \tag{20}$$

In words, $V^\pi(\boldsymbol{s})$ is equal to the current expected reward plus the expected future rewards, which is calculated by assuming that the policy π is followed until stage $n \to \infty$.

Our objective is to determine the optimal policy π^*, which maximizes (20) for all $\boldsymbol{s}$ (or, equivalently, maximizes (9) for all $\boldsymbol{s}^{n_0}$). The optimal state-value function is defined as

$$V^*(\boldsymbol{s}) = \max_{\pi \in \Pi} \{V^\pi(\boldsymbol{s})\}, \ \forall \boldsymbol{s} \in \boldsymbol{\mathcal{S}}, \tag{21}$$

and the optimal policy π^* is defined as

$$\pi^*(\boldsymbol{s}) = \arg\max_{\pi \in \Pi} \{V^{\pi}(\boldsymbol{s})\}, \ \ \forall \boldsymbol{s} \in \boldsymbol{\mathcal{S}}\,. \tag{22}$$

The optimal value function V^* and the corresponding optimal policy π^* can be iteratively computed using a technique called value iteration (VI) [2] as follows:

$$V_k^*(\boldsymbol{s}) = \max_{\boldsymbol{\xi} \in \boldsymbol{\mathcal{X}}} \left\{ R(\boldsymbol{s}, \boldsymbol{\xi}) + \gamma \sum_{\boldsymbol{s}' \in \mathcal{S}} p\left(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{\xi}\right) V_{k-1}^*(\boldsymbol{s}') \right\}, \tag{23}$$

where k is the iteration number and $V_0^*(\boldsymbol{s})$ is an arbitrary initial estimate of the state-value function. The optimal stationary policy π^* defined in (22) is obtained by iterating until $k \to \infty$. In practice, VI requires only a few iterations before converging to a near optimal state-value function and policy. Importantly, V^* and π^* are computed offline; then, at run-time, all that is required to perform optimally is to follow the optimal state-to-action mappings π^* stored in a simple look-up table of size $|\boldsymbol{\mathcal{S}}|$.

B. Centralized Cross-Layer System

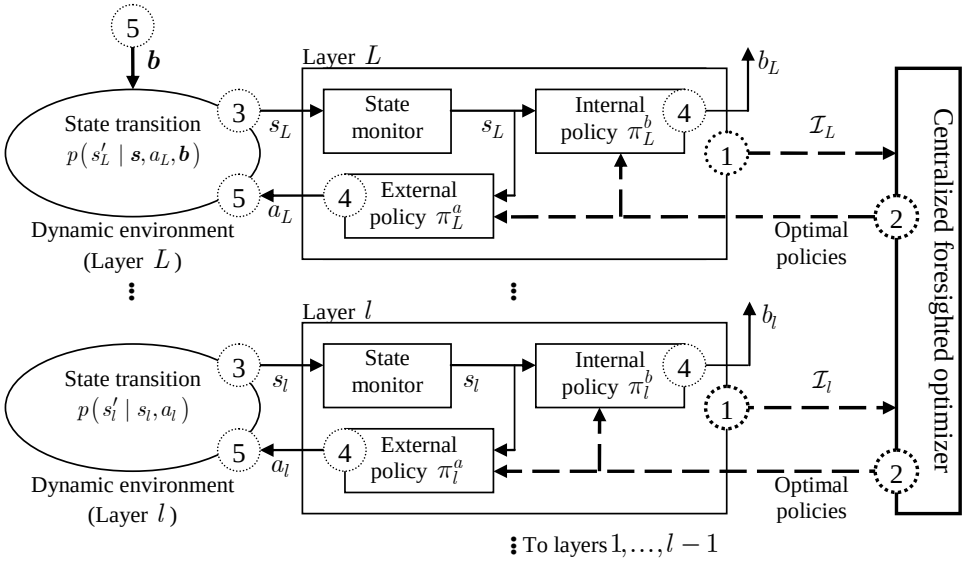

Fig. 2. Centralized cross-layer system optimization framework. The portions of the figure that are in bold comprise the centralized information exchange process. If the dynamics are stationary, then steps 1 and 2 only need to happen once.

The centralized foresighted framework introduced in the previous subsection requires each layer to convey the following information tuples to the centralized optimizer:

$$\mathcal{I}_l = \begin{cases} \left\langle \mathcal{S}_L,\ \mathcal{X}_L,\ p\left(s'_L \mid \boldsymbol{s}, a_L, \boldsymbol{b}\right),\ \alpha_L\left(s_L, a_L\right),\ \beta_L\left(s_L, b_L\right)\right\rangle, & \text{for layer } L \\ \left\langle \mathcal{S}_l,\ \mathcal{X}_l,\ p\left(s'_l \mid s_l, a_l\right),\ \alpha_l\left(s_l, a_l\right),\ \beta_l\left(s_l, b_l\right)\right\rangle, & \text{for layers } l < L. \end{cases}$$

Fig. 2 summarizes the centralized foresighted cross-layer framework.

There are several limitations to this centralized framework, which we have already described in the introduction (Section I). In summary, these limitations stem from steps 1 and 2 of the procedure described above because these steps require the layers to expose their parameters and algorithms, and to relinquish their decision making processes, to a centralized entity, thereby violating the layered system architecture.

V. Layered Optimization

To overcome the shortcomings of the centralized solution, we propose a layered optimization framework for maximizing the same objective function (i.e. the discounted sum of future rewards in (9)). The proposed layered framework allows layers to make optimal autonomous decisions with only a small overhead for exchanging messages between them. Importantly, the format of these messages is independent of the parameters, configurations, and algorithms at each layer, which makes the framework adaptable to different applications, operating systems, and hardware architectures. Fig. 3 illustrates the layered framework. The flow of information in Fig. 3 is the same as in Fig. 2 except that steps 1 and 2 are replaced by the layered optimizer, which we describe in detail in Section V.B, and the states and internal actions at the lower layers determine the QoS to the upper layers as we described in Section II.F.

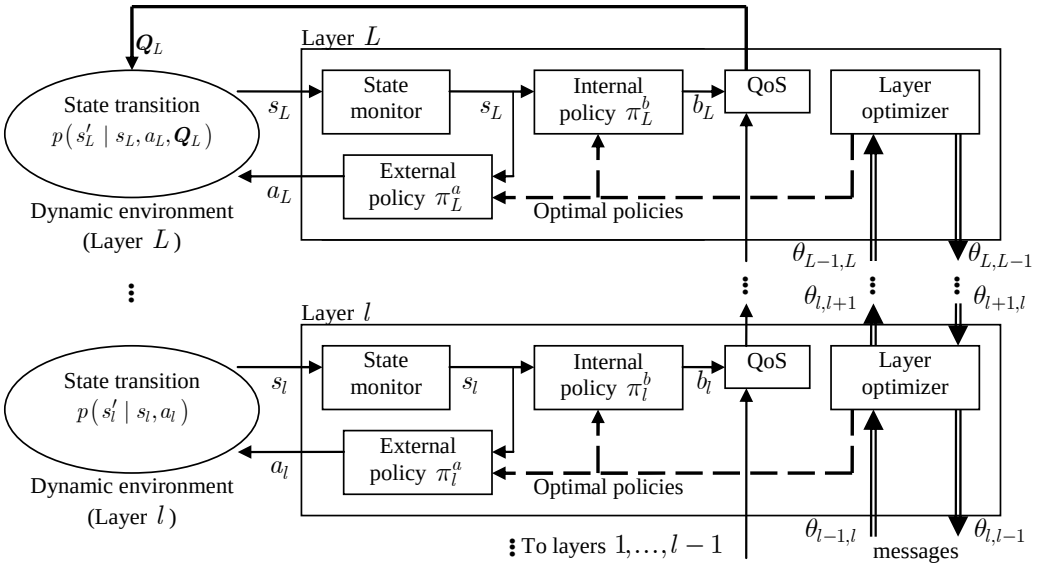

Fig. 3. Layered cross-layer system optimization framework with message exchanges.

A. *Layered MDP for Cross-layer Decision Making*

Definition: Layered MDP [3]**.** The layered MDP model is defined by the tuple $\langle \boldsymbol{\mathcal{L}}, \boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{X}}, \{\Theta_{l,l+1}\}_{l=1}^{L-1}, \{\Theta_{l,l-1}\}_{l=2}^{L}, p, R, \gamma \rangle$, where $\boldsymbol{\mathcal{L}} = \{1,\ldots,L\}$ is a set of L layers; $\boldsymbol{\mathcal{S}}$ is a finite set of states with elements $\boldsymbol{s} = (s_1,\ldots,s_L) \in \boldsymbol{\mathcal{S}}$; $\boldsymbol{\mathcal{X}}$ is a finite action space with elements $\boldsymbol{\xi} = (\xi_1,\ldots,\xi_L) \in \boldsymbol{\mathcal{X}}$, where $\xi_l = (a_l, b_l)$ contains the lth layer's external action a_l and internal action b_l; $\Theta_{l,l+1}$ is a set of "upward messages," which layer l can send to layer $l+1$ ($l \in \{1,\ldots,L-1\}$), and $\theta_{l,l+1} \in \Theta_{l,l+1}$ is one such message; $\Theta_{l,l-1}$ is a set of "downward messages," which layer l can send to layer $l-1$ ($l \in \{2,\ldots,L\}$), and $\theta_{l,l-1} \in \Theta_{l,l-1}$ is one such message; p is a transition probability function $p : \boldsymbol{\mathcal{S}} \times \boldsymbol{\mathcal{X}} \times \boldsymbol{\mathcal{S}} \mapsto [0,1]$; R is a reward function $R : \boldsymbol{\mathcal{S}} \times \boldsymbol{\mathcal{X}} \mapsto \mathbb{R}$; and, γ is the discount factor.

B. *Layered Value Iteration*

In this subsection, we describe the proposed offline algorithm for evaluating the optimal state-value function V^* (see (21)) and the corresponding optimal policy π^* (see (22)) without a centralized optimizer. This is done using an algorithm called *layered value iteration* (layered VI) [3], which is illustrated in Fig. 4.

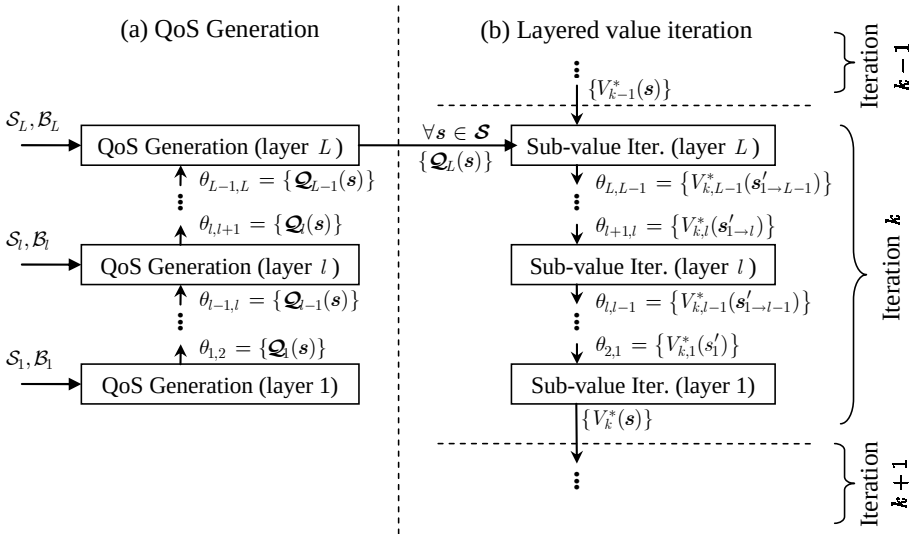

Fig. 4. Layered VI with upward and downward messages. (a) During an initial QoS generation period, upward messages are used to tell the application layer which QoS levels are supported in each system state. The set of QoS levels at layer L is required to perform layered VI. (b) Downward messages are generated in every iteration of the layered VI algorithm.

Before layered VI can be performed, layer L needs to know which QoS levels are supported by the lower layers for all system states $\boldsymbol{s} \in \boldsymbol{\mathcal{S}}$. This is achieved through a message exchange process in which *upward messages* $\theta_{l,l+1} \in \Theta_{l,l+1}$ are sent from layer l to layer $l+1$ for all $l \in \{1,\ldots,L-1\}$. The contents of these upward messages are QoS sets defined as

$$\boldsymbol{\mathcal{Q}}_l(\boldsymbol{s}) = \left\{ Q_l \middle| Q_l = \vec{F}_l(s_l, b_l, Q_{l-1}), \ldots, Q_1 = \vec{F}_1(s_1, b_1),\ \forall b_l \in \mathcal{B}_l, \ldots, \forall b_1 \in \mathcal{B}_1 \right\},\ \forall \boldsymbol{s} \in \boldsymbol{\mathcal{S}},$$

where the QoS Q_l and the QoS mapping function $\vec{F}_l(s_l, b_l, Q_{l-1})$ are defined in Section II.F. Fig. 4(a) illustrates the QoS generation period that immediately precedes layered VI. During this period, each layer $l \in \{1,\ldots,L-1\}$, starting from layer $l=1$, generates its QoS set for all $\boldsymbol{s} \in \boldsymbol{\mathcal{S}}$ and sends its upward messages $\theta_{l,l+1} = \{\mathcal{Q}_l(\boldsymbol{s})\}$ to layer $l+1$ (e.g. the HW layer tells the OS layer its available frequency and power combinations). We let the notation $\{\mathcal{Q}_l(\boldsymbol{s})\}$ denote the set of QoS values for all $\boldsymbol{s} \in \boldsymbol{\mathcal{S}}$. After the Lth layer's set of QoS levels $\boldsymbol{\mathcal{Q}}_L(\boldsymbol{s})$ is generated, the layered VI algorithm illustrated in Fig. 4(b) begins.

We derive the layered VI algorithm by substituting the factored transition probability function and the factored reward function defined in (3) and (4), respectively, into the centralized

VI algorithm defined in (23): i.e.,

$$V_k^*(\boldsymbol{s}) = \max_{\boldsymbol{\xi}\in\boldsymbol{\mathcal{X}}} \left\{ \begin{array}{l} g(\boldsymbol{s},\boldsymbol{b}) - \sum_{l=1}^{L} \omega_l^a \alpha_l(s_l,a_l) - \sum_{l=1}^{L} \omega_l^b \beta_l(s_l,b_l) + \\ \gamma \sum_{\boldsymbol{s}'\in\mathcal{S}} p\left(s_L' \mid \boldsymbol{s}, a_L, \boldsymbol{b}\right) \prod_{l=1}^{L-1} p\left(s_l' \mid s_l, a_l\right) V_{k-1}^*(\boldsymbol{s}') \end{array} \right\}. \quad (24)$$

Recall from (7) and (8) in Section II.F that the internal reward and the transition probability function at layer L can be expressed using the QoS from the lower layers, which is known by layer L because of the upward messages. The QoS based expressions allow us to change the optimization variables in (24) from the joint actions $\boldsymbol{\xi} \in \boldsymbol{\mathcal{X}}$ to $\boldsymbol{a} \in \boldsymbol{\mathcal{A}}$ and $Q_L \in \boldsymbol{\mathcal{Q}}_L(\boldsymbol{s})$. Hence, we can rewrite (24) as:

$$V_k^*(\boldsymbol{s}) = \max_{\substack{\boldsymbol{a}_{1\to L}\in\boldsymbol{\mathcal{A}}_{1\to L}, \\ Q_L\in\boldsymbol{\mathcal{Q}}_L(\boldsymbol{s})}} \left\{ \begin{array}{l} R_{\text{in}}(s_L,Q_L) - \sum_{l=1}^{L} \omega_l^a \alpha_l(s_l,a_l) + \\ \gamma \sum_{\boldsymbol{s}'_{1\to L}\in\boldsymbol{\mathcal{S}}_{1\to L}} \prod_{l=1}^{L-1} p\left(s_l' \mid s_l, a_l\right) p\left(s_L' \mid s_L, a_L, Q_L\right) V_{k-1}^*(\boldsymbol{s}') \end{array} \right\}, \quad (25)$$

where we have written the next-state vector $\boldsymbol{s}' \in \boldsymbol{\mathcal{S}}$ as $\boldsymbol{s}'_{1\to L}=(\boldsymbol{s}'_1,\ldots,\boldsymbol{s}'_L) \in \boldsymbol{\mathcal{S}}_{1\to L}$ and the joint external action vector $\boldsymbol{a} \in \boldsymbol{\mathcal{A}}$ as $\boldsymbol{a}_{1\to L}=(a_1,\ldots,a_L) \in \boldsymbol{\mathcal{A}}_{1\to L}$.

We observe that $R_{\text{in}}(s_L,Q_L) - \omega_L^a \alpha_L(s_L,a_L)$ is independent of the next-states $\boldsymbol{s}'_{1\to L-1} \in \boldsymbol{\mathcal{S}}_{1\to L-1}$ and that $\sum_{\boldsymbol{s}'_{1\to L-1}\in\boldsymbol{\mathcal{S}}_{1\to L-1}} \prod_{l=1}^{L-1} p\left(s_l' \mid s_l, a_l\right) = 1$, therefore, we can move $R_{\text{in}}(s_L,Q_L) - \omega_L^a \alpha_L(s_L,a_L)$ into the product term as follows:

$$V_k^*(\boldsymbol{s}) = \max_{\boldsymbol{a}_{1\to L-1}\in\boldsymbol{\mathcal{A}}_{1\to L-1}} \left\{ \begin{array}{l} -\sum_{l=1}^{L-1} \omega_l^a \alpha_l(s_l,a_l) + \\ \sum_{\boldsymbol{s}'_{1\to L-1}\in\boldsymbol{\mathcal{S}}_{1\to L-1}} \prod_{l=1}^{L-1} p\left(s_l' \mid s_l, a_l\right) \underbrace{\max_{\substack{\boldsymbol{a}_L\in\boldsymbol{\mathcal{A}}_L, \\ Q_L\in\boldsymbol{\mathcal{Q}}_L}} \left\{ \begin{array}{l} R_{\text{in}}(s_L,Q_L) - \omega_L^a \alpha_L(s_L,a_L) + \\ \gamma \sum_{s_L'\in\mathcal{S}_L} p\left(s_L' \mid s_L, a_L, Q_L\right) V_{k-1}^*(\boldsymbol{s}') \end{array} \right\}}_{\text{Sub-value iteration at layer } L} \end{array} \right\}, \quad (26)$$

where we have also moved the maximization over the external action set and QoS set at layer L inside the product term because the other terms are independent of these parameters. The right-hand term of (26) is the *sub-value iteration* (sub-VI) at layer L, the result of which we denote by

$V^*_{k-1}(\boldsymbol{s}'_{1\to L\text{-}1})$ for all $\boldsymbol{s}'_{1\to L-1} \in \boldsymbol{\mathcal{S}}_{1\to L-1}$:

Sub-value iteration at layer L**:**

$$V^*_{k-1}(\boldsymbol{s}'_{1\to L\text{-}1}) = \max_{\substack{\boldsymbol{a}_L\in\mathcal{A}_L,\\ Q_L\in\boldsymbol{\mathcal{Q}}_L}} \left\{ \begin{matrix} R_{\text{in}}(s_L,Q_L) - \omega^a_L \alpha_L(s_L,a_L) + \\ \gamma \sum_{s'_L\in\mathcal{S}_L} p(s'_L \mid s_L,a_L,Q_L) V^*_{k-1}(\boldsymbol{s}') \end{matrix} \right\}. \tag{27}$$

The sub-VI at layer L determines the optimal internal actions at every layer (by selecting the optimal QoS). The external actions, however, are determined by each individual layer based on local models of their dynamics (i.e. transition probability functions and reward functions).

We observe that $-\omega^a_l \alpha_l(s_l,a_l)$ is independent of the next-states $\boldsymbol{s}'_{1\to l-1} \in \boldsymbol{\mathcal{S}}_{1\to l-1}$ and that $\sum_{(\boldsymbol{s}'_{1\to l\text{-}1})\in\boldsymbol{\mathcal{S}}_{1\to l-1}} \prod_{l=1}^{l-1} p(s'_l \mid s_l,a_l) = 1$. Hence, similar to how we obtained (27) from (25), the sub-VI at layers $l = 2,\ldots,L-1$ for all $\boldsymbol{s}'_{1\to l-1} \in \boldsymbol{\mathcal{S}}_{1\to l-1}$ can be performed as follows:

Sub-value iteration at layer $l = 2,\ldots,L-1$**:**

$$V^*_{k-1}(\boldsymbol{s}'_{1\to l\text{-}1}) = \max_{\boldsymbol{a}_l\in\mathcal{A}_l} \left\{ -\omega^a_l \alpha_l(s_l,a_l) + \gamma \sum_{s'_l\in\mathcal{S}_l} p(s'_l \mid s_l,a_l) V^*_{k-1}(\boldsymbol{s}'_{1\to l}) \right\}, \tag{28}$$

where $V^*_{k-1}(\boldsymbol{s}'_{1\to l})$ (on the right hand side) is the result of the sub-VI at layer $l+1$ for all $\boldsymbol{s}'_{1\to l} \in \boldsymbol{\mathcal{S}}_{1\to l}$, which is sent as a *downward message* from layer $l+1$ to layer l, i.e. $\theta_{l+1,l} = \{V^*_{k-1}(\boldsymbol{s}'_{1\to l})\}$.

Finally, the sub-VI at layer $l = 1$ is performed as follows:

Sub-value iteration at layer 1**:**

$$V^*_k(\boldsymbol{s}) = \max_{\boldsymbol{a}_1\in\mathcal{A}_1} \left\{ -\omega^a_1 \alpha_1(s_1,a_1) + \gamma \sum_{s'_1\in\mathcal{S}_1} p(s'_1 \mid s_1,a_1) V^*_{k-1}(s'_1) \right\}, \tag{29}$$

where $\theta_{2,1} = \{V^*_{k-1}(s'_1)\}$ and $V^*_k(\boldsymbol{s})$ becomes the input to the sub-VI at layer L during iteration $k+1$.

We note that performing L sub-VIs (i.e. one for each layer) during one iteration of layered

VI is equivalent to one iteration of centralized VI. After performing layered VI, we obtain the state-value function, V_k^*, which will converge to the optimal state-value function V^* as $k \to \infty$. Subsequently, the optimal layered external and internal policies (i.e. π_l^{a*} and π_l^{b*}, respectively) can be found using (22), and can be stored in a look-up table at each layer to determine the optimal state-to-action mappings at run-time.

C. Complexity of Layered Value Iteration

It is well known that one iteration of the centralized VI algorithm has complexity $O(|\boldsymbol{\mathcal{S}}|^2|\boldsymbol{\mathcal{X}}|)$. In our layered setting, $\boldsymbol{\mathcal{S}} = \times_{l=1}^{L}\mathcal{S}_l$ is the state set and $\boldsymbol{\mathcal{X}} = \times_{l=1}^{L}\mathcal{X}_l = \times_{l=1}^{L}(\mathcal{A}_l \times \mathcal{B}_l)$ is the action set, which is comprised of external and internal action sets at each layer. Based on these definitions, $|\boldsymbol{\mathcal{S}}| = \prod_{l=1}^{L}|\mathcal{S}_l|$ and $|\boldsymbol{\mathcal{X}}| = \prod_{l=1}^{L}|\mathcal{X}_l| = \prod_{l=1}^{L}(|\mathcal{A}_l| \times |\mathcal{B}_l|)$.

In order to evaluate the complexity of the layered value iteration procedure, we must first look at the complexity of each sub-VI.

The sub-VI at layer L defined in (27) has complexity

$$Comp_L = O\left(|\boldsymbol{\mathcal{S}}||\mathcal{S}_L|\prod_{l'=1}^{L-1}|\mathcal{S}_{l'}||\mathcal{A}_L||\boldsymbol{\mathcal{Q}}_L|\right) = O\left(|\boldsymbol{\mathcal{S}}|^2|\mathcal{A}_L|\overbrace{\prod_{l'=1}^{L-1}|\mathcal{B}_{l'}|}^{|\boldsymbol{\mathcal{Q}}_L|}\right). \tag{30}$$

The sub-VIs at layers $l \in \{2,\ldots,L-1\}$ defined in (28) have complexity

$$Comp_l = O\left(|\boldsymbol{\mathcal{S}}||\mathcal{S}_l|\prod_{l'=1}^{l-1}|\mathcal{S}_{l'}||\mathcal{A}_l|\right). \tag{31}$$

Finally, the sub-VI at layer 1 defined in (29) has complexity

$$Comp_1 = O(|\boldsymbol{\mathcal{S}}||\mathcal{S}_1||\mathcal{A}_1|). \tag{32}$$

Hence, the total complexity of one iteration of the layered VI algorithm can be expressed as

$$Comp = \sum_{l=1}^{L} Comp_l\,. \tag{33}$$

We compare the centralized VI and layered VI complexities in Table I assuming that states and

actions are as defined in Table II.

Table I. Complexity of centralized value iteration and layered value iteration.

Layer	No. States	No. External Actions	No. Internal Actions
1 (Hardware)	1	1	3
2 (Operating System)	3	2	1
3 (Application)	29	3	1
Centralized VI Complexity	$O(\underbrace{\lvert\boldsymbol{\mathcal{S}}\rvert^2\lvert\boldsymbol{\mathcal{X}}\rvert}_{(29\cdot3)^2\cdot(3\cdot3\cdot2)}) = 136242$		
Layered VI Complexity	$O\left(\underbrace{\lvert\boldsymbol{\mathcal{S}}\rvert\lvert\mathcal{S}_1\rvert\lvert\mathcal{A}_1\rvert}_{(29\cdot3)\cdot1\cdot1}\right) + O\left(\underbrace{\lvert\boldsymbol{\mathcal{S}}\rvert\lvert\mathcal{S}_2\rvert\lvert\mathcal{S}_1\rvert\lvert\mathcal{A}_2\rvert}_{(29\cdot3)\cdot3\cdot1\cdot2}\right) + O\left(\underbrace{\lvert\boldsymbol{\mathcal{S}}\rvert^2\lvert\mathcal{A}_3\rvert\prod_{l'=1}^{\overbrace{2}^{\lvert\boldsymbol{\mathcal{Q}}_3\rvert}}\lvert\mathcal{B}_{l'}\rvert}_{(29\cdot3)^2\cdot3\cdot3\cdot1}\right) = 87 + 522 + 68121 = 68730$		

VI. RESULTS

In this section, we test the performance of the proposed framework using the illustrative cross-layer system described in Section III. Table II details the parameters used at each layer in our simulator, which we implemented in MATLAB. 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 sequence (30 Hz, CIF resolution, quantization parameter 24) under three different encoding configurations. The chosen encoding parameters are listed in Table II. We let one DU comprise 11 aggregated MBs, and we let each simulation comprise encoding 20000 such DUs. Since real-time encoding is not possible with the selected encoding parameters, we set the buffer drain rate to $v = 32/11$ (DUs/sec).

We note that the illustrative results presented here depend heavily on the simulation parameters defined in Table II. Nevertheless, our most important observations are about the fundamental properties of the proposed framework, which are independent of the chosen example.

Table II. Simulation parameters used for each layer.

Layer	Parameter	Value
Application Layer (APP)	Buffer State	$\mathcal{S}_{\text{APP}} = \{0,\ldots,\rho^{\max}\}$, $\rho^{\max} = 28$ (DUs)
	Parameter Configuration	$\mathcal{A}_{\text{APP}} = \{1,2,3\}$ $a_{\text{APP}} = 1$: Quarter-pel MV, 8x8 block ME $a_{\text{APP}} = 2$: Full-pel MV, 8x8 block ME $a_{\text{APP}} = 3$: Full-pel MV, 16x16 block ME
	Gain parameters	$\Omega = 40$, $\Lambda = 4$
	External Cost Weight	$\omega^{a}_{\text{APP}} = 1$
	Rate-distortion Lagrangian	$\lambda_{\text{rd}} = 1/128$
	Buffer drain rate	$v = 32/11$ (DUs/sec)
	DU size	11 Macroblocks
Operating System Layer (OS)	Resource Allocation State	$\mathcal{S}_{\text{OS}} = \{0.80, 0.55, 0.35\}$
	Resource Allocation Weight	$\mathcal{A}_{\text{OS}} = \{1,3\}$
	External Cost Weight	$\omega^{a}_{\text{OS}} = 1$
Hardware Layer (HW)	State	Constant
	Processor Frequency Actions	$\mathcal{A}_{\text{HW}} = \{200, 600, 1000\}$ Mhz
	Internal Cost Weight	$\omega^{b}_{\text{HW}} = 9 \times 10^{-27}$

A. *Equivalence of Centralized and Layered Value Iteration*

Fig. 5 illustrates the state-value function obtained using layered and centralized VI with a discount factor of $\gamma = 0.9$. Both VI algorithms yield the same state-value function, so we only show one state-value function here.

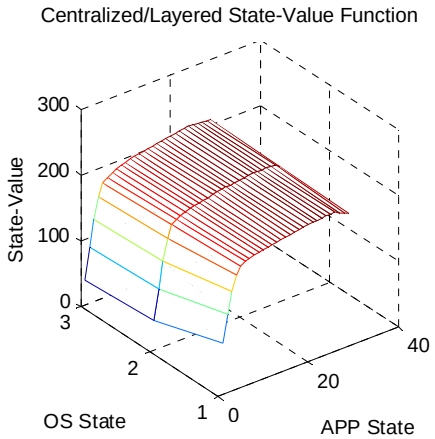

Fig. 5. State-value function obtained from centralized VI and layered VI $(\gamma = 0.9)$**.**

B. *Impact of the Discount Factor*

The discount factor γ impacts the average system performance and also the number of iterations required for the centralized and layered VI algorithms to converge. Table III illustrates the impact of the discount factor on the average reward (defined as the sum of (16) and (19))

achieved over five simulations of the centralized system and five simulations of the layered system. (Because the reward is very abstract, in Section VI.E we provide more detailed simulation results in a variety of simulation scenarios.) As expected, the results in Table III show that the centralized system and the layered system perform nearly identically. The small differences in the actual measured performance are due to random dynamics over the finite simulations (i.e. random resource allocations at the OS layer). We also observe that larger values of the discount factor improve the system's performance; however, larger values also increase the number of iterations (centralized or layered) required to find the optimal foresighted decision policy.

Table III. Impact of the discount factor on the average system reward.

Discount Factor (γ)	Number of Iterations	Avg. Reward (Centralized)	Avg. Reward (Layered)
0.00	2	1.67	1.35
0.15	15	1.66	1.76
0.30	23	6.24	6.50
0.45	33	16.95	16.97
0.60	51	17.63	17.60
0.75	87	17.83	17.78
0.90	227	17.93	17.92

C. Impact of Model Inaccuracy on the System's Performance

Using the centralized MDP or the layered MDP requires good prediction of future events in order to predict the future performance of the system. In this paper, we have assumed: (i) we know the expected reward for every possible state and action, (ii) the probability transition functions can be characterized by stationary controlled Markov chains, and (iii) the stochastic matrices describing the probability transitions are known perfectly such that the optimal policy to the MDP can be computed offline. If we relax these assumptions, it becomes necessary to analyze the affect of model inaccuracy on the overall performance. In the following analysis, we assume that model inaccuracy is due to the fact that the system's dynamics are not actually stationary and Markov.

If the state-transitions in the system are not truly stationary and Markov, then the expected reward predicted by the stationary Markov model will differ from the average reward that is actually achieved. We can quantify this model error as follows. We let $\mu^{\pi}(\boldsymbol{s})$ denote the steady-state probability of being in state $\boldsymbol{s}$ when following policy π. Using $\mu^{\pi}(\boldsymbol{s})$, we can compute the expected rewards that should be achieved if the system's dynamics are truly stationary and Markov:

$$\bar{R}^{\pi} = \sum_{\boldsymbol{s}\in\boldsymbol{\mathcal{S}}} \mu^{\pi}(\boldsymbol{s}) R(\boldsymbol{s},\pi(\boldsymbol{s})). \tag{34}$$

Now, consider an N-stage simulation of the system ($N \gg 0$), which traverses the sequence of states $\boldsymbol{s}^0, \boldsymbol{s}^1, \ldots, \boldsymbol{s}^N$ when following policy π. The average reward obtained over this simulation can be written as:

$$\bar{R}^{\pi}(N) = \frac{1}{N} \sum_{n=0}^{N-1} R(\boldsymbol{s}^n, \pi(\boldsymbol{s}^n)). \tag{35}$$

The absolute difference between the expected rewards and the N-stage average reward, $\left|\bar{R}^{\pi} - \bar{R}^{\pi}(N)\right|$, indicates how accurate the stationary Markov model is for the actual system. Large values of $\left|\bar{R}^{\pi} - \bar{R}^{\pi}(N)\right|$ indicate that the stationary Markov model is inaccurate.

Consider the following example in which we measure the impact of the model's inaccuracy on the system's performance. Fig. 6 illustrates the actual encoding complexity trace (for a fixed action) and the corresponding trace generated by a stationary model of the complexity trace. Clearly, the actual encoding complexity is not perfectly represented by the stationary model defined in (11). Because the actual complexity traffic is not stationary, the buffer transition model in (14) is not accurately represented as a stationary Markov chain. Nevertheless, the data in Table IV shows us that the predicted reward (based on the stationary Markov model of the buffer) does not differ significantly from the actual reward that is achieved when simulating the

system over $N = 20,000$ stages.

Table IV. Simulated ($N = 20,000$ stages) vs. predicted reward, PSNR, power, and resource allocation cost.

	Avg. Reward	Avg. PSNR (dB)	Avg. Power (W)	Avg. Rsrc. Alloc. Cost
Simulated	17.93	38.35	2.90	2.69
Predicted	18.91	38.40	2.44	2.98
Prediction error	0.98	0.05	0.46	0.29

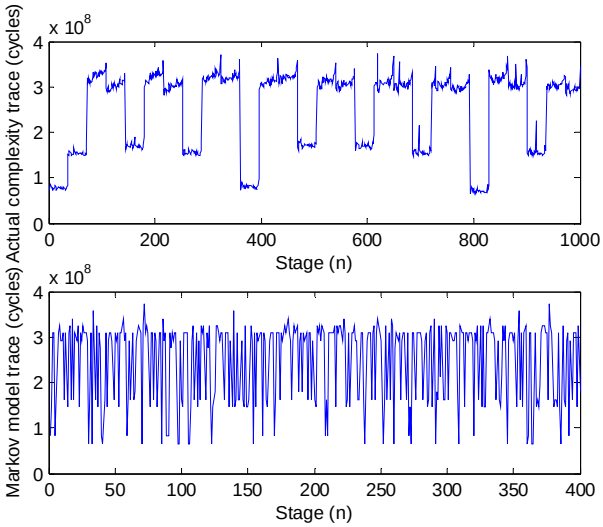

Fig. 6. Actual complexity trace (top) vs. trace generated by a stationary model (bottom). The stationary model's parameters are trained based on the actual complexity trace.

D. Myopic and Foresighted Simulation Trace Comparison

Fig. 7 shows detailed simulation traces of the APP actions, OS actions, HW actions, APP states, and OS states over time when a myopic decision policy is used ($\gamma = 0$) and when a foresighted decision policy is used ($\gamma = 0.9$). From these traces, it is clear why the myopic decision policy performs worse than the foresighted policy. We note that these traces are representative of the simulations with rewards shown in the first and last row of Table III.

First, under the myopic policy, the application receives a smaller CPU time fraction on average. This is because the immediate reward is always maximized when the OS selects the lower resource allocation weight (i.e. $\phi = 1$); therefore, the OS layer will never use a higher weight to reserve increased CPU time in the future. Meanwhile, the foresighted policy is able to

see that it is less costly to request a higher future CPU time fraction than it is to immediately increase the processor frequency.

Second, because the application receives a low fraction of the CPU time on average, the myopic policy selects the least complex, lowest quality, action (i.e. $a_{\mathrm{APP}} = 3$) and the highest processor speed (i.e. $f = 1000$ Mhz) much more frequently than in the foresighted case. In addition, despite reducing the encoding complexity and boosting the processor speed, the APP buffer repeatedly underflows. All of these factors result in the myopic policy incurring excessive costs compared to the foresighted policy.

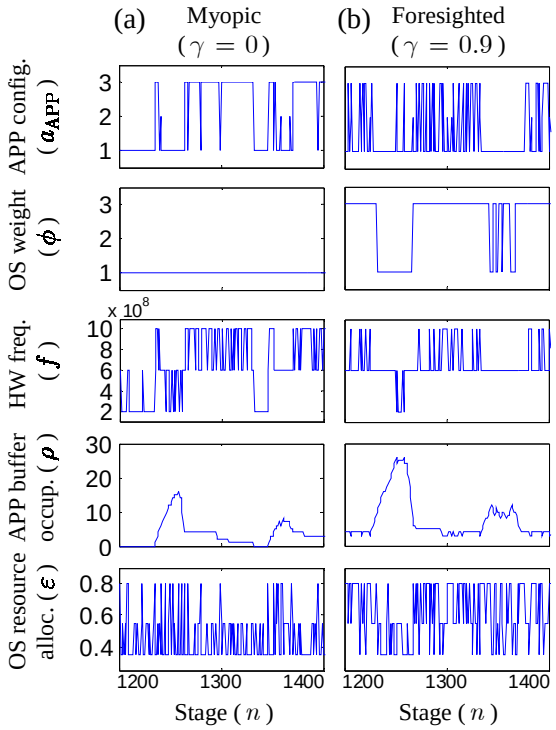

Fig. 7. Simulation traces from stage 1200 to 1400. (a) Simulation with myopic cross-layer optimization ($\gamma = 0$). (b) Simulation with foresighted cross-layer optimization ($\gamma = 0.9$).

E. Performance of Simplifications of the Proposed Framework

In Fig. 8, we illustrate the scalability of the proposed cross-layer framework by comparing the performance of the full cross-layer design to several simplified system configurations, which are similar to those that have been proposed in prior research. In this way, we also show that our

proposed framework is a superset of existing work. Detailed information about the average peak-signal-to-noise ratio (PSNR in dB), average power consumption, average resource allocation cost, and the average number of buffer overflows and underflows for each bar in Fig. 8 is shown in Table V.

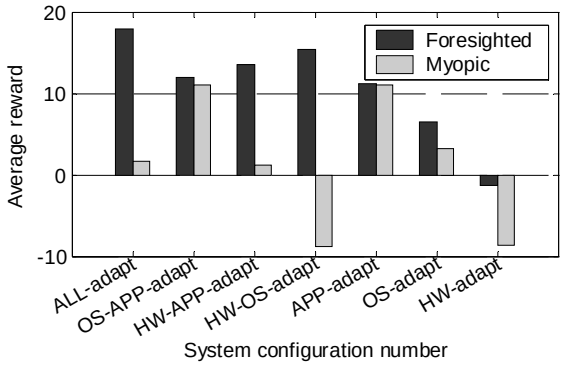

Fig. 8. Comparison of the performance of different system configurations.

Table V. Detailed simulation results for each configuration. In each column of the table, the value obtained in the foresighted case is on the left and the value obtained in the myopic case is on the right.

Configuration	Avg. PSNR (dB)		Avg. Power (W)		Avg. Resource Allocation Cost		Avg. No. Underflows		Avg. No. Overflows	
ALL-adapt	38.35	31.27	2.90	3.67	2.69	1.00	0	5516	0	0
OS-APP-adapt	36.30	36.29	9.00	9.00	1.07	1.00	0	0	810	813
HW-APP-adapt	38.16	31.13	5.36	3.60	1.00	1.00	4	5467	0	0
HW-OS-adapt	38.10	28.14	3.95	2.00	2.42	1.00	6	12874	0	0
APP-adapt	36.31	36.29	9.00	9.00	1.00	1.00	0	0	800	803
OS-adapt	34.05	32.85	9.00	9.00	1.09	1.00	2	1323	2201	2069
HW-adapt	31.40	28.15	6.52	1.99	1.00	1.00	5359	12848	0	0

In the following paragraphs we describe each system configuration used to obtain the results in Fig. 8 and Table V. For each of these configurations, the cross-layer system's parameters are defined as in Table II except where specified.

Configuration 1: ALL-adapt. Similar to [7] [8], every layer can adapt its action. As expected, this configuration achieves the highest reward out of all configurations when using a foresighted decision policy. Its myopic performance, however, is worse than several of the other configurations, which have the HW action fixed at $f = 1000$ Mhz. This initially seems counterintuitive because we would expect it to perform better given that it has more options available; however, this is only true if the best decisions are made given the additional options,

which is not the case with the myopic policy. Instead, the myopic policy frequently selects the lowest processor frequency in order to save on the immediate power costs; consequently, it is forced to aggressively encode at the highest processor frequency to avoid continually underflowing the application buffer. Due to the convexity of the power-frequency function (i.e. the HW internal cost), the average power consumed is higher than if it had just used the middle processor frequency more frequently (like the foresighted policy chooses to do).

Configuration 2: OS-APP-adapt. Under this configuration, the OS and APP layers can adapt their actions, but the HW layer's action is fixed at $f = 1000$ Mhz. We observe from Fig. 8 that this configuration's myopic policy performs nearly as good as its foresighted policy. This is because the fixed high processor frequency provides the application with enough resources to meet most of its delay constraints (i.e. avoid buffer underflows). This configuration's foresighted performance, however, is worse than that of the *ALL-adapt* configuration because a lot of power is wasted by using the high processor frequency throughout the duration of the simulation.

Configuration 3: HW-APP-adapt. Similar to [5] [6], the HW and APP layers can adapt their actions, but the OS layer's action is fixed at $\phi = 1$. The foresighted performance of this configuration is better than the myopic performance of the *ALL-adapt* configuration, even though both configurations have a static resource allocation weight at the OS layer (i.e. $\phi = 1$). This is because the foresighted policies modestly increase the processor's frequency and reduce the APP complexity before the buffer underflows. Meanwhile, the foresighted performance of this configuration is worse than that of the *ALL-adapt* configuration because it must use higher processor speeds (and therefore more power) to compensate for having a lower fraction of the CPU time on average.

Configuration 4: HW-OS-adapt. Under this configuration, the HW and OS layers can adapt

their actions, but the APP layer's action is fixed at $a_{\mathrm{APP}} = 2$. This configuration's myopic policy frequently selects the lowest processor frequency in order to save on the immediate power costs. This causes a huge number of buffer overflows, which severely degrade its performance. We note that this configuration's foresighted performance is better than that of the *HW-APP-adapt* and *OS-APP-adapt* configurations only because of the particular cost-weights chosen for our simulations (i.e. ω_l^a and ω_l^b, $\forall l \in \{1,\ldots,L\}$), which heavily penalize high CPU frequencies. In other words, different cost-weights would change the relative performance of the various configurations with two adaptive layers.

Configuration 5: APP-adapt. The APP layer can adapt its action, but the HW layer's action is fixed at $f = 1000$ Mhz and the OS layer's action is fixed at $\phi = 1$. For the same reason as in the *OS-APP-adapt* configuration, this configuration's myopic performance is nearly the same as its foresighted performance.

Configuration 6: OS-adapt. Similar to [1], the OS layer can adapt its action, but the HW layer's action is fixed at $f = 1000$ Mhz and the APP layer's action is fixed at $a_{\mathrm{APP}} = 2$. There are many overflows in this case because there are more resources allocated to the encoder than it requires, which results in DUs being encoded too quickly.

Configuration 7: HW-adapt. Similar to [9], the HW layer can adapt its action, but the OS layer's action is fixed at $\phi = 1$ and the APP layer's action is fixed at $a_{\mathrm{APP}} = 2$.

We make two final observations regarding the above results. First, we note the *HW-adapt* and *HW-OS-adapt*, *APP-adapt* and *OS-APP-adapt*, and the *HW-APP-adapt* and *ALL-adapt* configuration pairs perform nearly the same under their respective myopic policies. This is because the OS action is fixed under all myopic configurations (as we first illustrated in Fig. 7). Second, we note that buffer overflows can be avoided by increasing the size of the post encoding

buffer or by increasing the range of actions available at each layer.

VII. CONCLUSION

We have proposed a novel and general formulation of the cross-layer decision making problem in real-time multimedia systems. In particular, we modeled the problem as a layered Markov decision process, which enabled us to jointly optimize each layer's parameters, configurations, and algorithms while maintaining a separation between the decision processes and the design of each layer. Unlike existing work, which focuses on myopically maximizing the immediate rewards, we focus on *foresighted* cross-layer decisions. In our experimental results, we verified that our layered solution achieves the same performance as the centralized solution, which violates the layered architecture. Additionally, we demonstrated that our proposed framework can be interpreted as a superset of existing suboptimal multimedia system designs with one or more adaptive layers. Finally, we showed that dramatic performance gains are achievable when using foresighted optimization compared to myopic optimization.

REFERENCES

[1] J. Nieh, M.S. Lam, "The design, implementation and evaluation of SMART: a scheduler for multimedia applications," *Proc. of the Sixteenth ACM Symposium on Operating Systems Principles*, pp. 184-197, Oct. 1997.

[2] R. S. Sutton, and A. G. Barto, "Reinforcement learning: an introduction," Cambridge, MA:MIT press, 1998.

[3] F. Fu, M. van der Schaar, "A new theoretic framework for cross-layer optimization with message exchanges," *Infocom student workshop*, 2008, to appear.

[4] P. Pillai, K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *Proc. of the 18th ACM Symposium on Operating Systems Principles*, pp. 89-102, 2001.

[5] Z. He, Y. Liang, L. Chen, I. Ahmad, and D. Wu, "Power-rate-distortion analysis for wireless video communication under energy constraints," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 15, no. 5, pp. 645-658, May 2005.

[6] D. G. Sachs, S. Adve, D. L. Jones, "Cross-layer adaptive video coding to reduce energy on general-purpose processors," in *Proc. International Conference on Image Processing*, vol. 3, pp. III-109-112 vol. 2, Sept. 2003.

[7] W. Yuan, K. Nahrstedt, S. V. Adve, D. L. Jones, R. H. Kravets, "GRACE-1: cross-layer adaptation for multimedia quality and battery energy," *IEEE Trans. on Mobile Computing*, vol. 5, no. 7, pp. 799-815, July 2006.

[8] W. Yuan, K. Nahrstedt, S. V. Adve, D. L. Jones, R. H. Kravets, "Design and evaluation of a cross-layer adaptation framework for mobile multimedia systems," *Proc. of SPIE Multimedia Computing and Networking Conference*, vol. 5019, pp. 1-13, Jan. 2003.

[9] L. Benini, A. Bogliolo, G. A. Paleologo, G. D. Micheli, “Policy optimization for dynamic power management,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 6, pp. 813-833, June 1999.

[10] S. L. Regunathan, P. A. Chou, J. Ribas-Corbera, “A generalized video complexity verifier for flexible decoding,” *Proc. of the International Conference on Image Processing (ICIP 2003)*, vol. 3, pp. III-289-92 vol. 2, Sept. 2003.

[11] A. Ortega, K. Ramchandran, M. Vetterli, “Optimal trellis-based buffered compression and fast approximations,” *IEEE Trans. on Image Processing*, vol. 3, no. 1, pp. 26-40, Jan. 1994.

[12] S. Mohapatra, R. Cornea, H. Oh, K. Lee, M. Kim, N. Dutt, R. Gupta, A. Nicolau, S. Shukla, N. Venkatasubramanian, “A cross-layer approach for power-performance optimization in distributed mobile systems,” *19th IEEE International Parallel and Distributed Processing Symposium*, 2005.

[13] P. Pillai, H. Huang, and K.G. Shin, “Energy-Aware Quality of Service Adaptation,” Technical Report CSE-TR-479-03, Univ. of Michigan, 2003.

[14] Z. Ren, B. H. Krogh, R. Marculescu, “Hierarchical adaptive dynamic power management,” *IEEE Trans. on Computers*, vol. 54, no. 4, Apr. 2005.